# Geometric Approaches for 3-Dimensional Shape Approximation

by

Ria Sonecha

S.B. Electrical Engineering and Computer Science,
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023



Authored by: Ria Sonecha
Department of Electrical Engineering and Computer Science
May 12, 2023

Certified by: Russ Tedrake
Toyota Professor of EECS, Aero/Astro, MechE
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

# Geometric Approaches for 3-Dimensional Shape Approximation

by

Ria Sonecha

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

Having 3D simulation models which represent the visual geometry and contact dynamics of arbitrary objects is important for achieving robust planning and control for robotic manipulation tasks and sim2real transfer. Currently, the most common solution for obtaining such models is generating them by hand. However, this process is not generalizable or scalable. Neural Radiance Fields (NeRFs) are able to generate photorealistic 3D renderings of arbitrary objects based only on a few RGB images. 3D meshes that are extracted from NeRFs are often complex and hard to use in simulation. In this thesis we propose geometric approaches based on convex optimization for simplifying such meshes into unions of primitive shapes so that they are faster and more accurate to simulate.

Thesis Supervisor: Russ Tedrake
Title: Toyota Professor of EECS, Aero/Astro, MechE

# Acknowledgments

I'd like to start by thanking Russ Tedrake for welcoming me into his lab and being the most caring and supportive advisor I could have asked for. His thoughtful questions and insights throughout my MEng pushed me to think deeply about the problems we were working on, and I learned so much in my time at the Robot Locomotion Group. I'd also like to express my gratitude to the rest of the RLG. Their passion and curiosity for understanding challenging problems is inspiring. It has been an honor working with and learning from everyone in the lab. I especially want to thank Alex Amice for his mentorship throughout my MEng. I learned so much from our weekly meetings and I'm so grateful for the generosity with which he shared his time and knowledge.

Thank you to my parents, Vatsal and Monal, and my little brother, Rohan, for their constant love, support, and encouragement during my academic journey, and for making my dream of attending MIT a reality. Love you!

Funding for my MEng was generously provided by the Kanako Miura Fund, which was established to honor the legacy of MIT researcher Dr. Kanako Miura and "empower female students, researchers, and visiting faculty on our campus who share the same passion that Kanako had for robotics." I am honored and grateful to have been one of the beneficiaries of this fund.

# Contents

# List of Figures

# Chapter 1

# Introduction

In order to achieve robust planning and control for robotic manipulation tasks, it is crucial to be able to accurately simulate the visual geometry and contact dynamics of arbitrary objects. Currently, there is heavy reliance on handcrafted models of objects which represent their geometry and dynamics in simulation platforms. These models can take weeks to design, making it impossible to scale this solution to the number of objects that would be required for a robot to robustly handle a large variety of manipulation tasks. Furthermore, reliance on human-designed simulation models means that robots are incapable of manipulating objects they have not seen before. Instead, an ideal solution would be to automatically generate dynamically accurate simulation models from just a few unlabeled images. Another emerging use case for such "simulatable assets" is the idea of real2sim2real in which simulation models generated from real world data are used to collect large scale datasets of robots performing tasks in simulation, which can then be used to train policies on robots in the real world [21].

Previously, Neural Radiance Fields (NeRF) have been successful in creating photorealistic mesh representations of objects from a small set of input images [24]. However, while these models may be visually realistic, their underlying meshes are complex and often poor representations for accurately simulating the original object dynamics in simulation. Furthermore, the meshes generated by NeRF are generally made up of thousands of vertices and faces which makes them slow to simulate and

thus impractical to use for the desired tasks of generating robot plans or collecting data sets of trajectories for training robot policies in the real world. Our goal in this work is to use geometric approaches to simplify the meshes generated from NeRFs into unions of primitive shapes so that they are faster to simulate and have better simulation dynamics.

# Chapter 2

# Related Work

## 2.1 Neural Radiance Fields and 3D Object Representations

A Neural Radiance Field (NeRF) is a fully connected deep neural network which represents a scene or object [24]. During training, NeRFs are fed a sparse set of image and camera pose pairs. Then, at inference time, they take a camera position and viewing direction as input, and generate the volume density and emitted radiance at that location in 3D space. Classic volume rendering techniques can be used to project the output colors and densities onto an image. When trained effectively, NeRFs can produce photorealistic renderings of novel views of scenes or objects. In addition, some recent works such as [25] have utilized algorithms based on marching cubes to generate 3D meshes from NeRFs. These technologies have recently gained traction in the graphics community and are now being employed by companies like Luma AI and CSM.ai for various applications, such as generating photorealistic 3D assets for video games and e-commerce, and visual effects for movies [2, 1].

Given the tasks they are used for, meshes generated from NeRFs tend to be optimized for visual accuracy, which often results in messy, complex meshes with spurious faces. Given these artifacts, when used in simulation NeRF meshes generally do not have the same dynamic behavior as the real world object. The authors of [9] propose

augmenting NeRFs with dynamical properties which are obtained from images and videos of the object. Using synthetic data, the authors showed that this augmented object representation could be manipulated in simulation. However, in addition to the images required to train the NeRF, this approach also requires videos of the object interacting with its environment in order to generate the object representation. Additionally, this approach does not address the complexities of simulated NeRFs generated from real world data.

## 2.2 Mesh Simplification Methods

Many different types of mesh simplification and decomposition algorithms have been proposed in the past to reduce processing time and simulation complexity. These algorithms vary in their approach as well as the metrics they use to evaluate results.

One class of such algorithms is approximate convex decomposition in which a 3D shape is broken into a set of nearly convex regions in order to leverage efficient geometry processing algorithms specifically designed for convex shapes. Traditionally, these approaches use one of two metrics for their greedy decomposition algorithm – boundary-based-distance or volume-based-distance between the shape and its decomposed convex hull. As pointed out in [34], these metrics alone can be insufficient in preserving fine-grained details that are crucial for maintaining object functionality in simulation environments (eg. the slots in a toaster). To address this, the authors of [34] propose a different distance metric which simultaneously takes into account both boundary and interior distances.

Another class of algorithms are shape decomposition algorithms which instead break input shapes into a set of oriented geometric primitives such as spheres, cuboids, cylinders, and ellipsoids.

Papers such as [31] and [6] perform shape decomposition using sphere meshes and sphere trees. [31] decomposes the input shape into a collection of spheres connected by edges or faces based on a spherical quadric error metric inspired by [11]. [6] uses a medial-axis based algorithm to decompose shapes into a hierarchical sphere tree

structure.

[32] and [27] use unsupervised learning to decompose shapes into collections of cuboids. In [32] the authors formulate a geometric loss function which is a combination of the coverage loss and consistency loss. Coverage loss enforces that the input object is subsumed by the predicted parameterized shapes, while consistency loss does the opposite. Combining these losses enforces that the output shape is maximally consistent with the input. [27] uses a similar approach but the input shape is represented using an adaptive hierarchical cuboid abstraction. A loss function combining volume consistency (minimizing the difference in volume), surface consistency (minimizing the difference in surface area), mutual exclusion (minimizing cuboid overlap), abstraction compactness (minimizing the total abstraction volume), and hierarchical consistency is used to train the model.

[19] uses a supervised learning approach to fit a combination of different types of primitives to a mesh. While this method gives more general results, it requires labeled data for training.

The authors of [17] introduce a 3D object representation, which they call Fuzzy Metaballs, for differentiable rendering. Fuzzy Metaballs are an implicit representation of an object, made up of a set of general multidimensional Gaussians that form ellipsoids. Using a differentiable renderer, the authors optimize a Fuzzy Metaball representation from a set of images and show the success of using this representation to perform classic computer vision tasks such as shape from silhouette and pose estimation.

In this work we aim to use geometric reasoning, rather than learning-based methods, to fit collections of primitives to noisy meshes.

## 2.3 Point Cloud Registration

A different, but related problem to shape decomposition, is point cloud registration. In this class of problems the goal is to align two point clouds by computing the relative rotation and translation between them. A common use case for point cloud

registration is to localize an object in a scene given a model point cloud of the object of interest. This is similar to the problem in shape decomposition of finding poses for primitive shapes such that they align with the vertices of a mesh. Other applications of point cloud registration include robot localization, lidar scan matching, and mapping.

Works such as [38] and [14] use optimization methods and tight convex relaxations to find globally optimal and certifiable registration results. More specifically, [38] uses a truncated least squares objective function to handle outliers and decouples solving for the relative scale, rotation, and translation. Using a tight semidefinite relaxation of the rotation constraint, and an adaptive voting scheme to solve for scale and translation, allows the authors to guarantee the optimality of their results. [14] takes a similar approach, but uses mixed integer optimization to solve for globally optimal poses.

Another class of algorithms for point cloud registration are iterative, sampling-based methods which tradeoff global optimality in favor of faster runtimes. Examples of such algorithms include Iterative Closest Point (ICP) [20] and Random Sample Consensus (RANSAC) [10]. ICP starts by finding the closest points between two point clouds and then computes a translation and rotation. This process is repeated until convergence. RANSAC works by sampling many subsets of data points, computing a pose for each set of points, and picking the pose for which the number of inliers is largest. RANSAC is a widely used method for tasks such as fitting curves to data with outliers, finding planar surfaces in point cloud data [40], and landmark location estimation for visual odometry and SLAM [10, 16].

These methods are generally more computationally efficient than their globally optimal counterparts, but they are susceptible to getting stuck in local minima and finding suboptimal solutions.

## 2.4 Convex Optimization

Convex optimization, used in both [38] and [14] for point cloud registration, is a branch of mathematical optimization where both the cost function and constraints are convex functions of the decision variables. The most commonly solved convex optimization in practice are conic programs which include Linear Programs (LPs), Quadratic Programs (QPs), Second Order Cone Programs (SOCPs), and Semidefinite Programs (SDPs). For convex optimization problems, globally optimal solutions can be found by polynomial time algorithms [3].

A more complex class of optimization problems are Mixed Integer Programs (MIPs), in which some or all of the decision variables must be integers. MIPs are a powerful tool for modeling discrete decisions in optimization problems such as whether a data point is an inlier or outlier. In the worst case finding a solution to a MIP requires searching through an exponentially large search space, making it NP-hard in the number of discrete variables [39].

Despite these hardness results, effective strategies for solving MIPs in practice have been developed. Most commonly MIPs are solved using the branch and bound algorithm which recursively partitions the discrete search space into smaller subproblems, successively bounding the objective function within each subproblem, and selectively exploring other subproblems until an optimal solution is found or proven to be impossible [4]. In some cases branch and bound is able to eliminate large portions of the search tree and efficiently find optimal solutions to MIPs. However, this is not always the case and even with branch and bound, solving some MIPs can be computationally intensive and impractical at large scales.

# Chapter 3

# Optimization Methods for Fitting Primitives

## 3.1 Background

The problem of fitting primitive shapes to a mesh is similar to a version of the point cloud registration problem where the goal is to localize a known object in a scene by aligning a point cloud model of the object to a point cloud of the scene. In the case of mesh simplification, instead of a model point cloud we have a geometric primitive such as a sphere, ellipsoid, or box, and instead of a scene point cloud we have the vertices of the input mesh.

In standard point cloud registration without outliers the relative scale $s$, translation $b$, and rotation $R$ can be computed by solving the least squares problem which minimizes the distance between the transformed scene points $x_i$ and their corresponding model points $m_i$. [38] and [14] achieve robustness to outliers by instead using a truncated least squares objective function: $\sum_{i=1}^{N} \min(\|sRx_i + t - m_i\|^2, \bar{c})$. As written this objective is non-convex and both works take different approaches to finding optimal solutions to the problem.

[38] decouples each of the transforms (scale, translation, and rotation) by taking advantage of invariant measurements between scene points and model points, and converts the non-convex truncated least squares objective into a non-convex Quadrat-

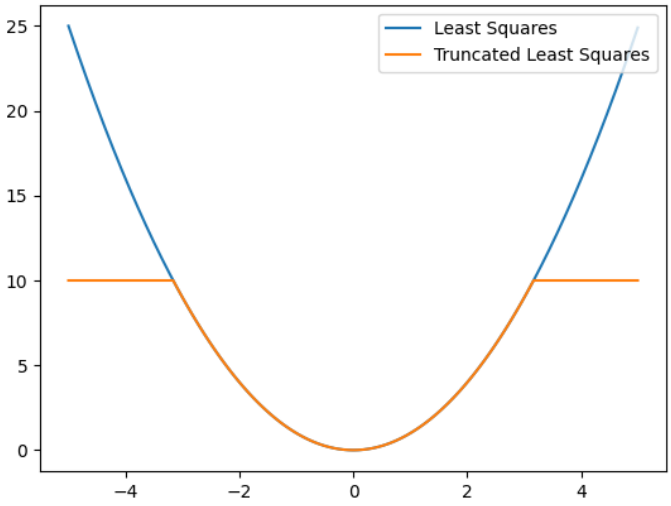



Figure 3-1: Plot showing Least Squares Cost and Truncated Least Squares Cost (with $\bar{c} = 10$) in one dimension.

ically Constrained Quadratic Program (QCQP). In practice the authors find that this QCQP has a tight semidefinite relaxation which can be solved to find certifiably optimal estimates of the relative transform.

Unfortunately, the decoupling procedure of [38] does not extend to the primitive fitting problem. Therefore, an approach based on relaxing the TLS objective using least squares necessarily relaxes a quartic objective which is less likely to have tight SDP relaxations. Instead, in this work we will generalize the approach taken in [14] for the point cloud registration problem to the primitive fitting problem. Specifically, we will formulate the truncated least squares objective as a mixed integer program.

## 3.2 Fitting Individual Primitives to Meshes

We seek to solve the problem

$$\min_{\alpha} \sum_{i=1}^{N} \|\alpha(x_i) - u_i\|^2,$$

where the points $x_i$ are vertices of the mesh, $\alpha$ is a function which transforms the points $x_i$ onto the primitive, and the points $u_i$ are points on the surface of the primitive shape. Essentially, our goal is to minimize the distance between the transformed scene points and a point on the surface of the primitive shape being fit. We parametrize the function $\alpha$ with a transformation matrix $A$ and translation vector $b$. Thus, the full optimization problem is:

$$\begin{aligned} \min_{A,b,u} \quad & \sum_{i=1}^{N} \|Ax_i + b - u_i\|^2, \\ \text{s.t.} \quad & \text{constraints on } u_i \text{ based on shape,} \\ & \text{constraints on } A. \end{aligned} \tag{3.1}$$

The presence of outliers can cause the solution to (3.1) to be biased, creating non-physical shapes. Therefore, classifying points as inlier or outliers is essential for robust, real world performance. This discrete classification decision can be modeled using binary variables, resulting in a mixed integer program similar to the point registration program described in [14]:

$$\begin{aligned} \min_{A,b,u,\phi,o} \quad & \sum_{i=1}^{N} \phi_i, \\ \text{s.t.} \quad & \phi_i \geq \|Ax_i + b - u_i\|^2 - \mathbb{M}o_i \quad \forall i, \\ & \phi_i \geq \phi_{max} o_i \quad \forall i, \\ & o_i \in \{0, 1\} \quad \forall i, \\ & \text{constraints on } u_i \text{ based on shape,} \\ & \text{constraints on } A. \end{aligned} \tag{3.2}$$

This formulation introduces $N$ new continuous decision variables $\phi_i$ which store the truncated least squares cost for each data point, and $N$ binary variables $o_i$ which indicate if a data point is an inlier ($o_i = 0$) or an outlier ($o_i = 1$). The variable $\phi_{max}$ is a threshold for classifying a point as an outlier and also acts as a penalty for outliers. The variable $\mathbb{M}$ is a sufficiently large constant which is used to turn on

and off constraints on $\phi_i$ based on the value that $o_i$ takes. Problem (3.2) is a mixed integer program which has to be solved with branch and bound.

In the following sections we show how to use this general formulation for fitting spheres, ellipsoids, and boxes to noisy input points.

### 3.2.1 Spheres

In order to transform the mesh vertices onto a sphere, $A$ is a scaled version of the identity matrix. Additionally, we require $u_i$ to be a point on the surface of the unit ball with the constraint $||u_i||^2 = 1$. This results in the following optimization problem:

$$\begin{aligned} \min_{s,b,u} \quad & \sum_{i=1}^{N} \|sx_i + b - u_i\|^2, \\ \text{s.t.} \quad & \|u_i\|^2 = 1 \quad \forall i, \\ & s \geq 0. \end{aligned} \tag{3.3}$$

The magnitude constraint on $u_i$ in program (3.3) is not convex, so we relax it to $\|u_i\|^2 \leq 1$ which can be implemented as a convex second-order-cone constraint. This leads to the convex SOCP optimization problem below:

$$\begin{aligned} \min_{s,b,u} \quad & \sum_{i=1}^{N} \|sx_i + b - u_i\|^2, \\ \text{s.t.} \quad & \|u_i\| \leq 1 \quad \forall i, \\ & s \geq 0. \end{aligned} \tag{3.4}$$

However, the convex relaxation on the $u_i$ constraint means that data points can be mapped to any point within the unit sphere, not just points on the surface of the unit sphere. This includes the trivial solution where $u = s = b = 0$.

A more sophisticated relaxation of (3.3) involves semidefinite programming. First,

we rewrite the quadratic cost in the following form:

$$\sum_{i=1}^{N} \|sx_i + b - u_i\|^2 = \sum_{i=1}^{N} \begin{bmatrix} s \\ b \\ u_i \end{bmatrix}^T Q_i \begin{bmatrix} s \\ b \\ u_i \end{bmatrix}. \tag{3.5}$$

Then, the relaxed program is:

$$\begin{aligned} \min_{s,b,u,Z} \quad & \sum_{i=1}^{N} \mathbf{TR}(Q_i Z_i,) \\ \text{s.t.} \quad & Z_i \succeq \begin{bmatrix} s \\ b \\ u_i \end{bmatrix} \begin{bmatrix} s \\ b \\ u_i \end{bmatrix}^T \quad \forall i, \\ & \mathbf{TR}(Z_{i,[-3:,-3:]}) = 1 \quad \forall i, \\ & \mathbf{TR}(Z_{i,[0:-3,0:-3]}) = \mathbf{TR}(Z_{j,[0:-3,0:-3]}) \quad \forall i, j, \\ & s \geq 0. \end{aligned} \tag{3.6}$$

However, without a term that is purely linear in the decision variables $s, b, u$, there is no cost to pushing optimal solutions towards $u_i^T u_i = 1$, resulting in solutions similar to those given by the SOCP (3.4) [29].

As written, both the SOCP and SDP relaxations of the program are loose unless we fix $s$, in which case the relaxations are tight but do not solve for the optimal $s$. However, we can do better than the above formulation by noticing that a point $p$ is on the surface of a sphere parametrized by center $c$ and $r$ if $\|p - c\| = r \implies \|p - c\|^2 = r^2 \implies \|p - c\|^2 - r^2 = 0$. Thus, we can reformulate the optimization problem as a penalty on mesh vertices $p$ being far from the surface of the fit sphere:

$$\begin{aligned} \min_{c,r} \quad & \sum_{i=1}^{N} (\|p_i - c\|^2 - r^2)^2, \\ \text{s.t.} \quad & r \geq 0. \end{aligned} \tag{3.7}$$

As written this problem has a quartic objective which is not convex. However, as

shown in [15] the objective can be expanded and rearranged to the following expression:

$$\begin{aligned}&\sum_{i=1}^{N}((p_{ix}^2+p_{iy}^2+p_{iz}^2)-(2p_{ix}c_x+2p_{iy}c_y+2p_{iz}c_z+r^2-c_x^2+c_y^2+c_z^2))^2\\=&\sum_{i=1}^{N}((2p_{ix}c_x+2p_{iy}c_y+2p_{iz}c_z+r^2-c_x^2+c_y^2+c_z^2)-(p_{ix}^2+p_{iy}^2+p_{iz}^2))^2.\end{aligned} \tag{3.8}$$

This expression can then be converted to vector notation:

$$\sum_{i=1}^{N}(a_i^TQ-f_i)^2,$$

$$\text{where: } a_i=\begin{bmatrix}2p_{ix}\\2p_{iy}\\2p_{iz}\\1\end{bmatrix},\quad Q=\begin{bmatrix}c_x\\c_y\\c_z\\r^2-(c_x^2+c_y^2+c_z^2)\end{bmatrix},\quad f_i=\begin{bmatrix}p_{ix}^2+p_{iy}^2+p_{iz}^2\end{bmatrix}.$$

Thus, the new problem can be written with a standard least squares objective, where vectors $a_i$ and $f_i$ are pre-computed from the data, and the only decision variable is $Q$:

$$\min_{Q}\quad\sum_{i=1}^{N}(a_i^TQ-f_i)^2. \tag{3.9}$$

This optimization problem is now a least squares program which has a unique and non-trivial solution for $Q$. The first three elements of the optimal $Q$ contain the coordinates of the center of the fit sphere, and can be used to recover the radius of the sphere from the fourth element of $Q$.

It is worth noting that the objective function for the new formulation is no longer exactly the squared distance between data points and the surface of the sphere. Instead, it is the difference between the squared distance between a point and the

center of the sphere, and the radius of the sphere squared. However, the benefit of this modified formulation is that we avoid the loose convex relaxation $\|u_i\|^2 \leq 1$.

Adding outlier rejection we have the following mixed integer quadratic program (MIQP):

$$\begin{aligned} \min_{Q,\phi,o} \quad & \sum_{i=1}^{N} \phi_i, \\ \text{s.t.} \quad & \phi_i \geq (a_i^T Q - f_i)^2 - \mathbb{M} o_i \quad \forall i, \\ & \phi_i \geq \phi_{max} o_i \quad \forall i, \\ & o_i \in \{0,1\} \quad \forall i. \end{aligned} \tag{3.10}$$

In practice it may be possible to modify the MIQP above so that it is easier to find solutions. Since the truncated least squares cost for this program is now quadratic in the decision variable $Q$, the SDP relaxation is more likely to be tight and result in optimal solutions. Alternatively, this optimization problem can be converted to a mixed integer linear program (MILP) by switching from the squared cost $(a_i^T Q - f_i)^2$ to the absolute value $|a_i^T Q - f_i|$. In practice, MILPs can be solved much faster than MIQPs due to the efficiency of simplex warm starting the branch and bound algorithm [28]. The MILP formulation is given below:

$$\begin{aligned} \min_{Q,\phi,o} \quad & \sum_{i=1}^{N} \phi_i, \\ \text{s.t.} \quad & \phi_i \geq +(a_i^T Q - f_i) - \mathbb{M} o_i \quad \forall i, \\ & \phi_i \geq -(a_i^T Q - f_i) - \mathbb{M} o_i \quad \forall i, \\ & \phi_i \geq \phi_{max} o_i \quad \forall i, \\ & o_i \in \{0,1\} \quad \forall i. \end{aligned} \tag{3.11}$$

### 3.2.2 Ellipsoids

We can take a similar approach to fitting ellipsoids as we used for fitting spheres. The general quadratic form $ax^2 + by^2 + cz^2 + 2dxy + 2exz + 2fyz + 2gx + 2hy + 2iz = 1$ describes an ellipsoid if all of the discriminants are less than zero:

$$
\begin{aligned}
(2d)^2 - 4ab < 0, \\
(2e)^2 - 4ac < 0, \\
(2f)^2 - 4bc < 0.
\end{aligned}
\tag{3.12}
$$

In vector notation this can be written as:

$$
a_i^T Q - 1 = 0,
$$

$$
\text{where: } a_i = \begin{bmatrix} x_i^2 \\ y_i^2 \\ z_i^2 \\ 2x_iy_i \\ 2x_iz_i \\ 2y_iz_i \\ 2x_i \\ 2y_i \\ 2z_i \end{bmatrix}, \quad Q = \begin{bmatrix} a \\ b \\ \vdots \\ h \\ i \end{bmatrix}.
$$

Now we write the problem of fitting ellipsoids as a minimization of $(a_i^T Q - 1)^2$ for all data points:

$$
\begin{aligned}
\min_{Q} \quad & \sum_{i=1}^{N} (a_i^T Q - 1)^2, \\
\text{s.t.} \quad & (2d)^2 - 4ab < 0, \\
& (2e)^2 - 4ac < 0, \\
& (2f)^2 - 4bc < 0.
\end{aligned}
\tag{3.13}
$$

As written here the objective function is convex but each of the discriminant constraints is non-convex due to the bi-linear terms. Empirically, we find that solutions to the unconstrained optimization problem almost always result in solutions that sat-

isfy the discriminant constraints if there are at least 20 data points. Thus it can be solved as a least squares problem and the solutions can be verified after the fact. This also holds for the formulation with outliers:

$$\begin{aligned}
\min_{Q,\phi,o} \quad & \sum_{i=1}^{N} \phi_i, \\
\text{s.t.} \quad & \phi_i \geq (a_i^T Q - 1)^2 - \mathbb{M} o_i \quad \forall i, \\
& \phi_i \geq \phi_{max} o_i \quad \forall i, \\
& o_i \in \{0, 1\} \quad \forall i, \\
& (2d)^2 - 4ab < 0, \\
& (2e)^2 - 4ac < 0, \\
& (2f)^2 - 4bc < 0.
\end{aligned} \tag{3.14}$$

A solution which guarantees ellipsoidal solutions can be achieved with an SDP. The original quadratic form can be rewritten as a matrix equation:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^T \begin{bmatrix} a & d & e & g \\ d & b & f & h \\ e & f & c & i \\ g & h & i & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0. \tag{3.15}$$

Call the matrix of coefficients $P$. In this form the discriminant constraints from (3.14) are equivalent to a positive definite constraint on $P$. Thus, an alternative optimization formulation is:

$$\begin{aligned}
\min_{P} \quad & \sum_{i=1}^{N} \begin{bmatrix} a_i, \\ 1 \end{bmatrix}^T P \begin{bmatrix} a_i \\ 1 \end{bmatrix} \\
\text{s.t.} \quad & P_{[0:3,0:3]} \succeq 0.
\end{aligned} \tag{3.16}$$

And with outliers:

$$\begin{aligned} \min_{P,\phi,o} \quad & \sum_{i=1}^{N} \phi_i, \\ \text{s.t.} \quad & \phi_i \geq \begin{bmatrix} a_i \\ 1 \end{bmatrix}^T P \begin{bmatrix} a_i \\ 1 \end{bmatrix} - \mathbb{M} o_i \quad \forall i, \\ & \phi_i \geq \phi_{max} o_i \quad \forall i, \\ & P_{[0:3,0:3]} \succeq 0, \\ & o_i \in \{0,1\} \quad \forall i. \end{aligned} \tag{3.17}$$

With this formulation the solutions that are found are guaranteed to be ellipsoidal for any number of data points, but solving SDPs and mixed integer SDPs is harder than the quadratic program proposed earlier and requires specialized solvers.

### 3.2.3 Boxes

In order to fit boxes we return to formulation (3.1), where for each scene point $x_i$, we search for the closest corresponding model point $u_i$. Unlike the sphere and ellipsoid cases, for boxes we can use linear constraints to constrain $u_i$.

We start by presenting the box fitting problem in two dimensions. In this case the model we are fitting is the square defined by the vertices $(-1,-1), (1,-1), (1,1), (-1,1)$, and $u_i$ must be exactly on one of the sides of this square. Each side is a closed line segment which can be represented by the expression $e_2 + \theta(e_1 - e_2)$ where $e_1$ and $e_2$ are the endpoints of the segment, and $\theta$ is a scalar variable which ranges from 0 to 1 [5]. Thus, $u_i$ can be represented using one binary variable per side to assign which side $u_i$ belongs to, and a scalar variable ranging from 0 to 1, to determine where along the chosen side $u_i$ lies. This mixed integer program is shown below:

$$
\begin{aligned}
\min_{A,b,\phi,\gamma} \quad & \sum_{i=1}^{N} \phi_i, \\
\text{s.t.} \quad & \phi_i \geq \|Ax_i + b - (e_{2,j} + \theta_{ij}(e_{1,j} - e_{2,j}))\|^2 - \mathbb{M}(1 - \gamma_{i,j}) \quad \forall i \in [1, N], \forall j \in [1, 4], \\
& 0 \leq \theta_{ij} \leq 1 \quad \forall i \in [1, N], \forall j \in [1, 4], \\
& \sum_{j=1}^{4} \gamma_{i,j} \geq 1 \quad \forall i \in [1, N], \quad \sum_{i=1}^{N} \gamma_{i,j} \geq 1 \quad \forall j \in [1, 4], \\
& \gamma_{ij} \in \{0, 1\}, \\
& A = \begin{bmatrix} a_{00} & -a_{01} \\ a_{10} & a_{11} \end{bmatrix}, \quad a_{00}, a_{11} \geq \epsilon, \quad a_{01}, a_{10} \geq 0.
\end{aligned}
\tag{3.18}
$$

In this formulation $e_{1,j}$ and $e_{2,j}$ represent the two endpoints that define side $j$. Since we are working with a box there are always 4 sides. The binary variable $\gamma_{i,j}$ is 1 if point $x_i$ is assigned to side $j$ and 0 otherwise. There are additional constraints that require each point to be assigned to at least one side, and each side to be assigned at least one point. Additionally, in order to avoid symmetries, each of the basis vectors formed by matrix $A$ are constrained to have rotation angle between 0 and 90 degrees. We do not restrict $A$ to form an orthogonal basis so shearing is allowed. This results in a more general formulation which allows us to fit parallelograms/parallelepipeds to the data, rather than just rectangles/cuboids. However, it is possible to add an additional set of constraints to the elements of $A$ to prevent shearing.

Scaling this program to 3 dimensions would require each data point to have a separate binary variable for each face, and two $\theta$ variables to parametrize where on each face $u_i$ lies. Adding 6 binary variables per data point would be very expensive, so we present an alternative formulation which only uses 4 binary variables, and 1 continuous variable per data point.

$$\begin{aligned}
\min_{A,t,u,b,z} \quad & \sum_{i=1}^{N} \|Ax_i + b - u_i\|^2, \\
\text{s.t.} \quad & b_{i,j} \in \{0,1\}, \quad u_{i,j} \in [-1,1], \quad z_i \in \{-1,1\} \quad \forall i \in [1,N], \forall j \in [x,y,z], \\
& z_i - 2b_{i,j} \leq u_{i,j} \leq z_i + 2b_{i,j} \quad \forall i \in [1,N], \forall j \in [x,y,z].
\end{aligned} \tag{3.19}$$

In order for the point $u_i$ to be on the surface of the 3D box centered at the origin with side length 2, the L-infinity norm of $u_i$ must equal 1. If the binary variable $b_{i,j}$ is 0 coordinate $j$ of point $u_i$ is constrained such that $u_{i,j} \in \{-1,1\}$, and if $b_{i,j} = 1$, then $u_{i,j} \in [-1,1]$. The binary variable $z_i \in \{-1,1\}$ helps enforce these constraints. Similar to the previous formulation, adding additional constraints limiting the axis rotation angles of the vectors formed by $A$ makes the optimization easier to solve by eliminating some symmetries.

## 3.3 Primitive Fitting Based on Greedy Set Cover

The problem of simplifying a surface mesh into the union of primitive shapes can be seen as a generalization of the set cover problem where, given a set of elements (often called the "universe") and a set of subsets, the goal is to find the smallest set of subsets that covers all the elements in the universe. In the context of simplifying a mesh the universe is the surface represented by the vertices of the input mesh, the subsets are primitive shapes, and the goal is to find the set of primitives which minimizes the cost of explaining all of the input vertices.

In general, the set cover problem is NP-hard, but if the function which measures coverage is submodular, a greedy algorithm can be used to find solutions that are approximately optimal [36]. Submodularity is a property of set functions which are functions $f(S)$ that assign a value to each subset $S$ of a finite set $V$ [18]. Submodular functions are characterized by diminishing marginal returns. Mathematically this means that a function $f(S)$ is submodular if for every $A \subseteq B \subseteq V$ and $e \in V \setminus B$:

$$\Delta(e|A) \geq \Delta(e|B),$$
$$\text{where, } \Delta(e|S) = f(S \cup \{e\}) - f(S) \quad [18]. \tag{3.20}$$

In our problem setting of fitting primitives to vertices of a mesh, the subsets $S$ are various primitive shapes fit using the outlier rejection formulations, and the coverage function $f(S)$ is the sum of the cost of the fit for each vertex. In the case where $S$ contains a single primitive, $f(S)$ is the sum of the distance between each inlier vertex and the primitive, plus a cost of $\phi_{max}$ for each outlier vertex. When a new primitive $e$ is fit to the outliers, the cost $f(S \cup \{e\})$ decreases because some subset of the outlier points will be classified as inliers for primitive $e$ and have lower cost than $\phi_{max}$. As more primitives are added, the number of outliers decreases and the marginal reduction in cost diminishes, making $f(S)$ a submodular function. Thus, using a greedy algorithm to simplify a mesh will result in approximately optimal solutions.

More specifically, the greedy algorithm requires iteratively solving for the best fit primitive with outlier rejection, removing all input data points classified as inliers, and then repeating these steps until no more input data points remain. This can be extended to fitting multiple primitives by performing the outlier rejection optimization for all primitive types, selecting the primitive with the lowest cost, removing all inlier points for that primitive, and again repeating the process until all input points have been removed. While this algorithm will give approximately optimal results, the final primitive decomposition is not guaranteed to be globally optimal.

## 3.4 Optimal Primitive Fitting Using Mixed Integer Programming

In order to avoid the sub-optimality that comes from taking the greedy approach described in the previous section, we can use mixed integer programming to solve the NP-hard problem of optimal set cover for primitive fitting. This formulation is an

extension of the k-means clustering problem [22] where our goal is to both cluster points together, and fit a primitive shape to each cluster.

In standard k-means the goal is to optimally assign a set of data points to clusters, which are defined by the location of their centroid. This is an NP-hard problem that is usually solved using an efficient iterative algorithm. This approach does not guarantee optimality. An optimal solution can be found by formulating k-means as a MIP and solving the MIP using branch and bound. [35] presents a formulation for doing this which uses binary variables to determine which cluster each data point is assigned to. Extending this formulation to both cluster points and fit the best sphere to each cluster yields the following MIP:

$$\begin{aligned}
\min_{R,\phi,b} \quad & \sum_{i=1}^{N} \phi_i, \\
\text{s.t.} \quad & \phi_i \geq (a_i^T R_j - f_i)^2 - \mathbb{M}(1 - b_{ij}) \quad \forall i \in [1, N], \forall j \in [1, K], \\
& \sum_{j=1}^{K} b_{ij} = 1 \quad \forall i \in [1, N], \\
& b_i j \in \{0, 1\} \quad \forall i \in [1, N], \forall j \in [1, K].
\end{aligned} \tag{3.21}$$

In this program, which we refer to as the "k-spheres" optimization problem, there are $N * K$ binary variables $b_{ij}$ which represent whether or not a data point $i$ is assigned to sphere $j$. Using the big-M method, constraints on the cost associated with each data point are turned on and off depending on which cluster the point is assigned to. Additionally, each data point is constrained to be assigned to exactly one cluster.

This MIP can be extended to also choose which primitive type to fit to each cluster. This requires adding at least $N * M$ additional binary variables where $N$ is the number of data points and $M$ is the number of primitive types.

## 3.5 Results

In this section we present the results of using the formulations above to fit primitive shapes to various datasets. All of the optimization programs were implemented using Drake's MathematicalProgram class and solved using the commercial optimization solver Gurobi [30, 12].

### 3.5.1 Fitting Individual Primitives

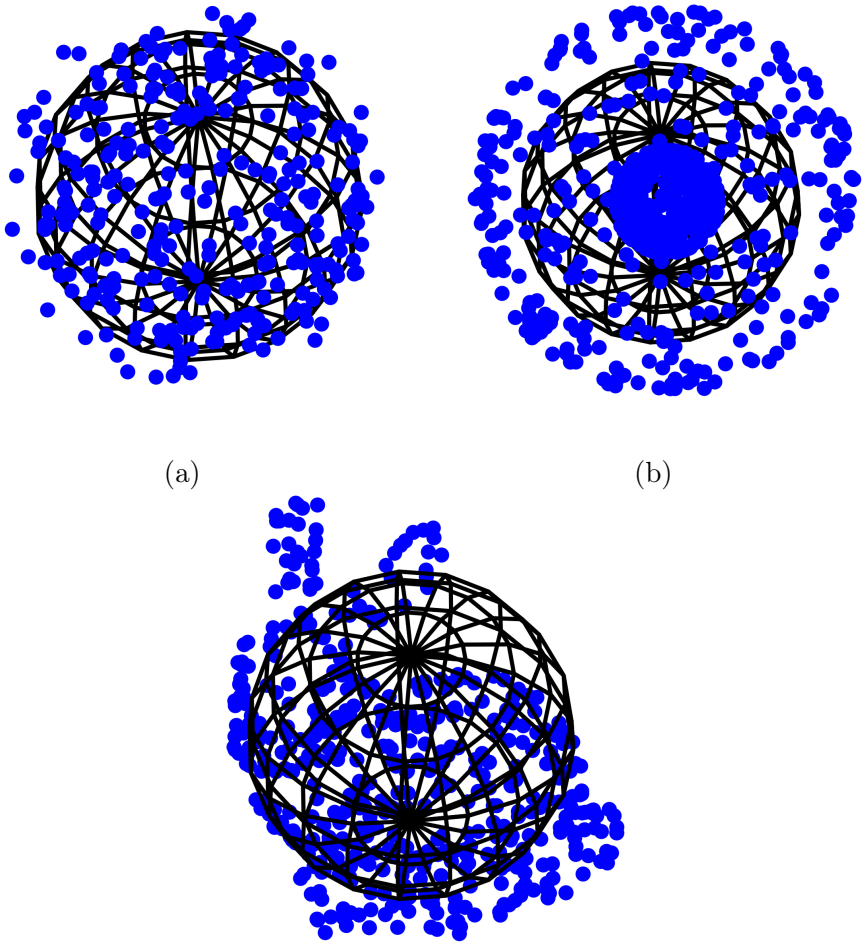
#### Spheres

Figure 3-2 shows the result of fitting spheres to various data distributions using formulation (3.9). Data points for Figure 3-2a are sampled from the surface of a sphere and perturbed with random noise. The data for the Figure 3-2b is drawn from two concentric spheres. Finally, for Figure 3-2c data is sampled from vertices of the Stanford Bunny Mesh [33]. In this case there is no outlier rejection so the optimal solution minimizes the distance between all points and the surface of the sphere.

Figure 3-3 shows the results of fitting spheres with outlier rejection. Data points classified as outliers are shown in red, while inlier data points are shown in blue. The data distributions used for the examples in Figure 3-3 are the same as those used in Figure 3-2, but due to the increased computation complexity of solving MIPs, only 20 data points are used for each example.

In, Figure 3-4 we show the time taken to solve optimization problem (3.9) for a range of data set sizes. For each data set, points are randomly sampled from the vertices of the Stanford Bunny Mesh [33]. Solving this convex, continuous optimization problem with Gurobi is fast, with average run times ranging from 0.2 to 0.4 ms. Furthermore, the solve time remains constant over data sets ranging from 100 points to 1000 points.

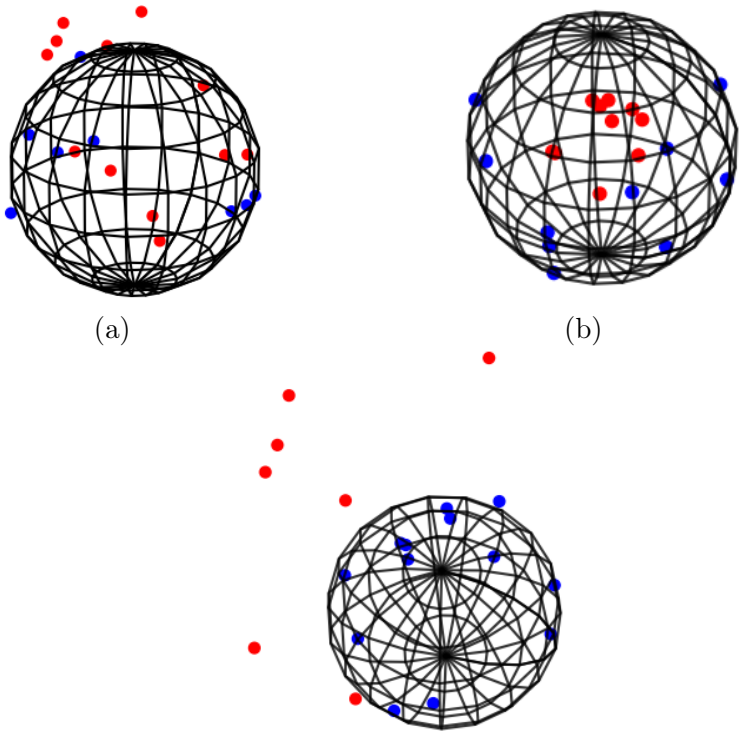
Once outlier rejection is introduced to the problem formulation in programs (3.10) and (3.11), solve times increase significantly. Figures 3-5 and 3-6 show the time taken for Gurobi to solve the MIQP and MILP for data sets ranging from 10 to 50 points. Note that the data is shown on a log scale. As the number of points increases from



(a) (b)

(c)

Figure 3-2: Fitting spheres without outlier rejection.



(a) (b)

(c)

Figure 3-3: Fitting spheres with outlier rejection. Points classified as outliers are shown in red.

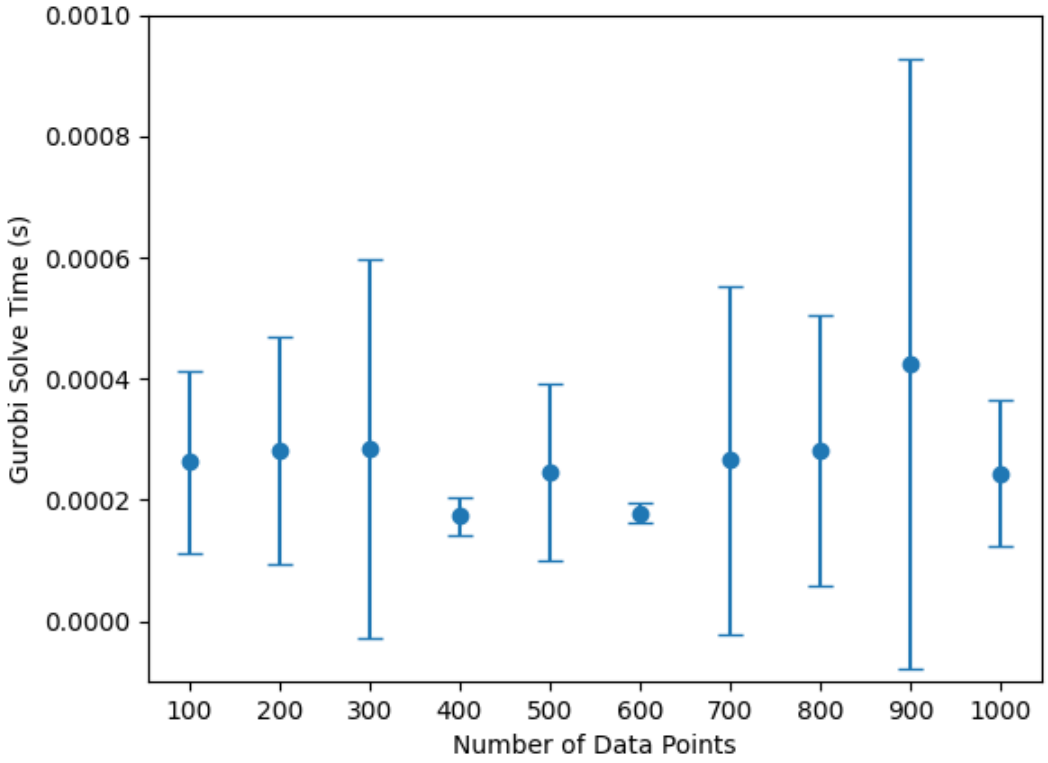



Figure 3-4: Gurobi solve times for fitting spheres without outlier rejection.

10 to 50, the solve times for the MIQP increase exponentially from an average of 0.2 seconds for 10 points to an average solve time of 3400 seconds (57 minutes) for 50 data points. For the MILP the solve times are much lower than the MIQP, but they still increase exponentially with the number of data points. For 10 points the average solve time is 0.015 seconds and this number increases to 190 seconds for 50 data points. With these mixed integer formulations, each new data point requires one new binary variable, thus increasing the branch and bound search space exponentially for every additionally data point.

### Ellipsoids

Examples of fitting ellipsoids with and without outlier rejection are shown in Figures 3-7 and 3-8. In both figures data for the first ellipsoid is drawn from the surface of a noisy ellipsoid and data for the second example is randomly sampled from the

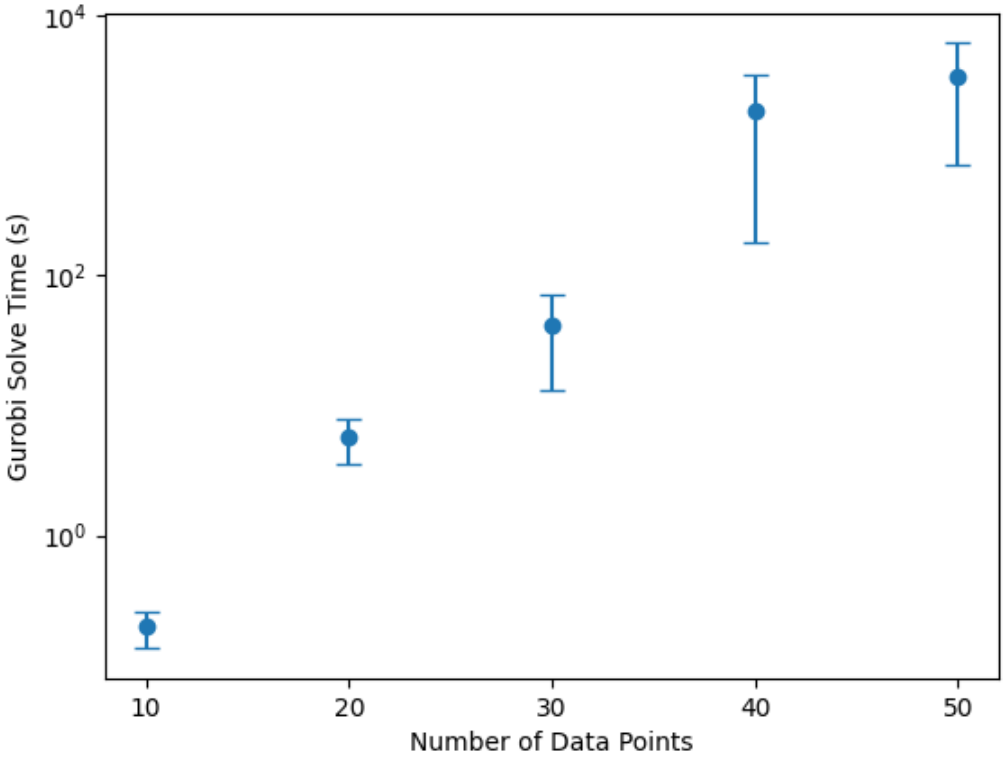



Figure 3-5: Gurobi solve times for fitting spheres with outlier rejection using the MIQP (3.10).

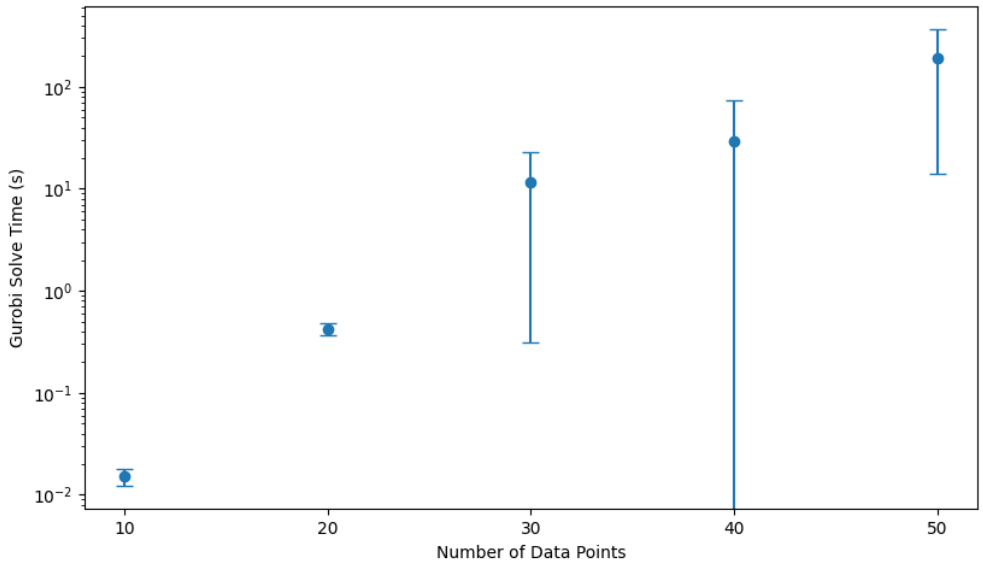


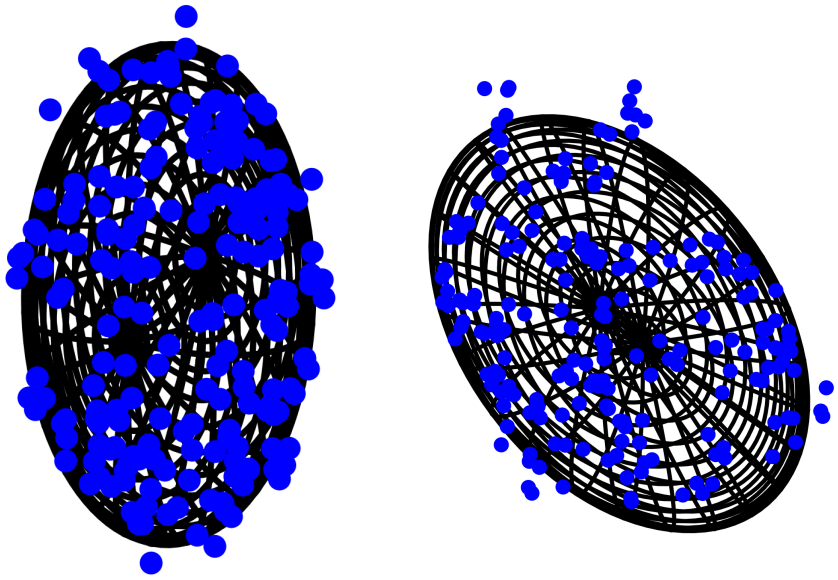

Figure 3-6: Gurobi solve times for fitting spheres with outlier rejection using the MILP (3.11).

Stanford Bunny Mesh [33].

The Gurobi solve times for fitting ellipsoids with and without outliers are shown in Figures 3-9 and 3-10. With no outlier rejection the average time to fit an ellipsoid ranges from about 0.2 to 0.7 ms, but the solve times do not scale with the number of data points. For the outlier rejection case, on the other hand, each additional data point requires a corresponding binary variable so the time to solve for an ellipsoid increases exponentially with the number of data points.

### Boxes

Figures 3-11 and 3-12 show the results of fitting boxes in two dimensions with and without outlier rejection. The mixed integer program for fitting boxes without outlier rejection requires 4 binary variables for each data point, while the program with outlier rejection uses 5 binary variables per data point. Thus, in the case of fitting boxes the computational complexity for both versions increases exponentially with the number of data points.



(a) (b)

Figure 3-7: Fitting ellipsoids without outlier rejection.

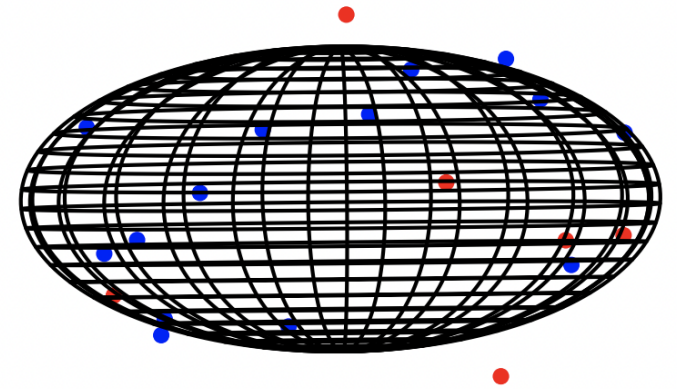
### 3.5.2 K-spheres Fitting

Figure 3-13 shows the results of fitting two spheres to different data distributions using the k-spheres program (Formulation (3.21)). The data for Figure 3-13a is drawn from two concentric spheres, the data for Figure 3-13b comes from two non-concentric spheres, and the data for Figure 3-13c is randomly sampled from the Stanford Bunny Mesh [33].

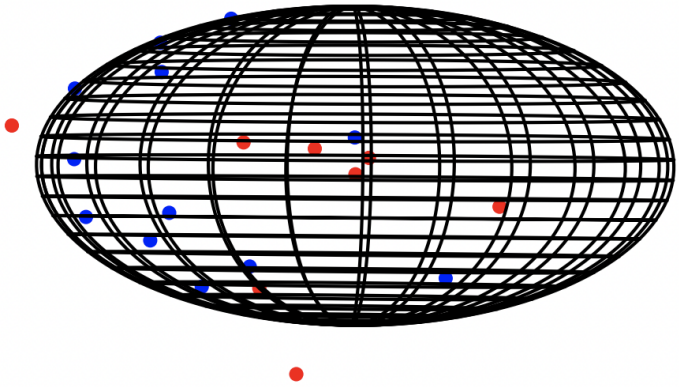
Figure 3-14 shows the time taken to solve the K-spheres problem for fitting two spheres to data sets of various sizes. As expected, the computational complexity increases exponentially with the number of data points because each additional data point requires one binary variable for each sphere being fit.

## 3.6 Discussion

For all of the optimization formulations presented in this section we can find globally optimal solutions. However, as discussed earlier, finding optimal solutions to mixed integer programs is NP-hard and the possible solution space grows exponentially with



(a)



(b)

Figure 3-8: Fitting ellipsoids with outlier rejection. Points classified as outliers are shown in red.

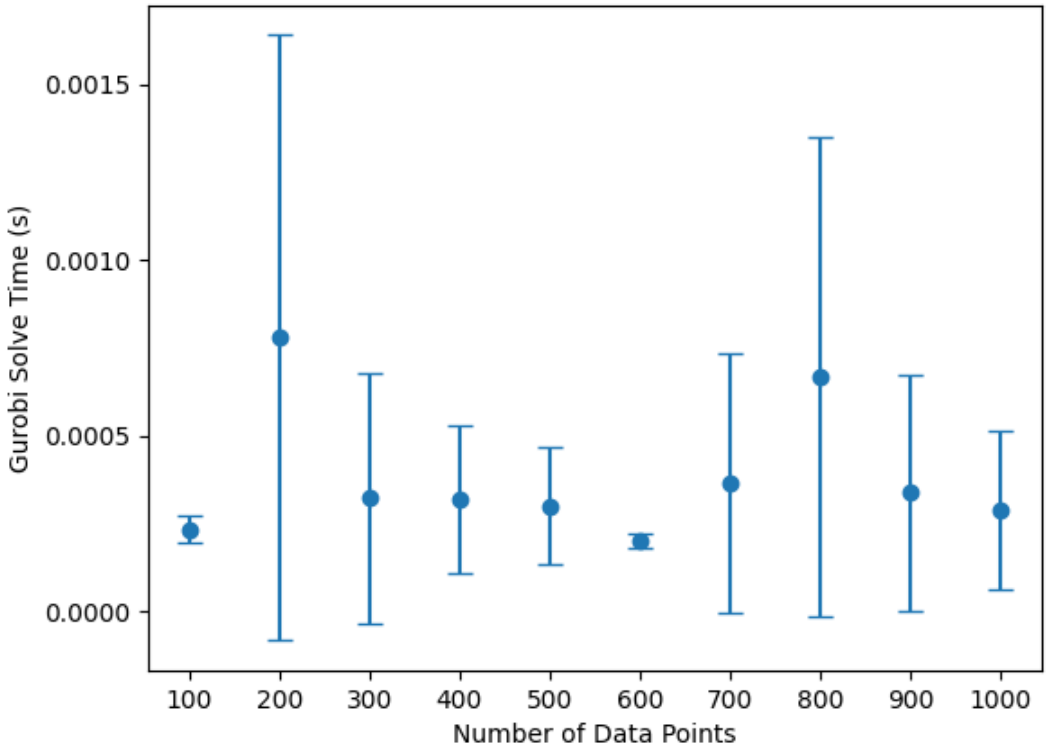



Figure 3-9: Gurobi solve times for fitting ellipsoids without outlier rejection.

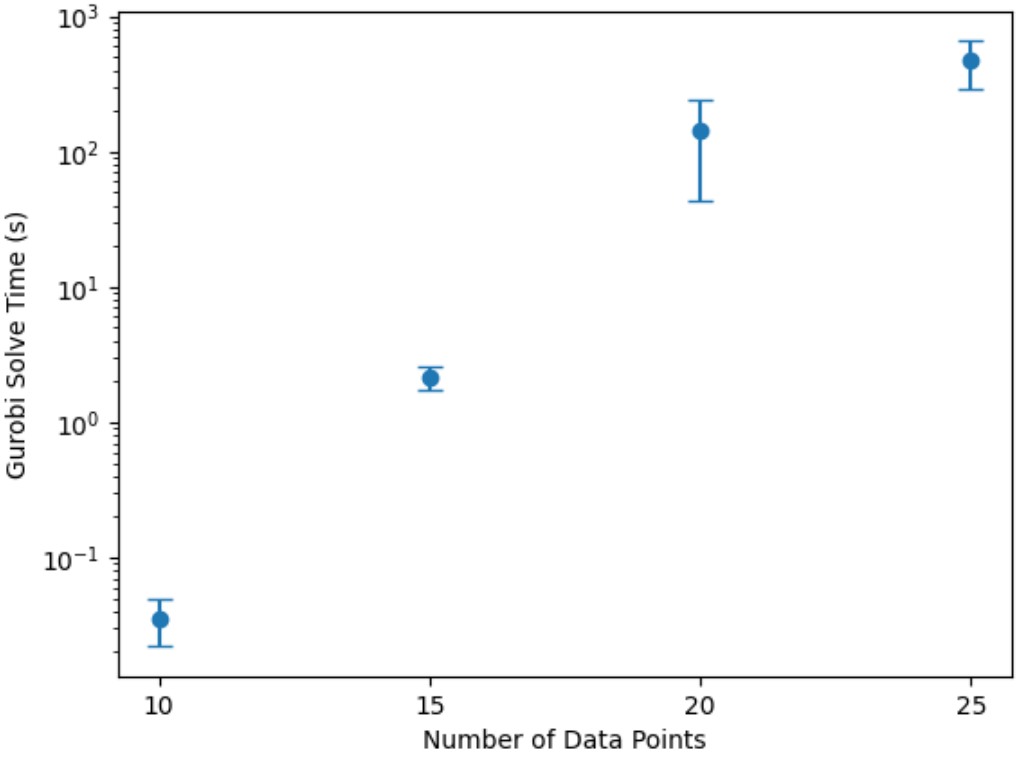
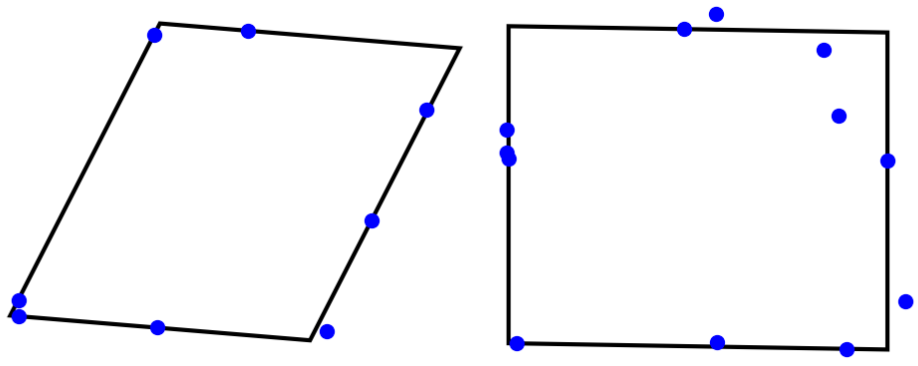
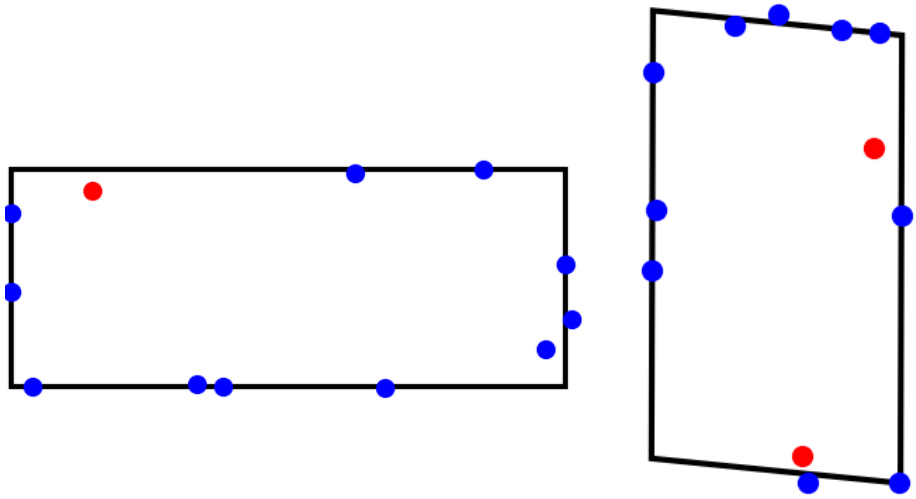



Figure 3-10: Gurobi solve times for fitting ellipsoids with outlier rejection.



(a) (b)

Figure 3-11: Fitting boxes without outlier rejection.



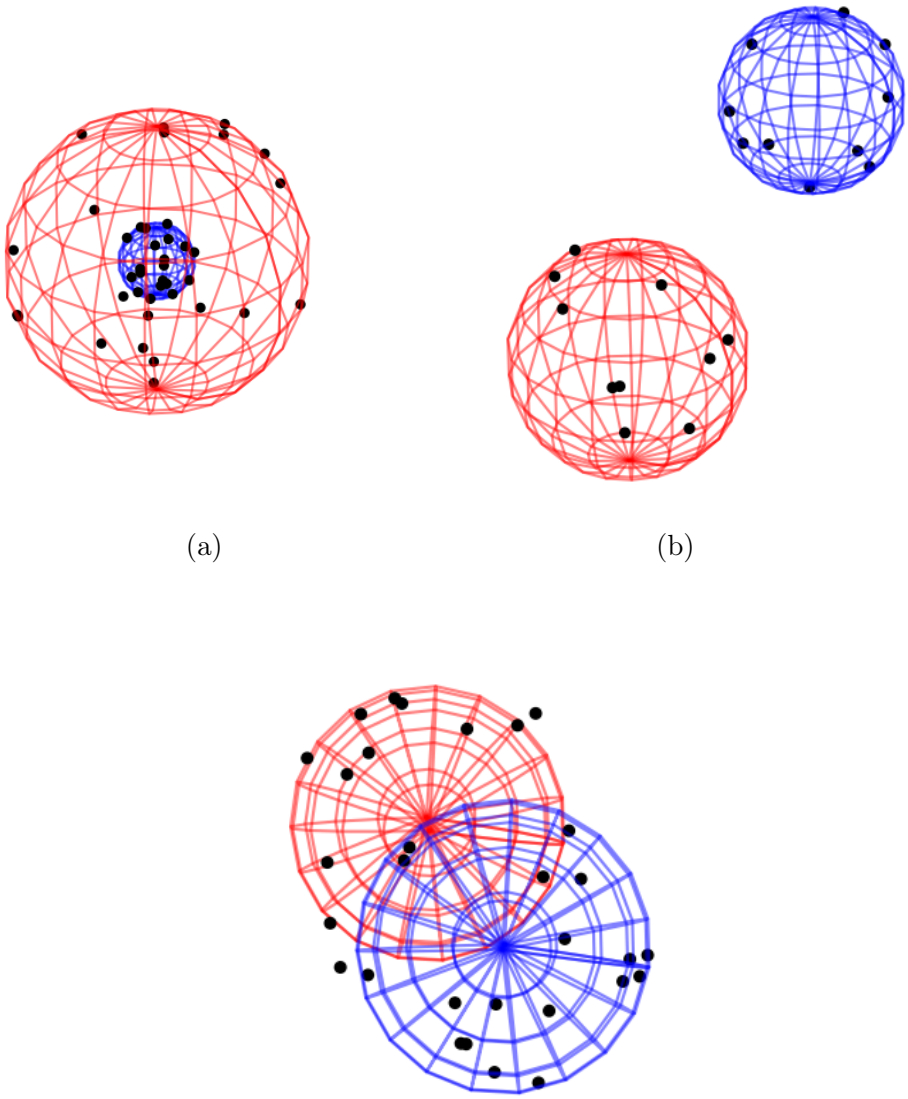
(a) (b)

Figure 3-12: Fitting boxes with outlier rejection. Points classified as outliers are shown in red.

the number of integer variables. As seen in Figures 3-5, 3-10, and 3-14, solving large mixed integer programs quickly becomes impractical.

Some of the computational difficulty of the mixed integer programs presented in this chapter comes from the presence of symmetries which result in symmetric optimal solutions which cannot be pruned by branch and bound. For example, in the K-spheres problem, the solution which assigns data points $1, \ldots, M$ to sphere 1, and $M+1, \ldots, N$ to sphere 2 is equivalent to the solution which assigns data points $1, \ldots, M$ to sphere 2, and $M+1, \ldots, N$ to sphere 1. However, branch and bound must explore both options before returning one of the them as the optimal solution.

Additionally, with this class of problems there are geometric heuristics that influence which types of solutions and binary assignments are likely to result in optimal solutions. For example, when fitting a single primitive with outlier rejection, it is likely that two points that are close to each other in Euclidean distance have the same outlier assignment. Similarly, in the K-spheres problem it is unlikely that points that are far away in Euclidean distance would be assigned to the same sphere. Encoding such heuristics in the branching function for branch and bound could help the algorithm



(a) (b)

(c)

Figure 3-13: Simultaneously fitting two spheres to different data distributions.

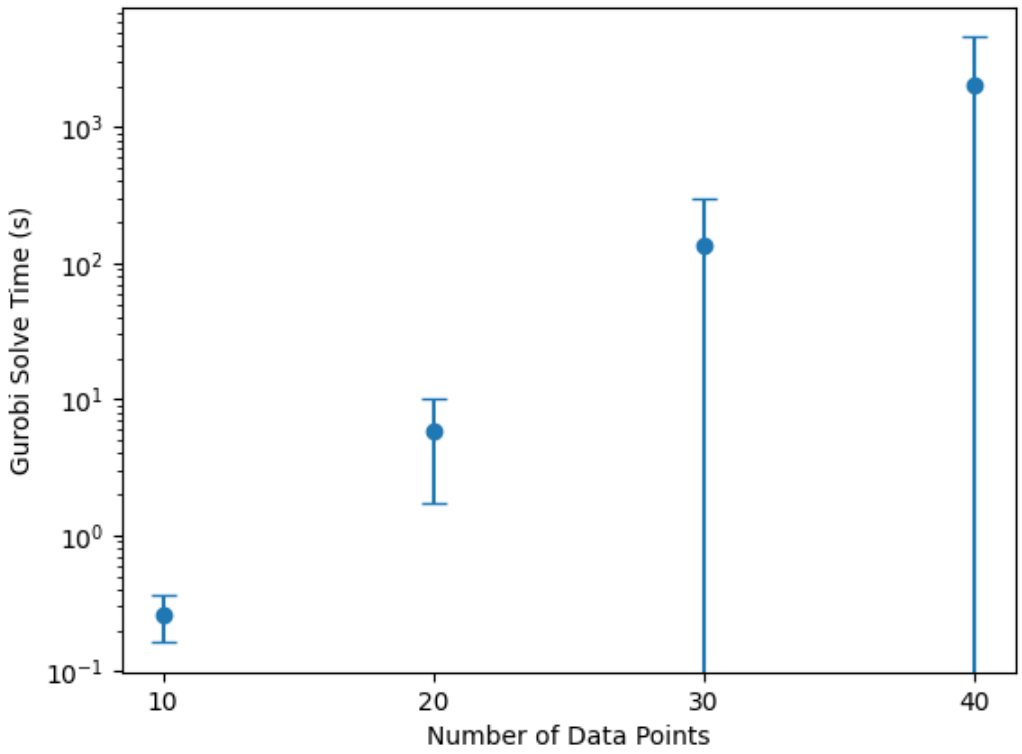



Figure 3-14: Gurobi solve times for fitting two spheres to a data set using Formulation (3.21).

explore the solution tree more efficiently and find optimal solutions faster. However, even with these strategies the optimization methods are hard to scale to the size of meshes like the Stanford Bunny which has 35947 vertices.

In Chapter 4 we present an alternative solution which uses a sampling based algorithm which is less computationally intensive, but does not have the same optimality guarantees given by the convex optimization formulations.

# Chapter 4

# Sampling-based Method for Fitting Primitives

Although the optimization-based techniques discussed in the previous chapter can ensure globally optimal solutions, their computational complexity makes them difficult to scale to the high number of vertices found in typical meshes. An alternative strategy is to reduce the computational complexity by using a sampling-based method at the cost of giving up guaranteed optimality. In this chapter we present a method for using the Random Sample Consensus (RANSAC) paradigm to fit primitives to meshes [10].

## 4.1 Algorithm Description

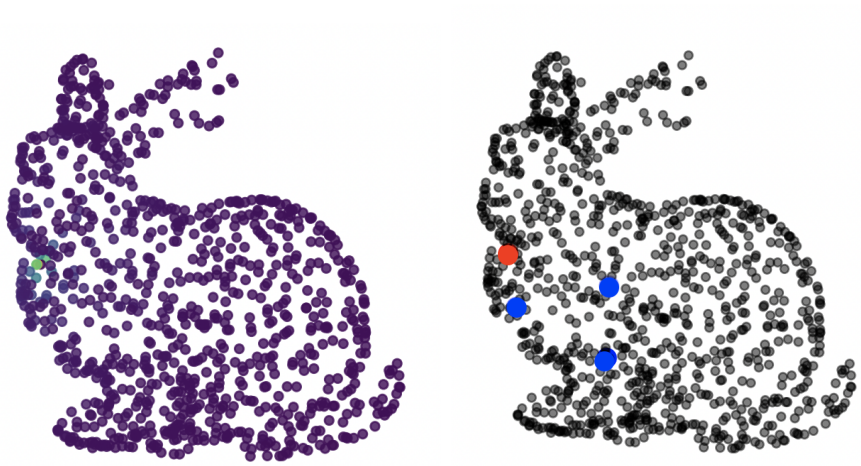
RANSAC provides an alternative to the optimization methods in Section 3.2 for fitting primitives in the presence of outliers. Rather than searching for the optimal assignment of outliers and inliers using a MIP, we can use RANSAC to implicitly perform outlier rejection by generating many candidate shape parameters and choosing the one that maximizes consensus. By using RANSAC to fit individual shapes we can scale the greedy mesh decomposition approach described in Section 3.3 to many more input mesh vertices.

The RANSAC algorithm for fitting a single primitive works by first sampling a

small subset of the input vertices. A shape is fit to this subset without any outlier rejection, and all the vertices in the original set which are within some distance threshold $\epsilon$ of the fit shape are added to an inlier set. A "refined" shape model is then fit to all of the inlier data points, again without any explicit outlier rejection. A new inlier set and cost of fit are computed for the updated shape model. This process repeats for a set number of iterations, after which the model with lowest cost is returned. Pseudo code for using the greedy set cover algorithm with RANSAC for fitting each shape is given below for clarity. In the rest of this section we explain in more detail each of the steps of the RANSAC algorithm for shape fitting.

**Algorithm 1** Sampling Based Primitve Decomposition Algorithm

$\mathcal{D} \leftarrow$ set of input mesh vertices
N $\leftarrow |\mathcal{D}|$
$\mathcal{P} \leftarrow \emptyset$
**while** $|\mathcal{D}| \geq \alpha * N$ **do**
  $i \leftarrow 0$
  $\phi_{\text{best}} \leftarrow$ None
  $c_{\text{best}} \leftarrow \infty$
  $\mathcal{I}_{\text{best}} \leftarrow \emptyset$
  **while** $i \leq$ max_iterations **do**
    $\mathcal{D}_r \leftarrow \text{GetRandomSubset}(\mathcal{D})$
    $\phi \leftarrow \text{BestFitModel}(\mathcal{D}_r)$
    $\mathcal{I} \leftarrow \text{GetInliers}(\mathcal{D}, \phi, \epsilon)$
    $\phi_{\text{new}} \leftarrow \text{BestFitModel}(\mathcal{I})$
    $\mathcal{I}_{\text{new}} \leftarrow \text{GetInliers}(\mathcal{D}, \phi_{\text{new}}, \epsilon)$
    $c \leftarrow \text{GetModelCost}(\mathcal{I}_{\text{new}}, \phi_{\text{new}})$
    **if** $c < c_{\text{best}}$ **then**
      $c_{\text{best}} \leftarrow c$
      $\phi_{\text{best}} \leftarrow \phi_{\text{new}}$
      $\mathcal{I}_{\text{best}} \leftarrow \mathcal{I}_{\text{new}}$
    **end if**
    $i \leftarrow i + 1$
  **end while**
  $\mathcal{D} \leftarrow \mathcal{D} \setminus \mathcal{I}_{\text{best}}$
  $\mathcal{P} \leftarrow \mathcal{P} \bigcup \phi_{\text{best}}$
**end while**
**return** $\mathcal{P}$



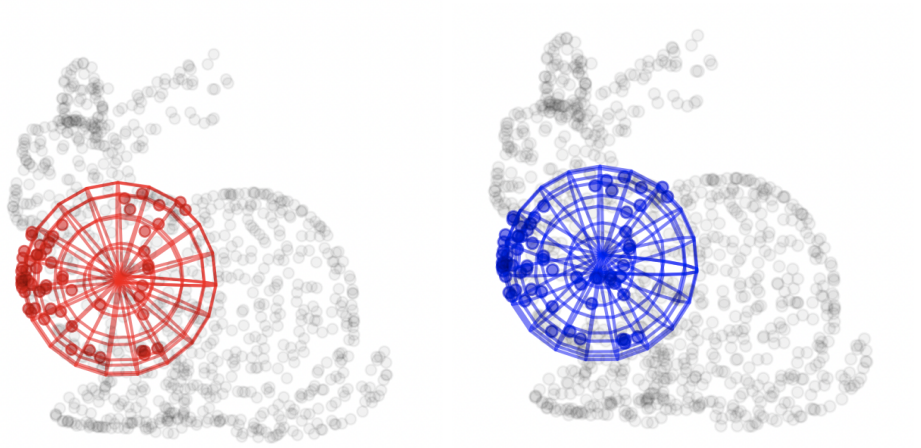
(a) (b)

Figure 4-1: Left: Colormap showing the euclidean-biased sampling distribution based on a seed point chosen on the left of the Bunny's torso. Right: The chosen sample points used for fitting a model. The red point is the seed point and the three blue points are the additional points sampled based on the euclidean-biased distribution.

**Data Sampling** RANSAC is sensitive to the seed points used to fit the initial models. On average, we expect a large sample of points that are far apart to be explained by different primitives while nearby points will generically belong to the same primitive. Therefore, we introduce the following Euclidean bias in our sampling algorithm. First, a single seed point is sampled uniformly from the data, then we compute a probability distribution such that points close to the seed point in Euclidean distance are more likely to be chosen than points far away. The remaining sample points are sampled using this distribution. Figure 4-1 shows a visual representation of this probability distribution.

By sampling in this way, we are more likely to fit a single primitive to points that are close to each other. This is preferable to using a single primitive to explain points that are far apart and would be better explained by multiple primitives.

**Shape Fitting** In this section we focus specifically on using spheres to approximate input meshes. In order to do this we use the least squares formulation presented in (3.9) to fit a single sphere to the sampled data points with no outlier rejection. It is also possible to instead fit other primitive shapes such as ellipsoids and boxes, or to



(a) (b)

Figure 4-2: Left: The best fit sphere for the four original sampled points, inlier vertices from the original data set are highlighted in red. Right: The refined sphere fit to all the red inlier points, with new inlier points shown in blue.

fit multiple shapes to the sampled data and select the best one based on the cost of the fit.

**Inlier Selection** In general, inliers for a given model are points whose distance from the surface of the model is within some threshold. Once an initial sphere radius $r$ and center $C$ have been computed, we compute the distance between every data point $x_i$ and the surface of the fit sphere using the equation $d_i = |\|(x_i - C)\|_2 - r|$. Given a maximum allowable threshold $d_{max}$, inliers are defined as all the points $x_i$ for which $d_i \leq d_{max}$. The threshold $d_{max}$ is a parameter of the RANSAC shape fitting algorithm which determines how close a point has to be to the surface of a sphere in order to be considered "explained" by that sphere.

**Cost of Fit** Once the inliers have been computed, a new, refined sphere is fit using all the inlier points as data. The set of inliers is re-computed for the new sphere using the same method as described above. The final cost used to determine which model is best is the average distance between the surface of the refined sphere and the updated set of inliers.

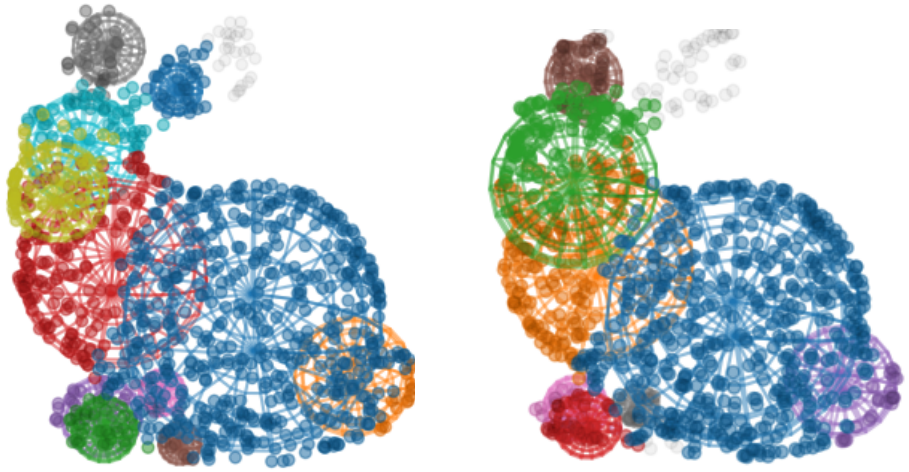
**Additional Algorithm Details** Before beginning the shape decomposition algorithm, the input mesh is scaled such that it fits within the unit sphere. Knowing this, it is reasonable to constrain any sphere that makes up the decomposition to have radius of at most 1. Additionally, we know that the center of any sphere used to decompose the input mesh should be on the inside of the mesh. This can be enforced by checking if the signed distance function (SDF) of the original input mesh is negative for the center of the sphere. If either the radius or center constraints are violated by a sphere output by the RANSAC shape fitting algorithm, it is discarded and its inlier points are not removed from the input data set.

## 4.2 Results & Discussion

Unlike the mixed integer programs from Chapter 3, the RANSAC-based algorithm for fitting spheres with outlier rejection is fast. Fitting a single sphere with RANSAC, using max_iterations = 1500 takes $3.2 \pm 2.6$ seconds. The RANSAC run time does not depend on the total number of data points because for each iteration a model is first fit for exactly 4 points (the minimum required to fit a sphere with least squares), then the refined model is fit to all the inlier points. The number of inlier points can vary depending on the size of the input and the value of the threshold $\epsilon$; however, as seen in Figure 3-4, the time to fit a sphere without outlier rejection does not scale with the number of data points. Thus, run time of each step of the RANSAC process is independent of the number of data points.

The time taken to decompose a full mesh into spheres using Algorithm 1 is dependent on the values of the parameters $\alpha$, which determines how many data points must be assigned to a sphere before stopping, $\epsilon$, the inlier threshold value, and the size of the input set of vertices. For the results shown in the rest of this section the fit times range from 25 to 35 seconds.

The results of applying Algorithm 1 to simplify the Stanford Bunny Mesh [33] are visualized in Figure 4-3. The figure shows the output of running the algorithm twice with the same parameters ($\alpha = 0.05, \epsilon = 0.1$). Each sphere and its inliers are shown in



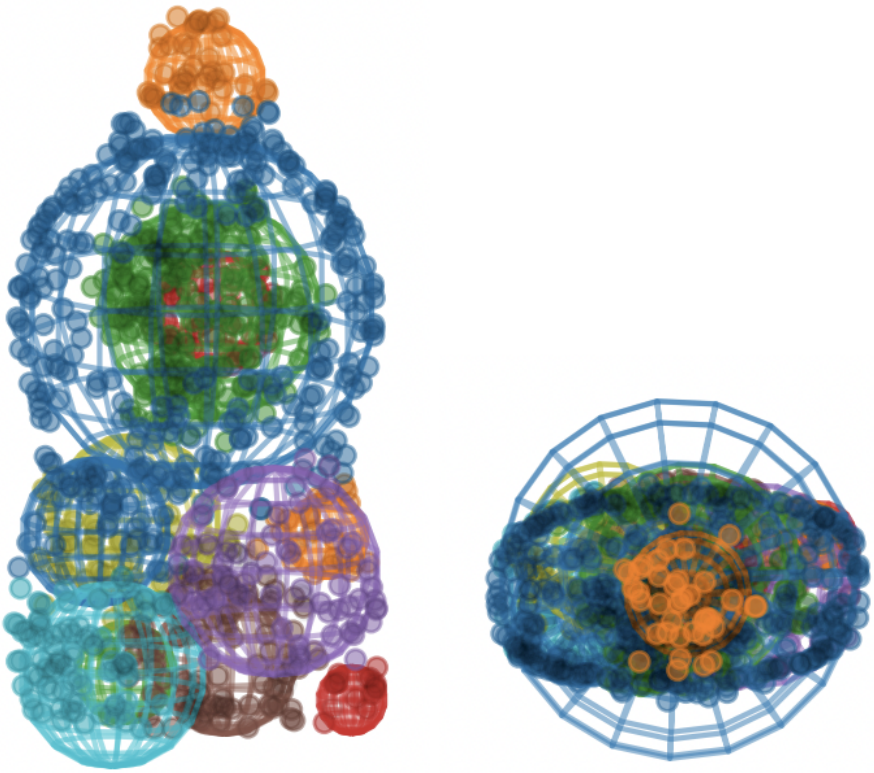
(a) (b)

Figure 4-3: RANSAC sphere decomposition applied to the Stanford Bunny.

a distinct color. Due to the randomness of the algorithm, every run results in slightly different results, hence the difference between the decomposition on the right and left. Given the shape of the bunny, spheres are a good primitive for approximating most of the body. However, looking at the ears we see that the results can be improved if we also included ellipsoids as an option for the decomposition.

Figure 4-4 shows the result of applying Algorithm 1 to a mesh generated from an Instant NGP NeRF of the YCB mustard bottle [25, 7]. The algorithm parameters used are the same as those used for Figure 4-3. On the left is a side view of the decomposition and on the right is a top view. From the top view we can see that the large blue sphere is a good approximation for points on the side of the bottle, but the rest of the sphere protrudes out from the planar surfaces that make up the the front and back of the mustard bottle. The planar surfaces present on the mustard bottle would be better approximated by ellipsoids or boxes, rather than just spheres.

Additionally, Algorithm 1 enforces that the input mesh points be well explained by a sphere, but does not enforce that the entire surface of each sphere fits the mesh. In the next chapter we explore a variation on the previous approaches which uses a denser cost in order to enforce both conditions.



(a) (b)

Figure 4-4: RANSAC sphere decomposition applied to the YCB Mustard Bottle.

# Chapter 5

# Signed Distance Field Approaches

## 5.1 Background

Both the optimization-based methods and sampling-based approaches from the previous chapters aim to minimize a cost based on the distances between input mesh vertices and the surface of the geometric shapes which make up the decomposition. As seen in some of the results from Chapter 4, this cost does not enforce that the whole surface of the geometric primitives closely matches the original mesh. In this chapter we explore an alternative optimization formulation which uses the Signed Distance Function (SDF) to implicitly represent the input mesh. The SDF of a mesh is the distance between any point in space and the surface of the mesh. The SDF is positive on the exterior of the mesh, zero on the surface of the mesh, and negative on the interior of the mesh. The SDF has been used as an effective implicit representation of 3D objects in works such as [26, 8, 23], and [13] has expanded on this idea to perform semantic primitive decomposition of meshes.

## 5.2 Using the Mesh SDF for Fitting Primitives

In this section we reformulate the optimization as minimizing the mesh SDF at points sampled from the surface of the primitive. In other words, we search for a primitive whose surface points are close to the surface of the input mesh. By re-formulating

the problem in this way, we better express the goal of finding a primitive that closely matches the surface of the mesh and reduce the likelihood of finding primitives which closely approximate some mesh vertices but otherwise differ significantly from the mesh surface.

We call the SDF of the input mesh $\text{SDF}_{\text{mesh}}$ and points on the surface of the primitive are called $\alpha(x, \theta)$ where points $x$ are sampled from the surface of a general version of the primitive (e.g. the unit sphere), $\theta$ represents the parameters of the fit primitive (e.g. the center and radius of a sphere), and the function $\alpha(x, \theta)$ maps the points $x$ onto the surface of the primitive defined by the parameters $\theta$. With these definitions we write the new optimization problem:

$$\min_{\theta} \sum_{i=1}^{N} (\text{SDF}_{\text{mesh}}(\alpha(x_i, \theta)))^2. \tag{5.1}$$

This optimization problem is non-convex due to the fundamental non-convexity of the SDF. Thus, the Formulation (5.1) is subject to getting stuck in local minima and solutions are sensitive to the initial conditions passed to the solver.

## 5.3 Results & Discussion

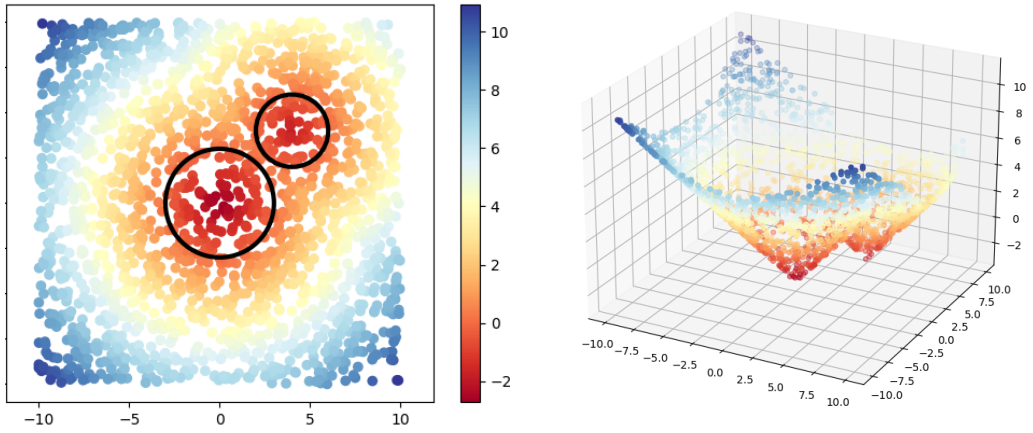
We focus on understanding this method with a simple 2D example where instead of an input mesh we have two circles implicity represented by their joint SDF, and our goal is to fit one circle to the input using Formulation (5.1). For our input of two circles with centers $c_1, c_2$ and radii $r_1, r_2$, the SDF is defined as

$$\text{SDF}_{\text{input}}(x) = \min(\|x - c_1\| - r_1, \|x - c_2\| - r_2).$$

Figure 5-1 shows a visual representation of this function.

We are searching for an optimal center $c^*$ and radius $r^*$ to define a circle, and we transform points $x_i$ sampled from the surface of the unit sphere onto the fit sphere using the function

$$\alpha(x_i, c^*, r^*) = r^* x_i + c^*.$$




(a) (b)

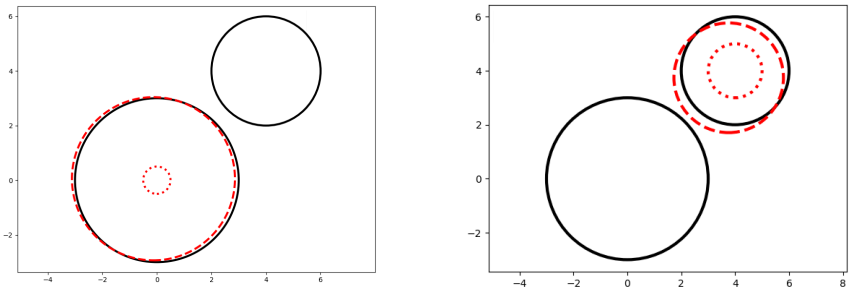
Figure 5-1: Signed Distance Function for two circles in two dimensions.

With this we write the following optimization problem:

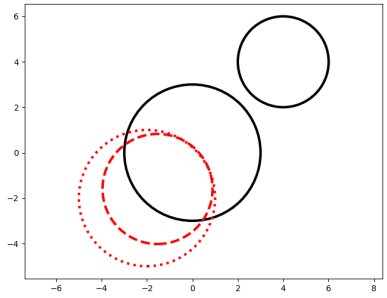
$$\min_{c^*, r^*} \sum_{i=1}^{N} (\text{SDF}_{\text{input}}(r^* x_i + c^*))^2. \tag{5.2}$$

We implement this program with Drake's MathematicalProgram class and solve using the nonlinear optimizer IPOPT [30, 37]. In Figure 5-2 we show the results for solving this program with different initial guesses.

As we can see from these figures, the solutions vary significantly depending on the initial guesses, and it is possible to end up in suboptimal local minima. However, with a good initial guess this method is able to implicitly perform outlier rejection and fit the circle such that it closely approximates the surface of the input. This method can be extended to fit different types of primitive shapes by modifying $\alpha$ and $\theta$, as well as to fit multiple primitives at once. More work remains to be done on selecting good initial guesses in order to obtain better results, however the approach of fitting points sampled from the surface of a primitive to the SDF of a mesh does, in theory, address some of the shortcomings of the previous methods.



(a) (b)



(c)

Figure 5-2: Fitting cirlces using formulation (5.2).

# Chapter 6

# Conclusion

In this thesis we presented geometric approaches for simplifying complex meshes with primitive shapes. The goal of simplifying meshes with primitives is to generate geometrically and dynamically accurate representations of arbitrary objects to use in simulation.

We began by presenting optimization formulations for fitting individual primitives to input meshes, both with and without outlier rejection. Using a greedy approach these individual primitives can be combined to represent the full input mesh. We also propose an alternative to the greedy method, which simultaneously performs assignment of the input vertices to primitives and computes the primitive parameters. These methods guarantee globally optimal solutions. However, finding optimal solutions to mixed integer programs is NP-hard, making these approaches hard to scale.

Next, we propose a sampling based method to perform shape decomposition. This approach builds off of the optimization methods but trades off optimality guarantees in order to reduce computational complexity. We show the results of using this algorithm to simplify meshes with thousands of input vertices. There are some cases where the algorithm outputs high quality results, but there are also cases where the fit spheres do not accurately represent the whole input mesh.

In order to mitigate this, in Chapter 5 we propose an alternative optimization formulation which uses an implicit representation of the input mesh, and aims to

minimize the distance between points sampled from the fit primitive and the surface of the mesh. While this approach results in a non-convex optimization problem which is susceptible to local minima and sensitive to initial guesses, we find that it is often able to find good solutions and can be improved by finding methods to select good initial guesses.

# Bibliography


[1] Common sense machines, inc. https://csm.ai.

[2] Luma labs. https://lumalabs.ai.

[3] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming*, 95(1):3–51, January 2003.

[4] Stephen Boyd and Jacob Mattingley. Branch and bound methods. Notes for EE364b, Stanford University, Winter 2006-07, 2007.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 1st edition, 2004.

[6] Gareth Bradshaw and Carol O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. Graph.*, 23(1):1–26, jan 2004.

[7] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, April 2017.

[8] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *CoRR*, abs/1812.02822, 2018.

[9] Simon Le Cleac'h, Hong-Xing Yu, Michelle Guo, Taylor A. Howell, Ruohan Gao, Jiajun Wu, Zachary Manchester, and Mac Schwager. Differentiable physics simulation of dynamics-augmented neural objects, 2022.

[10] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[11] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 209–216, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[12] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[13] Zekun Hao, Hadar Averbuch-Elor, Noah Snavely, and Serge J. Belongie. Dualsdf: Semantic shape manipulation using a two-level representation. *CoRR*, abs/2004.02869, 2020.

[14] Gregory Izatt, Hongkai Dai, and Russ Tedrake. Globally optimal object pose estimation in point clouds with mixed-integer programming. 12 2017.

[15] Charles F Jekel. *Digital Image Correlation on Steel Ball*. 2016.

[16] Songmin Jia, Zeling Zheng, Guoliang Zhang, Jinhui Fan, Xiuzhi Li, Xiangyin Zhang, and Mingai Li. An improved ransac algorithm for simultaneous localization and mapping. *Sensors*, 18(5):1402, 2018.

[17] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces, 2022.

[18] Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, February 2014.

[19] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J. Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

[20] Peng Li, Ruisheng Wang, Yanxia Wang, and Wuyong Tao. Evaluation of the icp algorithm in 3d point cloud registration. *IEEE Access*, 8:68030–68048, 2020.

[21] Vincent Lim, Huang Huang, Lawrence Yunliang Chen, Jonathan Wang, Jeffrey Ichnowski, Daniel Seita, Michael Laskey, and Ken Goldberg. Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8282–8289, 2022.

[22] J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA, 1967.

[23] Lars M. Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. *CoRR*, abs/1812.03828, 2018.

[24] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[25] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.

[26] Jeong Joon Park, Peter R. Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *CoRR*, abs/1901.05103, 2019.

[27] Chun-Yu Sun, Qian-Fang Zou, Xin Tong, and Yang Liu. Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM Trans. Graph.*, 38(6), nov 2019.

[28] Ted Ralphs, Menal Guzelsoy, Ashutosh Mahajan, Svetlana Oshkai. Warm starting for mixed integer linear programs. https://coral.ise.lehigh.edu/~ted/files/talks/INFORMS07.pdf, 11 2007.

[29] Russ Tedrake. *Underactuated Robotics*. 2023.

[30] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019.

[31] Jean-Marc Thiery, Émilie Guy, and Tamy Boubekeur. Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Trans. Graph.*, 32(6), nov 2013.

[32] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives, 2016.

[33] Greg Turk and Marc Levoy. The stanford bunny. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 277–286, New York, NY, USA, 1994. ACM.

[34] Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics*, 41(4):1–18, jul 2022.

[35] Hannah Werner. k-means clustering as a mixed integer programming problem. Master's thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2022.

[36] Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2:385–393, 1982.

[37] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. preprint.

[38] Heng Yang, Jingnan Shi, and Luca Carlone. Teaser: Fast and certifiable point cloud registration. 2020.

[39] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Computers & Industrial Engineering*, 154:107136, 2021.

[40] Liaomo Zheng, Ruiduan Wang, Shiyu Wang, Xinjun Liu, and Shipei Guo. Point cloud plane fitting based on ransac and robust eigenvalue method. In *2022 IEEE 8th International Conference on Computer and Communications (ICCC)*, pages 1368–1372, 2022.