TU WIEN Informatics

Konstruktion und Visualisierung von Gaussian Mixture Models aus Punktwolken für 3D-Objektrepräsentation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Simon Maximilian Fraiss, BSc
Matrikelnummer 01425602

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Adam Celarek

Wien, 4. Jänner 2022

Simon Maximilian Fraiss	Michael Wimmer

TU WIEN Informatics

Construction and Visualization of Gaussian Mixture Models from Point Clouds for 3D Object Representation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Simon Maximilian Fraiss, BSc
Registration Number 01425602

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Adam Celarek

Vienna, 4th January, 2022

Simon Maximilian Fraiss	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Simon Maximilian Fraiss, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 4. Jänner 2022

Simon Maximilian Fraiss

Danksagung

Mein Dank gilt Adam Celarek, welcher mich beim Verfassen dieser Arbeit wesentlich unterstützte, und bei Fragen und Problemen jederzeit mit Rat und Tat zur Seite stand. Weiters bedanke ich mich bei Professor Michael Wimmer für seine Unterstützung und sein Feedback, insbesondere in den mathematischen Teilen dieser Arbeit, sowie dafür, dass er mit der Rendering-and-Modelling Gruppe eine tolle Forschungsgruppe an der TU Wien leitet, in welcher man sich als Student sehr wohl und willkommen fühlt. Mein Dank gilt ebenso Simon Flöry für seine Geduld und dafür, dass ich trotz mehrmaliger Verzögerungen während dem Schreiben meiner Diplomarbeit im Rechenraum arbeiten durfte.

Weiters bedanke ich mich herzlichst bei meinen Eltern, welche mich die letzten Jahre auf vielerlei Art intensiv unterstützt haben. Ein großer Dank gilt ebenfalls all meinen Freund*innen, mit denen ich die letzten Jahre verbringen durfte, für all die lustigen und berührenden Zeiten, die wir zusammen verbracht haben. Dies inkludiert auch alle Mitglieder des Discord-Servers $\forall$*aus1endlichenRaum*. Speziellen Dank möchte ich Patrick, Steve, Chelsea, Kasia und Sandra ausdrücken.

Acknowledgements

I would like to thank Adam Celarek, who supported me substantially in writing this thesis, and who was always on hand with help and advice whenever I had problems or questions. I would also like to thank Professor Michael Wimmer for his support and feedback, especially in the mathematical parts of this work, as well as for leading the rendering and modeling group at TU Wien, in which I felt very comfortable and welcome as a student. I also thank Simon Flöry for his patience and for allowing me to work at Rechenraum while writing my thesis despite several delays.

I would also like to express sincere thanks to my parents, who have given me a lot of support in many ways over the past few years. A big thank you also goes out to all my friends with whom I have been able to spend the last few years, for all the touching and fun times that we spent together. This includes everyone from our Discord server $\forall$*aus1endlichenRaum*. I also want to express special thanks to Patrick, Steve, Chelsea, Kasia, and Sandra.

Kurzfassung

Punktwolken sind eine häufig genutzte Art, dreidimensionale Objekten zu beschreiben. In manchen Anwendungen sind jedoch andere Repräsentationen nützlicher. Gaussian Mixture Models (GMMs) können als solch eine alternative Repräsentation genutzt werden. Ein GMM ist eine konvexe Summe von Normalverteilungen, welche die Dichte einer Punktwolke beschreibt. In dieser Arbeit untersuchen wir sowohl die Visualisierung, als auch die Konstruktion von GMMs. Für die Visualisierung haben wir ein Werkzeug implementiert, welches eine Isoellipsoid- und eine Dichtevisualisierung ermöglicht. Wir beschreiben den mathematischen Hintergrund, die Algorithmen, und die Implementierung dieses Werkzeugs. Für die Konstruktion von GMMs untersuchen wir mehrere Algorithmen, welche bereits zum Konstruieren von GMMs im Kontext von 3D-Daten-Verarbeitung genutzt wurden. Wir präsentieren unsere Implementierungen des Expectation-Maximization(EM)-Algorithmus und des Top-Down HEM Algorithmus. Weiters haben wir die Implementierung von Geometrically regularized Bottom-Up HEM angepasst, um eine fixe Anzahl an Gaussians zu erzeugen. Wir evaluieren diese drei Algorithmen in Bezug auf die Qualität ihrer konstruierten GMMs. In vielen Fällen ist die statistische Likelihood, welche durch den EM Algorithmus maximiert wird, kein zuverlässiger Indikator für die Qualität eines GMMs. Daher nutzen wir stattdessen den Rekonstruktionsfehler einer rekonstruierten Punktwolke basierend auf der Chamfer-Distanz. Weiters definieren wir Metriken zur Messung der Uniformität der rekonstruierten Punktwolken und der Variation der Gaussians des GMMs. Wir demonstrieren, dass EM die besten Resultate bezogen auf diese Metriken erzeugt. Geometrically regularized Bottom-Up HEM ist unterlegen für niedrigere Anzahlen an Gaussians aber kann gute GMMs mit deutlich mehr Gaussians sehr effizient erzeugen.

Abstract

Point clouds are a common representation of three-dimensional shapes in computer graphics and 3D-data processing. However, in some applications, other representations are more useful. Gaussian Mixture Models (GMMs) can be used as such an alternative representation. A GMM is a convex sum of normal distributions, which aims to describe a point cloud's density. In this thesis, we investigate both visualization and construction of GMMs. For visualization, we have implemented a tool that enables both isoellipsoid and density visualization of GMMs. We describe the mathematical backgrounds, the algorithms, and our implementation of this tool. Regarding GMM construction, we investigate several algorithms used in previous papers for constructing GMMs for 3D-data processing tasks. We present our implementations of the expectation-maximization (EM) algorithm and top-down HEM. Additionally, we have adapted the implementation of geometrically regularized bottom-up HEM to produce a fixed number of Gaussians. We evaluate these three algorithms in terms of the quality of their generated GMMs. In many cases, the statistical likelihood, which is maximized by the EM algorithm, is not a reliable indicator for a GMM's quality. Therefore, we instead rely on the reconstruction error of a reconstructed point cloud based on the Chamfer distance. Additionally, we provide metrics for measuring the reconstructed point cloud's uniformity and the GMM's variation of Gaussians. We demonstrate that EM provides the best results in terms of these metrics. Top-down HEM is a fast alternative, and can produce even better results when using fewer input points. The results of geometrically regularized bottom-up HEM are inferior for lower numbers of Gaussians but it can create good GMMs consisting of high numbers of Gaussians very efficiently.

Contents

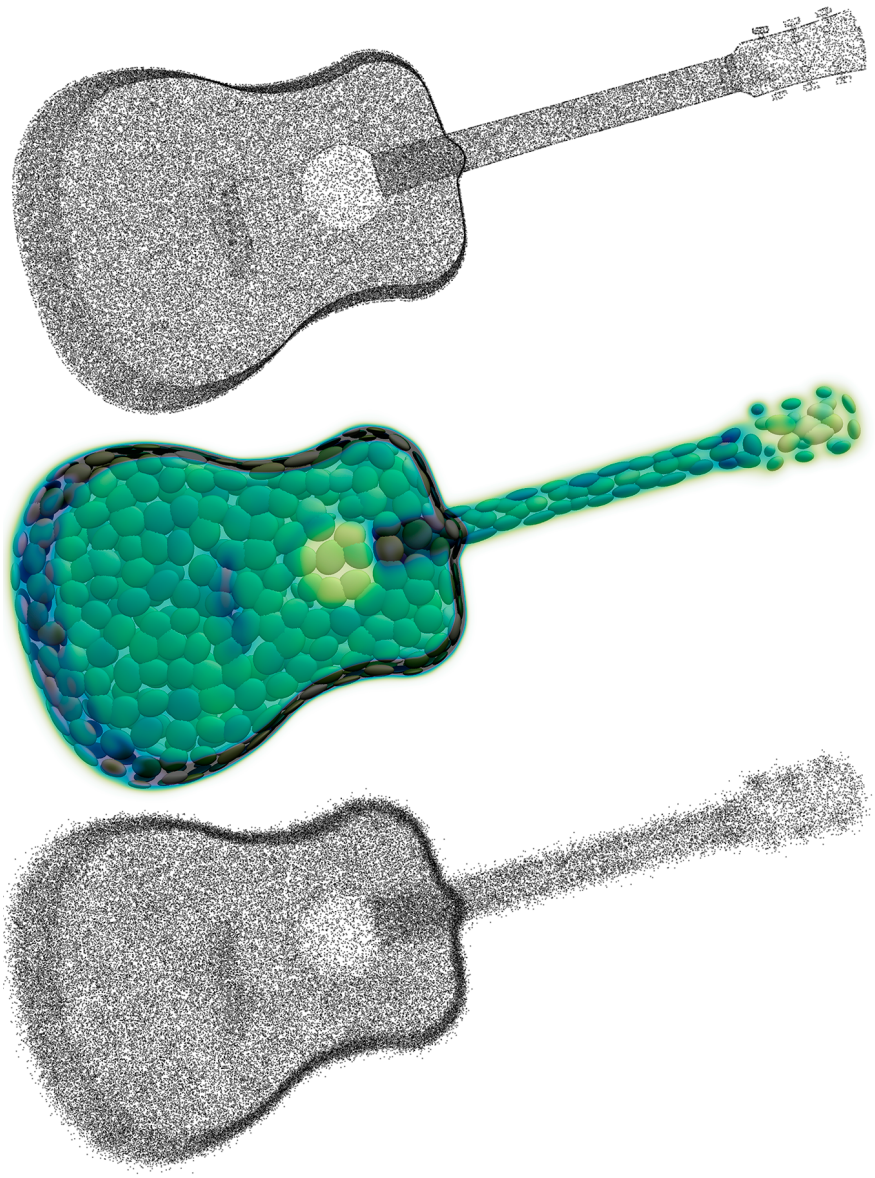


Figure 1: Example of mixture fitting on a guitar point cloud (based on the model *guitar_0001* from ModelNet 40 [WSK+15]). Top: Original point cloud. Center: Overlaid density and ellipsoid visualization (see Chapter 3) of a GMM consisting of 512 Gaussians. The GMM has been fitted to the original point cloud using the EM algorithm (see Chapters 2 and 4). Bottom: A new point cloud reconstructed from the GMM. The general shape of the object is preserved, but details are lost.

CHAPTER 1

Introduction

1.1 Motivation

Point clouds are a common representation of three-dimensional shapes in computer graphics and 3D-data processing. They commonly occur as the result of 3D-scanning techniques, such as laser scanning or photogrammetry. However, for some processing tasks, other representations are more useful. In particular, applying deep learning techniques on point clouds is challenging, because of the unordered and irregular nature of point clouds. A successful deep learning approach for three-dimensional point clouds enables tasks such as classification, segmentation, or hole filling, all of which are useful when working with point clouds representing three-dimensional objects. While deep learning techniques that work directly on point clouds have been proposed [QSMG17], various other methods transform the point cloud to another representation, such as voxel grids [QSN+16].

A research project at TU Wien is currently exploring a novel technique for applying deep learning methods based on Gaussian Mixture Models (GMMs) [Ano22]. A GMM is a probabilistic model whose probability density function is a convex sum of normal distributions. GMMs can be used as an alternative representation for 3D shapes and have been successfully used for 3D-processing tasks such as surface reconstruction [PMA+14][Eck17], point cloud registration [EK13][EKK18], and deep learning [BSLF18]. A GMM can be fitted to a point cloud so that its probability density function approximates the point cloud's density. This is illustrated in Figure 1. The construction of GMMs has been researched for many decades [RW84], therefore there are several algorithms available. However, it is not clear which one is most suitable for this use case. Therefore, we want to implement, evaluate and possibly adapt several GMM construction algorithms, focussing on algorithms that have been used for other 3D-data-processing tasks.

Additionally, to facilitate the development and the configuration of construction algorithms, and to enable deeper insight into the novel deep learning architecture researched at TU Wien, a visualization tool for GMMs is required. Such a tool should be able to visualize GMMs in

realtime while navigating freely through the scene. Furthermore, a Python interface is required to access the visualization functionality from Python code. We are not aware of any 3D GMM visualization tool that meets our requirements. The tool "gmsStudio" by Preiner et al. [PMA+14] is able to visualize 3D GMMs, but lacks density visualization techniques. Also, it does not provide an interface to access its functionality from Python code.

1.2 Aim of the Work

The aim of the work is two-fold.

The first goal is the development of a visualization tool for Gaussian Mixture Models. This tool should be able to visualize GMMs with several thousands of Gaussian components in realtime using various techniques, such as an isoellipsoid visualization, which shows one ellipsoid per Gaussian, and a density visualization, which is based on volume rendering. It should also be capable of displaying point clouds. The user should be able to change the parameters of the selected visualization techniques, as well as inspect the details of single Gaussians. The application will be written in C++ using OpenGL and Qt. Besides providing a stand-alone application, the visualizer also has to provide a Python interface for accessing its functionality from Python code.

The second goal of the thesis is to implement, adapt, and evaluate existing GMM-construction algorithms. We want to compare the EM algorithm [DLR77], top-down hierarchical EM by Eckart et al. [EKT+16], and geometrically regularized bottom-up hierarchical EM by Preiner et al. [PMA+14]. The latter two are more efficient adaptations of the EM algorithm and have been used in 3D-data-processing tasks, but it is unclear how much the results differ in their quality. The first two algorithms will be implemented by ourselves in Python, while we will adapt the C++ source code of the third one to our purposes and add Python bindings. We will define appropriate metrics for evaluation of the algorithms. Our evaluation focuses more on the difference in the quality of the results rather than the algorithms' efficiency.

1.3 Contributions

The main contributions of this work are as follows:

Visualization:

- We developed a tool for visualizing GMMs, providing both a stand-alone application and a Python interface.
- We describe the mathematical background for the density visualization. In particular, we constructed a theorem describing Gaussian density values along a ray. Additionally, we found a mistake in a previous work on a similar problem by Jakob et al. [JRJ11] and provide the correct solution.

Construction:

- PyTorch-based implementations of the EM-algorithm and the top-down HEM-algorithm.
- Adaptation of geometrically regularized bottom-up HEM: The number of Gaussians created by this algorithm is not fixed but stochastic and depends on the configurations of the algorithm. It is often not able to create lower numbers of Gaussians (512 or less). We adapted the algorithm to create a fixed number of Gaussians which can be arbitrarily low.
- As the likelihood and similar measures do not describe the quality of our GMMs in a useful way, we defined suitable metrics for evaluating the quality of GMMs regarding reconstruction error, irregularity, and variation of Gaussians.
- Evaluation of EM, top-down HEM, and geometrically regularized bottom-up HEM regarding the influence of their most important parameters.
- Comparison of EM, top-down HEM and geometrically regularized bottom-up HEM. We focus especially on the difference in quality of the results.

1.4 Structure of the Work

Chapter 2 provides the necessary background information on Gaussian Mixtures, as well as an overview of related work in the fields of GMM construction and visualization. We describe our visualization algorithms and the implementation of our visualization tool in Chapter 3. In Chapter 4, we discuss our implementations and adaptations of the selected GMM construction algorithms. These algorithms are evaluated in Chapter 5 and compared in terms of several suitable criteria. Finally, Chapter 6 concludes the thesis and reflects on possible future work.

CHAPTER 2

Background

2.1 Gaussian Mixture Models

A Gaussian Mixture Model (GMM) is a probabilistic model whose probability density function (PDF) is a weighted sum of Gaussian (normal) distributions. The PDF of a multi-dimensional Gaussian distribution is defined as

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{2.1}$$

where n is the number of dimensions, $\boldsymbol{\mu} \in \mathbb{R}^n$ the *mean*, and $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$ the *covariance matrix*. $\boldsymbol{\Sigma}$ is positive-definite and symmetric. The density is highest at $\boldsymbol{x} = \boldsymbol{\mu}$ and decreases in all directions in a way defined by $\boldsymbol{\Sigma}$. The eigenvectors of $\boldsymbol{\Sigma}$ are orthogonal and correspond to the axes of the PDF's isoellipsoids.

A GMM $\boldsymbol{\Theta}$ is described by K Gaussians $\boldsymbol{\Theta}_k = \{\omega_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$. Each $\boldsymbol{\Theta}_k$ represents a weighted normal distribution by its weight, mean, and covariance matrix, respectively. The weights fulfill $0 \leq \omega_k \leq 1$ and $\sum_{k=1}^{K} \omega_k = 1$. The probability density function of a GMM is defined as

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \sum_{k=1}^{K} \omega_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{2.2}$$

Just like $\mathcal{N}$, $f_{\boldsymbol{\Theta}}$ is a valid probability density function. In particular, its definite integral is one and it is positive for all choices of $\boldsymbol{x}$. Figure 2.1 shows an example for a one-dimensional GMM.

Any continuous density function can be approximated to arbitrary accuracy by a GMM with appropriately chosen parameters and sufficiently many Gaussians [GBC16] [LSW17].

GMMs have been successfully used for 3D-point-cloud-processing tasks such as surface reconstruction [PMA+14] [Eck17], point cloud registration [EK13] [EKK18] and classification [BSLF18].

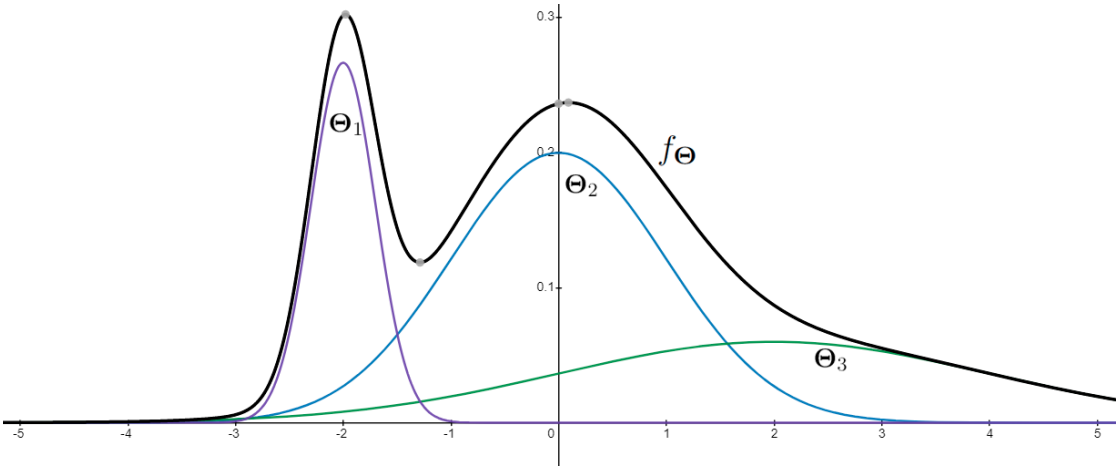


Figure 2.1: The plot shows the PDF $f_{\boldsymbol{\Theta}}$ of a one-dimensional GMM consisting of the three Gaussians $\boldsymbol{\Theta}_1 = \{0.2, -2, 0.3\}$, $\boldsymbol{\Theta}_2 = \{0.5, 0, 1\}$ and $\boldsymbol{\Theta}_3 = \{0.3, 2, 2\}$. $f_{\boldsymbol{\Theta}}$ is equal to the sum of the Gaussians' density functions.

2.2 Construction

2.2.1 Introduction

The problem of finding the weights, means, and covariance matrices of a (Gaussian) mixture model $\boldsymbol{\Theta}$ that best describe the distribution of a given point dataset $\boldsymbol{X}$ is called the *mixture density estimation problem*. According to Redner and Walker [RW84], this problem has been studied for a long time, with the first publications going back as far as the 19th century. One of the first methods for solving this problem for the case of two univariate Gaussian mixtures was proposed by Pearson [Pea94] using the *method of moments*. This method works by equating the empirical moments of the sampled data and the theoretical moments of the mixture. This process results in a set of equations that can be solved for the parameters.

Before the rise of computers, the method of moments remained the most studied way of approaching the mixture density estimation problem [RW84]. However, increased computing power made it possible to further investigate the more powerful *method of maximum likelihood*. The goal of this method is to choose the mixture model's parameters $\boldsymbol{\Theta}$, such that it maximizes the so-called *likelihood function*

$$L_{\boldsymbol{X}}(\boldsymbol{\Theta}) = \prod_{n=1}^{N} f_{\boldsymbol{\Theta}}(\boldsymbol{x}_n), \tag{2.3}$$

where N is the number of points $\boldsymbol{x}_n$ in the dataset $\boldsymbol{X}$ [RW84] [MP04]. Commonly, instead of using the likelihood itself, the logarithmic likelihood (*log-likelihood*) is used. It is often easier to handle algebraically. Because of the strict monotonicity of the logarithm, it has the same maximums.

$$\log\left(L_{\boldsymbol{X}}(\boldsymbol{\Theta})\right) = \sum_{n=1}^{N} \log\left(f_{\boldsymbol{\Theta}}(\boldsymbol{x}_n)\right) \tag{2.4}$$

In theory, the local maximums of $L_{\boldsymbol{X}}$ could be found by setting its partial derivatives to zero, which results in a set of so-called *likelihood equations*, whose solutions are the local maximums. However, in the case of mixture density problems, these equations can usually not be solved by analytic means [RW84]. Therefore, algorithms that provide approximate solutions are needed. Classical techniques such as Newton's method, quasi-Newton methods, and gradient-based methods can be applied to solve this problem [RW84]. Another algorithm for this problem, which is considered superior to the previous approaches [XJ96], is the expectation-maximization (EM) algorithm.

2.2.2 The Expectation-Maximization Algorithm

The EM algorithm was introduced by Dempster et al. [DLR77], who formulated the algorithm as a way of obtaining maximum likelihood estimates for incomplete data problems. They provide a very general formulation of the algorithm which is applicable for many problems. When applying it to the mixture density estimation problem for GMMs, it is equivalent to algorithms suggested by other authors before [Has66] [Day69].

We explain the EM algorithm for GMMs by seeing the mixture density problem as an incomplete data problem, following the explanation and equations by Bishop [Bis06]:

The algorithm works on the assumption that the point cloud has been sampled from a GMM consisting of K Gaussians. Therefore, each point belongs to the Gaussian it was originally sampled from. As we do not know which point belongs to which Gaussian, this is an *incomplete data problem*, and the EM algorithm can be applied. A latent K-dimensional variable $\boldsymbol{z}_n$ describing the association between points and Gaussians is introduced for each sample point $\boldsymbol{x}_n$. Exactly one element of $\boldsymbol{z}_n$ is one, while the other ones are zero. z_{nk} is one if and only if the point $\boldsymbol{x}_n$ belongs to the kth Gaussian.

Based on this, the idea of the EM algorithm is to calculate the expected value of the latent data $\boldsymbol{z}$ based on the point data and an initial model (*expectation (E) step*). Afterward, a new model that maximizes the likelihood of the model is constructed, given the expected values of the latent data (*maximization (M) step*). These two steps are repeated until a termination criterion is fulfilled (e.g. the change of the likelihood is below a certain threshold).

The E-step calculates the expected values of z_{nk}, which are denoted as $\gamma(z_{nk})$ or γ_{nk} for short. These values are referred to as *responsibilities*. They are calculated using Bayes' theorem:

$$\gamma_{nk} = p(z_{nk} = 1|\boldsymbol{x}_n) = \frac{p(z_{nk} = 1)p(\boldsymbol{x}_n|z_{nk} = 1)}{\sum_{j=1}^{K} p(z_{nj} = 1)p(\boldsymbol{x}_n|z_{nj} = 1)} \tag{2.5}$$

The prior probability of z_{nk} being one without any information about the sample point itself is equal to the Gaussian's weight ($p(z_{nk}) = \omega_k$). The probability density of a sampled point being

equal to $\boldsymbol{x}_n$, given knowledge about its assignment, is simply the assigned Gaussian distribution $p(\boldsymbol{x}_n|z_{nk} = 1) = \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. Therefore

$$\gamma_{nk} = \frac{\omega_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \omega_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{2.6}$$

In the M-step, the new means, covariance matrices, and weights are calculated using the responsibilities calculated in the E-step, as described by the following equations. These formulas can be derived from rearranging the likelihood equations:

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x}_n, \tag{2.7}$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{\text{new}})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T, \tag{2.8}$$

$$\omega_k^{\text{new}} = \frac{N_k}{N}, \tag{2.9}$$

where $N_k = \sum_{n=1}^{N} \gamma_{nk}$.

Repeating the E- and the M-step leads to an increase in likelihood. The steps are repeated until a convergence criterion is fulfilled.

One problem with the EM-algorithm, and maximum likelihood methods in general, is that the likelihood can become arbitrarily high through so-called singularities: If a Gaussian's mean falls exactly on an input point, an arbitrarily high likelihood can be produced by choosing smaller and smaller values in the covariance matrix, as illustrated in Figure 2.2. Equation 2.10 describes this for a three-dimensional Gaussian with a covariance matrix of the shape $\sigma^2 \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix.

$$\lim_{\sigma^2 \to 0} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{x}_n, \sigma^2 \boldsymbol{I}) = \lim_{\sigma^2 \to 0} \frac{1}{(2\pi)^{\frac{3}{2}} \sigma^3} = \infty \tag{2.10}$$

Measures have to be set in place to avoid the approaching of singularities, for example by setting minima for the covariances.

The EM algorithm has several advantages, such as reliable convergence, low cost per iteration, and ease of implementation. However, its convergence can be quite slow [RW84]. Several variants and extensions have been developed to improve the algorithm in various ways. Some extensions focus on speeding up convergence, for example by applying Aitken's acceleration method [LLS87] [Lou82] or combining it with other methods, such as conjugate gradient methods [JJ93] [SRG03], quasi-Newton methods [Lan95] or Fisher's scoring method [Ike00]. Some extensions focus on speeding up the algorithm by only updating some of the data in each iteration, making them more applicable to large data sets [NH98] [BFR+98]. Other extensions include online versions of the EM algorithm [HG09], variants that are less prone to overfitting [KRT99], variants with increased

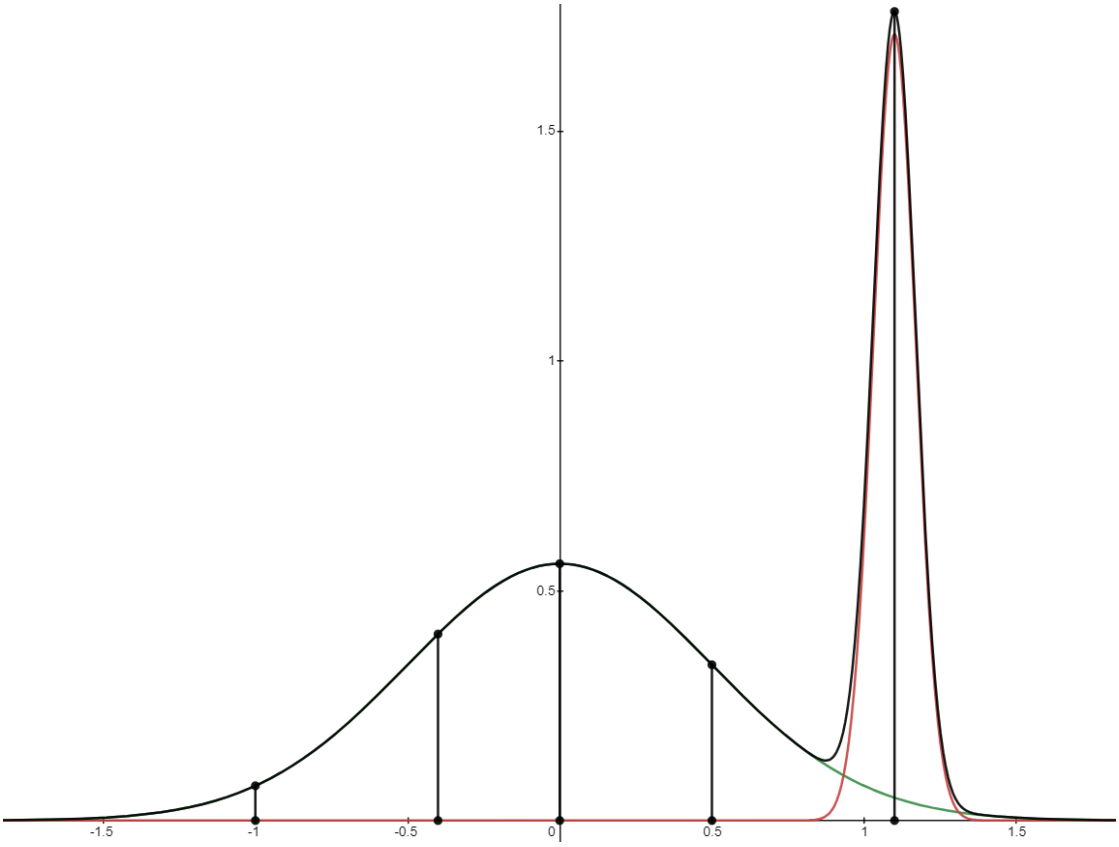


Figure 2.2: A GMM on a one-dimensional point cloud consisting of two Gaussians, one of which is collapsing onto one point. By reducing its variance, its height would increase even further, increasing the likelihood arbitrarily.

clustering accuracy [LCH10], semi-supervised variants [GSH15] and stochastic variants that are more likely to escape from local maxima [CCD95]. Not all of these algorithms were applied in a 3D setting. Their applications cover a wide range of different settings with different numbers of dimensions and Gaussians.

In the following, we will describe two extensions that create so-called *hierarchical* GMMs. These algorithms also lead to a significant speed-up of the algorithm. Both of these techniques have been successfully used in 3D point cloud processing tasks.

2.2.3 Hierarchical Top-Down EM

Eckart et al. proposed an EM algorithm for the construction of hierarchical GMMs from 3D point cloud data [EKT+16]. A hierarchical GMM consists of several levels of GMMs, where each new level replaces the Gaussians from the previous level with new sub-GMMs, resulting in a more complex GMM overall. The algorithm starts by fitting a mixture consisting of j Gaussians plus one additional uniform noise cluster using the EM algorithm. j is usually a low number, such

as eight. Afterward, each Gaussian ("parent") is replaced by a new sub-GMM consisting again of j Gaussians ("children") and a noise cluster. Each data point is assigned to one or more of those sub-GMMs using a partitioning scheme (see below). Each sub-GMM is then fitted to the points that have been assigned to the respective parent. This procedure of subdividing and fitting is repeated until the desired number of levels is reached.

When creating a new sub-GMM, the Gaussians are initialized on the corners of the bounding box of the relevant points. The sub-GMMs in the last level can be recombined into one final GMM by multiplying each Gaussian's weight with the weights of all its parents.

The authors suggest two partitioning schemes to assign the points to the sub-GMM: *Hard partitioning* assigns each point to exactly one sub-GMM, namely the one the point has the highest parent responsibility to. *Soft partitioning* on the other hand assigns a point to all sub-GMMs with a parent responsibility higher than a predefined partitioning threshold λ_l. In this case, additional weights derived from the responsibilities are stored per point-parent-pair, describing the point's influence on the sub-GMMs. The weight p_{nk} for the nth point and the kth Gaussian is calculated as $p_{nq}\gamma_{nk}/\xi$, where q is the index of the kth Gaussian's parent, γ is the responsibility, and ξ is a normalization term, which is the sum of all new p_{nk} in the qth sub-GMM.

The algorithm accelerates the construction of the GMMs by parallelizing on the GPU. The E-step is parallelized over all points and the M-step over all Gaussians. To further accelerate the algorithm, the E-step calculates the zeroth, first and second moments T_k^m of the points weighted by their responsibilities for each Gaussian:

$$T_k^0 = \sum_n \gamma_{nk}, \quad T_k^1 = \sum_n \gamma_{nk}\boldsymbol{x}_n, \quad T_k^2 = \sum_n \gamma_{nk}\boldsymbol{x}_n\boldsymbol{x}_n^T, \tag{2.11}$$

where k is the index of the Gaussian, γ_{nk} is the responsibility of the nth point to the kth gaussian and the $\boldsymbol{x}_n$ are the points. These can be calculated efficiently in parallel as part of the E-step.

The formulas of the M-step as defined in Equations 2.7, 2.8 and 2.9 are adapted for this algorithm and rewritten in a way to make use of the moments, which simplifies the M-step.

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{\sum_n \gamma_{nk}\boldsymbol{x}_n}{\sum_n \gamma_{nk}} = \frac{T_k^1}{T_k^0} \tag{2.12}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{\sum_n \gamma_{nk}\boldsymbol{x}_n\boldsymbol{x}_n^T}{\sum_n \gamma_{nk}} - \boldsymbol{\mu}_k^{new}\boldsymbol{\mu}_k^{newT} = \frac{T_k^2}{T_k^0} - \boldsymbol{\mu}_k^{newT}\boldsymbol{\mu}_k^{new} \tag{2.13}$$

$$\omega_k^{\text{new}} = \sum_n \frac{\gamma_{nk}}{N} = \frac{T_k^0}{N} \tag{2.14}$$

When using soft partitioning, the calculation of the moments T_k^m is adapted to

$$T_k^0 = \sum_n p_{nq}\gamma_{nk}, \quad T_k^1 = \sum_n p_{nq}\gamma_{nk}\boldsymbol{x}_n, \quad T_k^2 = \sum_n p_{nq}\gamma_{nk}\boldsymbol{x}_n\boldsymbol{x}_n^T \tag{2.15}$$

where q is the index of the parent of Gaussian k. Additionally, the calculation of the new weights is adapted to

$$\omega_k^{\text{new}} = \frac{T_k^0}{\sum_n p_{nq}}. \tag{2.16}$$

This algorithm speeds up GMM construction exponentially compared to classical EM.

2.2.4 Geometrically Regularized Hierarchical Bottom-Up EM

Geometrically regularized bottom-up HEM was developed by Preiner et al. [PMA+14] and is based on an earlier work by Vasconcelos and Lippman [VL98]. We will first describe the original technique and then describe the adaptations by Preiner et al.

Vasconcelos and Lippman proposed a hierarchical EM algorithm that works in a bottom-up-fashion: It starts from a very fine model consisting of many Gaussians. A coarser model consisting of fewer Gaussians is fitted to describe this initial model. This new model is then again replaced by a coarser one and so on. This approach differs from the classical EM algorithm, as the mixtures on each level l are not fitted to points but to the mixtures from the previous level $l + 1$.

The ith Gaussian's mean, covariance matrix and weight at level l are denoted by $\boldsymbol{\mu}_i^l$, $\boldsymbol{\Sigma}_i^l$ and ω_i^l respectively, and the number of Gaussians at level l is denoted by C^l.

A simple way to fit mixtures to mixtures would be to sample a new point cloud from the source mixture and fit the new model to those points. Following this intuition, the central idea in the derivation of the algorithm is to consider a *virtual sample* $\boldsymbol{X} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{C^{l+1}}\}$ of the model, where $\boldsymbol{X}_n$ is a set of points drawn from the nth Gaussian of the model. The number of points for set n is $M_n = \omega_n^l |P|$, where $|P|$ is the total number of virtual points and ω_n^l is the weight of the nth Gaussian. On this virtual sample, the EM algorithm is applied. Using the law of large numbers, this calculation becomes independent from the actual values of the sample and ultimately results in a modified E-Step, where the responsibility of Gaussian n from the last level $l+1$ and Gaussian k from the new level l is calculated by

$$\gamma_{nk} = \frac{\left[\mathcal{N}\left(\boldsymbol{\mu}_n^{l+1} | \boldsymbol{\mu}_k^l, \boldsymbol{\Sigma}_k^l\right) \exp\left(-\frac{1}{2}\operatorname{tr}\left(\left(\boldsymbol{\Sigma}_k^l\right)^{-1} \boldsymbol{\Sigma}_n^{l+1}\right)\right)\right]^{M_n} \omega_k^l}{\sum_j \left[\mathcal{N}\left(\boldsymbol{\mu}_n^{l+1} | \boldsymbol{\mu}_j^l, \boldsymbol{\Sigma}_j^l\right) \exp\left(-\frac{1}{2}\operatorname{tr}\left(\left(\boldsymbol{\Sigma}_j^l\right)^{-1} \boldsymbol{\Sigma}_n^{l+1}\right)\right)\right]^{M_n} \omega_j^l}, \tag{2.17}$$

where tr is the trace (the sum of the elements in the main diagonal of a matrix).

The formulas of the M-step to calculate the new model based on the responsibilities are modified as follow:

$$\omega_k^l = \frac{\sum_n \gamma_{nk}}{C^{l+1}} \tag{2.18}$$

$$\boldsymbol{\mu}_k^l = \frac{\sum_n \gamma_{nk} M_n \boldsymbol{\mu}_k^{l+1}}{\sum_n \gamma_{nk} M_n} \tag{2.19}$$

$$\boldsymbol{\Sigma}_k^l = \frac{\sum_n \gamma_{nk} M_n \left(\boldsymbol{\Sigma}_n^{l+1} + (\boldsymbol{\mu}_n^{l+1} - \boldsymbol{\mu}_k^l)(\boldsymbol{\mu}_n^{l+1} - \boldsymbol{\mu}_k^l)^T\right)}{\sum_n \gamma_{nk} M_n} \tag{2.20}$$

None of these formulas depend on the values of the sample $\boldsymbol{X}$, so no actual sample needs to be created.

The motivation of the extension by Preiner et al. is that a statistically optimal fit, such as provided by maximum likelihood approaches, is prone to smoothing the signal in a way that prevents robust reconstruction. Therefore, they extend the hierarchical EM method by geometric constraints to prevent the signal and noise from blending.

The regularization aims to stop Gaussians that are too far away from each other from merging in the reduction step. To measure the distance between two Gaussians the Kullback-Leiber divergence is used:

$$D_{KL}(\boldsymbol{\Theta}_t, \boldsymbol{\Theta}_s) = \frac{1}{2}\left(d_M(\boldsymbol{\mu}_t, \boldsymbol{\Theta}_s))^2 + \mathrm{tr}(\boldsymbol{\Sigma}_s^{-1}\boldsymbol{\Sigma}_t) - 3 - \ln\frac{|\boldsymbol{\Sigma}_t|}{|\boldsymbol{\Sigma}_s|}\right) \tag{2.21}$$

where $\boldsymbol{\Theta}_t$ and $\boldsymbol{\Theta}_s$ are two Gaussians, consisting of their corresponding ω, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ values, and $d_M(\boldsymbol{\mu}_t, \boldsymbol{\Theta}_s)$ is the Mahalanobis distance $\sqrt{(\boldsymbol{\mu}_t - \boldsymbol{\mu}_s)^T\boldsymbol{\Sigma}_s^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_s)}$. A threshold ρ is defined to be the maximum distance between two Gaussians. Their modified HEM algorithm only calculates γ_{nk} for which $D_{KL}(\boldsymbol{\Theta}_n^{l+1}, \boldsymbol{\Theta}_k^l) < \rho$. All other γ_{nk} will be zero. To provide intuitive control over this maximum distance, the authors provide a parameter α, such that $\rho = \alpha^2/2$.

Another change to the original algorithm is that the calculation of the new weights (see Equation 2.18) is modified to use a weighted sum

$$\omega_j^l = \sum_i \gamma_{ij}\omega_i^{l+1}. \tag{2.22}$$

Furthermore, the authors propose an initialization technique that places a Gaussian on each point. The initial covariance matrices are chosen in a way to reflect the local distribution of the nearby points inside a configurable radius r.

2.2.5 Alternative Approaches

Hosseini and Sra investigated using Riemannian optimization to generate GMMs [HS17]. Other classical techniques have been advanced as well such as stochastic gradient descent [GP19] or moment-based methods [MV10]. These methods are not designed in the context of finding a representation for 3D shapes.

PointGMM by Hertz et al. worked specifically on 3D point clouds [HHGCO20]. It uses deep learning techniques to generate hierarchical GMMs from point clouds. Their neural network learns class-specific priors (categories such as chair, table, airplane), which are then used to generate a GMM for the given point cloud. In our use case, we do not have such priors.

2.3 Visualization

In this section, different techniques for visualizing three-dimensional Gaussian Mixture Models are reviewed.

2.3.1 Gaussian Isoellipsoids

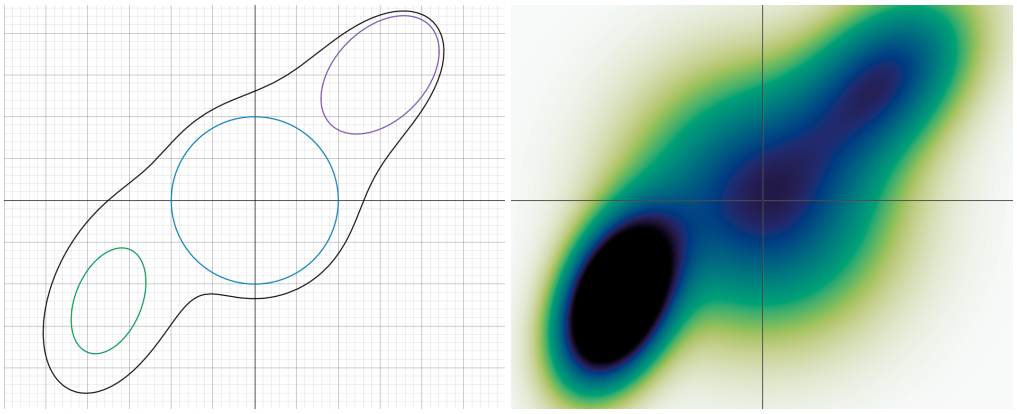
The rendering of Gaussian isoellipsoids is a method that has been commonly used for the visualization of GMMs in both 2D [Bis06] [VL98] and 3D [PMA+14] [Eck17] [HHGCO20]. In this visualization technique, one ellipsoid is displayed per Gaussian, resembling a contour of constant density. A two-dimensional example for this is shown in Figure 2.3a.

These ellipsoids can be expressed mathematically as all points $\mathbf{x}$ where the Mahalanobis distance is equal to the desired contour value c:

$$(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) = c^2 \tag{2.23}$$

If only one contour per Gaussian is rendered, c is commonly chosen to be one. This resembles the 1-σ-distance in the one-dimensional case. The resulting ellipsoids have axes $\pm c \sqrt{\lambda_i} \mathbf{e}_i$ where λ_i are the eigenvalues and $\mathbf{e}_i$ the normalized eigenvectors of $\boldsymbol{\Sigma}$ [Baj11].

This visualization technique gives insight into the internal structure of the GMM and the orientation and extent of the individual Gaussians.



(a) Isoellipsoids (colored) and isosurface (black) (b) Density heatmap (darker colors represent higher density)

Figure 2.3: Different visualizations of a two-dimensional GMM consisting of three Gaussians.

2.3.2 Isosurfaces

As an alternative to rendering iso-ellipsoids for each Gaussian, one can also display one or several isosurfaces of the GMM's probability density function. Isosurfaces have been used in both 2D

GMM visualization [Bis06] as well as in 3D [Kaw18]. A two-dimensional example is displayed in Figure 2.3a.

One approach for rendering three-dimensional isosurfaces of arbitrary three-dimensional functions is to cast rays through the image pixels and find the position of the desired isovalue on that ray. This could be done by using simple *ray-marching* [TT84], where the GMM is sampled at equidistant sample points along the ray until an isovalue crossing is found. This method has the risk of missing isovalue crossings when the interval step size is too high for the given GMM. More sophisticated methods use numerical methods such as Newton's method or regula falsi [Har93]. One such method, that works on sums of isotropic Gaussians has been proposed by Blinn [Bli82] and could be generalized to work on arbitrary GMMs.

An alternative approach is to sample the GMM's density function in a grid, from which the surface can then be extracted as a mesh using Marching Cubes [LC87]. The resolution of the grid needs to be high enough to not miss important details. Eckart [Eck17] proposed an extension of Marching Cubes specifically for isosurface extraction of hierarchical GMMs.

2.3.3 Density Visualization

Instead of visualizing individual Gaussians or isosurfaces, one can also aim for the visualization of the values of the three-dimensional probability density function itself, analogous to a heatmap visualization in 2D (Figure 2.3b). Density visualizations of three-dimensional GMMs have been used in [PMA+14], [Eck17], and [JRJ11]. Using this visualization technique the users can easily get an impression of what distribution or object the GMM is representing. Information about the parameters of the individual Gaussians is lost, however.

The rendering of scalar fields is well researched in the field of *volume rendering*, which deals with the generation of images from 3D volumetric data. These techniques are commonly used in medical visualization to visualize data acquired through computed tomography (CT) or magnetic resonance imaging (MRI). The following content of this section is based on [EHK+06].

To create realistic volume renderings, mathematical models based on physics are required. One such model is the emission-absorption model. In this model, the gas particles are able to emit and absorb light, but more complex effects, such as scattering or indirect illumination, are not possible. We will take a look at this model in more detail here:

In the emission-absorption-model, particles both emit light and absorb incoming light.To calculate the color of a point on the image plane, we shoot a ray from the eye through the point into the volume. We then integrate the values along this ray.

If we assume a model based on absorption only, the background radiance I_0 is absorbed by the particles in front. The light reaching the eye would therefore be calculated by:

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} \tag{2.24}$$

s_0 represents the position where the light enters the volume from the back. D is the position where the light leaves the volume. $\kappa(t)$ is the non-negative absorption coefficient at ray position t.

If the emission of every particle along the ray is considered as well, the formula is adapted to the so-called *volume-rendering integral*:

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{D} q(s) e^{-\int_{s}^{D} \kappa(t)dt} ds \tag{2.25}$$

The term left of the plus sign is simply the absorption formula defined before. The right term takes the particles' emissions into account. $q(s)$ is called the *source term* and describes the light radiance emitted at ray position s. The exponential function inside the integral ensures that this newly emitted light is absorbed by the particles in front in the same way as the background radiance.

This integral is the basis for many volume rendering techniques. In practice, this model is usually not applied directly, as it is not always an option to solve this integral. In most applications, the data is provided as a voxel grid rather than a continuous function. Therefore, the integral is instead approximated by Riemann sums, using a combination of ray-marching and alpha blending.

When visualizing GMMs, the volume-rendering integral can be evaluated analytically for simple choices of q and κ. Existing visualizations by Eckart et al. [EKT+16] and Preiner et al. [PMA+14] simply accumulate all density values along the viewing ray, which corresponds to $q(s)$ being the GMM's density at the ray position s, and no absorption taking place (i.e. $\kappa(t) = 0$). Jakob et al. use GMMs to render homogeneous participating media [JRJ11]. They choose $q(s)$ to be the density value as well, and choose $\kappa(t)$ to be a constant σ_t.

CHAPTER 3

Visualization

This chapter describes the algorithms we use for visualizing GMMs. Example applications of these methods are the examination of fitting results and the introspection of deep learning layers that process GMMs.

Two visualization techniques have been implemented: The rendering of Gaussian isoellipsoids, and additive density visualization. Sections 3.1 and 3.2 describe these techniques respectively. In Section 3.3 the graphical user interface is described.

3.1 Isoellipsoid Rendering

A very straightforward GM visualization technique is displaying a 1-σ-ellipsoid per Gaussian, indicating its mean and covariance. These ellipsoids are defined as described in Section 2.3.1 as the surface which fulfills:

$$(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k) = 1 \tag{3.1}$$

The three axes of the isoellipsoids are defined as $\pm \sqrt{\lambda_i} \boldsymbol{e}_i$, where λ_i is the ith eigenvalue of $\boldsymbol{\Sigma}_k$, and $\boldsymbol{e}_i$ is the normalized eigenvector of $\boldsymbol{\Sigma}_k$. We use instanced rendering to display the ellipsoids. That is, one sphere (described by 1,984 triangles) is rendered multiple times, once per Gaussian, with corresponding transformation matrices. The 4×4 transformation matrix for a Gaussian (using homogeneous coordinates) is calculated from the three-dimensional eigenvectors in the following way:

$$\begin{pmatrix} \sqrt{\lambda_1}\boldsymbol{e}_1 & \sqrt{\lambda_2}\boldsymbol{e}_2 & \sqrt{\lambda_3}\boldsymbol{e}_3 & \boldsymbol{\mu}_k \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

In case the determinant of the resulting transformation matrix is negative, we switch the signs of the first three columns to provide proper face ordering. Backface culling has to be enabled to prevent otherwise occurring z-fighting for very thin ellipsoids.

Isoellipsoid visualization provides insight into the means and covariances of the Gaussians. A rendering of the original point cloud can be easily combined with this visualization. However,

the Gaussians' weights ω_k are not considered in this visualization. To provide information about these, we have implemented the option to color the ellipsoids by their weight (see Figure 3.1).

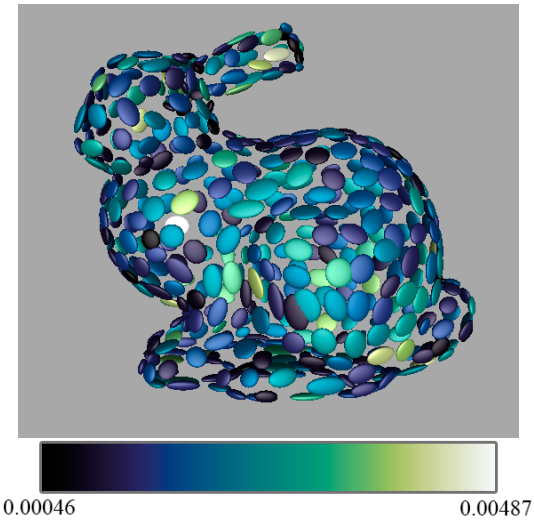
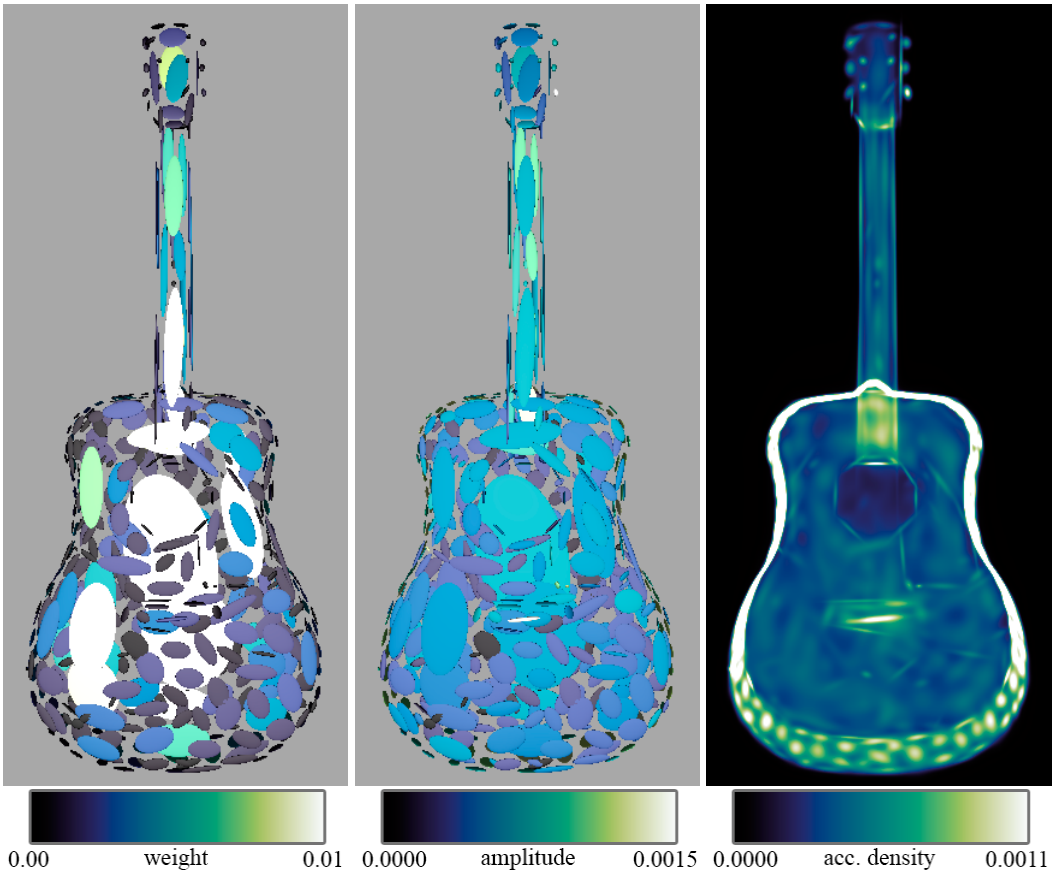


Figure 3.1: Results of ellipsoid rendering of a GMM of 512 Gaussians representing the Standford Bunny. The colors of the Gaussians indicate their weights.

Alternatively to coloring by the weights ω_k, the ellipsoids can also be colored by the Gaussian *amplitudes*. While the weights describe the total amount of influence of the Gaussian on the GMM, the Gaussian's amplitude is the highest density value of the Gaussian and is calculated by $\omega_k / \sqrt{(2\pi)^3 |\boldsymbol{\Sigma_k}|}$. Gaussians with smaller isoellipsoids have a higher amplitude than Gaussians with larger isoellipsoids of the same weight. Therefore, the small-ellipsoid Gaussians may have a stronger impact on the GMM's density values in the area of their ellipsoid. Because of this, the isoellipsoid rendering is often more similar to the density visualization when using amplitude coloring rather than weight coloring. This is demonstrated in Figure 3.2.

3.2 Density Visualization

While the isoellipsoid rendering provides information about the location and extent of the individual Gaussians, it does not provide a clear visualization of the density values the mixture assigns to each point in 3D space. Density visualizations aim to visualize the probability density function more directly. In this section, we will discuss our approach for density visualization and our implementation. Section 3.2.1 describes the approach for our technique. Section 3.2.2 describes how to calculate the volume-rendering integral, which is a central part of our algorithm. Section 3.2.3 describes further implementation details and Section 3.2.4 describes a technique for accelerating the rendering. Section 3.2.5 describes open challenges and future work.



(a) Ellipsoids colored by weights (b) Ellipsoids colored by amplitudes (c) Density visualization

Figure 3.2: Comparison of different visualization techniques on a GMM representing the model *guitar_0001* from ModelNet 40 [WSK+15]. In (a), we see that the larger Gaussians have the highest weights. However, in (b), it becomes visible that some small Gaussians have equal or higher amplitudes, such as the ones at the tuning pegs or on the bridge. These Gaussians produce higher density values, which can also be seen in the accumulated density visualization in (c).

3.2.1 Approach

Our density visualization is based on the *volume-rendering integral* (Equation 2.25). Usually, volume data is provided as a 3D grid of values, and the volume-rendering integral is approximated using ray marching and alpha blending. Our problem differs from classical volume visualization in that we do not start from a grid of samples but an analytical density function. By using a ray-marching-based approach, we would lose the advantage of having an exact analytical description of the GMM. Therefore, we will integrate the density along the ray analytically.

The volume-rendering integral is defined for each pixel on the viewing ray

$$\boldsymbol{r}(t) = \boldsymbol{x}_0 + t\boldsymbol{d}, \tag{3.3}$$

where $\boldsymbol{x}_0$ is the origin of the ray (the camera position), $\boldsymbol{d}$ the normalized direction from the camera to the pixel on the image plane, and t the distance from the origin. By using the GMM's density $f_{\boldsymbol{\Theta}}$ as source term q and using neither absorption nor background radiance (i.e. $\kappa(t) = 0, I_0 = 0$), the volume-rendering integral from Equation 2.25 becomes the accumulation of density values along the viewing ray:

$$\int_0^\infty f_{\boldsymbol{\Theta}}(\boldsymbol{r}(t))\mathrm{d}t \tag{3.4}$$

The next section describes how this integral is solved. The resulting value is then mapped to a color scheme to produce the final pixel color.

This approach is similar to what other authors have done to visualize their GMMs, such as Eckart et al. [EKT+16] and Preiner et al. [PMA+14]. These authors did not provide the formulas required for the rendering, so we will explore the details of this technique further in the next sections.

3.2.2 Solving the Volume-Rendering Integral

In this section, we discuss the volume-rendering integral in Equation 3.4, i.e. integrating a GMM's density values along a ray. The GMM's probability density function (*PDF*) is defined as

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \sum_{k=1}^{K} \omega_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{3.5}$$

where K is the number of Gaussians, ω_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the weight, mean and covariance matrix of the corresponding Gaussian respectively (see Section 2.1). The volume-rendering integral – which we will denote as $F_{\boldsymbol{\Theta}}(\boldsymbol{r})$ from now on – is therefore:

$$\begin{aligned} F_{\boldsymbol{\Theta}}(\boldsymbol{r}) = \int_0^\infty f_{\boldsymbol{\Theta}}(\boldsymbol{r}(t))\mathrm{d}t &= \int_0^\infty \sum_{k=1}^{K} \omega_k \mathcal{N}(\boldsymbol{r}(t)|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\mathrm{d}t \\ &= \sum_{k=1}^{K} \int_0^\infty \omega_k \mathcal{N}(\boldsymbol{r}(t)|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\mathrm{d}t \end{aligned} \tag{3.6}$$

To calculate the integral over the GMM's PDF, it is sufficient to calculate the integral of each Gaussian, and then sum up the results. To help solve this integral, we developed Theorem 1 (illustrated in Figure 3.3):

Theorem 1. *The values of a weighted Gaussian's PDF with parameters $\omega, \boldsymbol{\mu}, \boldsymbol{\Sigma}$ along a ray $\boldsymbol{r}(t) = \boldsymbol{x}_0 + t\boldsymbol{d}$ are described by a weighted one-dimensional Gaussian with mean μ', variance σ'^2 and weight ω':*

$$\omega \mathcal{N}(\boldsymbol{r}(t)|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \omega' \mathcal{N}(t|\mu', \sigma'^2) \tag{3.7}$$

where the parameters have the following values:

$$\mu' = \frac{\boldsymbol{d}^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu} - \boldsymbol{x}_0)}{\boldsymbol{d}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{d}} \tag{3.8}$$

$$\sigma'^2 = \frac{1}{\boldsymbol{d}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{d}} \tag{3.9}$$

$$\omega' = \omega \sqrt{2\pi\sigma'^2} \mathcal{N}(\boldsymbol{r}(\mu')|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.10}$$

Proof. By Replacing the $\mathcal{N}$-notations by the definition of the Gaussian distribution, Equation 3.7 can be written as

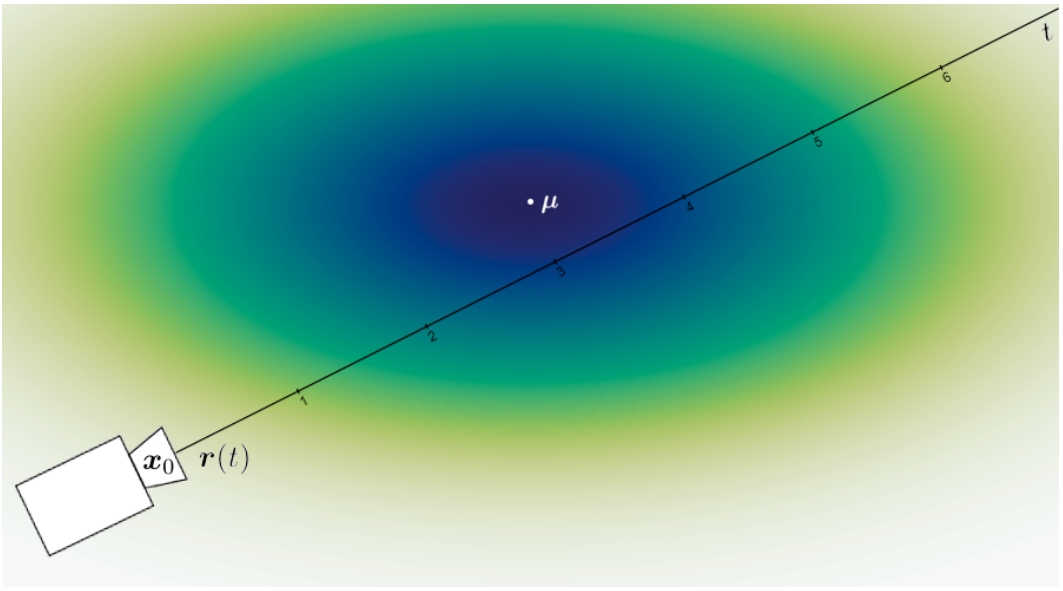
$$\frac{\omega}{\sqrt{(2\pi)^3 |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\boldsymbol{r}(t) - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{r}(t) - \boldsymbol{\mu})\right) = \frac{\omega'}{\sqrt{2\pi\sigma'^2}} \exp\left(-\frac{1}{2}\left(\frac{(t-\mu')^2}{\sigma'^2}\right)\right). \tag{3.11}$$

We plug in the definition of $\boldsymbol{r}(t)$ into the left side and rewrite it as follows:

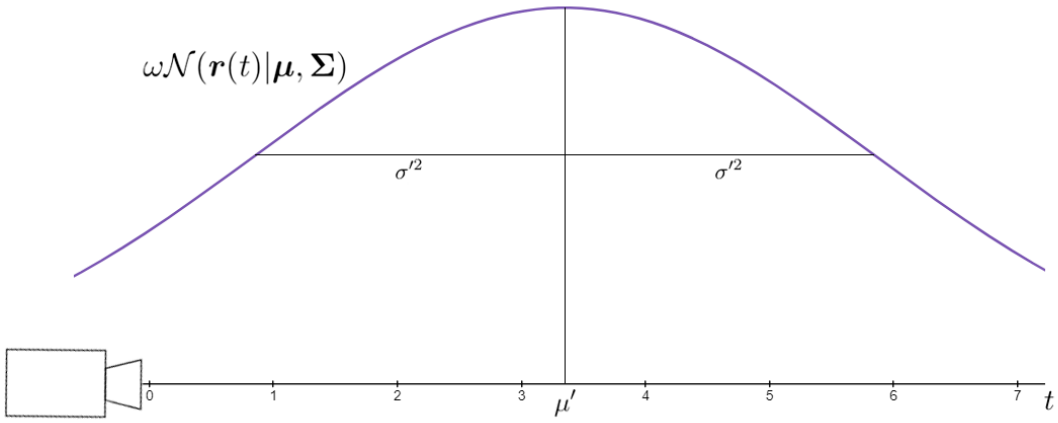
$$\begin{aligned}
&\frac{\omega}{\sqrt{(2\pi)^3 |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\boldsymbol{r}(t) - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{r}(t) - \boldsymbol{\mu})\right) \\
=&\frac{\omega}{\sqrt{(2\pi)^3 |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\boldsymbol{x}_0 + t\boldsymbol{d} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_0 + t\boldsymbol{d} - \boldsymbol{\mu})\right) \\
=&\frac{\omega}{\sqrt{(2\pi)^3 |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} \left((\boldsymbol{d}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{d}) t^2 + (2\boldsymbol{d}^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_0 - \boldsymbol{\mu})) t + \right.\right. \\
&\qquad \left.\left. (\boldsymbol{x}_0^T \boldsymbol{\Sigma}^{-1} \boldsymbol{x}_0 - 2\boldsymbol{x}_0^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu})\right)\right)
\end{aligned} \tag{3.12}$$

The right side of Equation 3.11 is rewritten as follows:

$$\frac{\omega'}{\sqrt{2\pi\sigma'^2}} \exp\left(-\frac{1}{2}\left(\frac{(t-\mu')^2}{\sigma'^2}\right)\right) = \frac{\omega'}{\sqrt{2\pi\sigma'^2}} \exp\left(-\frac{1}{2}\left(\frac{1}{\sigma'^2} t^2 - \frac{2\mu'}{\sigma'^2} t + \frac{\mu'^2}{\sigma'^2}\right)\right) \tag{3.13}$$



(a) 2D-Illustration of the viewing ray $\boldsymbol{r}(t)$ from the camera $\boldsymbol{x}_0$ through a Gaussian (represented by a density heatmap – darker values represent higher density). $\boldsymbol{\mu}_k$ marks the mean of the Gaussian.



(b) Graph of the the Gaussian's PDF along the ray. As Theorem 1 states, this function is itself a one-dimensional Gaussian with mean μ' and variance σ'^2, scaled by ω'.

Figure 3.3: Illustration of Theorem 1.

Taking the logarithm on both sides enables us to compare the coefficients. Comparing the coefficients of t^2 we get

$$\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{d} = \frac{1}{\sigma'^2} \longrightarrow \sigma'^2 = \frac{1}{\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{d}}, \tag{3.14}$$

confirming Equation 3.9. Comparing the coefficients of t results in:

$$2\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_0-\boldsymbol{\mu}) = -\frac{2\mu'}{\sigma'^2} \longrightarrow \mu' = -\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_0-\boldsymbol{\mu})\sigma'^2 = \frac{\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{x}_0)}{\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{d}}, \tag{3.15}$$

confirming Equation 3.8. Comparing the remainder of the equation results in

$$\frac{\omega}{\sqrt{(2\pi)^3|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}\left((\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x}_0 - 2\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})\right)\right) = \frac{\omega'}{\sqrt{2\pi\sigma'^2}}\exp\left(-\frac{\mu'^2}{2\sigma'^2}\right). \tag{3.16}$$

By rearranging for ω' we get

$$\omega' = \frac{\omega\sqrt{2\pi\sigma'^2}}{\sqrt{(2\pi)^3|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}\left((\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x}_0 - 2\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}) - \frac{\mu'^2}{\sigma'^2}\right)\right). \tag{3.17}$$

Because of

$$-\frac{\mu'^2}{\sigma'^2} = (1-2)\frac{\mu'^2}{\sigma'^2} = \frac{1}{\sigma'^2}\mu'^2 - \left(2\frac{\mu'}{\sigma'^2}\right)\mu', \tag{3.18}$$

the exponent in Equation 3.17 can be rewritten as

$$\begin{aligned}
&-\frac{1}{2}\left(\left(\frac{1}{\sigma'^2}\right)\mu'^2 - \left(2\frac{\mu'}{\sigma'^2}\right)\mu' + (\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x}_0 - 2\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})\right)\\
&= -\frac{1}{2}\left((\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{d})\mu'^2 + (2\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_0-\boldsymbol{\mu}))\mu' + (\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x}_0 - 2\boldsymbol{x}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})\right)\\
&= -\frac{1}{2}\left(\boldsymbol{r}(\mu')-\boldsymbol{\mu}\right)^T\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{r}(\mu')-\boldsymbol{\mu}\right).
\end{aligned} \tag{3.19}$$

The last step applies the inverse operations of the rearrangements in Equation 3.12. We can conclude

$$\omega' = \frac{\omega\sqrt{2\pi\sigma'^2}}{\sqrt{(2\pi)^3|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}\left(\boldsymbol{r}(\mu')-\boldsymbol{\mu}\right)^T\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{r}(\mu')-\boldsymbol{\mu}\right)\right) = \omega\sqrt{2\pi\sigma'^2}\mathcal{N}(\boldsymbol{r}(\mu')|\boldsymbol{\mu},\boldsymbol{\Sigma}), \tag{3.20}$$

confirming Equation 3.10 and proving Theorem 1. □

Having proven this theorem, we use this knowledge for solving the volume-rendering integral $F_{\boldsymbol{\Theta}}(\boldsymbol{r})$. Each evaluation of a Gaussian $\{\omega_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$ along the ray can be replaced by an evaluation of the corresponding one-dimensional Gaussian $\{\omega'_k, \mu'_k, \sigma'^2_k\}$:

$$\begin{aligned}
F_{\boldsymbol{\Theta}}(\boldsymbol{r}) &= \sum_{k=1}^{K}\int_0^\infty \omega_k\mathcal{N}(\boldsymbol{r}(t)|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\mathrm{d}t\\
&= \sum_{k=1}^{K}\omega'_k\int_0^\infty \mathcal{N}(t|\mu'_k,\sigma'^2_k)\mathrm{d}t
\end{aligned} \tag{3.21}$$

We know that $\int_{-\infty}^{x} \mathcal{N}(t|\mu, \sigma^2)\mathrm{d}t = \Phi\left(\frac{x-\mu}{\sigma}\right)$, where $\Phi(x)$ is the cumulative distribution function of the standard Gaussian distribution, i.e. $\Phi(x) = \int_{-\infty}^{x} \mathcal{N}(t|0,1)\mathrm{d}t$. Using this knowledge as well as Φ's properties regarding limits and symmetry, the integral can be replaced:

$$\begin{aligned}\sum_{k=1}^{K} \omega'_k \int_0^{\infty} \mathcal{N}(t|\mu'_k, \sigma'^2_k)\mathrm{d}t &= \sum_{k=1}^{K} \omega'_k \lim_{x\to\infty}\left(\Phi\left(\frac{x-\mu'_k}{\sigma'_k}\right) - \Phi\left(\frac{0-\mu'_k}{\sigma'_k}\right)\right) \\ &= \sum_{k=1}^{K} \omega'_k \left(1 - \Phi\left(\frac{-\mu'_k}{\sigma'_k}\right)\right) \\ &= \sum_{k=1}^{K} \omega'_k \Phi\left(\frac{\mu'_k}{\sigma'_k}\right)\end{aligned} \tag{3.22}$$

This means the volume-rendering integral can be calculated by evaluating Equation 3.23.

$$F_{\boldsymbol{\Theta}}(\boldsymbol{r}) = \sum_{k=1}^{K} \omega'_k \Phi\left(\frac{\mu'_k}{\sigma'_k}\right) \tag{3.23}$$

Alternatively, this can also be expressed in terms of the *error function* $\mathrm{erf}(x) = 2\Phi(x\sqrt{2}) - 1$ or the *complementary error function* $\mathrm{erfc}(x) = 2\Phi(-x\sqrt{2})$.

$$F_{\boldsymbol{\Theta}}(\boldsymbol{r}) = \sum_{k=1}^{K} \frac{\omega'_k}{2}\left(1 + \mathrm{erf}\left(\frac{\mu'_k}{\sqrt{2}\sigma'_k}\right)\right) = \sum_{k=1}^{K} \frac{\omega'_k}{2}\mathrm{erfc}\left(-\frac{\mu'_k}{\sqrt{2}\sigma'_k}\right) \tag{3.24}$$

Comparison with Related Work

Jakob et al. [JRJ11] examined the following integral along a ray $\boldsymbol{r}(t) = \boldsymbol{x}_0 + t\boldsymbol{d}$:

$$\int_a^b \mathcal{N}(\boldsymbol{r}(t)|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \exp(-\kappa t)\mathrm{d}t \tag{3.25}$$

When setting $\kappa = 0$, this is equal to our volume-rendering integral per Gaussian without the weight ω_k (Equation 3.6). However, the closed-form solution provided in the paper by Jakob et al. does not match ours and appears to be wrong. We found the correct solution for this integral and present it in this section.

For this, we consider a generalization of integral 3.25 with an added variable t_0:

$$\int_a^b \mathcal{N}(\boldsymbol{r}(t)|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \exp(-\kappa(t - t_0))\mathrm{d}t \tag{3.26}$$

When setting $t_0 = a$, this integral is a volume-rendering integral for a single Gaussian with a constant absorption term κ, which we believe could be the integral that should actually be solved

in that paper [JRJ11]. The exponential term reduces the Gaussian's impact to the final image with increasing distance from the ray origin.

The following table compares the solutions provided by Jakob et al. and ours. The differences are highlighted.

Jakob et al.	Ours
$\int_a^b \mathcal{N}(\boldsymbol{r}(t)\|\boldsymbol{\mu},\boldsymbol{\Sigma})\exp(-\kappa\, t)\mathrm{d}t =$ $C_0\left(\operatorname{erf}\left(\frac{C_3+2C_2b}{2\sqrt{C_2}}\right)-\operatorname{erf}\left(\frac{C_3+2C_2a}{2\sqrt{C_2}}\right)\right)$ (3.27)	$\int_a^b \mathcal{N}(\boldsymbol{r}(t)\|\boldsymbol{\mu},\boldsymbol{\Sigma})\exp(-\kappa\,(t-t_0))\mathrm{d}t =$ $C_0'\left(\operatorname{erf}\left(\frac{C_3+2C_2b}{2\sqrt{C_2}}\right)-\operatorname{erf}\left(\frac{C_3+2C_2a}{2\sqrt{C_2}}\right)\right)$ (3.30)
$C_0 = \frac{\exp\left(\frac{C_3^2}{4C_2}-C_1\right)}{(2\pi)^{3/2}\sqrt{\lvert\boldsymbol{\Sigma}\rvert}}$ (3.28)	$C_0' = \frac{\exp\left(\frac{C_3^2}{4C_2}-C_1'\right)}{(4\pi)\sqrt{2C_2}\sqrt{\lvert\boldsymbol{\Sigma}\rvert}}$ (3.31)
$C_1 = \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})-\kappa\, b$ (3.29)	$C_1' = \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})-\kappa\, t_0$ (3.32)

$$C_2 = \frac{1}{2}\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{d} \tag{3.33}$$

$$C_3 = \boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_0-\boldsymbol{\mu})+\kappa \tag{3.34}$$

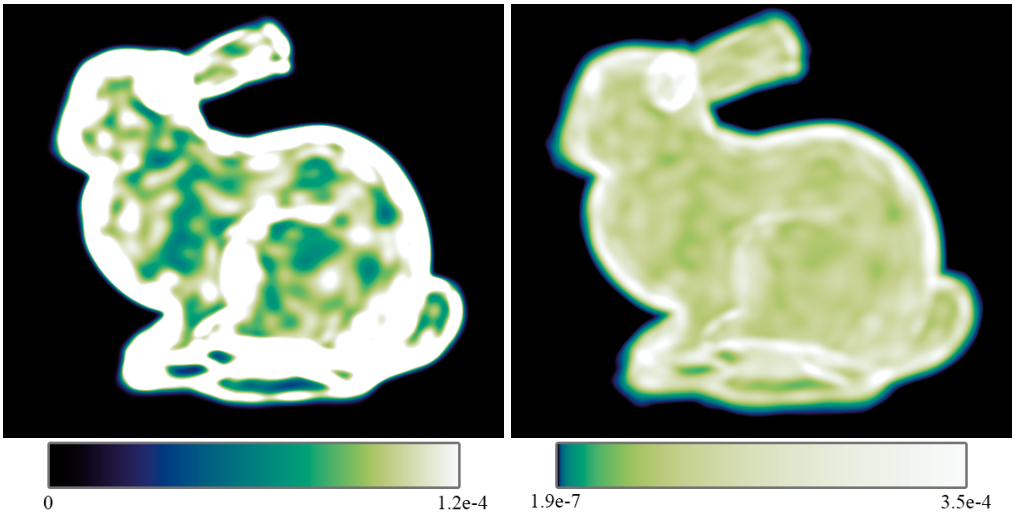
To solve integral 3.26, we first applied Theorem 1 to simplify it and replace the vectors and matrices with scalar values. This made the integral easier to use as input to a symbolic integral solver to find the solution. The integral becomes:

$$\int_a^b \omega'\mathcal{N}(t|\mu',\sigma'^2)\exp(-\kappa(t-t_0))\mathrm{d}t, \tag{3.35}$$

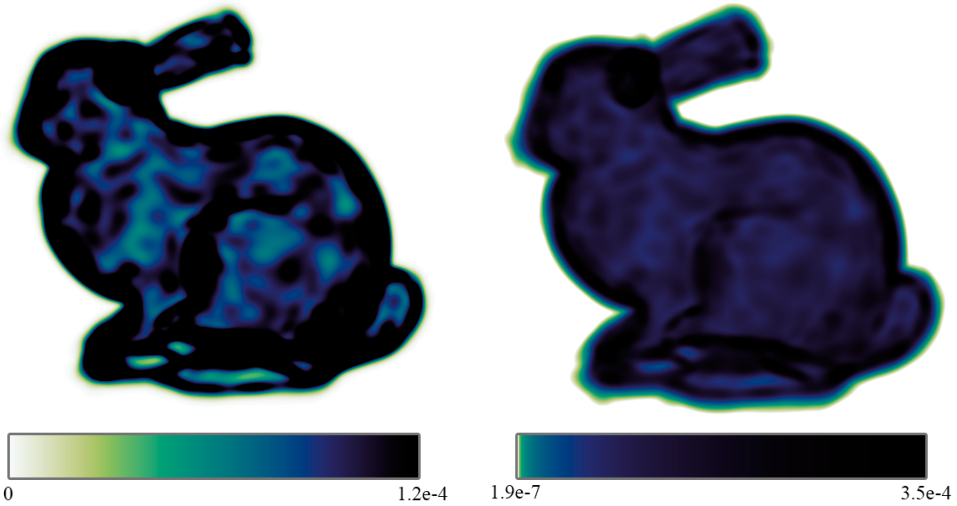
where $\mu' = \frac{\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{x}_0)}{\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{d}}$, $\sigma'^2 = \frac{1}{\boldsymbol{d}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{d}}$, and $\omega' = \sqrt{2\pi\sigma'^2}\mathcal{N}(\boldsymbol{r}(\mu')|\boldsymbol{\mu},\boldsymbol{\Sigma})$. The solution to this integral is

$$\frac{\omega'}{2}\exp\left(\frac{\kappa(\sigma'^2\kappa-2\mu'+2t_0)}{2}\right)\left(\operatorname{erf}\left(\frac{b+\sigma'^2\kappa-\mu'}{\sqrt{2\sigma'^2}}\right)-\operatorname{erf}\left(\frac{a+\sigma'^2\kappa-\mu'}{\sqrt{2\sigma'^2}}\right)\right) \tag{3.36}$$

By rearranging this formula and replacing μ', σ'^2, and ω' with their definitions, we get the result shown in Equation 3.30.



(a) Default density visualization (b) Logarithmic density visualization



(c) Density visualization with inverted colors (d) Inverted log-density visualization

Figure 3.4: Density visualizations of a GMM consisting of 512 Gaussians representing the Stanford Bunny model. Logarithmic visualization makes the object appear more solid, while the default visualization is showing the density differences along the surface more clearly. Values outside of the corresponding colorbar are mapped to the nearest color value.

3.2.3 Implementation Details

GLSL does not provide a function to evaluate Φ or one of the error functions directly in the shader. However, C++'s standard library contains the function `std::erfc`, which approximates the complementary error function. In a preparation step, this function is evaluated at regular sample points. The sample points are stored in an OpenGL texture, which is later sent to the GPU for rendering.

A naive implementation of this rendering principle uses a compute shader to calculate the accumulated density per pixel by calculating ω'_k, μ'_k and σ'_k for each Gaussian and using Equation 3.23 to accumulate all those values. After calculating the accumulated density $F_{\boldsymbol{\Theta}}(\boldsymbol{r})$, a predefined color mapping is applied to generate colors from the calculated values. As other authors have used logarithmic visualizations ([EKT+16]), we also provide the option to calculate the logarithm of the integrals before mapping them to the color table.

This approach produces visually satisfying results (Figure 3.4) and is also able to render generalized Gaussian mixtures with negative weights (Figure 3.5). However, it takes too long to render for real-time purposes, as for each Gaussian, several expensive calculations have to be performed per pixel. For example, rendering one of our test mixtures with 20,000 Gaussians took between 130ms and 140ms (using an Nvidia RTX 2060), which is not practical for an interactive application.

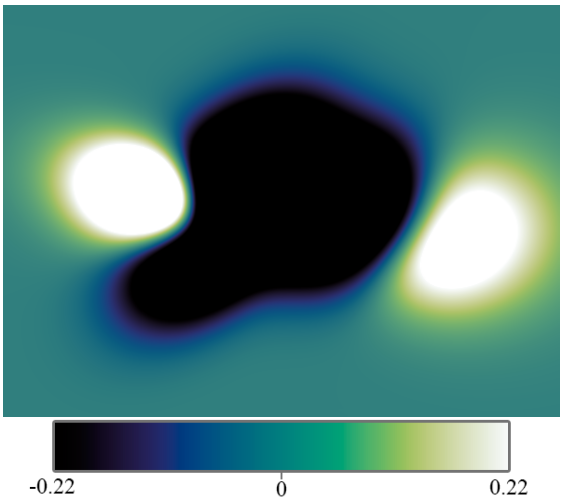


Figure 3.5: Density visualization of a Gaussian mixture with 5 Gaussians, three of which have negative weights. Values outside of the colorbar are mapped to the nearest color value.

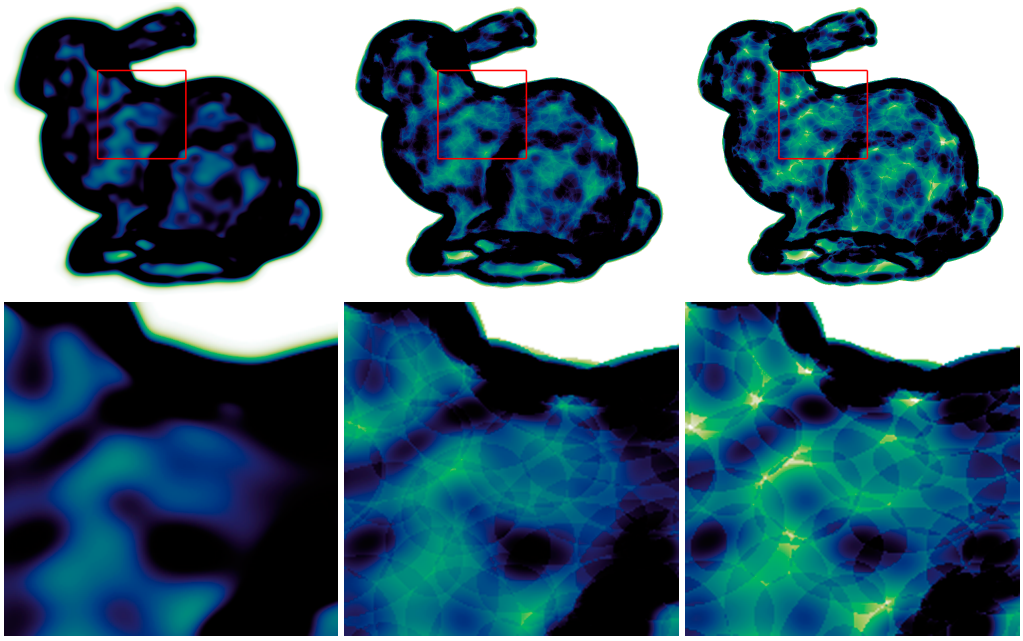
3.2.4 Acceleration

To accelerate the rendering of the density visualization, we set the densities of each Gaussian below a threshold τ to zero. This way, each Gaussian's influence is limited to the inside of its τ-isoellipsoid. These τ-isoellipsoids are rendered in the same way as done in the isoellipsoid visualization (Section 3.1). However, instead of rendering a color into the output texture, the accumulated density of the corresponding Gaussian at the current pixel is calculated in the fragment shader. This is done by calculating μ'_k, σ'_k, ω'_k and $\omega'_k \Phi\left(\frac{\mu'_k}{\sigma'_k}\right)$ (Equation 3.23). The single contributions of all relevant Gaussians are then added automatically into a floating-point texture by using OpenGL's additive blending. For this to work correctly, writing into the depth buffer has to be turned off, so that every Gaussian's contribution is rendered into the buffer independent of the rendering order. Additionally, front-face culling is enabled, so that each Gaussian only

calculates its contributions one time. The resulting texture is then passed to a compute shader, which maps each accumulated density value to a color. This approach has also been described by Jakob et al. [JRJ11].

The choice of τ influences both rendering time and accuracy. A too low τ results in poor accuracy and visible ellipsoid outlines, while choosing τ too high leads to a long rendering time. A simple heuristic we use for determining τ automatically is by multiplying the density value that maps to the maximum of the color table with 0.0001.

This approach improves the rendering time significantly. Rendering a GMM with 20,000 components is accelerated from 140ms to below 10ms, with hardly visible changes in the output result. Figure 3.6 illustrates the difference in accuracy when changing τ.



(a) $\tau = 0.125 \cdot 10^{-8}$ (b) $\tau = 30.125 \cdot 10^{-8}$ (c) $\tau = 50.125 \cdot 10^{-8}$

Figure 3.6: Density visualizations of a GMM consisting of 512 Gaussians representing the Stanford Bunny model with different acceleration parameters τ. (a) is indistinguishable from the exact rendering, while the others show a decrease in accuracy with an increase of τ.

3.2.5 Future Work

The approach described in the previous sections does not include shading effects and depth information. Therefore, important visual cues are missing. Surfaces behind the closest surface shine through and may make it impossible to interpret the result correctly without rotation, especially for scenes with higher depth complexity. The results could be improved by rendering the depth of the nearest isoellipsoids' back faces plus a fixed offset into the depth buffer before the main pass. This would prevent further away Gaussians from being rendered and shining through

the Gaussians in the front. An alternative to this approach would be to use more sophisticated choices for the source term $q(s)$ and the absorption coefficient $\kappa(t)$. Ideally, κ would depend on the density at position t, with higher densities absorbing densities behind them. q would include an illumination model using the PDF's gradient as the normal vector. We are unaware of an exact solution to such an integral. Approximating the integral using ray-marching and alpha-compositing is another possibility. This would require a fast way of transforming the GMM's PDF to a voxel grid.

3.3 Visualizer GUI

We have implemented a graphical user interface using Qt in C++. Figure 3.7 shows a screenshot of the visualizer in use.

The user can load a point cloud and a mixture file. When loading a point cloud or a mixture, the camera position is chosen automatically to fit the data in the viewport. Using mouse controls the camera position can be changed. The available rendering techniques can be enabled and disabled individually. Besides point cloud-, ellipsoids, and density rendering, we also provide a Gaussian positions rendering, which renders a colored point at each Gaussian's center. This is useful for debugging purposes and detecting small Gaussians in the model. Gaussians with zero weight or non-positive definite covariance matrices can be filtered out if desired.

The interface also enables changing rendering parameters, such as the isoellipsoid's threshold value, coloring options for all visualizations, the acceleration threshold, etc.

In the right sidebar, a list of Gaussians is provided, each one represented by an index and its mean position. When selecting a Gaussian from the list, the Gaussian's ellipsoid color changes to red, and additional information is shown in the bottom right part of the interface. It is also possible to pick a Gaussian by clicking on it in the ellipsoid or position visualization.

Besides the graphical user interface, we also created a Python library interface. This is realized using the library Pybind11 [Jak].

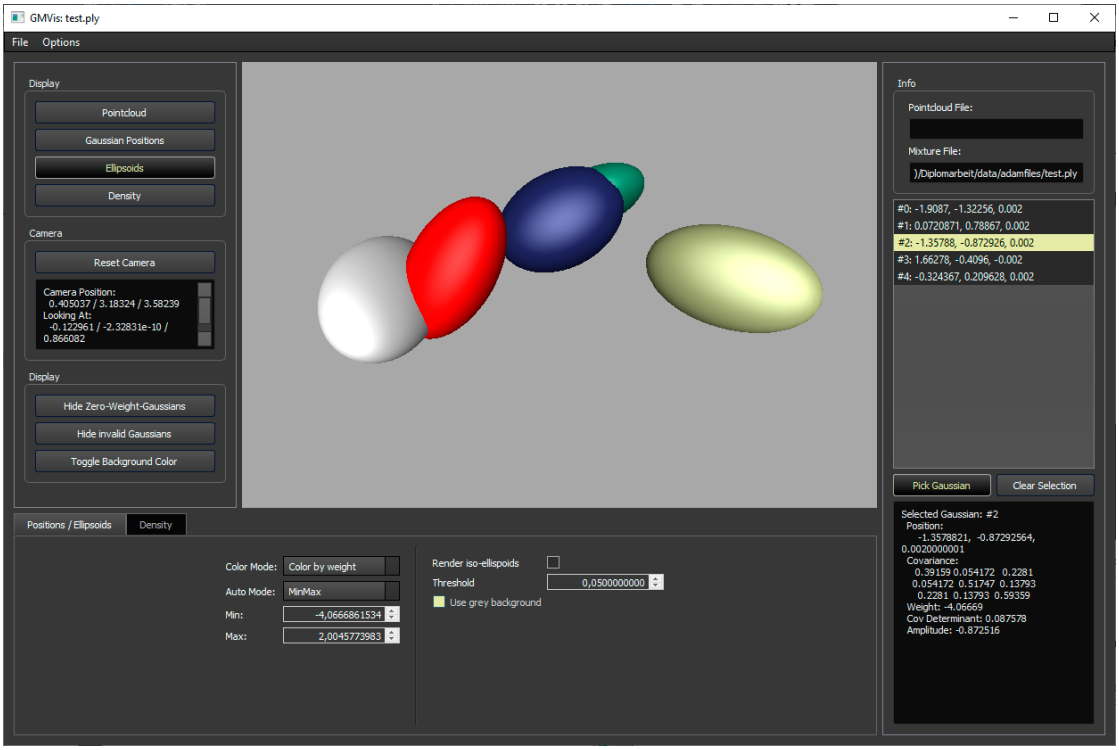


Figure 3.7: Screenshot of our graphical user interface with ellipsoid-rendering active. Gaussian 2 is selected and therefore displayed in red. The other Gaussians are colored by their weight.

CHAPTER 4

Construction: Implementation

In this chapter, we describe our implementations and adaptations of classical EM, top-down HEM by Eckart et al. [EKT+16] and geometrically regularized bottom-up HEM by Preiner et al. [PMA+14]. Both HEM algorithms have been used in 3D-data-processing tasks. We have implemented classical EM and top-down HEM ourselves. The code for geometrically regularized bottom-up HEM was provided by Reinhold Preiner and we adapted it to our needs.

Our goal is to create GMMs with hundreds of Gaussians that describe surfaces in 3D space. This goal leads to other requirements on the algorithms than when using GMMs for different applications, such as describing a low number of clear clusters. In particular, care has to be taken to keeping the algorithms numerically stable, even when working with low-valued responsibilities. Additionally, each Gaussian's minimum thickness has to be limited.

4.1 Classical EM

The EM algorithm we have implemented is based on the description in chapter 2.2.2. CPU-based Python implementations such as the one in *scikit-learn* [scia] are too slow for our use case. We use the GPU to accelerate the computation, which allows us to process thousands of input points and fit several hundred Gaussians. We accomplished this using Python and PyTorch, which enable performing matrix operations on the GPU.

As stated in Section 2.2.2, the EM algorithm for GMM construction works by repeating the so-called expectation (E) and maximization (M) steps. The E-step calculates the responsibilities γ_{nk}, which are the probabilities that point n belongs to Gaussian k, by using the following formula:

$$\gamma_{nk} = \frac{\omega_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \omega_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{4.1}$$

where ω_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the weight, mean and covariance matrix of the kth Gaussian respectively. K is the number of Gaussians. $\boldsymbol{x}_n$ is the nth point of the point cloud.

The M-step follows the E-step and calculates the new values for the Gaussians from the responsibilities calculated in the E-Step using the following equations:

$$N_k = \sum_{n=1}^{N} \gamma_{nk} \tag{4.2}$$

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x}_n \tag{4.3}$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k')(\boldsymbol{x}_n - \boldsymbol{\mu}_k')^T \tag{4.4}$$

$$\omega_k^{\text{new}} = \frac{N_k}{N} \tag{4.5}$$

where N is the number of points.

These two steps are repeated until a convergence criterion is fulfilled. In our implementation, the convergence criterion is fulfilled if the change of the average log-likelihood is less than a given threshold for a given number of iterations.

Implementing these formulas naively causes certain instabilities in the algorithm. The following sections will discuss these problems and the solutions we apply to them.

4.1.1 Numerical Problems

The formula for the weighted Gaussian probability density function, which is used in the E-Step of the algorithm, has some numerically unfavorable properties. The formula is:

$$\omega_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{\omega_k}{\sqrt{(2\pi)^3 |\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2} M(\boldsymbol{x}_n, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right) \tag{4.6}$$

where M is the squared Mahalanobis distance

$$M(\boldsymbol{x}_n, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k). \tag{4.7}$$

If a point $\boldsymbol{x}_n$ is far away from a Gaussian's mean $\boldsymbol{\mu}_k$, M becomes high, which leads to $\exp\left(-\frac{1}{2}M\right)$ becoming small. Therefore, when evaluating Equation 4.6 in code, the result becomes zero very easily, which leads to responsibilities becoming zero as well, even though the responsibilities' real values would be higher and still in the range of floating-point precision. In the worst case, this problem may cause all responsibilities for a Gaussian becoming zero, which leads to undefined means and covariances in the M-step.

To solve this problem, we use an approach taken from an implementation of the EM algorithm by Mo Chen [Che20]. This implementation applies several strategies to increase the numerical

accuracy of the algorithm. One of those strategies is to perform several operations in a logarithmic space. Instead of performing the calculation according to Equation 4.6, we calculate the logarithmized values ϱ_{nk} using the following formula:

$$\varrho_{nk} = \log\left(\omega_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right) = \log\left(\frac{\omega_k}{\sqrt{(2\pi)^3|\boldsymbol{\Sigma}_k|}}\right) - \frac{1}{2}M(\boldsymbol{x}_n, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.8}$$

This way, small values outside the floating-point precision are represented by negative numbers. For example, instead of trying to store 10^{-80}, which exceeds 32-bit floating-point precision, we would store $\log(10^{-80}) = -184.2068$, which can be stored with sufficient precision.

Using this approach, the calculation for the responsibilities γ_{nk} has to be adapted as well. The formula for the responsibilities can be rewritten as follows:

$$\begin{aligned}
\gamma_{nk} &= \frac{\omega_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \omega_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \\
&= \exp\left(\log\left(\omega_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right) - \log\left(\sum_{j=1}^{K} \omega_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right)\right) \\
&= \exp\left(\varrho_{nk} - \log\left(\sum_{j=1}^{K} \exp\left(\varrho_{nj}\right)\right)\right)
\end{aligned} \tag{4.9}$$

The last part of this equation, $\log\left(\sum_{j=1}^{K} \exp\left(\varrho_{nj}\right)\right)$, can be evaluated in a numerically stable way using PyTorch's function *logsumexp* [Con19].

By moving the *exp* evaluation to the end of the calculation of the responsibilities, we keep higher numerical accuracy throughout the calculations. This way, we do not run into the same numerical problems.

Despite the increased numerical stability, in rare cases, it still happens that all responsibilities for a Gaussian become zero. In this case, the position and covariances of such a Gaussian become *not-a-number* (NaN), and the weights become zero. If that happens, our implementation sets the positions and covariances to default values and leaves the weight at zero, so the algorithm can continue without problems.

Another numerical problem occurred during the calculations of the covariance matrices in the M-Step. In principle, the formulas we are using should always create positive definite matrices. However, due to numerical issues, this is not always the case. To ensure the valid continuation of the algorithm, we check if the new matrices and their inverses are still positive definite by checking the signs of the matrix's principal minors. If they are not, we do not use them and keep the covariance matrices from the previous step.

4.1.2 Regularization

A common problem with the EM algorithm, or maximum likelihood approaches for GMM construction in general, is that a Gaussian's position could become equal to a point's position and then reduce its covariances, approaching a singularity, therefore increasing the likelihood of that single point (Section 2.2.2). This could result in a very high overall likelihood, while the mixture model itself becomes useless. Similarly, the smallest eigenvalue of the covariance matrix of a Gaussian that describes a surface can also become smaller and smaller, creating arbitrarily high density values on the surface and creating high densities outside the surface (illustrated in Figure 4.1). To avoid Gaussians becoming arbitrarily small like this we add a small value ε to the eigenvalues of the covariance matrices. This approach was also used in other EM implementations such as [Che20] and [scia]. To do this, we make use of Theorem 2:

Theorem 2. *Let $\boldsymbol{A} = \boldsymbol{B} + \varepsilon \boldsymbol{I}$, where $\boldsymbol{B}$ is a positive-definite symmetric $n \times n$-matrix, $\boldsymbol{I}$ is the identity matrix, and ε is a real number. $\boldsymbol{A}$ has the same eigenvectors as $\boldsymbol{B}$ and its eigenvalues are $\boldsymbol{B}$'s eigenvalues + ε.*

Proof. $\boldsymbol{B}$ can be written as its eigenvalue decomposition $\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$, where $\boldsymbol{Q}$ is the matrix containing the eigenvectors and $\boldsymbol{\Lambda}$ the diagonal matrix containing the eigenvalues. $\varepsilon \boldsymbol{I}$ can be written as $\boldsymbol{Q}\varepsilon\boldsymbol{I}\boldsymbol{Q}^{-1}$. Therefore, $\boldsymbol{A}$ can be written as $\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1} + \boldsymbol{Q}\varepsilon\boldsymbol{I}\boldsymbol{Q}^{-1} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \varepsilon\boldsymbol{I})\boldsymbol{Q}^{-1}$ [sta]. As the latter is an eigenvalue decomposition of $\boldsymbol{A}$, this shows that $\boldsymbol{A}$ has the same eigenvectors as $\boldsymbol{B}$, while the eigenvalues are increased by ε. □

Using this theorem, the M-step is adapted such that it adds ε_{abs} to the diagonal elements of the original covariance matrix:

$$\boldsymbol{\Sigma}_k^{\text{reg}} = \boldsymbol{\Sigma}_k^{\text{new}} + \varepsilon_{\text{abs}} \boldsymbol{I} \tag{4.10}$$

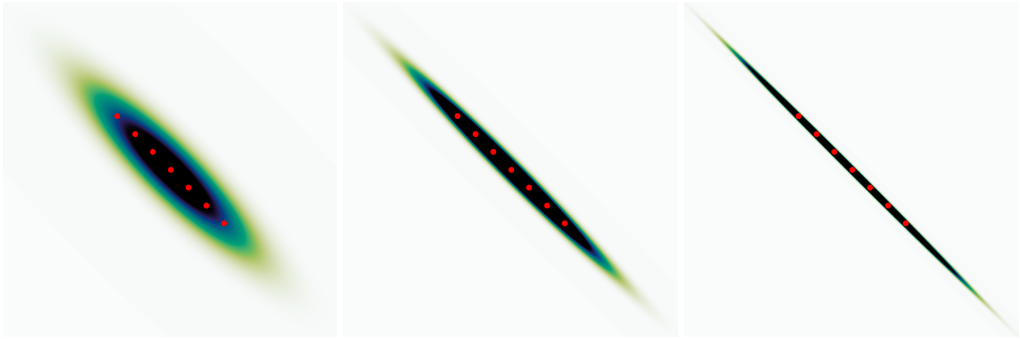


Figure 4.1: Illustration of several 2D-Gaussians fitted on points (red) with the only difference being the smaller eigenvalue decreasing from left to right. Without regularization, Gaussians can shrink onto the points of a surface, increasing densities on those points arbitrarily high. Densities outside of the points (upper left and lower right in the graphics) also increase this way. With regularization, Gaussians cannot be arbitrarily thinned.

$\boldsymbol{\Sigma}_k^{\text{new}}$ comes from Equation 4.4 and $\boldsymbol{I}$ is the identity matrix.

To preserve scale invariance of the results, we chose ε_{abs} depending on the point cloud's bounding box size. We chose abs = $\max(\varepsilon_{\text{rel}} \cdot p, 10^{-9})$, where p is the length of the longest side of the point cloud's bounding box and ε_{rel} is a free parameter. We chose 10^{-9} as a lower limit for ε_{abs} to avoid numerical problems. Choosing the right ε_{rel} controls the Gaussian's "thickness", avoids convergence to singularities and numerical problems, and also prevents the algorithm from creating high densities far away from the surface at the cost of decreased likelihood.

4.1.3 Initialization

We tested several initialization strategies for the EM algorithm.

Method *rnp* (*random normal positions*) is a method described in [MP04]. Here, the Gaussian positions are sampled from a normal distribution:

$$\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k \overset{\text{i.i.d}}{\sim} \mathcal{N}(\overline{\boldsymbol{x}}, \boldsymbol{V}) \tag{4.11}$$

where $\overline{\boldsymbol{x}}$ is the average sample point and $\boldsymbol{V}$ is the sample covariance matrix of the point cloud. The covariance matrices of the Gaussians are initialized as $\boldsymbol{V}$ and the weights are initialized as $1/K$, where K is the number of Gaussians.

Method *k-means* applies the k-means clustering algorithm on the point data. The centers of the clusters are used as Gaussian means. The initial covariances are calculated from all the points that have been assigned to the respective cluster. As a complete k-means execution can be very time-consuming, we only use 20 iterations for this initialization. We use the k-means implementation by scikit-learn [scib].

Method *fps* serves as a faster alternative to k-means and performs farthest point sampling (code from [QYSG17]) to sample K roughly equally spaced points from the point cloud as candidates for Gaussian means. Each point from the original point cloud is then assigned to its nearest Gaussian mean and a single M-step is performed to calculate the initial model.

The impact of the initialization techniques on the results is discussed in Section 5.5.1.

4.1.4 PyTorch Implementation

PyTorch offers mathematical functions on vectors, matrices, and tensors, similar to Matlab. To perform the necessary operations on tensors, we need to expand the input points, which are stored in a tensor of shape $(N \times 3)$, along a new axis K times, where K is the number of Gaussians. Similarly, the GMM's parameters need to be repeated N times, where N is the number of points. This way, both input points as well as Gaussian means are described by $(N \times K \times *)$-tensors, which makes them easy to combine and the formulas straightforward to implement (* is 3 for positions and Gaussian means, 3×3 for covariances, and 1 for weights). The implementation can be extended to construct several GMMs of the same size at once by adding a batch dimension to the beginning of each tensor. When using high numbers of points or

Gaussians, the implementation needs to be extended such that only a certain number of point-Gaussian combinations are considered at once to avoid out-of-memory errors. However, storing the responsibility matrix stays a memory bottleneck and limits the maximum numbers of points and Gaussians.

While our Python implementation is faster than CPU-based implementations such as the one by scikit-learn, it does not reach the speed described for the C++/CUDA-implementation by Eckart et al. [EKT+16]. Execution times for different settings are described in Section 5.5.1.

4.2 Top-Down HEM

Eckart et al. [EKT+16] described a hierarchical EM algorithm (Section 2.2.3). The key idea of this algorithm is to start by fitting a mixture containing j Gaussians, and then repeatedly refining it by replacing each Gaussian with a new sub-GMM consisting of j new Gaussians. The final number of Gaussians is j^l, where l is the number of levels. This process is illustrated in an example in Figure 4.2.

In each subdivision step, points are assigned to sub-GMMs using a partitioning scheme. This way, only a subset of all point-Gaussian pairs has to be considered each iteration, which leads to the increase in performance. Hard partitioning assigns each point to one sub-GMM, soft partitioning assigns each point to all sub-GMMs with corresponding responsibility larger than a threshold λ_l. The original implementation of Eckart et al. calculates zeroth, first, and second moments for each Gaussian in the E-step (formulas for soft partitioning) by

$$T_k^0 = \sum_n p_{nq}\gamma_{nk}, \quad T_k^1 = \sum_n p_{nq}\gamma_{nk}\boldsymbol{x}_n, \quad T_k^2 = \sum_n p_{nq}\gamma_{nk}\boldsymbol{x}_n\boldsymbol{x}_n^T, \tag{4.12}$$

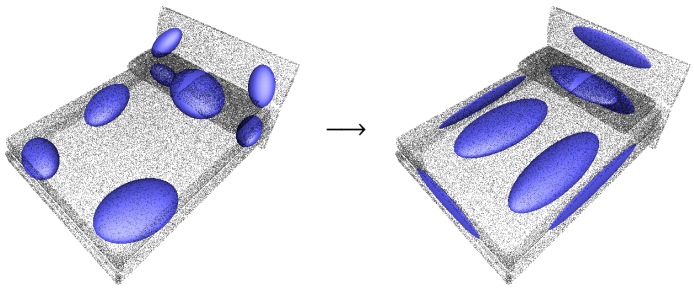
where k is the index of the Gaussian, q is the index of the parent Gaussian, γ_{nk} is the responsibilities of the Gaussian to the nth point, and p is the respective point weighting factor. These moments are used to calculate the new Gaussian parameters in the M-step:

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{\sum_n p_{nq}\gamma_{nk}\boldsymbol{x}_n}{\sum_n p_{nq}\gamma_{nk}} = \frac{T_k^1}{T_k^0} \tag{4.13}$$

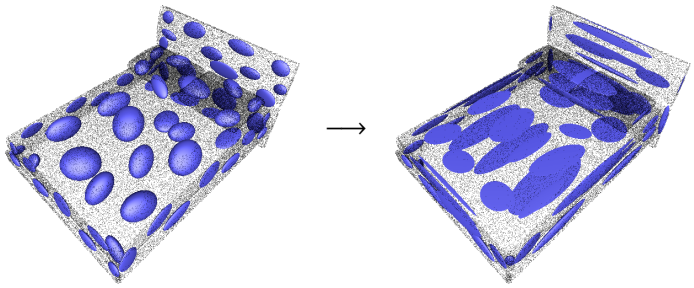
$$\boldsymbol{\Sigma}_k^{new} = \frac{\sum_n p_{nq}\gamma_{nk}\boldsymbol{x}_n\boldsymbol{x}_n^T}{\sum_n p_{nq}\gamma_{nk}} - \boldsymbol{\mu}_k^{new}\boldsymbol{\mu}_k^{newT} = \frac{T_k^2}{T_k^0} - \boldsymbol{\mu}_k^{newT}\boldsymbol{\mu}_k^{new} \tag{4.14}$$

$$\omega_k^{\text{new}} = \sum_n \frac{p_{nq}\gamma_{nk}}{N} = \frac{T_k^0}{N} \tag{4.15}$$

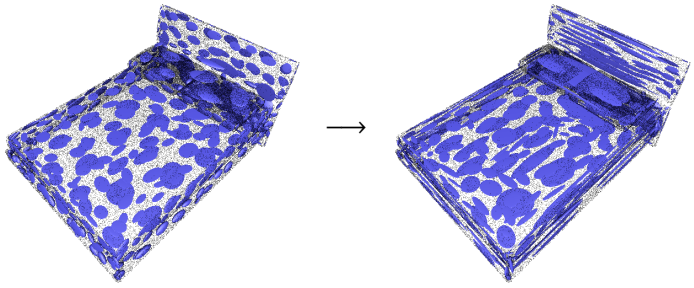
We have implemented the algorithm in Python using PyTorch [pyt], which required some adaptations of the original algorithm so that it fits the tensor-based programming scheme. Also, we have left out the noise cluster from the original algorithm, as we currently work with models that do not contain noise.



(a) Level 1



(b) Level 2



(c) Level 3

Figure 4.2: Isoellipsoid-visualization of the generated GMM at different times during the top-down HEM algorithm using $j = 8$, three levels, and the *fps*-initialization technique. Each row represents one level, the left is the result of the initialization, the right the GMM after the final iteration on that level.

4.2.1 Numerical Problems

To increase the numerical stability of the algorithm, we used the same strategies as in classical EM (Sections 4.1.1, 4.1.2).

Eckart et al. use Equation 4.14 for calculating the covariance matrices. This formula is mathematically equivalent to the original formula from the EM algorithm (Equation 4.4). This

moment-based-version is suggested as it simplifies the parallel computation of the covariance matrices. However, as two high values are subtracted from each other, a lot of accuracy is lost in the later digits, leading to decreased numerical stability and more frequent creation of non-positive definite matrices.

Therefore, we use the original formula from the EM algorithm. As we use a tensor-based strategy to parallelize the calculation of the responsibilities, we are not able to calculate the moments conveniently during the E-step. That is why we experience no loss of efficiency by using the original formula.

4.2.2 Initialization

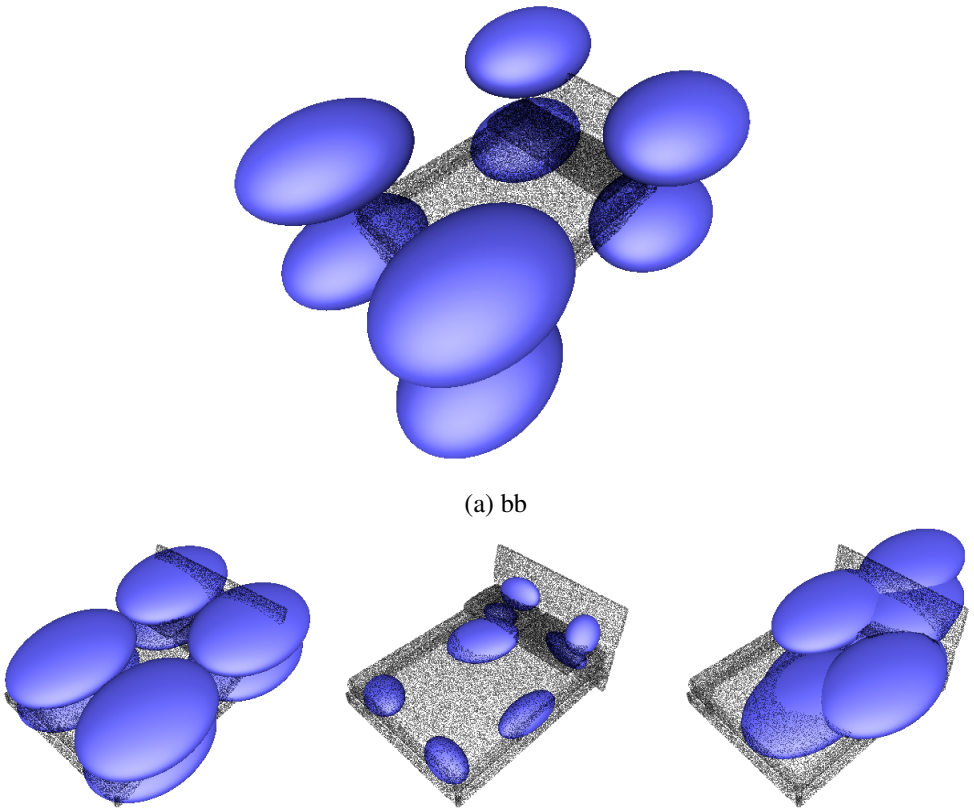
Initialization is performed several times in this algorithm, as both the first coarse mixture and the later sub-GMMs need to be initialized. We implemented the following initialization techniques:

Method *bb* is based on the initialization technique from the original paper. The Gaussian means are initialized at the corners of the axis-aligned bounding box of the respective data. One disadvantage with this approach is that it is not rotation invariant. For example, if we would fit a GMM to a 2D plane embedded in 3D space, we would get different results depending on the rotation of the plane. In the worst case, if the point cloud is planar and close to parallel to the coordinate system's axes, this can lead to several Gaussians ending up at the same position, which effectively reduces the representative power of the GMM. Our test dataset contains a lot of surfaces parallel to the axes, so this is a relevant problem.

We have implemented the method *eigen* as an alternative to *bb* to avoid the problems described above. *eigen* places the initial means on the corners of a box which is centered on the mean of the points. The box's side lengths correspond to twice the eigenvectors of the points' covariance matrix multiplied by the square roots of their corresponding eigenvalues. If the points are planar (i.e. the smallest eigenvalue is smaller than the regularization parameter ε_{abs} (Section 4.1.2)), we instead place the Gaussians on the corners and the middle points of the sides of a rectangle defined by the two largest eigenvectors. The initial covariance matrices are equal to the covariance matrix of the points.

Furthermore we have implemented methods *rnp* and *fps*, which are adaptations of the same-named techniques for classical EM (Section 4.1.3). The four initialization techniques are illustrated in Figure 4.3.

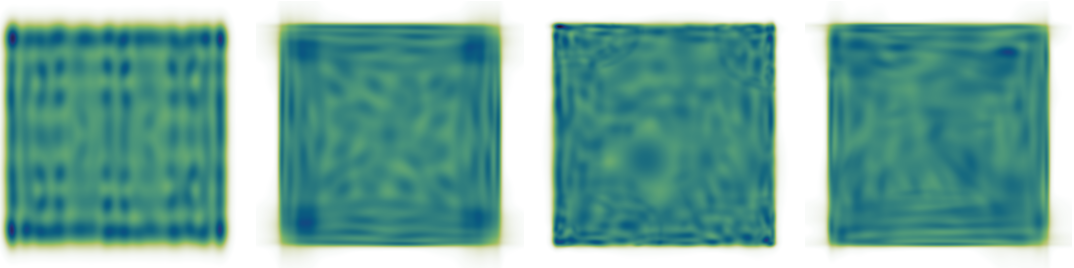
We will discuss the difference of the methods regarding the quality of their results in Chapter 5. Typical results of the algorithm with initialization strategies are shown in Figure 4.4.



(a) bb

(b) eigen (c) fps (d) rnp

Figure 4.3: Results of different initialization techniques for $j = 8$ on the model *bed_0001*.



(a) *bb* (b) *eigen* (c) *fps* (d) *rnp*

Figure 4.4: Final results on a planar point cloud using soft partitioning, initialized by different techniques using $j = 8$ and $l = 3$. *bb* and *eigen* shows symmetries, while *fps* and *rnp* appear more chaotic.

4.2.3 PyTorch Implementation

To get the speedup that the hierarchical algorithm provides we need to make sure that the E-step only calculates the responsibilities of the relevant Gaussian-point-pairs, as all other ones will be zero anyway. To do this, we create a tensor `mask_indices` of shape $(P \times 2)$, where P is the number of relevant Gaussian-point-pairs. For each pair, the index of the point and the index of the Gaussian are stored. To create `mask_indices`, we retrieve all the indices of non-zero point weighting factors p_{nk}. Using these indices, we can fetch all the relevant points and Gaussians and store them in tensors of shape $(P \times *)$ (* is 3 for positions and Gaussian means, 3×3 for covariances, and 1 for weights). This way, we can similarly combine them as in the classical EM algorithm. To calculate the sums of Gaussian densities per point, we make use of PyTorch's `sum`-function by placing all the previously calculated densities into one tensor, on which we can then perform the sum operation. Similarly to regular EM, the algorithm can be extended such that the densities for only a certain number of pairs are calculated at once to prevent out-of-memory errors. Still, as we need to store the whole responsibility matrix, our implementation is limited in its maximum number of points and Gaussians.

In some cases, no points may be assigned to a sub-GMM at all. This leads to the weights of all children becoming zero. To still end up with a correct final GMM in such a case, we replace these children with their parent Gaussian in a postprocessing step.

Our implementation does not reach the speed reported for the C++/CUDA-implementation by Eckart et al. [EKT+16]. Execution times for different settings are described in Section 5.5.2.

4.3 Geometrically Regularized Bottom-Up HEM

The C++-code for the geometrically regularized Bottom-Up HEM [PMA+14] (Section 2.2.4) is open source. We added Python bindings using Pybind, enabling us to call the algorithm from our Python framework. However, for our use case and our comparisons with the other algorithms, we require GMMs with a fixed number of Gaussians. While we could still work with smaller GMMs by filling them up with zero-weight Gaussians, we cannot use GMMs with a higher number of Gaussians. However, this HEM algorithm often convergences before reaching the desired number of Gaussians and therefore produces a much higher number of Gaussians than desired. Therefore, we adapted it as described below.

The algorithm starts with one Gaussian per point and then repeatedly reduces the mixture to a smaller one by a constant reduction factor δ (per default $\delta = 3$). In the original implementation, for each Gaussian it is individually decided if it is copied to the new level using δ^{-1} as the probability. This way, fewer Gaussians than our goal number can be selected as the number of selected Gaussians is not deterministic. To prevent this from happening, our modification always selects $\max(C\delta^{-1}, G)$ random Gaussians, where C is the number of Gaussians on the last level and G is the desired final number of Gaussian.

Afterward, the actual geometrically regularized HEM steps are performed, which adapt the Gaussians in the new level by taking the previous level's Gaussians in its neighborhood into account. However, some of the previous level's Gaussians will not influence any of the new

Gaussians, as they are too far away from every selected Gaussian. These so-called *orphans* will simply be added to the new level after the HEM steps, increasing the number of Gaussians again. This reduction process is repeated until the size of the new model does not differ from the size of the model in the previous level. As this convergence might be reached before reaching the desired Gaussian count, our modification assigns each orphan to the selected Gaussian with the smallest KL-divergence. This way, no orphans remain, and the number of Gaussians is always reduced.

CHAPTER 5

Construction: Evaluation

5.1 Introduction

In this chapter, we will discuss our evaluation and comparison of the selected GMM construction algorithms. Throughout this evaluation, we will fit GMMs on point clouds sampled from meshes from the ModelNet 40 dataset [WSK+15]. For our experiments, we use 200 geometric models, the first five from each category. See Figure 5.1 for a few examples.

We will start by taking a look at the evaluation metrics in Section 5.2, then we discuss normalization concerning spatial scaling in Section 5.3, and finally, we look at the evaluation strategy in Section 5.4. We will discuss the results of the individual algorithms in Section 5.5 and compare them in Section 5.6.

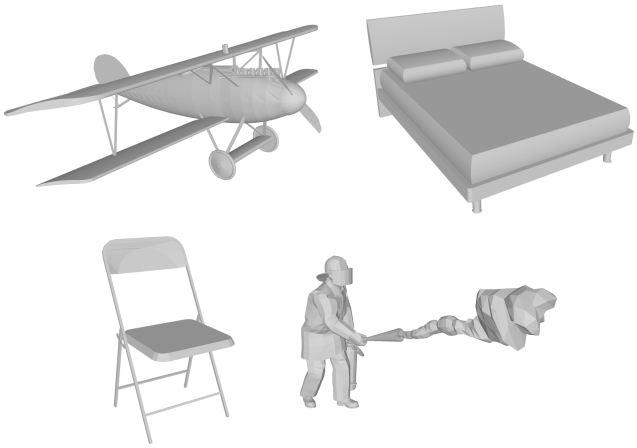


Figure 5.1: Four exemplary models from the ModelNet40 dataset from the categories airplane, bed, chair and person.

5.2 Metrics

We discuss several properties of GMMs and propose metrics to quantify them. In this chapter, we will refer to two different kinds of point clouds: For each mesh, we sample a *source point cloud* P_S uniformly from the mesh's surfaces, used to fit the GMM, and an *evaluation point cloud* P_E used as reference when computing certain metrics. We use P_E to avoid giving high scores to overfitted GMMs that are not a good representation of the underlying surface. Our evaluation point clouds have different sizes for different metrics. We are interested in measuring the log-likelihood, the reconstruction error, irregularity, and Gaussian variation.

5.2.1 Log-Likelihood

The likelihood of GMM $\boldsymbol{\Theta}$ on the point cloud P is defines as

$$L_P(\boldsymbol{\Theta}) = \prod_{\boldsymbol{x} \in P} f_{\boldsymbol{\Theta}}(\boldsymbol{x}), \tag{5.1}$$

where $f_{\boldsymbol{\Theta}}$ is the probability density function of the GMM $\boldsymbol{\Theta}$. We can write the log-likelihood as

$$\log(L_P(\boldsymbol{\Theta})) = \sum_{\boldsymbol{x} \in P} \log(f_{\boldsymbol{\Theta}}(\boldsymbol{x})). \tag{5.2}$$

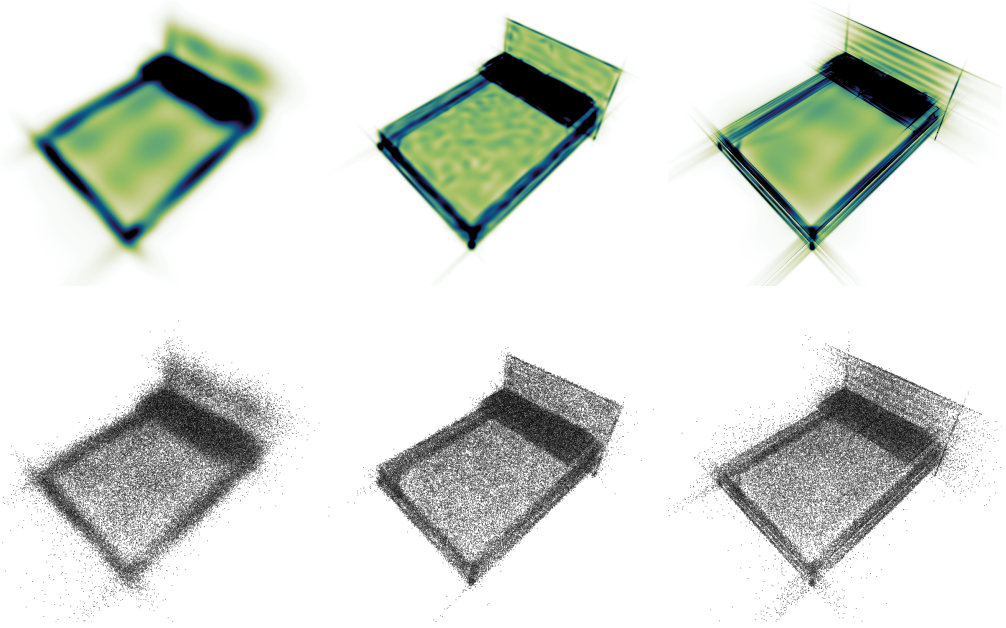
The likelihood on the source point cloud P_S is maximized by the EM algorithm. It is the basis for model selection methods like the Akaike information criterion or the Bayesian information criterion [Bie04]. High density values on the mesh's surface generally lead to a high log-likelihood, making it a candidate for measuring the goodness of fit.

For our evaluation, we track the average log-likelihood per point on the evaluation point cloud P_E

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{|P_E|} \sum_{\boldsymbol{x} \in P_E} \log(f_{\boldsymbol{\Theta}}(\boldsymbol{x})). \tag{5.3}$$

Dividing by the number of points leads to results that are independent of the point count. As we evaluate on an evaluation point cloud P_E rather than the original point cloud P_S, we avoid giving high scores to overfitted models. In our experiments, we chose $|P_E| = 1,000,000$ for calculating $\mathcal{L}$.

The likelihood has several disadvantages: Densities close to zero get logarithmized to large negative numbers, which may skew the result more than appropriate for our use case. Another disadvantage is that the likelihood only considers densities on the surface while ignoring undesired high densities elsewhere. These properties make the likelihood unfavorable for evaluating the GMM's representative power regarding the underlying model. Therefore, we will rarely consider $\mathcal{L}$ in the discussion of the results.



(a) $\mathcal{L} = -10.843$, $\mathcal{R} = 1.780$ (b) $\mathcal{L} = -9.831$, $\mathcal{R} = 0.700$ (c) $\mathcal{L} = -7.739$, $\mathcal{R} = 1.693$

Figure 5.2: Density visualization and reconstructed point clouds for different fits on the same model with the respective log-likelihood $\mathcal{L}$ (higher is better) and reconstruction error $\mathcal{R}$ (lower is better). While (a) gets bad scores for both $\mathcal{L}$ and $\mathcal{R}$, (c) gets a better $\mathcal{L}$ than (b) because it is using thinner Gaussians with higher densities on the surface. However it has worse $\mathcal{R}$ because of reconstructed points outside the original surface.

5.2.2 Reconstruction Error

Another approach to measuring the quality of a fit is to sample a new point cloud $P_{\boldsymbol{\Theta}}$ from the generated GMM $\boldsymbol{\Theta}$ and compare it with the evaluation point cloud P_E. Eckart et al. [EKT+16] used this approach. They calculate the distance from each point in a reference point cloud to its nearest neighbor in the reconstructed point cloud. They then use the root mean square distance as the basis for their metric. However, this does not take potential outlying points in the reconstructed point cloud into consideration. We extend their approach by using the Chamfer distance to calculate the difference between these two point clouds, which is defined as

$$d_{\mathrm{CD}}(P_1, P_2) = \frac{1}{|P_1|} \sum_{\boldsymbol{x} \in P_1} \min_{\boldsymbol{y} \in P_2} \|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \frac{1}{|P_2|} \sum_{\boldsymbol{x} \in P_2} \min_{\boldsymbol{y} \in P_1} \|\boldsymbol{x} - \boldsymbol{y}\|_2^2. \tag{5.4}$$

This technique calculates the squared distance of each point of both clouds to its nearest neighbor in the other point cloud. Averaging these distances gives the result.

The Chamfer distance has been used in Deep Learning to compare point clouds reconstructed from learned representations with their original versions [AYG19][GFK+18]. Because we also generate representations for point clouds, this is a very similar use case, which motivates using the same strategy.

We calculate the reconstruction error $\mathcal{R}$ as the square root of the Chamfer distance between P_E and $P_{\mathbf{\Theta}}$.

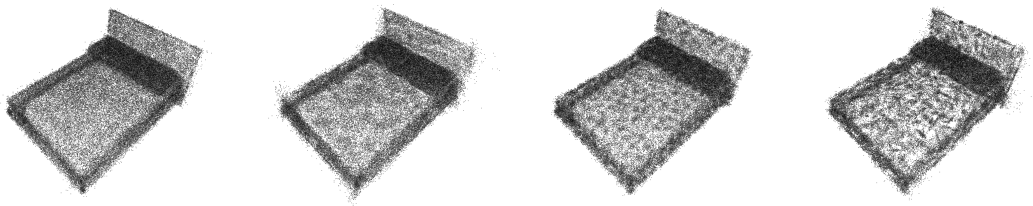
$$\mathcal{R}(\mathbf{\Theta}) = \sqrt{d_{\mathrm{CD}}(P_E, P_{\mathbf{\Theta}})} \tag{5.5}$$

For our experiments, we chose $|P_E| = |P_{\mathbf{\Theta}}| = 100,000$ for the calculation of $\mathcal{R}$.

Other than the likelihood, this metric also responds to high densities far away from the original surface. Figure 5.2 shows both $\mathcal{L}$ and $\mathcal{R}$ values on some exemplary GMMs. As this Figure illustrates, these values do not always correlate. High outlying densities create a worse reconstruction error while having no impact on the likelihood. As we are interested in outlying densities, $\mathcal{R}$ is better suited for our evaluation than $\mathcal{L}$.

5.2.3 Irregularity

An ideal density representation of the mesh's surface should have the same density value on each surface point. A point cloud sampled from the GMM should appear uniform along the surface. In practice, due to the nature of GMMs, the density will vary along the surface, and the points will not be perfectly uniformly distributed.



(a) $\mathcal{I} = 0.562$ (b) $\mathcal{I} = 0.586$ (c) $\mathcal{I} = 0.602$ (d) $\mathcal{I} = 0.719$

Figure 5.3: Reconstructed point clouds from different fits on the same model with increasing irregularity. Point clouds with lower irregularity look more uniform along the surface.

To evaluate this irregularity, we sample a point cloud $P_{\mathbf{\Theta}}$ from the generated GMM. We then filter out outliers by removing all points in $P_{\mathbf{\Theta}}$ whose nearest neighbor in P_E is further away than the maximum distance of a point in P_E to its nearest neighbor in $P_{\mathbf{\Theta}}$. We then project the points onto the nearest mesh surface, i.e. each point in $P_{\mathbf{\Theta}}$ is replaced by the nearest surface point of the original mesh (using the library Open3D [ZPK18]). Our irregularity metric $\mathcal{I}$ is defined as the coefficient of variation of the nearest-neighbor-distances from the evaluation point cloud P_E to the processed sampled point cloud $\varrho(P_{\mathbf{\Theta}})$:

$$\mathcal{I}(\mathbf{\Theta}) = \frac{\sqrt{\frac{1}{|P_E|-1}\sum_{\boldsymbol{x}\in P_E}\left(d_{\mathbf{\Theta}}(\boldsymbol{x}) - \overline{d_{\mathbf{\Theta}}}\right)}}{\overline{d_{\mathbf{\Theta}}}} \tag{5.6}$$

where $d_{\mathbf{\Theta}}(\boldsymbol{x})$ is the distance from $\boldsymbol{x}$ to the nearest-neighbor in $\varrho(P_{\mathbf{\Theta}})$:

$$d_{\mathbf{\Theta}}(\boldsymbol{x}) = \left(\min_{\boldsymbol{y}\in\varrho(P_{\mathbf{\Theta}})} \|\boldsymbol{x}-\boldsymbol{y}\|_2\right) \tag{5.7}$$

and $\overline{d_{\mathbf{\Theta}}}$ is the average nearest-neighbor distance

$$\overline{d_{\mathbf{\Theta}}} = \frac{1}{|P_E|}\sum_{\boldsymbol{x}\in P_E} d_{\mathbf{\Theta}}(\boldsymbol{x}). \tag{5.8}$$

Using the projected point cloud is necessary to get reliable results, as otherwise, the distances from the surface to the points may skew the result if there is a high amount of scattering around the surface (which is considered by $\mathcal{R}$). Outlier filtering before projection is performed because the outlier's projections could skew the nearest neighbor distances on the edges of the surfaces. A more uniform GMM produces a lower $\mathcal{I}$ than a more irregular one. Figure 5.3 shows GMMs with varying levels of uniformity and their respective $\mathcal{I}$ values.

5.2.4 Gaussian Variation

For certain applications, the variation of the Gaussian's sizes may be important. While each Gaussian technically has an infinite extent, we describe the Gaussian's "size" by calculating the volume of the Gaussian's 1-σ-ellipsoid $\frac{4}{3}\pi\sqrt{|\mathbf{\Sigma}|}$. As a metric for Gaussian variation, we track the coefficient of variation of those volumes. This is equal to calculating the coefficient of variation of the Gaussians' determinants' square roots:

$$\mathcal{V}(\mathbf{\Theta}) = \frac{\sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\left(v_{\mathbf{\Theta}}(k) - \overline{v_{\mathbf{\Theta}}}\right)^2}}{\overline{v_{\mathbf{\Theta}}}}, \tag{5.9}$$

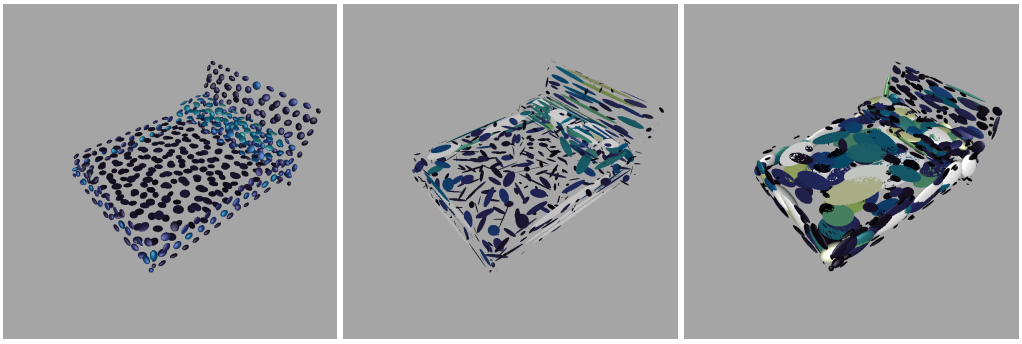
where K is the number of Gaussians of the GMM $\mathbf{\Theta}$, $v_{\mathbf{\Theta}}(k)$ is the square root of the kth Gaussian's determinant

$$v_{\mathbf{\Theta}}(k) = \sqrt{|\mathbf{\Sigma}_k|}, \tag{5.10}$$

$\mathbf{\Sigma}_k$ is the kth covariance matrix, and $\overline{v_{\mathbf{\Theta}}}$ is the average $v_{\mathbf{\Theta}}$:

$$\overline{v_{\mathbf{\Theta}}} = \sum_{k=1}^{K} v_{\mathbf{\Theta}}(k). \tag{5.11}$$

Figure 5.4 shows GMMs with varying levels of Gaussian variation and their respective $\mathcal{V}$ values.



(a) $\mathcal{V} = 0.904$ (b) $\mathcal{V} = 1.743$ (c) $\mathcal{V} = 2.439$

Figure 5.4: Different fits on the same model with increasing variation in Gaussians.

5.2.5 Additional Statistics

Additionally, we track the number of *0-Gaussians*, which are Gaussians with weight 0 or Gaussians that have been removed during the algorithm. We also track the algorithms' execution times. We measured these execution times on a PC with an Intel Core i7-6700 CPU, 16 GB RAM, and an NVIDIA GeForce GTX 1060 GPU. These are very implementation-dependent and should not be generalized to the algorithms themselves, as we did not focus on reducing computation time. This applies especially to geometrically regularized bottom-up HEM, as it is the only algorithm implemented in C++ instead of Python.

5.3 Metric Normalization

The results of some of our metrics depend on the spatial extent of the input model. Calculating the same metric on the same GMM in two different scales will produce different results. This is problematic as results from different meshes are not comparable, and results of larger models will dominate the mean values.

To solve this problem, we normalize the scale of the models. A naive approach is to scale the bounding boxes to a uniform size. This does not lead to satisfying results, as the scaled sizes of the models still differ too much.

Our approach calculates the average nearest neighbor distance $\hat{\eta}$ in the evaluation point cloud P_E. A scaling factor γ is then calculated by dividing a reference average nearest neighbor distance η_{ref} by our observed distance.

$$\gamma = \frac{\eta_{ref}}{\hat{\eta}} \tag{5.12}$$

This way, we can scale the point cloud and the GMM using the scaling factor γ, such that all point clouds have the same average nearest neighbor distance. For our experiments, we chose evaluation point clouds of size $|P_E| = 100,000$ and $\eta_{ref} = 0.2027$.

Instead of scaling the models in advance, we calculate the metrics on the original scale and then normalize the results. We scale $\mathcal{R}$ by multiplying with γ and $\mathcal{L}$ by subtracting $3\ln(\gamma)$. $\mathcal{I}$ and $\mathcal{V}$ are scale-independent and do not need to be scaled.

The effect of this normalization process is illustrated on exemplary $\mathcal{L}$ values in Figure 5.5.

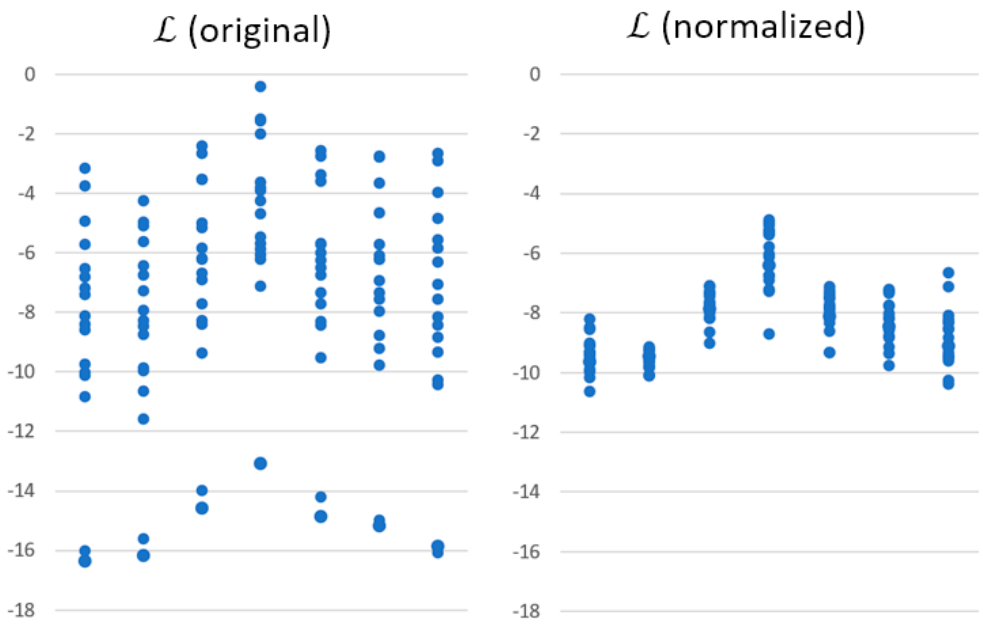


Figure 5.5: Comparison of $\mathcal{L}$ values of 20 models (original and normalized) for seven different algorithm configurations (each column represents one algorithm configuration). The normalized values show less variation.

5.4 Strategy

Our evaluation will take place in two stages: First, we examine each algorithm and the behaviors of its parameters individually. This provides insight into the mechanics of the algorithm and facilitates the selection of algorithm configurations to use for the second stage. In the second stage, we compare the algorithms to each other.

To evaluate the impact of a parameter, we fix the other parameters of the algorithm to reasonable values and test different values for the parameter of interest. For each experiment, we execute the selected methods on our 200 model dataset. Afterward, the previously discussed metrics on the results are calculated and normalized. We compare the methods by comparing the average metrics (denoted as AVG) of their results. Additionally, for $\mathcal{L}$, $\mathcal{R}$, $\mathcal{V}$ and $\mathcal{I}$, we also calculate the estimated standard error of the mean (SEM) to determine if the differences between two means are expected to be significant or not. For the numbers of 0-Gaussians and the execution times, we calculate the standard deviation of the results to describe the scattering of the measurements.

We cannot rule out the parameters behaving differently for other choices than tested in our experiments. Inspecting more parameter configurations would exceed this thesis's scope.

5.5 Results

In this section, we discuss the results of our experiments and the impacts of each algorithm's parameters.

5.5.1 EM

In this section, we will describe the results of our experiments done using our EM implementation (Section 4.1). We will inspect the influence of the regularization parameter ε_{rel} (abbreviated in this chapter by ε), the initialization technique, the number of input points, and the number of Gaussians. In all our experiments, the algorithm ran until the change of the average log-likelihood was less than 0.1 for 20 iterations. We configured the algorithm to use sub batches of 10,000 points in the E- and M-steps to not exceed the available memory.

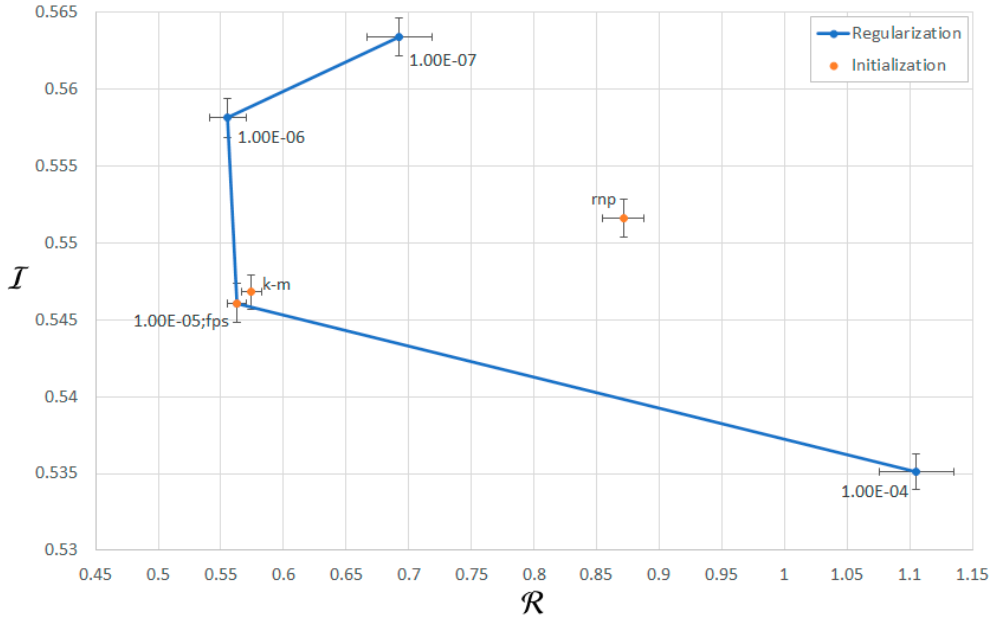


Figure 5.6: Results for EM regarding average reconstruction error $\mathcal{R}$ and average irregularity $\mathcal{I}$ of our experiments for the regularization parameter (using fixed initialization *fps*) and the initialization method (using fixed regularization parameter 10^{-5}). Results of the regularization experiments are connected by a line. Bars indicate standard errors of the means. Results are interpreted in the corresponding sections.

Regularization

In our EM implementation, we apply a regularization by adding a small value to the eigenvalues of the covariance matrices (Section 4.1.2). This way, we control the "thickness" of the Gaussians. The choice of ε defines this value relative to the bounding box size. We tested the values $10^{-4}, 10^{-5}, 10^{-6}$, and 10^{-7} for ε, using 100,000 input points, 512 Gaussians, and the *fps*-initialization. The results for our metrics are displayed in Table 5.1. Average $\mathcal{R}$ and $\mathcal{I}$ values are plotted in Figure 5.6. Exemplary GMMs are shown in Figure 5.7.

ε	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
10^{-4}	-10.402	0.030	1.105	0.030	**0.5351**	0.0011	**0.349**	0.006	0.000	0.000	**31.072**	0.540
10^{-5}	-9.512	0.022	**0.563**	0.008	0.5461	0.0013	0.497	0.015	0.000	0.000	39.417	11.654
10^{-6}	-8.545	0.030	**0.555**	0.015	0.5581	0.0013	0.873	0.032	0.005	0.071	57.440	23.926
10^{-7}	**-7.553**	0.076	0.692	0.026	0.5634	0.0012	1.254	0.043	0.005	0.071	64.643	26.268

Table 5.1: Results of ε-experiments for EM. Best values are formatted in bold. $\mathcal{L}$, $\mathcal{I}$, $\mathcal{V}$, and execution time rise with decreasing ε. $\mathcal{R}$ is best for 10^{-5} and 10^{-6}.

Choosing a lower ε leads to a higher likelihood $\mathcal{L}$. This is expected, as the regularization deliberately rejects a higher likelihood in exchange for numerical stability and avoiding of singularities.

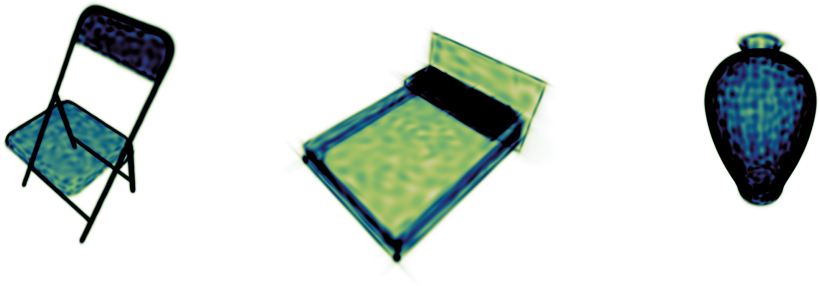
The reconstruction error $\mathcal{R}$ results in a U-shape (see Figure 5.6), with the lowest and highest ε-values creating worse results than the middle ones, which are on a similar level. With a too high ε, the reconstructed points are scattered widely around the surface, leading to higher nearest-neighbor distances. With a too low ε, the algorithm creates high densities on the extensions of the surfaces (see Section 4.1.2), which leads to reconstructed points far away from the surface, resulting in a high reconstruction error. This is especially true for models with long, even surfaces, on which the algorithm places long Gaussians. Round, smooth objects do not show this problem to such an extent. This behavior is illustrated in Figure 5.7.

Both the irregularity $\mathcal{I}$ and the Gaussian variation $\mathcal{V}$ are higher when using a lower ε. When the Gaussian ellipsoids are allowed to become smaller, there is more room for variation in their size than otherwise.

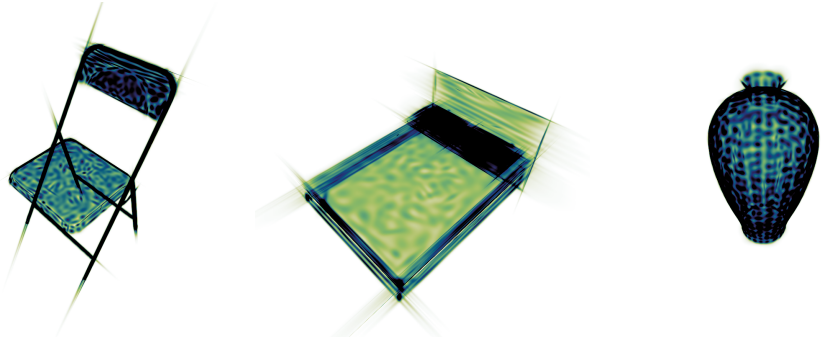
A lower ε also leads to a higher execution time, as the model needs more iterations to converge. The difference ranges from 31 seconds for 10^{-4} to 65 seconds for 10^{-7}. The standard deviation increases from 0.5 seconds to 26.

In conclusion, in this experiment, an ε of 10^{-5} is a good trade-off between having a low and more reliable reconstruction error $\mathcal{R}$, as well as providing good results for $\mathcal{I}$, $\mathcal{V}$ and execution times, compared to the higher values.

(a) $\varepsilon = 10^{-4}$



(b) $\varepsilon = 10^{-5}$



(c) $\varepsilon = 10^{-7}$

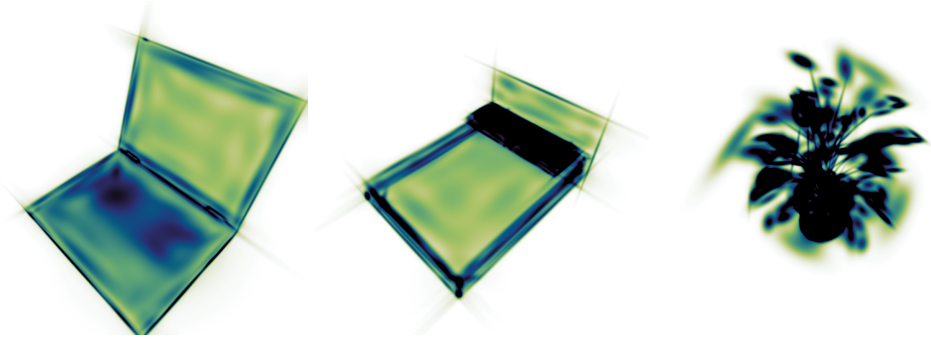
Figure 5.7: Resulting GMMs for the models *chair_0001*, *bed_0001* and *vase_0001* using EM with different values for ε. Results for 10^{-4} are very blurry. Results for 10^{-7} are more detailed, but the results for *chair_0001* and *bed_0001* contain high densities outside of the surface. *vase_0001* does not show this problem as it contains less flat surfaces.

Initialization

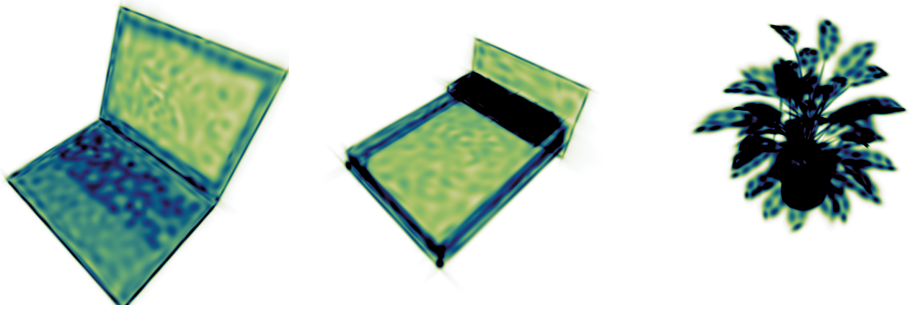
We have implemented the EM-initialization techniques *rnp*, *fps*, and *k-means* (Section 4.1.3). We tested these initialization techniques using 100,000 input points, 512 Gaussians, and $\varepsilon = 10^{-5}$. The results for our metrics are displayed in Table 5.2. Average $\mathcal{R}$ and $\mathcal{I}$ values are plotted in Figure 5.6.

init.	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
rnp	-9.572	0.025	0.871	0.017	0.5516	0.0013	1.045	0.026	0.050	0.478	65.125	16.855
fps	**-9.512**	0.022	**0.563**	0.008	**0.5461**	0.0013	**0.497**	0.015	0.000	0.000	**39.417**	11.654
k-means	**-9.525**	0.023	**0.574**	0.008	**0.5468**	0.0011	**0.518**	0.019	0.000	0.000	55.446	12.332

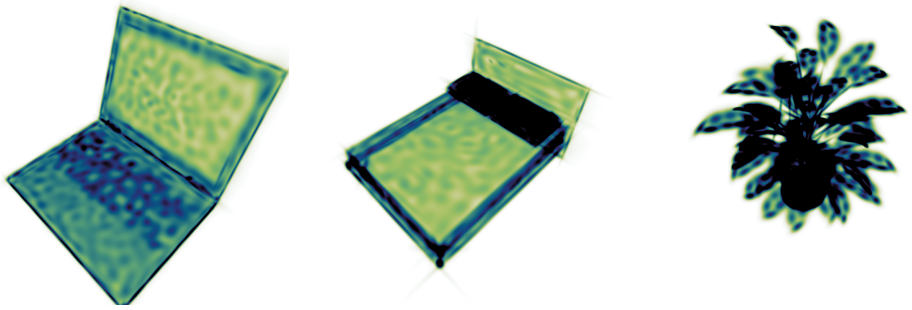
Table 5.2: Results of initialization-technique-experiments for EM. Best values are formatted in bold. *fps* and *k-means* are superior to *rnp* for all metrics. *fps* is the fastest technique.



(a) *rnp*



(b) *fps*



(c) *k-means*

Figure 5.8: Resulting GMMs for the models *laptop_0004*, *bed_0001* and *plant_0004*, using EM with different initialization strategies. *fps* and *k-means* produce similar results, *rnp* produces less detailed and smoother results.

No significant difference can be observed for *fps* and *k-means* regarding $\mathcal{L}$, $\mathcal{R}$, $\mathcal{I}$, or $\mathcal{V}$. The *rnp*-method initializes the positions randomly, while the other two aim to distribute small Gaussians equally along the surface. This leads to the final GMMs containing larger Gaussians when using *rnp* (as shown by the average determinants for *rnp*, *fps* and *k-means*, which are 555.067, 5.361 and 5.450, respectively). These larger Gaussians tend to extend outside of the actual surface, similar to what happens when using low ε-values, which plays a role in the higher reconstruction error $\mathcal{R}$. The larger Gaussians make the GMMs appear smoother in the density visualization, which can be seen in Figure 5.8. However, regarding the irregularity $\mathcal{I}$ of the reconstructed point cloud, *rnp* also appears slightly worse than the other two techniques. As results of *rnp* typically contain larger Gaussians, the Gaussian variation $\mathcal{V}$ is higher than the other two techniques.

When using *rnp*, more iterations are needed on average to converge, resulting in the longest execution time of the three methods. *k-means* and *fps* need similarly many iterations, however our implementation of *fps* is faster than our *k-means* implementation, which makes it the fastest technique in this experiment.

Concluding, *rnp* performs worse in terms of all our metrics but generates visually smoother results. *fps* and *k-means* create similar results. Our specific implementation of *fps* is faster, making it the more favorable technique in this examination.

Number of Points

Using fewer input points decreases execution time, but may harm the quality of the results. To inspect this, we tested the algorithm using 10,000, 50,000, and 100,000 sample points, with 512 Gaussians, $\varepsilon = 10^{-5}$ and both *fps*-initialization as well as *rnp*-initialization. The results are shown in Table 5.3. Results for $\mathcal{R}$ and $\mathcal{I}$ are plotted in Figure 5.9. Exemplary GMMs are shown in Figure 5.10.

init.	N	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
		AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
fps	10,000	-9.688	0.071	0.578	0.007	0.6622	0.0031	0.745	0.027	0.000	0.000	**4.491**	1.211
fps	50,000	**-9.514**	0.023	**0.560**	0.008	0.5531	0.0012	0.540	0.018	0.000	0.000	20.279	6.500
fps	100,000	**-9.512**	0.022	**0.563**	0.008	**0.5461**	0.0013	**0.497**	0.015	0.000	0.000	39.417	11.654
rnp	10,000	-9.615	0.054	0.836	0.016	0.6006	0.0029	1.281	0.031	0.055	0.578	7.773	2.081
rnp	50,000	-9.576	0.025	0.874	0.018	0.5542	0.0013	1.102	0.035	0.030	0.299	32.521	8.155
rnp	100,000	-9.572	0.025	0.871	0.017	0.5516	0.0013	1.045	0.026	0.050	0.478	65.125	16.855

Table 5.3: Results of point-count-experiments for EM. Best values are formatted in bold. The difference between 50,000 and 100,000 points is small. Using 10,000 points leads to worse results.

In terms of $\mathcal{L}$ and $\mathcal{R}$ there are only small differences between different point counts, where 10,000 appears only slightly inferior to the other two numbers for *fps*. When using the *rnp*-initialization, 10,000 even produces slightly better results regarding $\mathcal{R}$.

We can see a strong difference when inspecting the irregularity $\mathcal{I}$: While 50,000 points create only slightly more irregular GMMs than 100,000, reducing to 10,000 points increases the irregularity strongly. When using *fps*-initialization this leads to the highest average $\mathcal{I}$-value in all of our

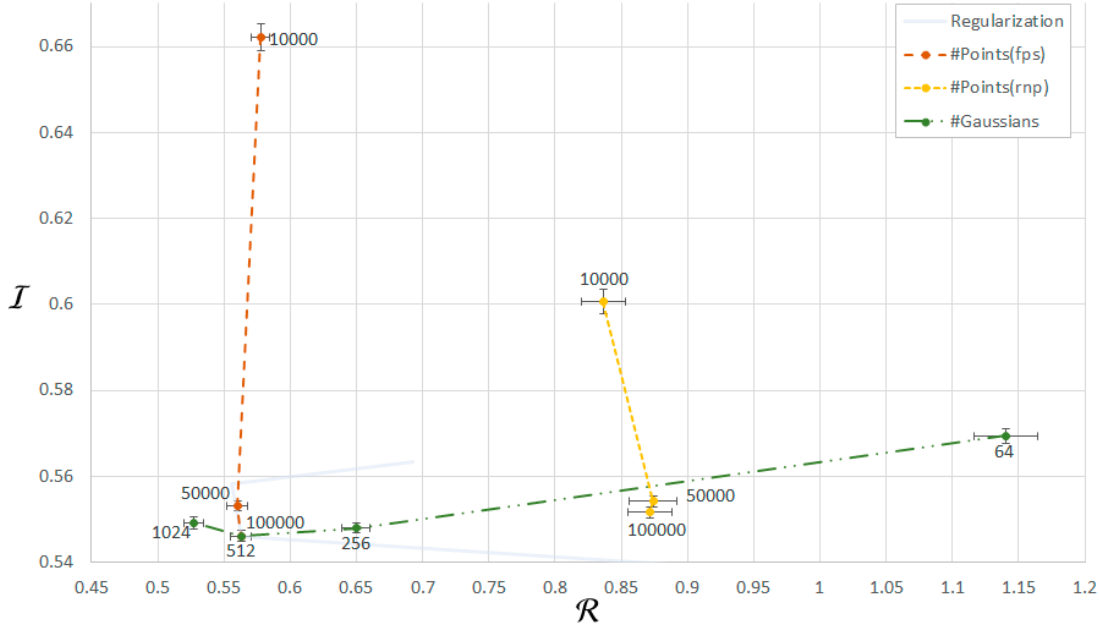


Figure 5.9: Results for EM regarding average reconstruction error $\mathcal{R}$ and average irregularity $\mathcal{I}$ of our experiments for the number of points (using initialization *fps* or *rnp* and 512 Gaussians) and the number of Gaussians (using fixed number of points 100,000). For easier interpretation, results from the same experiments are connected by lines in order of the changing parameter. Bars indicate standard errors of the means. For comparison with Figure 5.6, the results of the regularization-experiments are also shown in the background. Reducing the point count leads to an increase in $\mathcal{I}$. Reducing the Gaussian count leads to an increase in $\mathcal{R}$ and also affects $\mathcal{I}$.

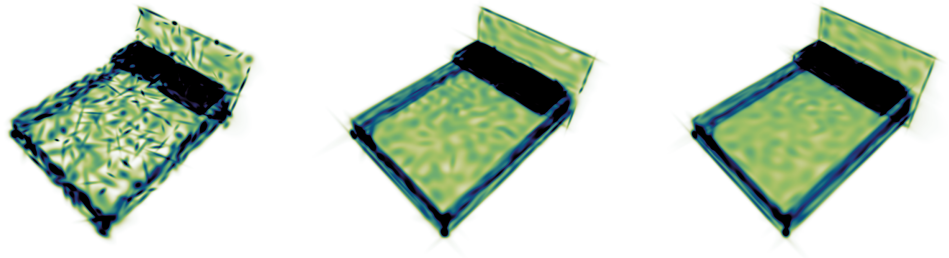
experiments. The increased irregularity is also clearly visible when looking at the visualizations: For 10,000 points and *fps*, the GMMs appear chaotic as Gaussians of different sizes are scattered on the surface (Figure 5.10). An alternative for getting smoother results when using fewer points is the use of the *rnp*-initialization, which results in lower irregularities for 10,000 points. However, it comes at the cost of an increased reconstruction error.

Using fewer points also leads to an increase in Gaussian variation $\mathcal{V}$. This variation is much higher when using *rnp* rather than for *fps*.

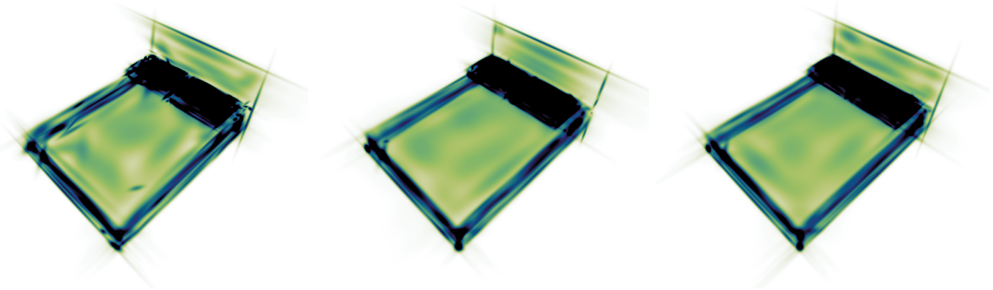
Reducing the number of points reduces the time per iteration linearly. Switching from 100,000 to 10,000 points speeds up the algorithm from 40 to 4 seconds.

Number of Gaussians

The number of Gaussians should have a direct impact on the results, as it defines the modeling power of the GMM. We tested the algorithm using 64, 256, 512, and 1,024 Gaussians, with 100,000 points, *fps*-initialization and $\varepsilon = 10^{-5}$. The results are shown in Table 5.4. The average $\mathcal{R}$ and $\mathcal{I}$ values are plotted in Figure 5.9. Exemplary GMMs are shown in Figure 5.11.



(a) 10,000 points, *fps* (b) 50,000 points, *fps* (c) 100,000 points, *fps*



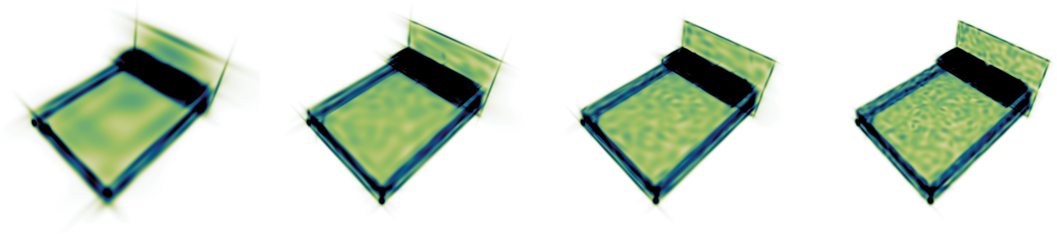
(d) 10,000 points, *rnp* (e) 50,000 points, *rnp* (f) 100,000 points, *rnp*

Figure 5.10: Resulting GMMs for the model *bed_0001*, using different input point counts and initialization strategies. A clear increase in irregularity can be observed when reducing the point count.

As expected, the reconstruction error $\mathcal{R}$ decreases with increased Gaussian count. The irregularity $\mathcal{I}$ is highest for 64 Gaussians. For the other Gaussian counts, it is on a similar level, with 512 being slightly superior to the others. The Gaussian variation $\mathcal{V}$ decreases with increasing Gaussian count. Similar to the point count, the number of Gaussians directly impacts the execution time by changing the time of each iteration.

K	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
64	-9.880	0.032	1.140	0.024	0.5694	0.0017	0.649	0.014	0.000	0.000	**7.257**	2.731
256	-9.594	0.024	0.650	0.010	**0.5479**	0.0011	0.547	0.015	0.000	0.000	22.162	6.865
512	-9.512	0.022	0.563	0.008	**0.5461**	0.0013	0.497	0.015	0.000	0.000	39.417	11.654
1024	**-9.463**	0.023	**0.527**	0.008	**0.5492**	0.0014	**0.460**	0.016	0.000	0.000	71.709	16.804

Table 5.4: Results of Gaussian-count-experiments for EM. Best values are formatted in bold. Most metrics improve with higher Gaussian count. $\mathcal{I}$ stays roughly on the same level for 256 and higher numbers. The execution time rises when using more Gaussians.



(a) 64 Gaussians (b) 256 Gaussians (c) 512 Gaussians (d) 1,024 Gaussians

Figure 5.11: Resulting GMMs for the model *bed_0001*, using EM with different numbers of Gaussians. A rise in accuracy can be observed when increasing the Gaussian count.

5.5.2 Top-Down HEM

In this section, we will describe the results of our experiments done using our implementation of top-down HEM (Section 4.2). We will inspect the influence of the regularization parameter ε, the partitioning method, the initialization technique, the number of input points, and the number of Gaussians. In our experiments, a level was accepted as converged when the change of the average log-likelihood was less than 0.1 for 20 iterations. To not exceed the available memory, we configured the algorithm to only process up to 5,120,000 point-Gaussians pairs at once in the E-step and use sub batches of 10,000 points and 512 Gaussians in the M-step. The execution times we measured for our PyTorch implementation are higher than the ones reported for the original C++/CUDA implementation by Eckart et al. [EKT+16].

Regularization

For top-down HEM, we apply the same regularization technique as with classical EM, where we add a small value to the eigenvalues of the covariance matrices (see Section 4.1.2). This value is defined relative to the bounding box size. To explore the effect of the regularization parameter ε for top-down HEM, we tested the values 10^{-4}, 10^{-5}, 10^{-6}, and 10^{-7} for ε, using 100,000 input points, 512 Gaussians ($j = 8, l = 3$), soft partitioning with $\lambda_l = 0.1$, and the *fps*-initialization. The results are displayed in Table 5.5. Results for the average $\mathcal{R}$ and $\mathcal{I}$ values are plotted in Figure 5.12.

ε	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
10^{-4}	-10.431	0.030	1.131	0.029	**0.5384**	0.0011	**0.639**	0.017	1.530	5.043	**7.937**	0.929
10^{-5}	-9.627	0.024	0.642	0.010	0.5483	0.0012	1.205	0.034	0.785	4.770	8.817	1.923
10^{-6}	-8.803	0.034	**0.567**	0.009	0.5568	0.0014	1.812	0.069	0.325	1.396	10.702	3.147
10^{-7}	**-7.993**	0.065	**0.585**	0.009	0.5598	0.0013	2.404	0.074	0.445	1.853	12.145	3.946

Table 5.5: Results of ε-experiments for top-down HEM. Best values are formatted in bold. Most metrics rise with decreasing ε.

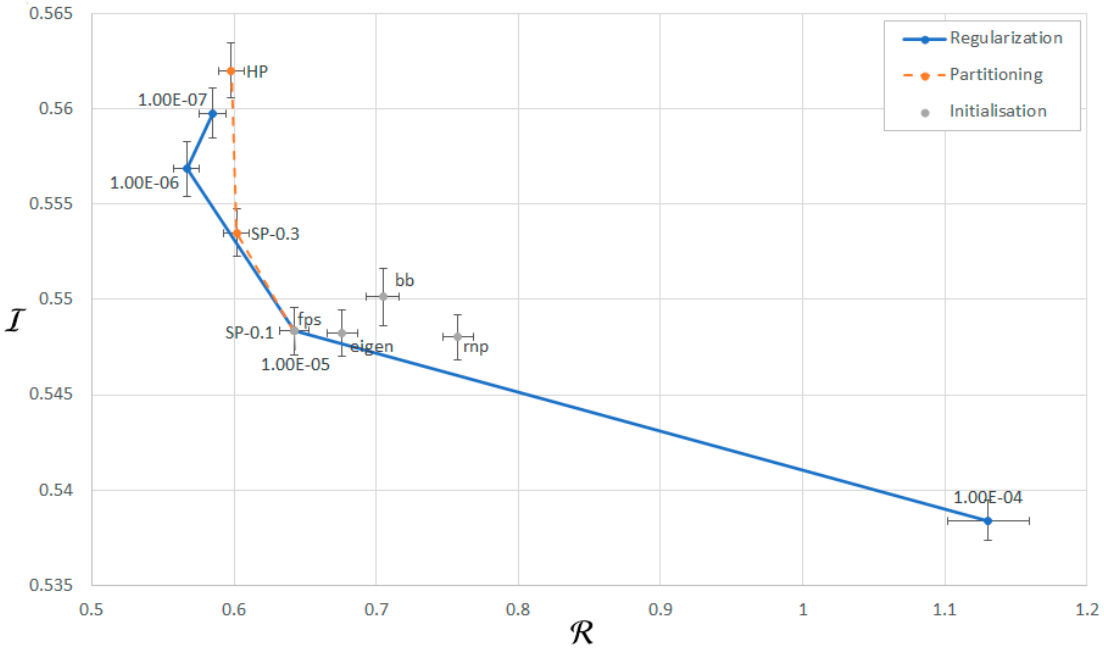


Figure 5.12: Results for top-down HEM regarding reconstruction error $\mathcal{R}$ and irregularity $\mathcal{I}$ of our experiments for the regularization parameter, the partitioning method and the initialization method. Results are interpreted in the corresponding sections.

For a high ε of 10^{-4}, the reconstruction error $\mathcal{R}$ is much higher than for the other values. The lowest error is achieved at 10^{-6}. A low ε of 10^{-7} does not cause such a high increase of the reconstruction error as for EM. This is because of the hierarchical structure, which restricts the final Gaussians from becoming as large as in EM. This way, it reduces the amount of high density outside the model.

Similarly to EM, a smaller ε leads to increased irregularity $\mathcal{I}$, increased Gaussian variation $\mathcal{V}$, and higher execution time.

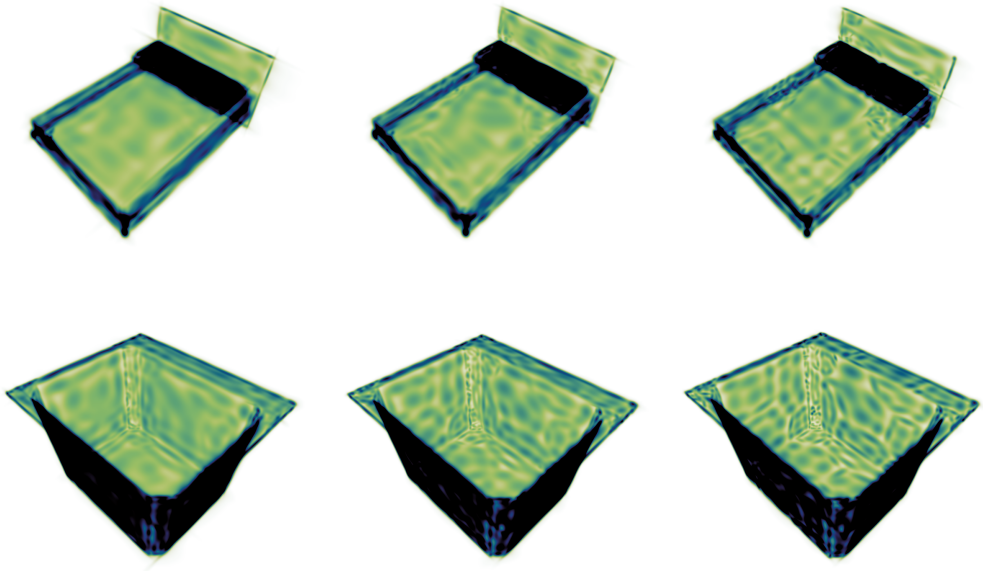
Partitioning

When initializing a new level, top-down HEM can either use *hard partitioning*, which assigns each point to the sub-GMM with the highest corresponding responsibility, or *soft partitioning*, which may assign each point to multiple sub-GMMs. When using soft partitioning, the parameter λ_l controls the minimum responsibility required for a point to be added to a sub-GMM. To analyze the impact of the different partitioning schemes, we tested the algorithm using both hard partitioning and soft partitioning with $\lambda_l = 0.1$ and $\lambda_l = 0.3$. Otherwise, 100,000 input points, 512 Gaussians ($j = 8, l = 3$), the *fps*-initialization, and $\varepsilon = 10^{-5}$ were used. Table 5.6 shows the results. The average $\mathcal{R}$ and $\mathcal{I}$ are plotted in Figure 5.12. Figure 5.13 shows exemplary GMMs.

The results show a bigger reconstruction error $\mathcal{R}$ for SP, $\lambda_l = 0.1$ than for SP, $\lambda_l = 0.3$ and HP.

Part.	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
SP, $\lambda_l = 0.1$	-9.627	0.024	0.642	0.010	**0.5483**	0.0012	1.205	0.034	0.785	4.770	8.817	1.923
SP, $\lambda_l = 0.3$	-9.601	0.023	**0.602**	0.009	0.5535	0.0013	1.100	0.028	0.720	5.063	7.879	1.392
HP	-9.614	0.022	**0.598**	0.009	0.5620	0.0014	1.152	0.031	1.035	5.505	**6.653**	1.036

Table 5.6: Results of partitioning-experiments for top-down HEM. Best values are formatted in bold. Harder partitioning methods result in lower reconstruction errors but higher irregularity.



(a) Soft partitioning, $\lambda_l = 0.1$ (b) Soft partitioning, $\lambda_l = 0.3$ (c) Hard partitioning

Figure 5.13: Resulting GMMs for the models *bed_0001* and *sink_0002*, using top-down HEM with different partitioning methods. Harder methods appear more irregular than more relaxed ones.

The more relaxed a scheme is, the larger the Gaussians can become, which may lead to outlying densities. This may partially explain the higher error.

The GMMs constructed by softer techniques appear smoother, while for the harder partitioning methods the boundaries of the sub-GMMs can be seen (see Figure 5.13). This is also reflected by the irregularity $\mathcal{I}$, with $\lambda_l = 0.1$ resulting in the most uniform reconstructed point clouds, and hard partitioning in the most irregular ones. Regarding the Gaussian variation $\mathcal{V}$, soft partitioning with $\lambda_t = 0.1$ results in a higher value than the harder methods.

The partitioning scheme affects the execution time, as the harder the partitioning scheme, the fewer Gaussian-point-pairs have to be considered each iteration.

To summarize, on average, GMMs from softer partitioning schemes are more uniform, but have higher reconstruction errors and take longer to construct.

Initialization

For top-down HEM we have implemented the initialization techniques *rnp*, *fps*, *bb*, and *eigen* (Section 4.2.2). We tested these initialization techniques using soft partitioning with $\lambda_l = 0.1$, 100,000 points, 512 Gaussians ($j = 8, l = 3$), and $\varepsilon = 10^{-5}$. The results are shown in Table 5.7. The average $\mathcal{R}$ and $\mathcal{I}$ are plotted in Figure 5.12. Exemplary GMMs are shown in Figure 5.14.

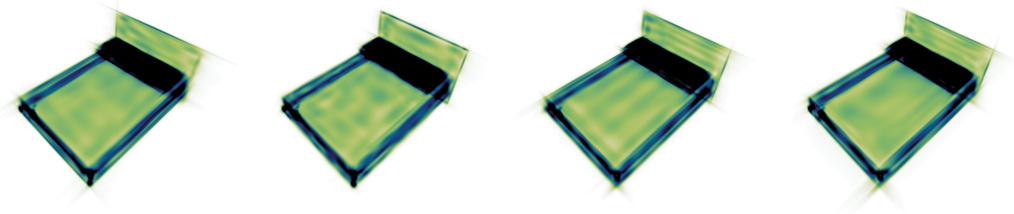
init.	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
rnp	**-9.558**	0.025	0.757	0.011	0.5480	0.0012	1.037	0.025	2.335	5.304	7.868	1.374
fps	-9.627	0.024	**0.642**	0.010	0.5483	0.0012	1.205	0.034	0.785	4.770	8.817	1.923
bb	**-9.564**	0.025	0.705	0.012	0.5501	0.0015	**0.947**	0.026	6.425	19.021	7.628	1.305
eigen	**-9.554**	0.025	0.676	0.011	0.5482	0.0012	**0.909**	0.028	0.175	1.301	8.059	1.313

Table 5.7: Results of initialization-technique-experiments for top-down HEM. Best values are formatted in bold. *fps* shows the lowest $\mathcal{L}$, but best $\mathcal{R}$. Regarding $\mathcal{V}$, *bb* and *eigen* are the best.

Initialization technique *rnp* shows the highest average reconstruction error $\mathcal{R}$. *eigen* has been introduced as an alternative to *bb* and proves to create more accurate results regarding $\mathcal{R}$. The lowest reconstruction errors are produced by *fps*.

Regarding irregularity $\mathcal{I}$, no clear difference between the methods can be observed. The average $\mathcal{I}$ for *bb* is higher than the others. However, the standard errors are high enough to make this difference unreliable. *fps* shows the largest Gaussian variation $\mathcal{V}$, followed by *rnp*. *bb* and *eigen* are on a similar level.

Further differences become visible when looking at the amount of Gaussians that were removed during the algorithm: While *eigen* shows the lowest and most reliable average amount of removed Gaussians from all four methods, *bb* shows the highest average of 6.425, with a standard deviation of 19.



(a) *rnp* (b) *fps* (c) *bb* (d) *eigen*

Figure 5.14: Resulting GMMs for the models *bed_0005*, *bed_0001* and *sink_0002*, using top-down HEM with different initialization methods.

Number of Points

For examining the impact of the number of input points, we tested the algorithm using 10,000, 50,000, and 100,000 points, for soft partitioning with $\lambda_l = 0.1$, 512 Gaussians ($j = 8, l = 3$), and $\varepsilon = 10^{-5}$. As we suspect the initialization methods to have a bigger impact, we test the techniques *fps*, *eigen*, and *rnp* for all point counts. The results are shown in Table 5.8. The average $\mathcal{R}$ and $\mathcal{I}$ values are plotted in Figure 5.15.

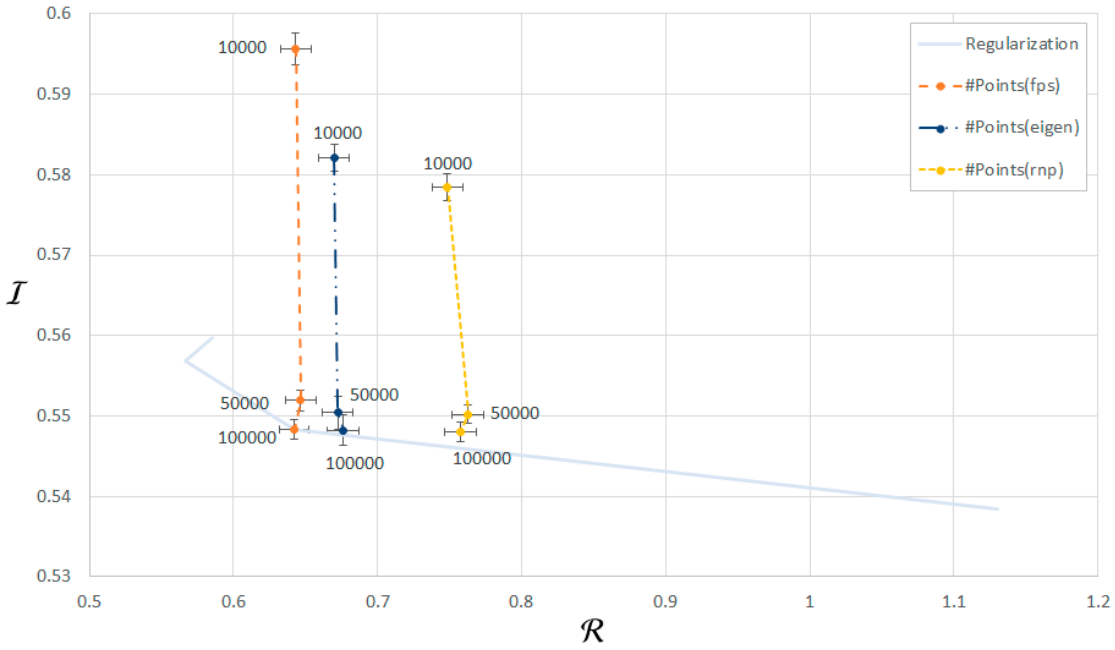


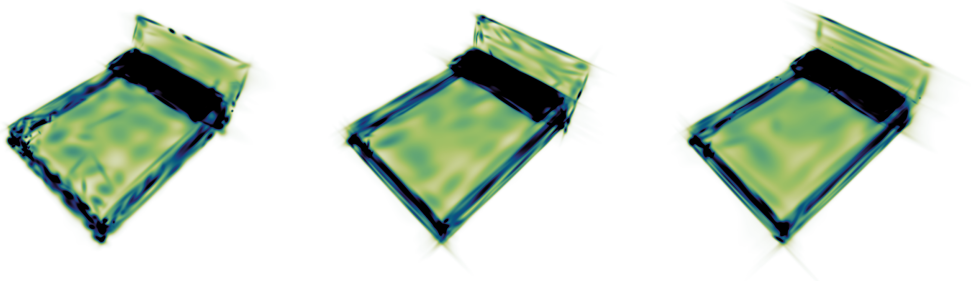
Figure 5.15: Results for top-down HEM regarding average reconstruction error $\mathcal{R}$ and average irregularity $\mathcal{I}$ of our experiments for the number of points using 512 Gaussians. The three initialization methods *fps*, *eigen*, and *rnp* were tested. For easier interpretation, results using the same initialization method are connected by lines in order of the point count. Bars indicate standard errors of the means. For comparison with Figure 5.12, the results of the regularization-experiments are also shown in the background. Reducing the point count increases $\mathcal{I}$, but hardly changes $\mathcal{R}$.

While $\mathcal{R}$ shows no clear difference for different point counts, the average irregularity $\mathcal{I}$ is much higher for 10,000 points, although not as high as when using classical EM. Both the *rnp*-initialization and the *eigen*-initialization provide more uniform results than *fps*, at the cost of higher reconstruction errors. This can be seen in Figure 5.16. The Gaussian variation $\mathcal{V}$ reduces when using more points.

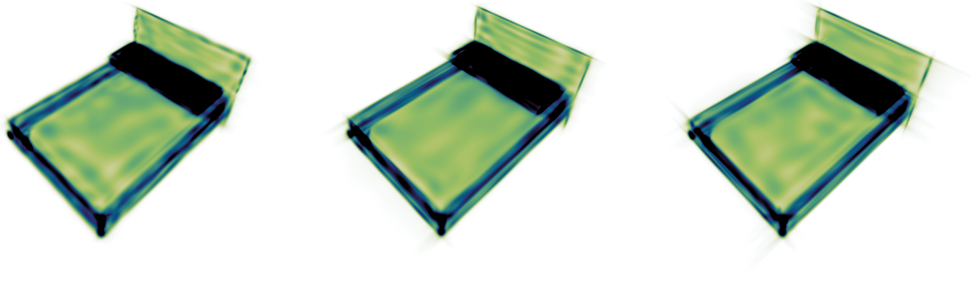
The execution time ranges from around two seconds for 10,000 points, to around 8 seconds for 100,000 points.

init.	N	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
		AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
fps	10,000	-9.680	0.023	0.643	0.010	0.5956	0.0020	1.388	0.037	1.225	4.724	**2.726**	0.455
fps	50,000	-9.628	0.024	0.647	0.011	**0.5519**	0.0013	1.291	0.047	0.445	1.896	5.326	1.080
fps	100,000	-9.627	0.024	0.642	0.010	**0.5483**	0.0012	**1.205**	0.034	0.785	4.770	8.817	1.923
eigen	10,000	-9.592	0.025	0.670	0.011	0.5821	0.0019	0.990	0.023	0.585	0.000	**1.782**	0.000
eigen	50,000	-9.556	0.025	0.672	0.010	**0.5504**	0.0012	**0.904**	0.027	0.245	0.000	4.422	0.000
eigen	100,000	-9.554	0.025	0.676	0.011	**0.5482**	0.0012	**0.909**	0.028	0.175	0.000	8.059	0.000
rnp	10,000	-9.598	0.025	0.749	0.011	0.5785	0.0017	1.167	0.035	2.700	6.868	**1.691**	0.329
rnp	50,000	-9.562	0.025	0.763	0.011	**0.5502**	0.0012	1.061	0.028	1.505	4.498	4.464	0.715
rnp	100,000	-9.558	0.025	0.757	0.011	**0.5480**	0.0012	**1.037**	0.025	2.335	5.304	7.868	1.374

Table 5.8: Results of initialization-technique-experiments for top-down HEM. Best $\mathcal{I}$ values, $\mathcal{V}$ values, and execution times per initialization method are formatted in bold. $\mathcal{I}$ and $\mathcal{V}$ are worse for 10,000 points than for higher numbers. $\mathcal{L}$ and $\mathcal{R}$ show hardly any differences for different point counts. *eigen* and *rnp* produce lower $\mathcal{I}$ values than *fps* at the cost of higher $\mathcal{R}$.



(a) *fps*, 10,000 points (b) *eigen*, 10,000 points (c) *rnp*, 10,000 points



(d) *fps*, 50,000 points (e) *eigen*, 50,000 points (f) *rnp*, 50,000 points

Figure 5.16: Resulting GMMs for the model *bed_0001* using top-down HEM with different numbers of input points and initialization strategies (results for 100,000 points are shown in Figure 5.14). An increase in irregularity can be observed when reducing the point count.

Number of Gaussians

In top-down HEM, the number of Gaussians K equals j^l, where l is the number of levels, and j is the number of Gaussians per sub-GMM. To examine the behavior of different numbers of Gaussians, we tested the numbers 4^3 (64), 8^2 (64), 4^4 (256), 8^3 (512), and 4^5 (1024). These were tested using soft partitioning with $\lambda_l = 0.1$, 100,000 input points, *fps*-initialization, and $\varepsilon = 10^{-5}$. The results are shown in Table 5.9. The average $\mathcal{R}$ and $\mathcal{I}$ values are plotted in Figure 5.17.

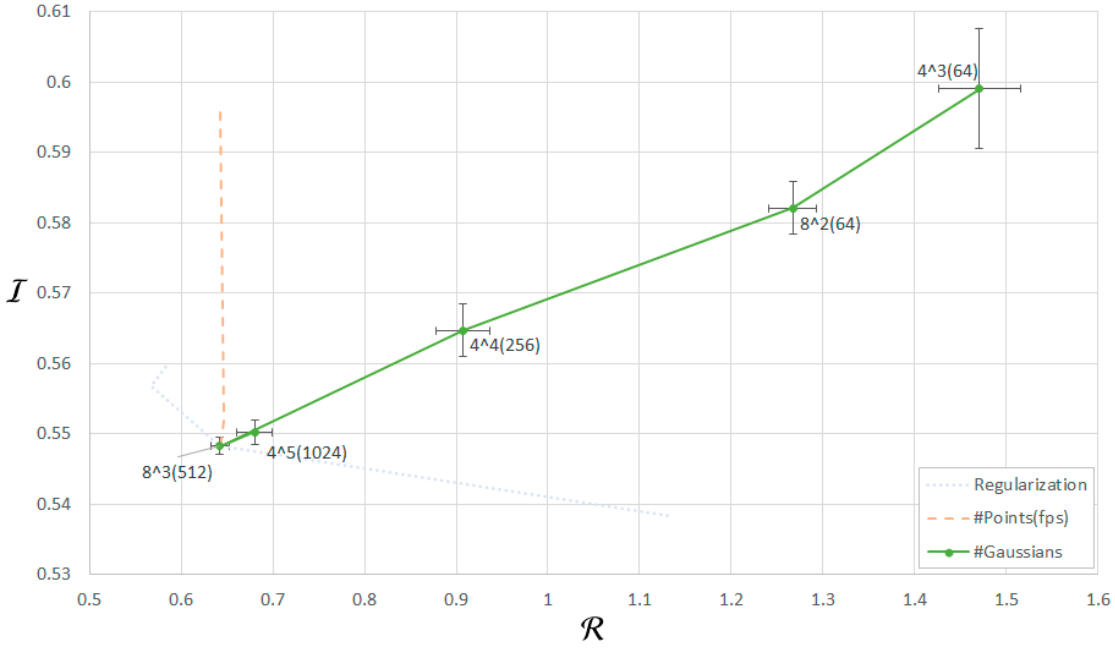


Figure 5.17: Results for top-down HEM regarding average reconstruction error $\mathcal{R}$ and average irregularity $\mathcal{I}$ of our experiments for the number of Gaussians using 100,000 points and the *fps*-initialization. Bars indicate standard errors of the means. For comparison with Figure 5.12 and 5.15, the results of the regularization-experiments and the point count-experiment (with *fps*-initialization) are also shown in the background. A lower number of Gaussians leads to higher $\mathcal{R}$ and $\mathcal{I}$.

j^l (K)	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
4^3(64)	-10.217	0.037	1.471	0.045	0.5991	0.0085	1.262	0.051	0.000	0.000	2.994	0.505
8^2(64)	-10.059	0.035	1.267	0.026	0.5821	0.0037	**1.039**	0.033	0.035	0.495	**2.856**	0.761
4^4(256)	-9.845	0.029	0.907	0.029	0.5648	0.0037	1.505	0.076	0.040	0.330	6.627	1.121
8^3(512)	**-9.627**	0.024	**0.642**	0.010	**0.5483**	0.0012	1.205	0.034	0.785	4.770	8.817	1.923
4^5(1024)	**-9.640**	0.025	0.680	0.020	**0.5503**	0.0017	1.661	0.086	1.480	3.069	17.630	2.377

Table 5.9: Results of initialization-technique-experiments for top-down HEM. Best values are formatted in bold. A higher number of Gaussians leads to better $\mathcal{L}$ and $\mathcal{I}$. $\mathcal{R}$ is best for 512 Gaussians. $\mathcal{V}$ is lower for $j = 8$ than for $j = 4$.

We see a decrease of the reconstruction error $\mathcal{R}$ for higher numbers of Gaussians, except for 1,024 Gaussians, where the reconstruction error is slightly higher than for 512 Gaussians. We also observe that the result for 64 Gaussians depends on the used j/l-combination, as 8^2 provides better results than 4^3. The irregularity $\mathcal{I}$ is lower when using more Gaussians. The Gaussian variation $\mathcal{V}$ is lower when using $j = 8$ than when using $j = 4$. The variations per j increase with higher l.

5.5.3 Geometrically Regularized Bottom-Up HEM

The implementation of geometrically regularized bottom-up HEM provides a large number of parameters. We limit this evaluation to the most important ones: The fixed initialization distance, the alpha-value, the number of points, and the number of Gaussians. The algorithm would also provide alternative initialization strategies, such as using k nearest neighbors for initialization rather than a fixed radius or initializing with isotropic Gaussians of a fixed standard deviation. The reduction factor we left at its default value of three.

Initialization-Distance

The initialization of the algorithm chooses the initial covariances by analyzing the distribution of the points' neighborhoods of radius r. In the implementation, this radius is given in percent of the diagonal of the input point cloud's bounding box.

To examine the impact of the initialization distance, we tested the values 0.8%, 0.9%, 1.0%, 1.1%, 1.2%, and 1.3% using $\alpha = 4$, 100,000 input points, and 512 Gaussians. The results are shown in Table 5.10. Average $\mathcal{R}$ and $\mathcal{I}$ are plotted in Figure 5.18. Exemplary GMMs are displayed Figure 5.19.

dist	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
0.8%	-8.268	0.071	0.933	0.023	0.5869	0.0009	4.534	0.231	0.000	0.000	1.482	1.085
0.9%	-8.348	0.072	0.853	0.021	0.5768	0.0007	3.364	0.174	0.000	0.000	1.351	0.708
1.0%	-8.420	0.073	0.829	0.021	0.5690	0.0005	2.719	0.128	0.000	0.000	1.400	0.777
1.1%	-8.507	0.074	0.834	0.025	0.5644	0.0005	2.304	0.123	0.000	0.000	1.535	0.901
1.2%	-8.611	0.074	0.845	0.027	0.5614	0.0005	2.036	0.111	0.000	0.000	1.655	0.998
1.3%	-8.707	0.075	0.866	0.028	**0.5586**	0.0006	**1.787**	0.086	0.000	0.000	1.847	1.168

Table 5.10: Results of initialization-distance-experiments for geometrically regularized bottom-up HEM. Best values are formatted in bold. $\mathcal{I}$ and $\mathcal{V}$ decrease with higher distance. The lowest $\mathcal{R}$ value is given for 1.0%, however, the standard errors are too high to show a clear result.

The average reconstruction error is highest at 0.8%. The other tested initial distances do not show a clear difference in $\mathcal{R}$, as their standard errors overlap. The irregularity $\mathcal{I}$ and Gaussian variation $\mathcal{V}$ both reduce when using higher distances.

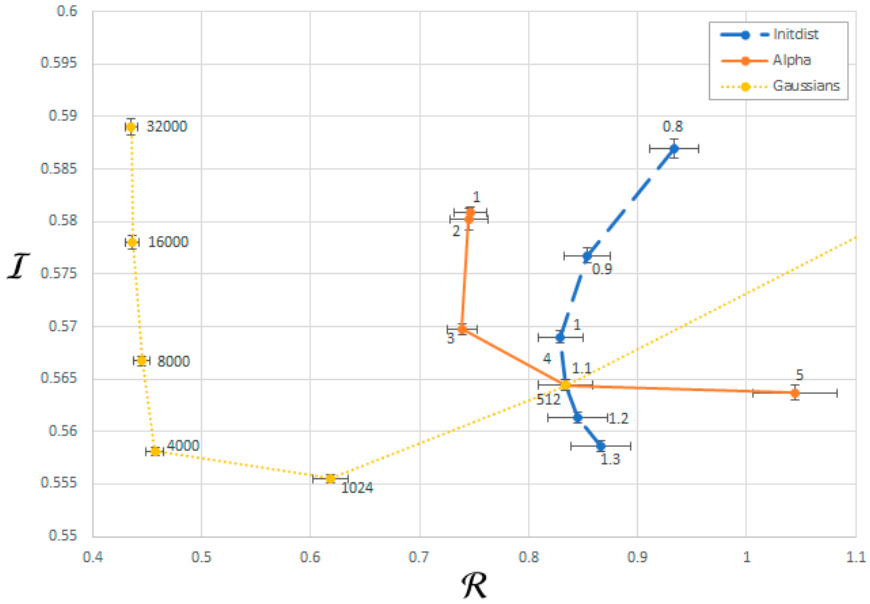


Figure 5.18: Results for geometrically regularized bottom-up HEM regarding average reconstruction error $\mathcal{R}$ and average irregularity $\mathcal{I}$ of our experiments for the initialization distance, alpha, and the number of Gaussians. Bars indicate standard errors of the means. All three parameters impact both $\mathcal{R}$ and $\mathcal{I}$. These results are interpreted in the corresponding sections.

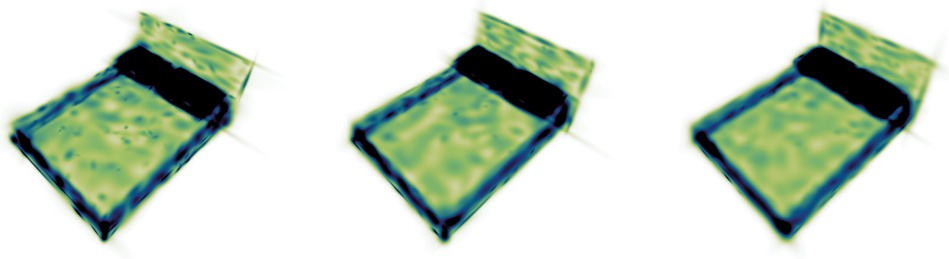
Alpha

The α-parameter controls the merging of Gaussians by determining the KL-divergence-threshold ρ, as $\rho = \alpha^2/2$. The higher α is, the farther away Gaussians can be from each other to be merged. To examine the impact of the α-parameter, we tested the values 1, 2, 3, 4, and 5 using 1.1% as initialization distance, 100,000 input points, and 512 Gaussians. The results are shown in Table 5.11. Average $\mathcal{R}$ and $\mathcal{I}$ are plotted in Figure 5.12. Exemplary GMMs are displayed Figure 5.20.

α	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
1	-8.467	0.067	**0.746**	0.015	0.5808	0.0006	3.210	0.133	0.000	0.000	12.536	8.497
2	-8.433	0.068	**0.745**	0.017	0.5803	0.0010	3.179	0.165	0.000	0.000	2.184	2.917
3	-8.444	0.070	**0.739**	0.014	0.5698	0.0005	2.701	0.138	0.000	0.000	**1.181**	0.834
4	-8.507	0.074	0.834	0.025	**0.5644**	0.0005	**2.304**	0.123	0.000	0.000	1.535	0.777
5	-8.635	0.079	1.044	0.039	**0.5637**	0.0007	**2.173**	0.097	0.000	0.000	1.867	1.022

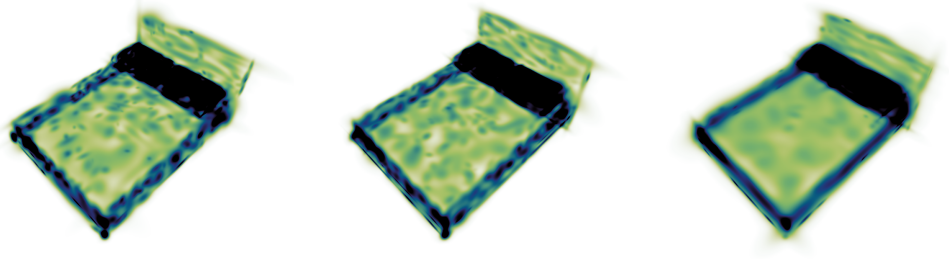
Table 5.11: Results of initialization-distance-experiments for geometrically regularized bottom-up HEM. Best values are formatted in bold. $\mathcal{R}$ is best for lower α, but $\mathcal{I}$ and $\mathcal{V}$ are better for higher α.

The reconstruction error is lowest for $\alpha \leq 3$. A higher α generally leads to smaller irregularities $\mathcal{I}$ and Gaussian variations $\mathcal{V}$.



(a) 0.9% (b) 1.1% (c) 1.3%

Figure 5.19: Resulting GMMs for the model *bed_0001* using geometrically regularized bottom-up HEM with different initialization distances. The decrease of irregularity and Gaussian variation can be observed.



(a) $\alpha = 1$ (b) $\alpha = 3$ (c) $\alpha = 5$

Figure 5.20: Resulting GMMs for the model *bed_0001* using geometrically regularized bottom-up HEM with different α-values. Low α leads to more irregular GMMs with many small Gaussians, while high α leads to smoother GMMs.

While most configurations lead to execution times below two seconds, using $\alpha = 1$ leads to a significantly higher execution time of 12.5 seconds. A reduced alpha leads to more Gaussians becoming orphans, as they are not close enough to any of the selected parent Gaussians. In this case, our extension has to find the nearest parents for the orphans, which increases the execution time.

Number of Points

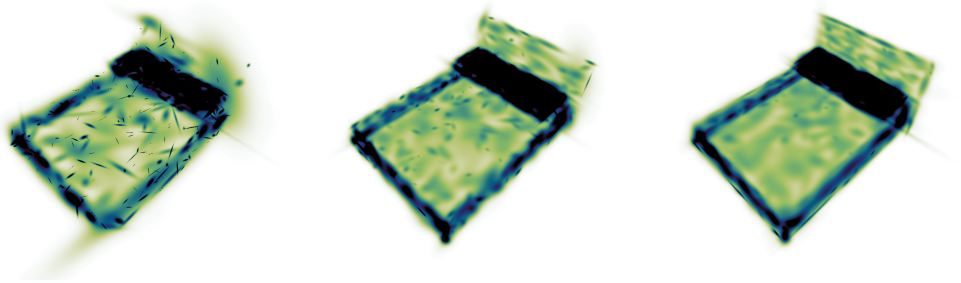
For inspecting the behavior of the algorithm with different numbers of input points, we tested 10,000, 50,000, and 100,000 points with initialization distance 1.1%, $\alpha = 4$, and 512 Gaussians. The results are shown in Table 5.12.

Reducing the number of points from 100,000 to 10,000 has a strong negative effect on $\mathcal{R}$, $\mathcal{I}$, and

$\mathcal{V}$. The quality of the results is reduced significantly, as illustrated in Figure 5.21. Using 50,000 points does not worsen the results as dramatically.

N	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
10,000	-8.967	0.063	1.529	0.054	0.6470	0.0017	7.317	0.311	0.000	0.000	**0.222**	0.154
50,000	**-8.531**	0.071	**0.856**	0.024	0.5784	0.0008	3.282	0.179	0.000	0.000	0.541	0.286
100,000	**-8.507**	0.074	**0.834**	0.025	**0.5644**	0.0005	**2.304**	0.123	0.000	0.000	1.535	0.901

Table 5.12: Results of point-count-experiments for geometrically regularized bottom-up HEM. Best values are formatted in bold. Using more points takes longer, but is favorable in regards of all our other metrics.



(a) 10,000 (b) 50,000 (c) 100,000

Figure 5.21: Resulting GMMs for the model *bed_0001* using geometrically regularized bottom-up HEM with different numbers of points. A clear decrease in accuracy and regularity can be seen for 10,000 points.

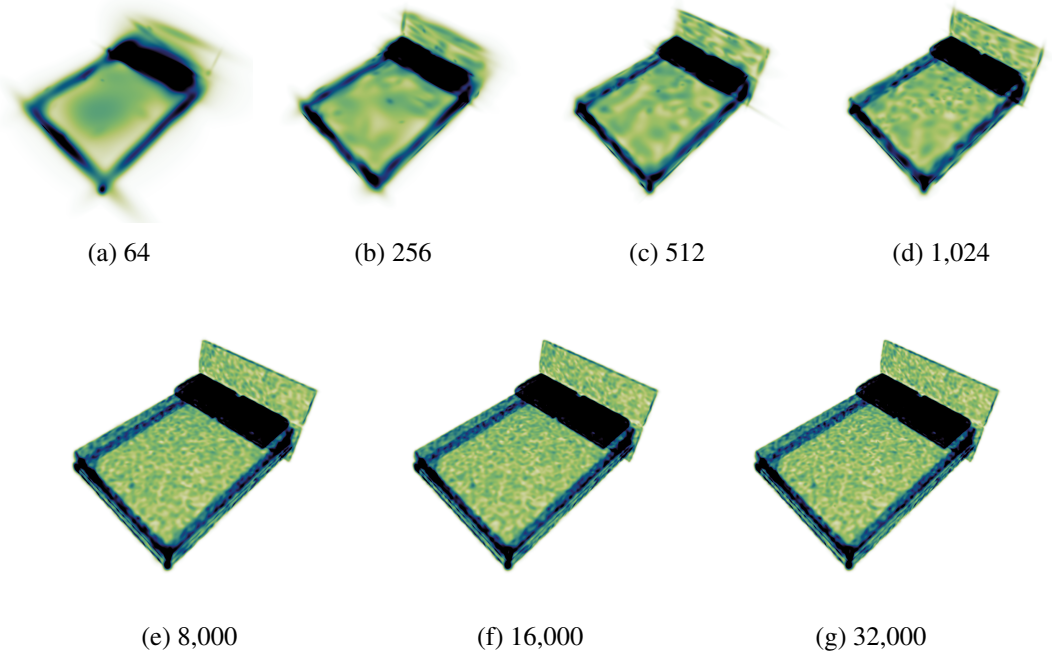
Number of Gaussians

Contrary to the other tested algorithms, this algorithm is intended for higher numbers of Gaussians and is faster the more Gaussians are used. We can therefore examine higher numbers that are not possible for the other algorithms. To examine the behavior of the algorithm we tested the algorithm with 64, 256, 512, 1,024, 8,000, 16,000, and 32,000 Gaussians using an initialization distance of 1.1%, $\alpha = 4$, and 100,000 input points. For 4,000 and more Gaussians, the desired Gaussian count is usually reached without taking extra care of orphans. Therefore, we disabled the respective part of our extension in those cases. The results are shown in Table 5.13. Average $\mathcal{R}$ and $\mathcal{I}$ for 512 and higher numbers are plotted in Figure 5.18. Example GMMs are shown in Figure 5.22.

The reconstruction error $\mathcal{R}$ is lower the more Gaussians are used. Using 8,000 or more Gaussians results in the lowest errors in this evaluation. From 64 to 1,024 Gaussians, the irregularity $\mathcal{I}$ decreases, but from 4,000 upwards it shows a rise again. The Gaussian variation $\mathcal{V}$ is reducing when using more Gaussians.

K	$\mathcal{L}$		$\mathcal{R}$		$\mathcal{I}$		$\mathcal{V}$		0-Gaussians		execution time [s]	
	AVG	SEM	AVG	SEM	AVG	SEM	AVG	SEM	AVG	STD	AVG	STD
64	-9.567	0.083	3.477	0.108	0.6709	0.0028	3.430	0.132	0.000	0.000	1.553	1.009
256	-8.807	0.078	1.401	0.046	0.5945	0.0014	3.158	0.175	0.000	0.000	1.605	1.017
512	-8.507	0.074	0.834	0.025	0.5644	0.0005	2.304	0.123	0.000	0.000	1.535	0.901
1,024	-8.307	0.072	0.619	0.016	**0.5555**	0.0004	1.971	0.099	0.000	0.000	1.592	1.013
4,000	**-8.006**	0.069	**0.458**	0.008	0.5581	0.0004	1.229	0.036	-0.005	0.071	**0.886**	0.623
8,000	**-7.939**	0.068	**0.445**	0.007	0.5667	0.0005	1.150	0.034	0.000	0.000	**0.859**	0.629
16,000	**-7.885**	0.068	**0.437**	0.006	0.5780	0.0006	**1.060**	0.031	0.000	0.000	**0.858**	0.609
32,000	**-7.851**	0.068	**0.436**	0.006	0.5890	0.0008	**1.049**	0.031	0.000	0.000	**0.815**	0.614

Table 5.13: Results of Gaussian-count-experiments for geometrically regularized bottom-up HEM. Best values are formatted in bold. Most metrics are best for higher numbers of Gaussians. $\mathcal{I}$ is an exception and is best for 1,024 Gaussians.



(a) 64 (b) 256 (c) 512 (d) 1,024

(e) 8,000 (f) 16,000 (g) 32,000

Figure 5.22: Resulting GMMs for the model *bed_0001* using geometrically regularized bottom-up HEM with different numbers of Gaussians.

5.6 Comparison

In this section, we will compare selected configurations of the three discussed algorithms. Additionally, we will also compare them to the *fps*-initialization technique (Section 4.1.3). We will first compare the results when using a sufficient amount of input points (100,000) and afterward talk about the results when using lower numbers of points (10,000).

5.6.1 Sufficient Points

In this section, we examine the algorithms when using 512 Gaussians and 100,000 points. To compare the algorithms, we will choose a few configurations per algorithm to compare.

- For EM, we will use the configurations using the *fps*-initialization and $\varepsilon = 10^{-5}$ (*EM5*) or $\varepsilon = 10^{-6}$ (*EM6*). We do not consider other initialization techniques, as they would not improve the results.
- For top-down HEM, we use soft partitioning with $\lambda_l = 0.1$ and the *fps*-initialization. For the regularization parameter ε we choose 10^{-5} (*TDS5*), 10^{-6} (*TDS6*), and 10^{-7} (*TDS7*).
- For geometrically regularized bottom-up HEM, we use an initialization distance of 1.1% and the alpha values 2 (*BU2*), 3 (*BU3*), and 4 (*BU4*).
- For the *fps*-initialization, we test different regularization parameter values: 10^{-5} (*FPS5*), 10^{-6} (*FPS6*), and 10^{-7} (*FPS7*).

Exemplary results for some of these methods are displayed in Figure 5.26.

The average execution times are plotted in Figure 5.23: EM is much slower than the other methods: It needs more than half a minute. Top-down HEM's execution time is around 10 seconds. The *fps*-initialization takes only two to three seconds. Geometrically regularized bottom-up HEM is the fastest technique with two seconds or less. As stated before, these times are specific to our implementations and do not describe the algorithms in general.

To accelerate the slower algorithms, it is an option to use only half of the points (50,000). This hardly changes the reconstruction error $\mathcal{R}$, and only slightly increases $\mathcal{I}$ and $\mathcal{V}$, while halving the execution time (see Section 5.5).

In figure 5.24, the average reconstruction errors $\mathcal{R}$, irregularities $\mathcal{I}$, and Gaussian variations $\mathcal{V}$ are plotted for the selected algorithms. While EM might be the slowest algorithm, it also provides the best results regarding the reconstruction error, and also provides the lowest irregularity when using $\varepsilon = 10^{-5}$. The results of top-down HEM regarding $\mathcal{R}$ and $\mathcal{I}$ are worse, but still close to EM. Geometrically regularized bottom-up HEM produces the worst result regarding $\mathcal{R}$ and is also worse than the previous two algorithms in terms of $\mathcal{I}$. It might be the fastest method but the results are inferior to the other algorithms for such "low" numbers of Gaussians. Finally, when using a low regularization parameter, the *fps*-initialization can create results with reconstruction errors close to EM and top-down HEM at the cost of a much higher irregularity.

For applications where the Gaussian variation matters, EM and *fps* are good options. Top-down HEM's GMMs are more varied, and geometrically bottom-up HEM shows the highest Gaussian variation.

In this comparison, geometrically regularized bottom-up HEM seems inferior. However, this algorithm is optimized for large numbers for Gaussians. It is the only one of our algorithms that can generate GMMs with 8,000 or more Gaussians, as the other ones would run into memory

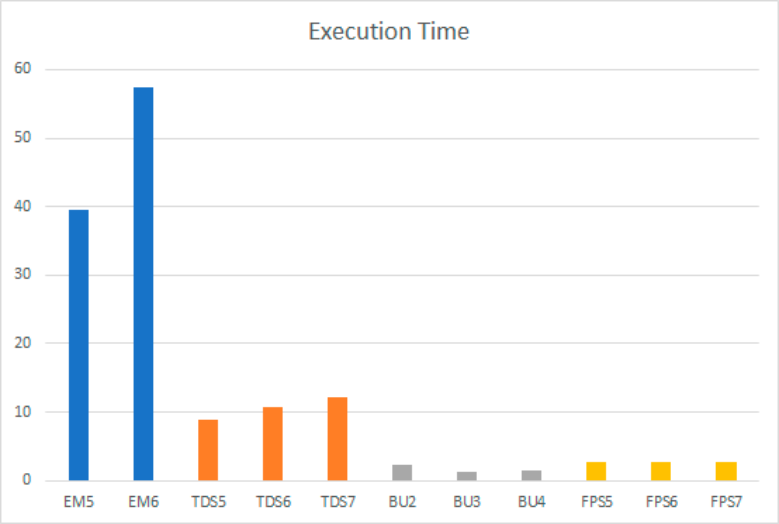


Figure 5.23: Comparison of execution times of EM (blue), top-down HEM (orange), bottom-up HEM (gray), and *fps*-initialization (yellow) using 512 Gaussians.

problems. Additionally, it is also much faster than the other algorithms would be, as it usually takes less than a second to execute. When using such high amounts of Gaussians is desired, geometrically regularized bottom-up HEM is a very efficient and powerful algorithm, that creates accurate results with low reconstruction errors (see Figures 5.25 and 5.27).

To summarize, EM produces the best results regarding all our metrics. Top-down HEM is a fast approximation for EM. The Gaussian sizes are more varied, but regarding reconstruction error and irregularity, the results are only slightly worse than EM. The *fps*-initialization by itself provides results quickly and with good reconstruction errors and low Gaussian variation. However, they have a high irregularity. Geometrically regularized bottom-up HEM is inferior for 512 Gaussians or less, but when using several thousand Gaussians it provides good results very fast.

5.6.2 Low Point Count

When using a lower number of points the results get worse for all algorithms. To examine which algorithms are favorable when only dealing with a low number of points, we examine the results of our algorithms when using 512 Gaussians and only 10,000 input points. Figure 5.28 shows the results for $\mathcal{R}$ and $\mathcal{I}$.

In this comparison, EM still creates the lowest average reconstruction error, at the cost of very high irregularity. Top-down HEM is inferior to EM regarding its reconstruction error, however, it shows much lower irregularity. Geometrically regularized bottom-up HEM provides inferior results regarding the reconstruction error but has still a lower irregularity than EM or *fps*. Examples are shown in Figures 5.10, 5.16, and 5.21.

We conclude that top-down HEM is the best suited algorithm in this case as it provides a good trade-off between low reconstruction error and low irregularity.

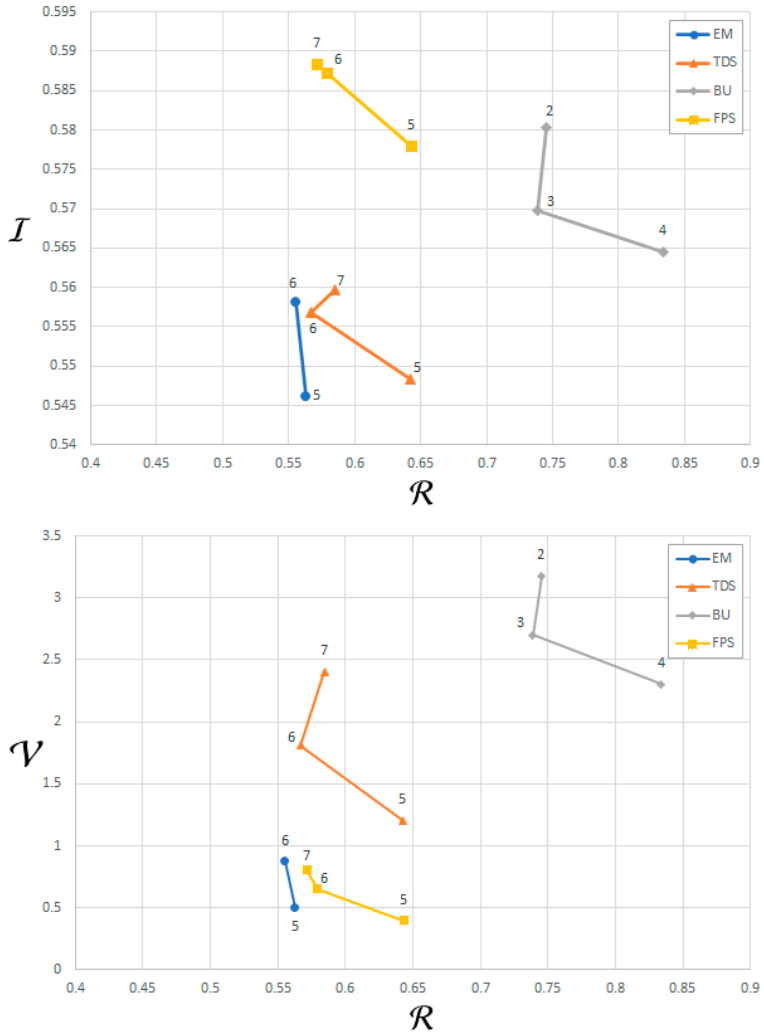


Figure 5.24: Average $\mathcal{R}$, $\mathcal{I}$, and $\mathcal{V}$ values for EM, top-down HEM with soft partitioning (TDS), geometrically bottom-up HEM (BU) and the furthest point sampling initialization technique (FPS) using 512 Gaussians. EM provides the best results in all three metrics.

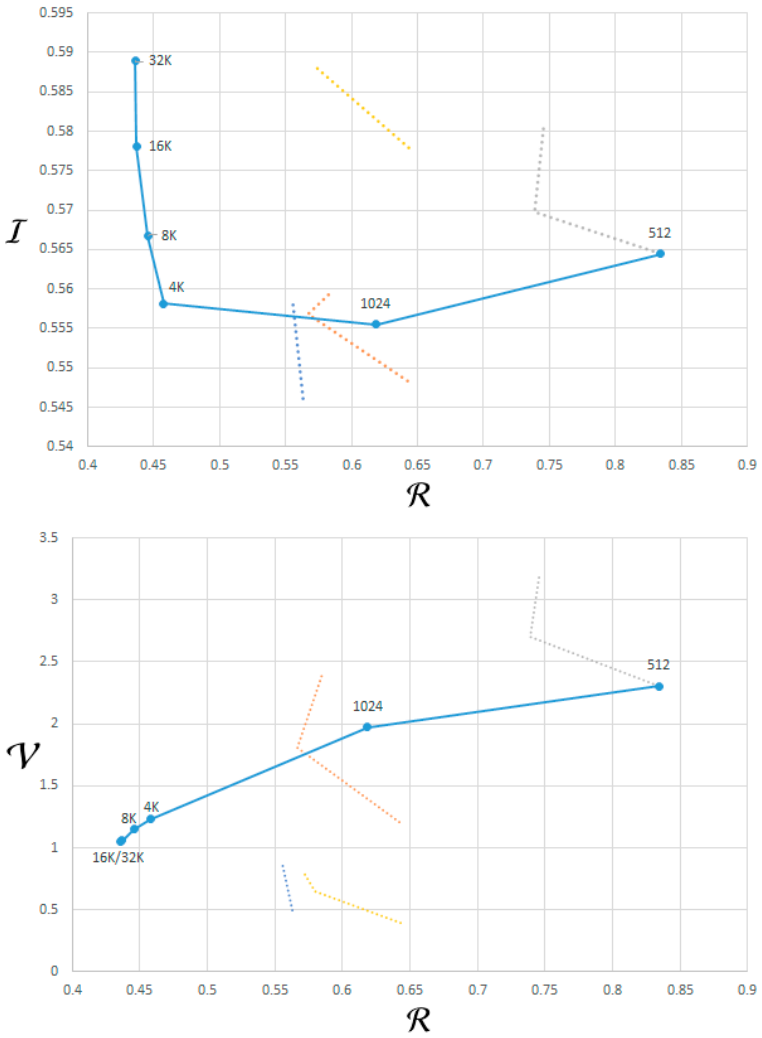
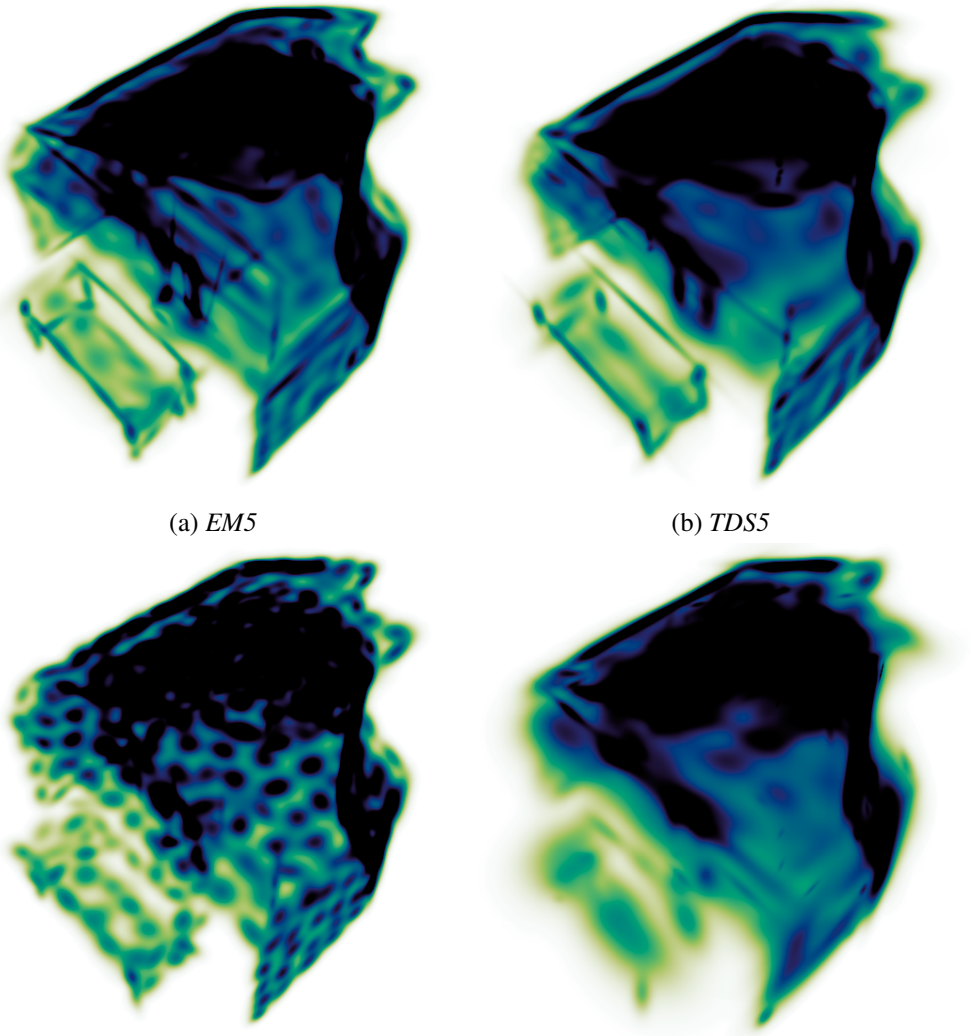


Figure 5.25: The solid line shows the average $\mathcal{R}$, $\mathcal{I}$, and $\mathcal{V}$ values for geometrically regularized bottom-up HEM with different numbers of Gaussians. For comparison with Figure 5.24, the results from the other algorithms for 512 Gaussians are shown as dotted lines. Higher number of Gaussians lead to the best $\mathcal{R}$-values in this evaluation. $\mathcal{V}$ reduces with higher numbers, but $\mathcal{I}$ rises from 4,000 Gaussians onwards.



(a) *EM5* (b) *TDS5*

(c) *FPS5* (d) *BU4*

Figure 5.26: Exemplary results of four selected algorithm configurations on the model *piano_0004*, using 512 Gaussians and 100,000 input points.

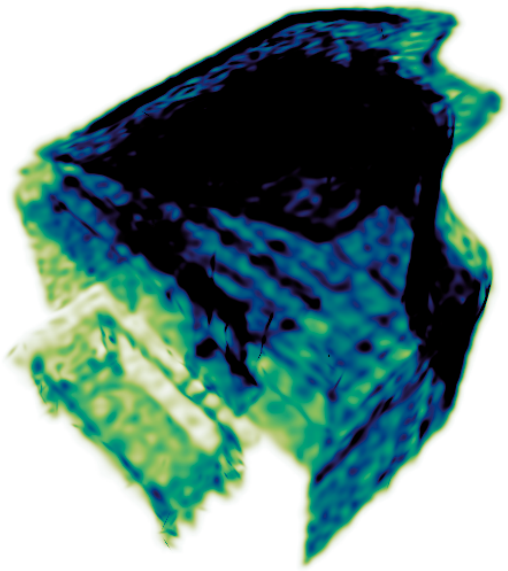


Figure 5.27: Exemplary results of geometrically regularized bottom-up HEM on the model *piano_0004*, using 16,000 Gaussians and 100,000 input points.

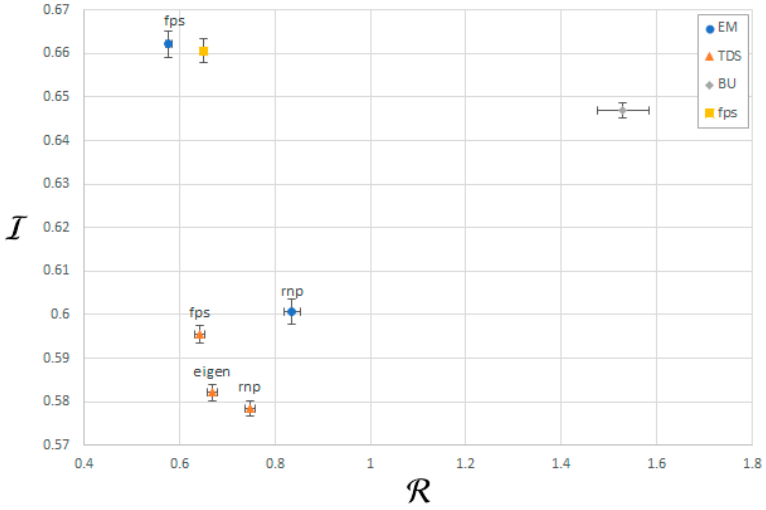


Figure 5.28: Average $\mathcal{R}$ and $\mathcal{I}$ values for the tested methods using 10,000 points. Labels indicate the used initialization method. Top-down HEM provides a good trade-off between low $\mathcal{R}$ and low $\mathcal{I}$.

CHAPTER 6

Conclusion

In this section, we summarize our work and discuss the most important findings and possible future work.

6.1 Visualization

We have implemented a tool for visualizing GMMs. This tool enables both isoellipsoid visualizations as well as density visualizations. For the density visualization, we have presented the mathematical foundations used for calculating the accumulated density per pixel. To accelerate the density visualization, a splatting approach was implemented that limits each Gaussian's extent at a configurable threshold. This way, real-time rendering of thousands of Gaussians is possible. The visualizations were integrated into a graphical user interface, which provides configuration of the visualization parameters, camera navigation, and inspection of individual Gaussians. Additionally, a Python interface was implemented that enables fast access to the visualization functionality from other programs.

Possible future work includes the extension of existing visualizations, as well as the design of further visualization techniques: A density visualization with more sophisticated options for absorption coefficient and source term would improve the quality of the results. To approximate the resulting volume-rendering integral, a fast conversion technique of GMMs into voxel grids would be required. Furthermore, isosurface visualization of GMMs remains an open topic. Isosurfaces could be approximated by volume-rendering techniques or be rendered by extending the approach by Blinn [Bli82] to work on the GPU with arbitrary GMMs.

6.2 Construction

We have implemented both EM and top-down HEM in Python using PyTorch. We added several strategies to make the algorithms numerically stable. Using a regularization is necessary to prevent

Gaussians from thinning arbitrarily and ultimately reducing the GMM's quality. Furthermore, we have implemented several initialization techniques for both algorithms.

We have adapted the source code of geometrically regularized bottom-up HEM by Preiner et al. [PMA+14] such that it always generates the desired number of Gaussians.

We have defined several criteria to evaluate the algorithms by the quality of their produced GMMs. The likelihood, which is maximized by the EM-algorithm, is not useful for our purposes, as it does not consider densities outside the surfaces. We gave examples of GMMs with high likelihoods that are not a good representation of the underlying models. Instead, we rely on a reconstruction error based on the Chamfer distance to measure the accuracy of the fit. Another important metric is the irregularity, which describes the uniformity of point clouds sampled from the GMMs. Additionally, we inspected variation of Gaussians, execution time, and the number of zero-weight-Gaussians.

We compared EM, top-down HEM, geometrically regularized bottom-up HEM, and our *fps*-initialization regarding those metrics. EM is a powerful algorithm that creates the best results given a sufficient amount of points. Using too few points, the results show a strong increase in irregularity. The irregularity can be reduced by changing the initialization technique, which comes at the cost of an increased reconstruction error. Our EM implementation is the slowest technique. When a faster technique is required, top-down HEM is a good alternative. The quality of its results regarding reconstruction error and irregularity is slightly inferior compared to EM. The *fps*-initialization can produce results with reconstruction errors close to EM or top-down HEM, however, they are much more irregular. Finally, geometrically regularized bottom-up HEM produces inferior results for 1,024 Gaussians or less regarding their reconstruction errors. However, it can produce high-quality GMMs for higher numbers of Gaussians very efficiently.

Possible future work includes faster implementations of the EM and top-down HEM algorithm, as well as the investigation of further GMM construction algorithms. We have illustrated in this thesis that the likelihood is not a perfect measurement for the quality of fit for our use case. However, many algorithms, including all EM-based algorithms, follow the maximum-likelihood approach. It might be an option to investigate more alternative approaches, and potentially design algorithms that specifically aim to reduce the reconstruction error rather than increase the likelihood.

List of Figures

List of Tables

Bibliography

[Ano22] Anonymous. Gaussian mixture convolution networks. In *Submitted to The Tenth International Conference on Learning Representations*, 2022. under review.

[AYG19] Nitin Agarwal, Sung-Eui Yoon, and M Gopi. Learning embedding of 3d models with quadric loss. *arXiv preprint arXiv:1907.10250*, 2019.

[Baj11] Peter Bajorski. *Statistics for imaging, optics, and photonics*, volume 808. John Wiley & Sons, 2011.

[BFR+98] Paul S Bradley, Usama Fayyad, Cory Reina, et al. Scaling em (expectation-maximization) clustering to large databases. *Microsoft Research*, pages 0–25, 1998.

[Bie04] Herman J Bierens. Information criteria and model selection. *Manuscript, Penn State University*, 2004.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[Bli82] James F Blinn. A generalization of algebraic surface drawing. *ACM transactions on graphics (TOG)*, 1(3):235–256, 1982.

[BSLF18] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3(4):3145–3152, 2018.

[CCD95] Gilles Celeux, Didier Chauveau, and Jean Diebolt. *On stochastic versions of the EM algorithm*. PhD thesis, INRIA, 1995.

[Che20] Mo Chen. Em algorithm for gaussian mixture model (em gmm), 2020. `https://www.mathworks.com/matlabcentral/fileexchange/26184-em-algorithm-for-gaussian-mixture-model-em-gmm`, MATLAB Central File Exchange. Retrieved October 18, 2020.

[Con19] Torch Contributors. Pytorch documentation: torch.logsumexp, 2019. `https://pytorch.org/docs/stable/generated/torch.logsumexp.html`, Retrieved October 18, 2020.

[Day69] Neil E Day. Estimating the components of a mixture of normal distributions. *Biometrika*, 56(3):463–474, 1969.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[Eck17] Benjamin Eckart. *Compact Generative Models of Point Cloud Data for 3D Perception*. PhD thesis, Carnegie Mellon University Pittsburgh, 2017.

[EHK+06] Klaus Engel, Markus Hadwiger, Joe Kniss, Christof Rezk-Salama, and Daniel Weiskopf. *Real-time volume graphics*. AK Peters/CRC Press, 2006.

[EK13] Benjamin Eckart and Alonzo Kelly. Rem-seg: A robust em algorithm for parallel segmentation and registration of point clouds. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4355–4362. IEEE, 2013.

[EKK18] Benjamin Eckart, Kihwan Kim, and Jan Kautz. Hgmr: Hierarchical gaussian mixtures for adaptive 3d registration. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 705–721, 2018.

[EKT+16] Benjamin Eckart, Kihwan Kim, Alejandro Troccoli, Alonzo Kelly, and Jan Kautz. Accelerated generative models for 3d point cloud data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5497–5505, 2016.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GFK+18] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018.

[GP19] Alexander Gepperth and Benedikt Pfülb. Gradient-based training of gaussian mixture models in high-dimensional spaces. *arXiv preprint arXiv:1912.09379*, 2019.

[GSH15] Haitao Gan, Nong Sang, and Rui Huang. Manifold regularized semi-supervised gaussian mixture model. *JOSA A*, 32(4):566–575, 2015.

[Har93] John C Hart. Ray tracing implicit surfaces. *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pages 1–16, 1993.

[Has66] Victor Hasselblad. Estimation of parameters for a mixture of normal distributions. *Technometrics*, 8(3):431–444, 1966.

[HG09] Bashar Awwad Shiekh Hasan and John Q Gan. Sequential em for unsupervised adaptive gaussian mixture model based classifier. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 96–106. Springer, 2009.

[HHGCO20] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. Pointgmm: A neural gmm network for point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12054–12063, 2020.

[HS17] Reshad Hosseini and Suvrit Sra. An alternative to em for gaussian mixture models: Batch and stochastic riemannian optimization. *arXiv preprint arXiv:1706.03267*, 2017.

[Ike00] Shiro Ikeda. Acceleration of the em algorithm. *Systems and Computers in Japan*, 31(2):10–18, 2000.

[Jak] Wenzel Jakob. pybind11 github repository. `https://github.com/pybind/pybind11`. Accessed: 28th March 2021.

[JJ93] Mortaza Jamshidian and Robert I Jennrich. Conjugate gradient acceleration of the em algorithm. *Journal of the American Statistical Association*, 88(421):221–228, 1993.

[JRJ11] Wenzel Jakob, Christian Regg, and Wojciech Jarosz. Progressive expectation-maximization for hierarchical volumetric photon mapping. In *Computer Graphics Forum*, volume 30, pages 1287–1297. Wiley Online Library, 2011.

[Kaw18] Takeshi Kawabata. Gaussian-input gaussian mixture model for representing density maps and atomic models. *Journal of structural biology*, 203(1):1–16, 2018.

[KRT99] Takamasa Koshizen, Yves Rosseel, and Yoshihiro Tonegawa. A new em algorithm using tikhonov regularization. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, volume 1, pages 413–418. IEEE, 1999.

[Lan95] Kenneth Lange. A quasi-newton acceleration of the em algorithm. *Statistica sinica*, pages 1–18, 1995.

[LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[LCH10] Jialu Liu, Deng Cai, and Xiaofei He. Gaussian mixture model with local consistency. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

[LLS87] Nan Laird, Nicholas Lange, and Daniel Stram. Maximum likelihood computations with repeated measures: application of the em algorithm. *Journal of the American Statistical Association*, 82(397):97–105, 1987.

[Lou82] Thomas A Louis. Finding the observed information matrix when using the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 44(2):226–233, 1982.

[LSW17] Yulong Lu, Andrew Stuart, and Hendrik Weber. Gaussian approximations for probability measures on rˆd. *SIAM/ASA Journal on Uncertainty Quantification*, 5(1):1136–1165, 2017.

[MP04] Geoffrey J McLachlan and David Peel. *Finite mixture models*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2004.

[MV10] Ankur Moitra and Gregory Valiant. Settling the polynomial learnability of mixtures of gaussians. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 93–102. IEEE, 2010.

[NH98] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[Pea94] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894.

[PMA+14] Reinhold Preiner, Oliver Mattausch, Murat Arikan, Renato Pajarola, and Michael Wimmer. Continuous projection for fast l1 reconstruction. *ACM Trans. Graph.*, 33(4):47–1, 2014. Link to Tool: `https://harvest4d.org/software/H4D_D9.3_CLOP.zip` (accessed on 23rd July 2020), Link to Source Code: `https://github.com/rpreiner/gmslib` (accessed on 16th August 2021).

[pyt] Pytorch homepage. `https://pytorch.org/`, Retrieved October 31, 2020.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[QSN+16] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[QYSG17] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[RW84] Richard A Redner and Homer F Walker. Mixture densities, maximum likelihood and the em algorithm. *SIAM review*, 26(2):195–239, 1984.

[scia] Scikit learn - gaussian mixture models. `https://scikit-learn.org/stable/modules/mixture.html`. Accessed: 10th August 2021.

[scib] Scikit learn - kmeans. `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`. Accessed: 10th August 2021.

[SRG03] Ruslan Salakhutdinov, Sam T Roweis, and Zoubin Ghahramani. Optimization with em and expectation-conjugate-gradient. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 672–679, 2003.

[sta] Stackexchange: Eigenvalues of the sum of two matrices: one diagonal and the other not. `https://math.stackexchange.com/questions/1374836/eigenvalues-of-the-sum-of-two-matrices-one-diagonal-and-the-other-not`. Accessed: 17th August 2021.

[TT84] Heang K Tuy and Lee Tan Tuy. Direct 2-d display of 3-d objects. *IEEE Computer Graphics and Applications*, 4(10):29–34, 1984.

[VL98] Nuno Vasconcelos and Andrew Lippman. Learning mixture hierarchies. In *NIPS*, pages 606–612. Citeseer, 1998.

[WSK+15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[XJ96] Lei Xu and Michael I Jordan. On convergence properties of the em algorithm for gaussian mixtures. *Neural computation*, 8(1):129–151, 1996.

[ZPK18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.