TU WIEN Informatics

Shape Shifting

A Multiscale Optimal Transport Approach to 3D Point Cloud Comparison

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Marwin Schindler, BSc
Matrikelnummer 01627754

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Asst.Prof. Dr.in Renata Georgia Raidou

Wien, 30. April 2025

Marwin Schindler	Renata Georgia Raidou

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Shape Shifting

A Multiscale Optimal Transport Approach to 3D Point Cloud Comparison

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Marwin Schindler, BSc
Registration Number 01627754

to the Faculty of Informatics

at the TU Wien

Advisor: Asst.Prof. Dr.in Renata Georgia Raidou

Vienna, April 30, 2025

Marwin Schindler	Renata Georgia Raidou

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Marwin Schindler, 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.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 30. April 2025

Marwin Schindler

Danksagung

Meinen tiefsten Dank will ich meiner Betreuerin Renata Raidou ausdrücken, für ihre Unterstützung und Mühen während der Arbeit an meiner Diplomarbeit. Aber auch, weil sie durch mein gesamtes Masterstudium die beste Professorin war, die ich mir hätte wünschen können. Danke, dass du mir vertraut hast, und Freiheiten gegeben hast meine Ideen zu verfolgen, selbst als sie noch ihre Form finden mussten. Danke schön für deine Leitung durch unsere gemeinsamen Projekte und all die Errungenschaften, die wir erarbeitet haben, die ich in Ehren halten werde.

Danke auch an all meine Freunde dafür, dass wir gemeinsam wachsen und lernen konnten und auch für die Ablenkung in den verdienten Pausen. Ich möchte mich bei Andi, Joey und Tobi bedanken, für die Projekte, die wir während unseres Studiums gemacht haben, von denen manche nach wie vor, auf die beste Art, Tränen des Lachens hervorrufen.

Vielen Dank an meine Familie, vor allem an meine Eltern für deren Unterstützung. Ihr habt es mir ermöglicht meinem Studium nachzugehen und wart beide immer für mich da, wegen und auch abseits des Studiums. Ich möchte meiner Mutter danken dafür, dass sie mir schon früh den Weg geebnet hat und ich möchte meinem Vater dafür danken, dass er mir immer den Rücken gestärkt hat. Ich weiß, es hätte dir viel bedeutet.

Danke Elli, dafür, dass du mir jeden Tag mit deiner Herzlichkeit erleichterst. Ich kann mir nicht vorstellen, dass ich all das ohne dir geschafft hätte. Ich freue mich darauf, was die Zukunft für uns bereithält. Ein kleines Danke auch an die Kleine auf dem Weg dafür, dass sie mir einen guten Ansporn gab die Implementierung fertigzustellen.

Acknowledgements

I want to express my deepest thanks to Renata Raidou for her support and efforts during my thesis. Also, for being the best professor that I could have wished for during my master's degree. I'm grateful that you trusted me and gave me the freedom to explore ideas even when they were still finding their shape. Thank you for guiding me through our projects and all the achievements we gained together, which I will cherish.

Thanks to all my friends for growing and learning alongside each other and distracting me during the well-deserved breaks. I want to thank Andi, Joey, and Tobi for the projects we did together during our studies, some of which, in the best way, still trigger tears of laughter.

Many thanks to my family, especially my parents, for their support. You enabled me to pursue my studies, and both of you have always been there for me, during and beyond my studies. I would like to thank my mother for paving my way early on, and I want to thank my father for always bolstering my back. I know it would have meant a lot to you.

Thank you, Elli, for easing every day with your kindness. I can not imagine having done all that without you. I'm looking forward to what the future holds for us. A little thank you also to the little one on the way, for giving me a good incentive to finish the implementation.

Kurzfassung

Fortschritte in der Vermessungstechnik und der dreidimensionalen Bildverarbeitung haben die Geschwindigkeit und Präzision, mit der Objekte und Landschaften in der realen Welt eingefangen und rekonstruiert werden können, erheblich verbessert. Diese virtuellen Rekonstruktionen sind relevant für Vermessungsanwendungen und werden häufig als Punktwolken abgespeichert, d.h., eine Reihe von 3D Koordinaten, mit optionalen zusätzlichen Attributen wie Farben und Normalen. Häufig werden solche Rekonstruktionen mehrmals über einen längeren Zeitraum hinweg aufgenommen, um Veränderung einzufangen. Intuitive vergleichende Visualisierungen können dabei helfen den Wandel dieser Rekonstruktionen zu veranschaulichen. Jedoch bergen die Ausmaße solcher Daten Herausforderungen für Methoden der Visualisierung. Des Weiteren ist es einfacher denn je, große Datensätze solcher Rekonstruktionen zu sammeln, selbst für unerfahrene Anwender. Jedoch gibt es abseits der zeitintensiven Algorithmen, die für den medizinischen Bereich konzipiert sind, wenige Anwendungen die ermöglichen Unterschiede in solchen Ensembles von Formen zu vergleichen. Die verfügbaren Algorithmen basieren auf Nachbarschaftsbeziehungen, welche bei der Anwendung mit komplexen Formen nicht gut skalieren und keine Strukturierung der Daten unterstützen. Umfangreiche Ensembles von räumlichen Daten zu ordnen kann die Analyse vereinfachen, wenn keine chronologische Sortierung vorliegt.

Wir haben ein Framework entworfen und implementiert, um die vergleichende Visualisierung von Ensembles von Punktwolken zu unterstützen. Wir nutzen dafür die umfangreiche mathematische Theorie des optimalen Transports und umgehen damit die Nachteile von Ansätzen, die auf Nachbarschaftsbeziehungen basieren. Wenn es keine implizite Reihenfolge der Daten gibt, ermöglicht unsere Methode das automatische Arrangieren der einzelnen Punktwolken, um Beziehungen zwischen ihnen herzustellen und die Analyse zu vereinfachen. Die visuelle Hervorhebung von Abweichungen innerhalb der Daten erleichtert das Auffinden von Mustern. Durch das Nutzen von massiv parallelen Implementierungen ermöglichen wir einen animierten Übergang zwischen den Punktwolken. Die visuelle Betonung von Charakteristiken und Veränderungen jeder einzelnen Form unterstützt dabei die Klarheit der Visualisierung. Unsere Methode prozessiert die Daten schnell und bietet eine umfangreiche Auswahl an Werkzeugen, um ein ganzes Ensemble von Punktwolken zu analysieren.

Abstract

Advances in measurement technologies and 3D vision have significantly enhanced the speed and precision with which real-world objects and landscapes can be captured and reconstructed. These virtual reconstructions are relevant for surveying applications and are often encoded as point clouds, i.e., a set of 3D coordinates, possibly accompanied by additional attributes like colors or normals. Often, reconstructions of objects or landscapes are acquired over time to monitor their changes. Intuitive visualization that allows one to comprehend the shifts over time in such reconstructions could be of help, but the vast size of the data imposes challenges on comparative visualization pipelines. On the other hand, it is simpler than ever to amass numerous reconstructions of real-world objects, even for novice users. Still, outside of computationally intensive algorithms tailored to applications for the medical domain, there is a gap in approaches that allow for comparing differences within ensembles of shapes. Available algorithms outside of medicine are built upon nearest neighbor queries, which do not scale well to complex shapes and lack guidance for the comparison. Extensive ensembles of spatial data need to be delivered in a structured way to avoid time-intensive manual ordering when there is no chronological ordering implied or known.

We designed and implemented a framework to support the comparative visualization of ensembles of point clouds. By utilizing the mature mathematical framework of optimal transport, we circumvent shortcomings of commonly employed nearest neighbor-based approaches and allow our method to compare a whole ensemble of reconstructions in a comprehensive representation. If there is no inherent ordering, our method enables the automatic arrangement of individual point clouds, establishing their relationships and simplifying the analysis process. We derive additional metrics about the whole ensemble, which are then used to enrich the visualization and help to detect patterns of variation within the data. By leveraging fast GPU-based implementations, we enable a smooth transition between displayed point clouds in an animation and offer visual aids that highlight the characteristics of each shape and how these change. Our method processes the data fast and provides comprehensive means to browse through a large ensemble of point clouds.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Definition

Over the past decades, advancements in measurement technology have significantly improved our ability to capture the geometric characteristics of our surroundings. The resulting datasets are vast and complex, and impose challenges on the algorithms designed to process them. Visualization pipelines are required to scale with these increasing sizes of spatial datasets, and even more challenges arise when the problem requires the comparison of multiple datasets. Addressing these challenges that result from extending the visualization and analysis pipeline to multiple large entities and putting them in context with each other, while still maintaining reasonable computational performance, defines the main endeavor of this work. In this chapter, we establish the general topics and the range of problems that have been considered within the context of this thesis.

Comparative visualization describes methods utilized to find similarities or differences within data [KCK17]. The resulting techniques are embedded in a broad range of scientific fields like medicine, geodesy, architecture, and industrial design. All of these disciplines work with different data modalities, but share their dependence on means to compare multi-dimensional data under task-specific requirements. With the present possibilities to simulate and collect data, a lot of research disciplines depend on an insightful comparison of different simulation outcomes. At the same time, measured data from the real world often needs to be investigated automatically, to reveal any changes that took place, using tools from comparative visualization.

Point clouds, stemming from laser scanning or photogrammetry, comprise a major part of topographic data. LiDAR (Light Detection and Ranging) scanning [VM10], where the surrounding environment is sampled using pulses of laser light, is steadily improving in capturing datasets, to an increasing extent and a finer resolution. Also, photogrammetry methods [Kra07] that reconstruct the 3D coordinates of a point cloud from image bundles

are becoming more accessible to the broad public and are rapidly improving to capture finer details. This data is often used for change detection, where given a multi-temporal scan of the same entity, it should automatically be detected if and where something has changed. Traditional comparison of point cloud data is mostly concerned with finding differences and similarities within data pairs, and new challenges arise when a large collection (i.e., more than two samples) of such datasets should be compared.

Ensembles describe sets of data instances, with a common context and similar characteristics [WHLS19]. The individual ensemble members, such as those in a multi-temporal point cloud, are often captured and displayed in chronological order. There are also scenarios where such a sorting order is not given, for instance, when comparing shape ensembles, as commonly done in medicine, biology, and epidemiology. Acquisition technologies like terrestrial scanners and photogrammetry deliver extensive amounts of data, whose analysis has been a central task in the field of computer graphics for a long time. Comparing multiple instances of spatial data is very challenging, due to the large amount of information that must be processed. Medical disciplines face similar challenges when comparing volumetric images, acquired by technologies like computed tomography (CT) or magnetic resonance imaging (MRI). The ability to derive differences between multiple instances can be very helpful, for instance, when investigating the anatomy of patients and occurring pathological changes, or when comparing the effect of a certain treatment in different situations. A framework to guide the comparison and indicate an order of the individual data entities could greatly simplify the difference detection. These shape ensembles lack algebraic operations to put them into relation, which makes sorting a challenging task. For instance, taking the sum of two point clouds is not defined in a traditional sense, but identifying that two shapes are close or far away from each other is still insightful [Fey20]. This requires a suitable metric that captures the similarity of the data.

Morphometrics refers to the quantitative analysis of shapes [HK15]. A shape is defined as what remains after removing all global geometric information from an object, like the transformations that position it in some reference frame. Larger ensembles of the resulting shapes are then studied, using tools from statistics to find variation. This comparison often takes place within a so-called shape space, a manifold in which each point encodes a shape for a chosen metric. The resulting methods can be utilized, for example, to differentiate between species in biology or to describe an evolutionary response to a gene mutation. An important related field is Computational Anatomy [GM98, You10], focusing on finding mathematical models and statistical analysis to study the variability and characteristics of anatomical shapes, for instance, to describe the effects of certain health conditions or to find insights from large population studies.

Animation is an intuitive form of presenting time-varying data, where changes can be displayed directly as if the intermediate states of the data had been captured. Therefore, interchangeable designs and animations are an established choice, out of the various methods, that one can resort to when attempting to build a comparative visualization pipeline [KCK17]. With spatial data, space in the visualization to further encode

information is very limited, making animation a well-suited method. Although using animation has also been considered sub-optimal within multiple contexts in visualization [LLPY07, Mun15], since it can lead to fatigue of users when more focus is required to observe changes. Another difficulty in this context, of comparing a multi-temporal point cloud, is that there are no explicit point-to-point correspondences, due to differences in the sampling of the surface and acquisition noise. This needs to be respected when computing the distances and transitions between data instances.

Correspondence matching between spatial data is a very challenging task that shares characteristics with scientific topics like feature detection and registration [Fey20]. Registration describes the process of aligning two or more data instances to each other [ZF03, DYDZ22], such as images, point clouds, or 3D models, which can be implemented at varying levels of complexity. Where rigid registration only allows for translation and rotation, affine registration further allows for scaling and shearing, as used in Morphometrics, to remove the differences between data instances that don't contribute to the shape. Non-rigid registration methods capture the actual differences in the shape by building a deformation field that transforms a source shape to form a target shape when applied. Methods that compute such a non-rigid deformation depend on costly optimization routines and are not directly applicable to real-time visualization applications of large spatial data.

Scalability is therefore a major concern when working with large spatial data ensembles. The computational cost of the comparison and difference detection between multiple point clouds is a large bottleneck in a comparative visualization pipeline. The runtime of the algorithms should be kept at a minimum to enable interactive exploration of the data. This is especially important when the data is used for decision-making processes, where the user needs to understand the differences between data instances quickly.

To sum up, traditional methods struggle to efficiently compare ensembles of multiple larger-scale point clouds, due to high computational demands, especially when point-to-point correspondences are not granted. Given the pressing need for a scalable data framework that enables efficient, interactive, and insightful comparison of large spatial data ensembles, in this thesis we will answer the following research question: *How can a scalable and computationally efficient comparative visualization framework enable interactive exploration and insightful comparison of large spatial data ensembles, while effectively highlighting areas of significant difference through intuitive animations?*

1.2 Aim of this Work

We envision an approach to identify and represent differences within spatial data ensembles of multiple large point clouds. For that, an order should be established semi-automatically within the given ensemble that eases the comparison for users by specifying an explicit sequence in which ensemble members can be compared. An incentive here is to model the associated shape space as simply as possible and let users quickly navigate through the whole dataset by morphing from one ensemble member to the next one, in a smooth

animation, which serves as an engaging overview for the recipients. The current position in the shape space itself should be represented in an auxiliary view.

To compute transitions between ensembles of large point clouds within a reasonable runtime, an existing optimal transport framework [Fey20] should be extended with a hierarchical data structure to create assignments between the ensemble members. This corresponds to performing a clustering of the data entities into coherent regions. These regions are the basis for the pairwise hierarchical correspondence estimation between point clouds. In our case, these clusters should stem from the nodes of an octree data structure [SOW20]. An advantage of using a hierarchical data structure for the matching process is that we can drastically increase the possible data size. Octrees are a common choice for the processing and structuring of point clouds and enable us to compute the differences at coarser granularity but with less computational costs [Fey20].

Further, an explicit encoding of regions of strong difference within the ensemble should be implemented using colors, complemented with contours to aid recipients in reasoning about the shape and regions of high variability. This requires us to implement a 3D viewer with the capability to render the data, animate the precomputed transformation assignments, and post-process the renderings to add visual landmarks.

These proposed undertakings come with the challenge that the established algorithms for non-rigid registration already fully utilize available computational resources, but for consumer-grade systems, the runtimes are far from enabling immediate comparison for large data entities. To perform studies on ensembles of large point clouds, the issues of speed and scalability have to be taken special care of.

To sum up, the **main contribution** of this thesis is a comparative visualization application that allows for computing and visualizing the outcome of non-rigid registrations between the members of large point cloud ensembles, to derive geometric differences. The fully registered ensemble can be automatically ordered within a linear shape space that is easy to navigate. To enable fast processing times for large ensembles of point clouds, we extend a state-of-the-art library for optimal transport computations. The results of this processing are visualized using a modern rendering framework and custom post-processing shader pipelines, allowing for various means of visual encoding to highlight changes between ensemble members, while the changes themselves are directly mapped to an animation using the compute capabilities of modern graphics hardware.

1.3 Structure of the Thesis

The following pages explain the achieved comparative visualization pipeline and the utilized concepts that allow for the fast processing of large spatial data. In Chapter 2 we list literature that shares the use of comparative visualization to achieve goals, similar to the ones defined in the previous introduction. Further, publications that utilize optimal transport theory for problems related to computer graphics and computer vision are discussed. In Chapter 3 we summarize concepts that are relevant to the remainder of this thesis, and the implementation of a solver for the optimal transport problem. We list prerequisites and defined requirements in Chapter 4. The methodology that we use to achieve the proposed comparative visualization framework is split into two chapters, first an explanation of our approach to the optimal transport problem in Chapter 5, which follows an explanation of our visualization pipeline in Chapter 6. Technical details of the implementation are summarized in Chapter 7. In Chapter 8 we show multiple different use cases for our comparative visualization approach and the results of a performance evaluation. A summarizing discussion of the implemented solution, together with limitations and an outlook to possible future work, can be found in Chapter 9.

CHAPTER 2

Related Work

Finding patterns of variation within spatial data and their causes is of interest in many research areas. For instance, investigating anatomical variation helps research in medicine to get valuable insights into how certain diseases alter, or how their prognosis is affected by the shapes of anatomical structures [CEW17, RCMA+18, FMCM+21]. Identifying the distinguishing features of multiple entities requires adequate comparison methods, built upon visualizations that support understanding, communication, and decision-making. Studying configurations of spatial data to find resemblances and differences is a general challenge for various research subjects. The different perspectives on the problem led to a range of terms within the visualization community. Works that seem closely related to the content of this thesis were published using keywords like “Comparative Visualization”, “Visual Shape Analytics”, “Morphometrics”, and “Ensemble/Cohort Visualization”, to name a few.

Easing the comparison process is challenging due to the ever-growing amounts of data that should be made observable, while allowing for details (potentially, on demand) and interaction with the data where necessary. This requires finely tuned visualization methods that highlight the qualities of the data as comprehensibly as possible. With an increase in data complexity, it becomes much harder to arrange the multitude of information in a single descriptive visualization. Comparative descriptiveness needs to be carefully crafted for large and high-dimensional data [KCK17]. Adequate difference metrics that emphasize the most meaningful discrepancies in the comparison have to be selected. Multiple options exist for established metrics, a few of which are compared later in Subsection 3.1.2. One of the choices is the Wasserstein distance, derived from the topic of optimal transport that has been researched for a long time [Kan60].

2.1 Comparative Visualization

Given the multitude of work that has been done in the research field of comparative visualization, several publications summarize the results that have been achieved so far. The state-of-the-art report by Kim et al. [KCK17] presents a comparative visualization taxonomy for 3D and 4D spatial data visualizations obtained by surveying the research field. Here, spatial has to be interpreted in the sense that the position, width, height, depth, and relative position of things and their shape all matter to a recipient, which complicates the creation of insightful visualizations. There is no single design approach that can fit all possible use cases, and different applications require appropriate handling. They identify the following four fundamental approaches as the key design methods.

- **Juxtaposition** describes approaches where multiple data instances are displayed simultaneously side by side.
- **Superimposition** indicates that multiple data instances are displayed at the same time in the same coordinate system, i.e., overlaid over each other.
- **Interchangeability** is described as displaying a single data instance at first, but over time, the visualization changes to display additional data, all of which is registered in the same coordinate system. An example thereof could be animated visualizations.
- **Explicit encoding** requires deriving a composite visualization from the data instances, rather than displaying all in their original form. For instance, this could be a difference boolean operation applied to pairs of volumetric data.

The four approaches are illustrated in Figure 2.1. Their findings point towards a general lack of comparative visualization tools for time-varying spatial 3D data. They also point out that most comparative visualizations for spatial data are built to support only a small number of data instances, often due to the computational cost of rendering 3D data. This is a common problem for volumetric data, where single data entities can easily consist of billions of values, and also for point clouds, which are often even larger. These scales entail the problem that a single design approach on its own might not be suitable to convey all relevant information. The most promising line of work that Kim et al. refer to is to upgrade the fundamental approaches into a hybrid approach and use a combination thereof to compensate for their respective weaknesses.

A lot of literature, which can be considered closely related to comparative visualization, has been published on the topic of ensemble visualization [KCK17]. This line of research often focuses on the different results from simulations when the initial conditions and parameters are modified. An example would be the work of Evers and Linsen [EL22], where the parameters-induced space is partitioned in a semi-automatic fashion based on clusters of similar simulation runs. Weather forecast ensembles are also well suited for methods from ensemble visualization. In the work of Ferstl et al. [FKRW17], the authors

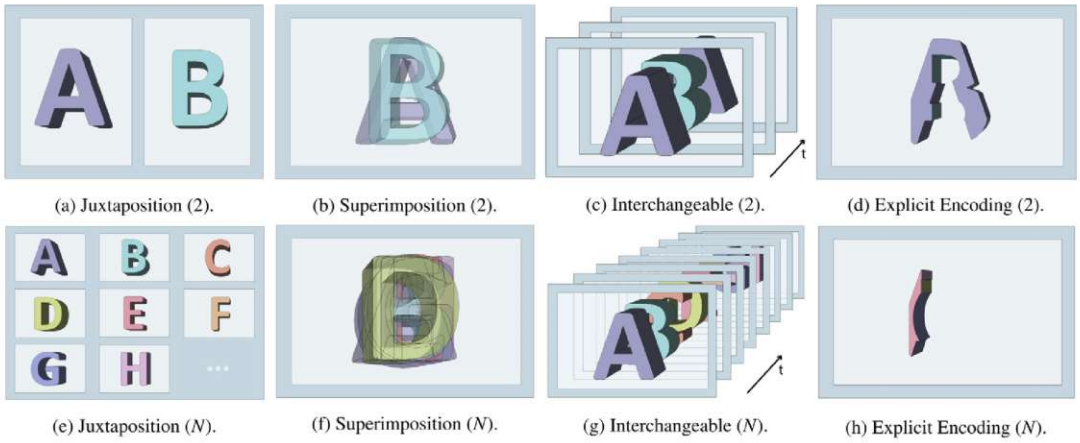


Figure 2.1: Illustrations of the four fundamental approaches to comparative visualization for spatial data from Kim et al. [KCK17].

employ clustering of iso-contours extracted from weather forecast ensembles to visualize the chaotic and thus highly uncertain nature of the weather over time. Unfortunately, their approach utilizes 2D data and isn't directly applicable to 3D point-cloud data. Volumetric data is also frequently used in the context of ensemble visualization, be it from medical scans or material studies. A publication by Weissenböck et al. [WFG+19] investigates volumes from industrial 3D X-ray computed tomography of foam samples. They unfold the volume to a 1D line by covering all its cells with a coherent space-filling curve. This curve is then unfolded and rendered in a nonlinear scaled line plot, where screen space is assigned to regions based on the variance across different ensemble members. Brushing and linking allow for the exploration of the original underlying data instances in a 3D rendering, where regions of interest from the plot are highlighted using colors. Although space-filling curves have been used for point cloud data before [GVOC18] and can handle large point counts in the range of billions, they demand heavy pre-processing with high computational cost, without computing the correspondences between the points of different point clouds in the process.

2.1.1 Visual Shape Analytics

A large body of literature, tightly coupled to comparative visualization and its design paradigms, is published within the scientific branch of visual shape analytics. A survey by Hermann and Klein [HK15] describes the subject as finding new ways to visualize shape variability, found by long-established techniques in the field of morphometrics [MS13]. Previous techniques in morphometrics employed automatic statistical evaluation of the data to find the occurrences and sources of variations in organic shapes, thus only providing the final results acquired [Roh90]. Visual shape analytics build upon an interactive workflow to display differences directly to the users where they are of interest. A core contribution is to link abstract representations of a high-dimensional shape space

with a 3D visualization, in a way that makes it intuitive to explore the shape space of the investigated data. The work of Busking et al. [BBP10], where navigation inside a 2D scatterplot steers the 3D visualization of shapes next to it, as seen in Figure 2.4b, could be an example. Other approaches allow for direct manipulation of the shapes within the 3D visualization, which can be seen in Figure 2.5c from the work of Hermann et al. [HSSK16].

It is important to provide additional statistical analysis and details when needed. Rigid transformations (and sometimes all affine transformations) are factored out before the actual statistical analysis. This way, only the remaining parts of the data are considered, which are captured by non-rigid deformations and constitute the shape of the entities. Often, a representative template shape is computed for the whole shape ensemble at hand and used as the baseline for the comparison within the shape space. From this reference shape, the modes of variation are computed, which are displacement vectors or functions that represent the patterns of variation within an ensemble [LAA+13]. An illustration of this workflow can be seen in Figure 2.2. These modes are often computed using Principal Component Analysis (PCA) to obtain the eigenvalues and eigenvectors of a deformation.

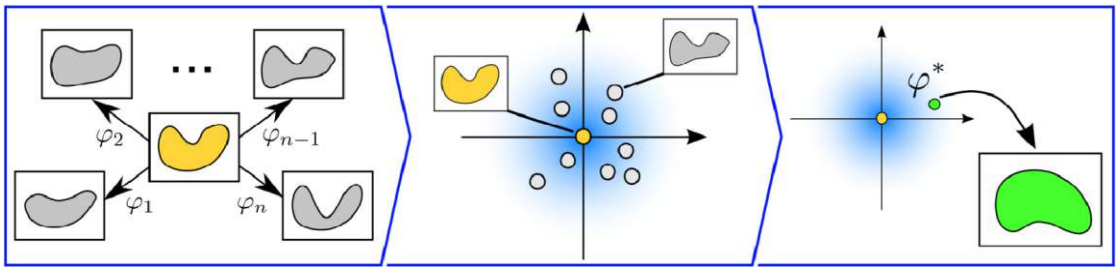


Figure 2.2: The visual shape analytics modeling pipeline in the work of Hermann et al. [HK15]. First, shape variability is represented by registering all entities to a template shape. Subsequently, the individual shapes can be represented by a deformation of the template shape. The last subfigure depicts the synthesis of novel shapes based on the statistical analysis of the deformations.

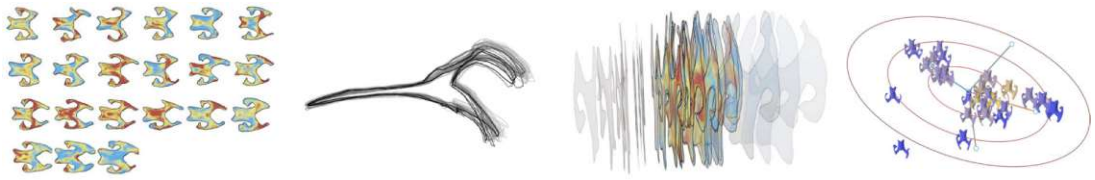
The differences between the shapes can be described using first and second-order moments, such as the mean and the covariance of a displacement vector field that can be constructed from methods within the framework of Large Deformation Diffeomorphic Metric Mapping (LDDMM) [BMTY05]. LDDMM provides a method to establish a deformation model for a given shape space, under certain regularization that makes the resulting deformations highly suitable for the field of computational anatomy. The method builds a bijective and smooth 3D map between two data instances, which also applies to the inverse of the map. An example of such a map can be seen in Figure 2.3a and the corresponding vector field in Figure 2.3b. Unfortunately, the complexity of these methods leads to a large computational overhead, and non-rigid registration is mostly performed as a time-intensive pre-processing step.

(a) (b) (c) (d) (e)

Figure 2.3: In Figure 2.3a an example mapping computed using LDDMM is shown, and Figure 2.3b depicts the vector field that induces the mapping, taken from Beg et al. [BMTY05]. From Figure 2.3c to Figure 2.3e, the progression of the deformation is depicted when the mapping from Figure 2.3a is applied to a 2D slice from an MRI of a canine heart. The mapping is computed between a normal canine heart in Figure 2.3c and a failing canine heart in Figure 2.3e.

A work by Busking et al. [BBP09] implements direct focus and context visualizations of deformation fields from non-rigid registration between two volumes. This analysis is referred to as deformation-based morphometry and is popular in the context of brain-imaging research. Their approach allows for visualizing certain characteristics of the deformation field on top of the scalar volume that is being deformed. Importance values are assigned to certain areas of the volume by an interest function to handle occlusion. The analysis of a deformation field often includes analyzing the Jacobian matrix to find the areas of positive and negative change within the volume. For medical data, this translates to a loss or a gain in tissue. This metric is overlaid with a visualization of the magnitude of the displacement field with respect to the pre-computed importance values to blend between both. The scalar context volume, to which the deformation is applied, is rendered sparsely as the first layer of the visualization. Additionally, the deformation field can be visualized superimposed on the volume. The result is a visualization that either depicts the dynamic context of the deformation as an animation or by superimposing and blending the different layers, using colors to differentiate between them.

In the later work of Busking et al. [BBP10], a scatterplot enables smooth navigation through the computed shape space. Each point in the plot corresponds to either an entity in the ensemble or a linear interpolation between those entities. Their shape space is built according to the Active Shape Model (ASM), where the distribution in space is based on the principal components of the volumetric dataset. The proposed application features three different views. The first one to visualize the shape space as a scatter plot formed by a customizable projection of the higher-dimensional points to 2D. Colors in the plot indicate the likelihood of each shape with respect to the underlying Gaussian distribution of the model, as seen in Figure 2.4b. The object view visualizes the shape corresponding to the currently selected point in shape space, with an optional explicit encoding of the distance to the ensemble mean. As a third view, they propose to use a shape evolution view as a compromise between the global shape space view and the local object view. Several shape samples within a 1D subspace of shape space are projected to



(a) (b)

Figure 2.4: Figure 2.4a shows the auxiliary views that can be added to a regular 3D view of the currently active shape in the visualization tool of Busking et al. [BBP10]. Figure 2.4b displays a scatterplot that allows for navigation through shape space, showing small depictions of the ensemble members colored according to their likelihood within the Gaussian distribution modeling the ensemble.

2D and either displayed in a 2D grid or as a stack of 2D shapes along the third axis in a 3D view, as depicted in Figure 2.4a. Users can decide if they want to display a stack of the actual ensemble members this way or a progression of the in-between shapes of a linear interpolation. The combination and the linked interaction of all views enable a holistic exploration of the shape space.

Kilian et al. [KMP07] present an approach to model variations of triangular meshes by manipulating their representation within a shape space. This allows for morphing, deformation, and exploration of the shapes. Their approach is based on letting the visualized shape move on a geodesic curve within the shape space to interpolate between two states. Being solely a geometric approach, it can be applied to a large class of problems without knowledge of the semantic features of the underlying physical models.

The aforementioned principles of visual shape analytics are employed in the later work of Hermann et al. [HSSK16], which utilizes stationary velocity fields to facilitate interactive non-linear image interpolation and plausible extrapolation of shape ensembles. Stationary velocity fields offer simpler-to-compute alternatives to the much more expensive methods from LDDMM, enabling efficient visualization. Establishing a template shape enables browsing between the individual images of the ensemble, which can be reconstructed by applying the respective displacement field to the template shape. The mean shapes of subsets of the ensemble are computed on the fly and can be compared to other mean shapes or the ensemble template shape. To visualize the whole shape ensemble in superimposition, they included likelihood volumes in their application, where each voxel encodes the likelihood of being occupied within the given ensemble, as seen in Figure 2.5a. All shapes can be subject to an interactive reformation, in which the shape is purposefully deformed to aid interpretation, as in Figure 2.5c. To allow direct insights into the animated deformations, they include the option to render certain parts of the deformation field as streamlines on top of the displayed volume, using common visualization techniques for vector fields, as displayed in Figure 2.5b.

A series of related works revolve around the comparison of pelvic organ segmentations. They use visualizations of the organ shapes that can be interacted with through linked

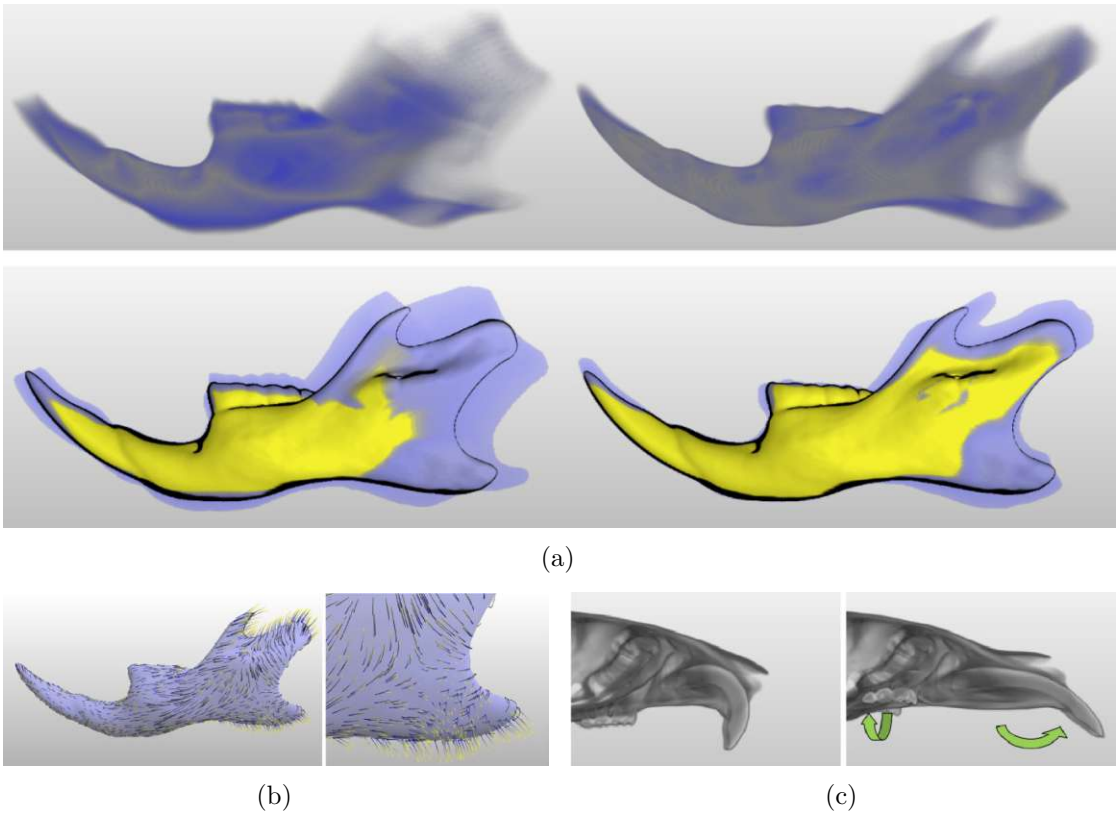


Figure 2.5: In Figure 2.5a two subgroups within a larger ensemble are shown. The different volumes are superimposed in the top row, and the likelihood volume is shown in yellow, with additional contours, in the second row. In Figure 2.5b, a streamline overlay is depicted, indicating the underlying vector field of the deformation, and Figure 2.5c shows an example for the interactive reformation of a volume.

auxiliary views. Pelvic organs, such as the bladder, exhibit large changes in shape throughout the treatment period. In prostate cancer radiotherapy, if anatomical variability is not taken into account, it can result in a high radiation dose to healthy tissue, causing side effects. Automatic segmentations of the pelvic organs, which are often used as input to radiotherapy planning systems, cannot robustly account for organ variability. An application that focuses on a use case close to the clinical radiotherapy process is the work of Raidou et al. [RCMA+18]. Using multiple linked views, clinical researchers can investigate individual patients over time and compare them with large cohorts, to understand which bladder shape characteristics are more prone to side effects during radiotherapy. A later extension to this approach is the work by Furmanová et al. [FMCM+21], providing means to predict the changes a patient's pelvic anatomy will exhibit during the treatment of prostate cancer. The anticipated changes are factored into the treatment plan evaluation to select an optimal strategy for the upcoming weeks.

To understand the circumstances (e.g., type of organ, specific shape, volume, etc.) under which an automatic algorithm for different pelvic organ segmentations may fail, Reiter et al. [RBGR18] developed a web-based application to extract shape variability and visualize all organs in an ensemble simultaneously. Their input data consists of meshes with per-triangle correspondences. As a distance metric, they utilize a shape descriptor based on 7th-order spherical harmonics that provides translation and rotation invariance, but not scale, as the size of the organs matters. The resulting 8D shape descriptor vector is projected to 2D for the visualization, using Principal Component Analysis (PCA) within an organ class and t-Distributed Stochastic Neighbor Embedding (t-SNE) [VdMH08] for multiple organ classes, as illustrated in Figure 2.6. Users can select one or multiple organ shapes in the PCA, t-SNE, or a corresponding Parallel Coordinates Plot, and the linked object viewer will display the mean shape of the organ with an explicit encoding of various attributes.

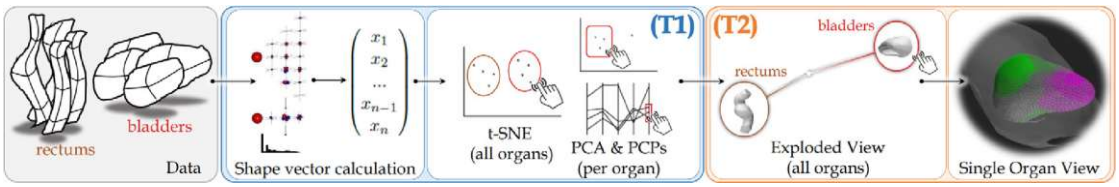


Figure 2.6: The workflow proposed by Reiter et al. [RBGR18]. Using triangular meshes of different pelvic organs as input data, shape descriptors are calculated for each one. An embedding of the shapes, computed with t-SNE, is used as an overview of the different organ classes. Additional views encode the same shapes using PCA and a Parallel Coordinates Plot populated with the shape descriptor vectors. In the illustration, these views are marked as task (T1), and they address the quantification and visualization of one or more pelvic organs. An additional exploded view shows various selected organs and their respective organ classes. A single organ view shows the mean shape for a class and visual encodings for the characteristics of different selected organs of that class. These views are marked as task (T2) and support the comparative visualization of several pelvic organ segmentations of a cohort of patients.

In the context of Computational Anatomy and Statistical Shape Modeling (SSM), shapes can be represented by particles, stemming from discrete samples of the underlying set of surfaces or volumes, as used in the application of Cates et al.[CEW17] called ShapeWorks. The core functionality of ShapeWorks involves finding the sample points that best capture the essence of an ensemble of anatomical shapes, but provide a description that is as simple as possible. The landmarks are automatically detected for an ensemble of binary volumes that represent 3D surfaces, which allows for an ensemble-specific metric that is less influenced by the individual variabilities each member possesses. ShapeWorks is utilized in a series of publications, for example, Orkild et al. [OZI+22] use it to find the main representative shape for the right ventricle of the heart within groups of patients having different conditions. Their goal was to discover significant group differences and classify characteristic shape changes that follow the tricuspid regurgitation (TR) disorder

of the heart. In their study featuring a cohort of patients with varying conditions of the heart, they observed that the variation that the shapes undergo resides within a linear spectrum. This means that the surface shape of the right ventricle can be used as an indication of the severity of a given heart condition in the future.

2.1.2 Point Clouds

Only a few approaches address comparative visualization pipelines solely for point clouds, although the data modality itself is commonly used to collect surveys of landscapes over time. The landscape scenery that such temporal point clouds depict behaves rather rigidly over time. Therefore, localized difference indicators are more of an interest than a scalar describing how much a scene has changed globally. Commonly used comparative visualization tools for point clouds revolve around different techniques that are often built on established metrics and explicit encoding of distances as colors. These are usually referred to as change detection methods. A free and open-source tool that comes with a wide range of functionality is CloudCompare [GM24]. An example of a cloud-to-cloud comparison in this software can be seen in Figure 2.7. Similar to some of the distance metrics described in Subsection 3.1.2, a point-wise distance to each nearest neighbor is computed for the cloud pair, where one point cloud is used as reference and the other as the target. The Euclidean distance is then mapped to color values and displayed for each point. Since the pointwise nearest neighbors between the two point clouds are, in general, not determined as the actual nearest neighbor of the sampled surface, CloudCompare supports *local modeling*, where the local surface is approximated to deliver more accurate distances.

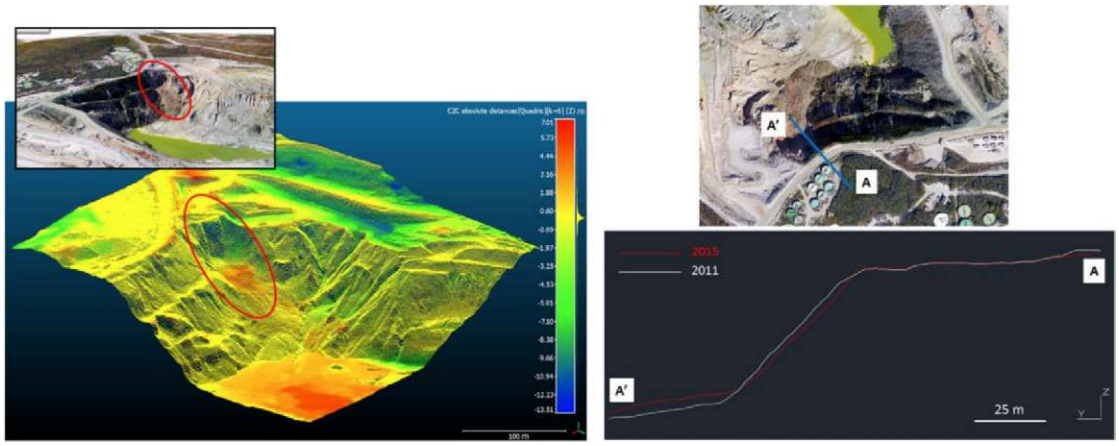


Figure 2.7: In the work of Vanneschi et al. [VEFC17], CloudCompare is used to monitor slope instability.

A survey by Kharroubi et al. [KPB+22] investigated a larger body of work dedicated to change detection using 3D point clouds, published from 2004 to 2022. In their work, they differentiate between two main categories for change detection. *Fast change*, where

features of the data are tracked over time to compute a displacement field, and *slow change*, where the global distance between two point clouds is computed. The presented approaches are further subdivided into *point-based*, which work directly with the data, *object-based*, by applying semantic segmentation first, and *voxel-based* comparison by subdividing the point cloud into a regular grid.

Schmidt et al. [SPA+14] developed an approach that does not directly utilize point clouds, but first reconstructs a polygonal mesh from the raw data and compares the differences that the reconstruction algorithm of choice incorporates. They focus on providing tools to compare whole ensembles of different reconstruction results for the same point cloud. Using a reference mesh, they compute the vertex-wise distance to all other surfaces and automatically extract high-variance areas. These areas are then shown as n-dimensional features in a parallel coordinates plot. Next to the plot, an overview renders the reference mesh, which can show different variance maps encoded in the texture.

To highlight the most relevant geometric differences in a 3D visualization of a time-varying scene, Palma et al. [PSCC18] developed an approach that enhances the perception of change in certain areas and hides others that are deemed irrelevant. As a starting point, they take two different 3D models of the same environment as input and construct colored triangular meshes. Furthermore, they precompute a probability map for the likelihood of change to take place in each area of the model. This map is used to remove variability that results from noise during the acquisition process. Taking inspiration from perception studies on the Change Blindness phenomenon [SFR00], they propose using different speed curves for different regions of the model. This means that the time-based interpolation values are transformed through different easing functions, leading to either slower or more rapid changes during the animation. This can be utilized in semiautomatic monitoring applications.

2.1.3 Animation

For ensembles where all entities are registered, methods utilizing interchangeability apply well to the task of displaying the changes over time. Since these techniques rely on the memory of the viewers and impose more cognitive load than a side-by-side visualization, they have to be designed with caution [Mun15, KCK17, LLPY07]. Therefore, animation for comparison should be justified, and caution should be taken to not overstimulate the user by displaying too much movement at the same time. One of the simplest forms of interchangeable design in visualization is blending the opacity when changing from one data instance to the next, in an animated transition. This leads to both data instances being transparent and visible at the same time, which can be advantageous for comparison, but also leads to clutter. Visualizations that employ smooth transitions and morphing between data instances are difficult to implement but can be beneficial, compared to hard transitions in the view, helping the recipients to stay focused on emphasized regions of interest [HR07, TMB02].

In the section titled 'Eyes Beat Memory' of Tamara Munzner's book [Mun15], she states

that it is often a good idea to display data simultaneously for comparison, since it induces a much smaller cognitive load on recipients than having to remember the view that was seen before. A difference between the use of animation in the storytelling of popular movies and the animation of datasets is that within movies the attention of the viewers has been carefully directed to some region of interest within the frame, which is not necessarily the case for data visualization, where changes might occur in many parts of the view simultaneously. Simpler forms of transitions, like a jump cut, can help with detecting small changes between views, but there is evidence that more sophisticated animation techniques can be beneficial since they enable users to also track larger changes in the position and shape of the entities [TMB02], given that the amount of global change between different views is limited.

It is important to keep these guidelines in mind when making use of animation in visualization, but it should not be disregarded that the transition between states can encode a lot of information itself. An example is the work of Amirkhanov et al. [AKS+19], where the trajectories of 4D differential equations are analyzed, using smooth, animated transitions. Differential equations are often visualized by displaying 2D subprojections of higher-dimensional trajectories in juxtaposition. The approach of Manylands lets users travel across *4D HyperLand* down to *3D SpaceLand*, and finally *2D FlatLand* by animating transitions between the projections. Additionally, *Timelines* were employed in an auxiliary plot to represent each variable of the system across time, as well as visual representations of dynamic segments of the trajectories. The work was later extended by Schindler et al. [SAR23] with means to compare multiple such trajectories. They added the option to combine trajectories into surfaces and an Order Independent Transparency (OIT) blending model to deal with the resulting occlusions of heavily convoluted surfaces. The result of this work for the Goldbeter model [Gol11] describing bipolar disorder can be seen in Figure 2.8.

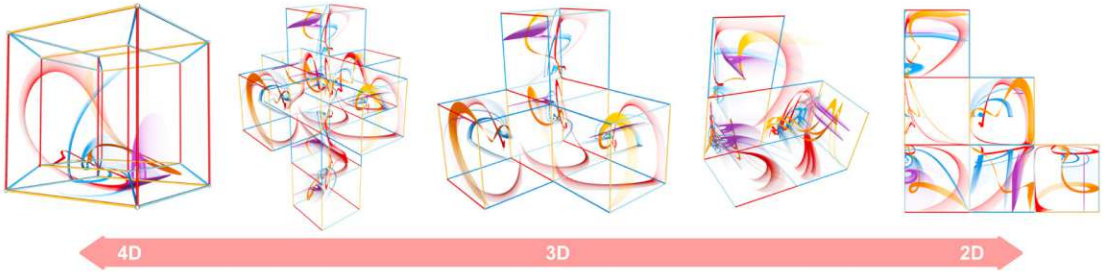


Figure 2.8: Smoke surfaces for the Goldbeter model describing bipolar disorder [Gol11]. The tesseract on the left-hand side contains four smoke surfaces indicated by four distinct colors. Each smoke surface corresponds to one seeding line that denotes multiple initial conditions. The tesseract is consecutively unfolded to its lower-dimensional subspaces, drilling down to 3D and finally unfolding to 2D.

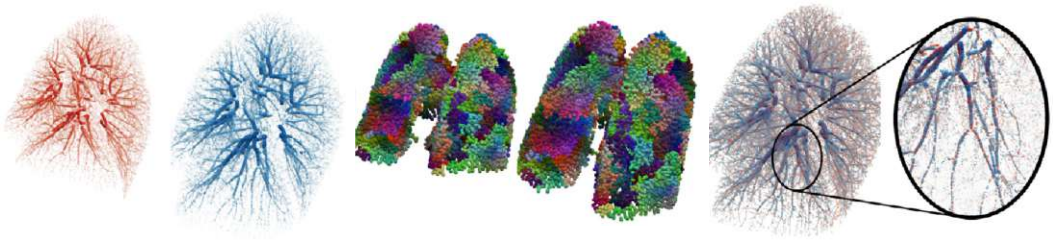
2.2 Optimal Transport for Registration

To compare multiple spatial entities, we have to ensure discrete topological correspondences between them, allowing for an explicit comparison with respect to certain characteristics of the shapes. When comparing ensembles of shapes, the analysis requires some form of affine registration first, to remove geometric differences between the individual entities, followed by a non-rigid registration to find the actual differences in the shape. A method commonly employed for the non-rigid registration of medical data is LDDMM. Established tools for finding spatial correspondences can also be found in the mathematical field of transportation theory, which studies the optimal transport of resources from one distribution to another. This is equivalent to nearest neighbor projections under a global constraint of bijectivity that enforces consistency in the matching [FCVP17]. Both methods suffer from immense computational overhead that scales poorly with the input data size. Modern approaches that benefit from massive concurrency by leveraging GPU processing capabilities or deep learning methods decrease the required run times. Still, the methods remain at a very high computational cost, preventing their use in real-time applications with large data ensembles.

Optimal transport is a powerful tool for a multitude of applications. By aligning two probability distributions and measuring the cost of transforming one into the other, it can be used to compare shapes, images, or other data instances. The transformation picked is the one that minimizes the total cost. Originating from the work of Kantorovich [Kan60], a large body of literature on the topic of optimal transport followed. His research led to the main tools of linear programming, which further led to him being awarded a Nobel Prize for his contributions. His method is regarded as a variant of the Simplex algorithm [KTT+24].

A survey by Bonneel et al. [BD23] covers the different groups of numerical methods used to solve the optimal transport problem. The first class of methods that they describe contains linear programming solvers, which allow for an exact solution of the discrete problem. For general problems with arbitrary cost functions, the network simplex algorithm is a popular and efficient method, but it can take minutes to compute a solution for relatively small problem settings. Although utilizing the hierarchical structure of problems that are embedded on a grid can lead to faster computation times by employing multiscale methods. Another popular approach is to reduce the complexity of the optimal transport problem by projecting high-dimensional distributions onto 1D subspaces and solving the simpler problem in the reduced space. Multiple such projections can be computed onto different subspaces, the optimal transport assignment is then approximated as the average over all these projections [BD23]. This method produces different results from native optimal transport, but the error bounds to the true optimal transport assignment can be computed.

A range of methods is based on the idea of adding the entropy of the transport plan as a regularization term, which leads to the so-called Sinkhorn algorithm [BD23]. The problem can then be solved using iterative matrix multiplications and element-wise



(a) (b) (c)

Figure 2.9: In Figure 2.9a, a pair of lungs captured at different states of a breathing motion are depicted. Figure 2.9b shows the deep learning features extracted from these point clouds in the work of Shen et al. [SFL+21] that are used as input for an optimal transport problem. The resulting assignment is used to aid the registration of the two point clouds as seen in Figure 2.9c.

division, which are easy to implement and can be parallelized efficiently. The price to pay for this simplicity and gained efficiency is that the entropic regularization adds an amount of blur to the transport plan. Also, using very small values for the entropy parameter leads to numerical instabilities, which can be mitigated by performing computations in the log domain, although again at the price of increased computational cost [Fey20].

In the context of his dissertation, Jean Feydy [Fey20] developed, together with colleagues, the KeOps library [CFG+21], which aims to bring graphics-like performances to the machine learning and medical imaging communities. Building on this library, they developed the GeomLoss package for Python [FSV+19], which solves optimal transport problems via a symmetric debiased multiscale Sinkhorn loop. For machine learning and shape analysis tasks, their implementation outperforms comparable algorithms by one to three orders of magnitude. It allows them to define affordable loss functions that exhibit desirable geometric properties for the processing of point clouds, random vectors, curves, or meshes.

Establishing feature correspondences between data instances is closely related to the task of shape registration, which often comprises the tasks of feature extraction, feature matching, and regularization. Shen et al.[SFL+21] focused on a technique to improve the feature matching step by building on the GeomLoss library and utilizing tools from optimal transport theory. In the process, they published a dataset of over a thousand pairs of lung vascular trees, as densely sampled point clouds, to perform the challenging task of registering pairs of respiratory lung CT scans with the proposed method. An example pair of these point clouds is depicted in Figure 2.9 together with the features used for their optimal transport problem. Their work shows that the optimal transport solvers improve the accuracy for point cloud registration with optimization-based and deep-learning methods at an affordable computational cost.

Good feature matching is also a major requirement in scene flow estimation, where the goal is to track the movement of objects in a scene. This often requires finding

correspondences between point clouds, captured in sequence by, e.g., an autonomous driving car. Building on GeomLoss, Puy et al. [PBM20] developed FLOT, an application that finds correspondences in multi-temporal point clouds utilizing tools from Optimal Transport theory. Their process is divided into the following two steps. First, soft correspondences between points of the input point clouds are extracted using a neural network and point cloud convolutions, and then the transport cost between these features is computed. Afterward, the soft correspondences are exploited to obtain a first scene flow estimate, which is refined using a residual network. Their approach delivers results as good as the state-of-the-art methods while requiring fewer parameters.

Frey et al. [FE17b, FE17a] developed a technique to generate a transformation between arbitrary volumes based on optimal transport, as depicted in Figure 2.10. Their approach does not require any additional input or user guidance, besides the volume pair, and efficiently computes pairwise assignments that are iteratively improved towards the optimum. They showcase their approach by developing a time series reduction method, which works by deriving a distance metric from the transformation that can be used to extract representative time-steps within a series. In a second step, they morph between selected time-steps to approximate the full data series. Another example application is the ordering of a data ensemble, based on similarity, by first computing the pairwise distance between all ensemble members to populate a similarity matrix, and then solving a traveling salesmen-like problem using that matrix. This yields a sequence in which each transition is comparably smooth, although the rotation of the data has a large impact on the final sorting in their experiment.

Figure 2.10: A depiction of the volumetric transformation from a Chameleon dataset into a Zeiss engine from the work of Frey et al. [FE17b].

2.3 Summary

The research areas related to this work have been extensively explored over the years, resulting in a substantial body of literature. An established taxonomy for comparative visualization in general, but also specifically for the analysis of shapes ease the continuation in this research direction. Nevertheless, we still see open challenges when it comes to the study of ensembles of large point clouds, stemming from reconstructions of real-world objects or landscapes.

Existing approaches for the general comparison of shapes rarely use the raw representation of point clouds to facilitate the analysis of ensembles. More often, triangular meshes are derived, which requires a costly pre-processing step for large data. Many of the reviewed approaches originate from the biological or medical domain and work with volumetric data, which imposes different requirements on these methods. We focus on point clouds that are directly derived from photogrammetry methods, and especially large datasets of landscapes are collected using this modality.

Approaches that facilitate the comparison of spatial data do so by focusing on the visualization of pairwise differences, or only very few data instances, given the scale of contemporary datasets. We attempt to visualize ensembles consisting of multiple point clouds as directly as possible without resorting to a smaller view space for each point cloud by using an interchangeable design and animating the transition between ensemble members. Taking inspiration from many of the mentioned publications, we also implement explicit encoding of derived metrics in the colors of the visualization, which helps to guide the focus during the progression of the animation.

The non-linear registration is often left out of the main processing pipeline and done as a pre-processing step. We focus on the performance of this step to approach the direction of minimal latency for the comparison of larger spatial data. We extend a state-of-the-art solver for optimal transport problems to achieve even faster computation of the non-linear registration. This can significantly speed up workflows for the comparison of point cloud ensembles. This approach also supplies us with a useful distance metric for point clouds, which we utilize to design the embedding of our shape space as simply and intuitively as possible. While the broad range of existing approaches for comparing spatial data offers many fundamental concepts to build upon, the specific requirements of this work make it necessary to develop a novel approach, extending on what has been done before.

CHAPTER 3

Background

3.1 Point Clouds

A point cloud is a set of points in $\mathbb{R}^D$. Since a lot of the data processed as a point cloud depicts true or synthetic spatial environments, commonly we assume that $D = 3$ and each point is represented by its coordinates (x, y, z). Due to their simplicity, point clouds are a common data structure in computer graphics and computer vision. Compared to volumetric data, which is defined on a regular grid, point clouds are irregularly sampled and can be seen as a sparse representation of surfaces.

The points in a point cloud can be generated using a variety of methods, such as 3D scanning, photogrammetry, or computer-generated graphics. The points can further be augmented with additional information such as weights, color, normal vectors, and other additional attributes, making them a versatile data structure for a variety of applications. Especially in the context of visualization, point clouds can be presented using a range of different rendering techniques from simple point rendering to specialized methods like Gaussian Splatting [KKLD23].

3.1.1 Spatial Data Structures

With point clouds having sizes upwards of a billion points nowadays, efficient data structures to store and process the data are required. Computations that involve neighborhood information can be very expensive to calculate, even for smaller models. Data structures that induce a spatial hierarchy can be used to speed up these computations.

Regular grids

Regular grids are the simplest data structure to store point clouds and allow for faster coordinate-based queries. Points can be assigned to bins with uniform extent over the

whole grid, as depicted on the left of Figure 3.1. Such an assignment can be computed very efficiently on the Graphics Processing Unit (GPU) using modern libraries [Fey20]. However, simple regular grids are not well suited for sparse data and can lead to high memory consumption for large point clouds.

KD-Trees

A k-dimensional tree [Ben75] is a binary space partitioning data structure that organizes points in a k-dimensional space by splitting the space into two half-spaces at each inserted point that is not a leaf. They are particularly useful for very fast nearest-neighbor and range searches. A schematic depiction can be seen in Figure 3.1 on the left.

Octrees

An octree is also a space partitioning data structure, but each non-leaf node has exactly eight children. Octrees are often utilized as Level of Detail (LOD) structures within modern point cloud viewers. The combination with a layered point cloud, where we also save points at the inner nodes of the data structure and not solely in the leaves, is a de facto standard in LOD for point cloud rendering engines [SOW20]. This allows for efficient rendering of point clouds with sizes well in the billions when combined with out-of-core techniques, where only a small part of the whole point cloud needs to be loaded to the Random-Access Memory (RAM) or GPU memory of the processing machine [SOW20]. A simple illustration of an octree can be seen in the middle of Figure 3.1.

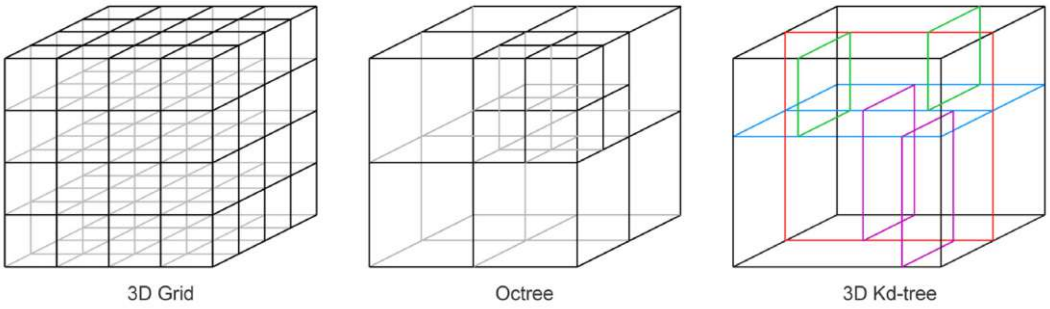


Figure 3.1: Illustrations of a simple regular grid, an octree, and a 3D kd-tree, in this order. The figure is taken from [OALRRRSS23].

3.1.2 Distance Metrics

The ability to describe the discrepancy between any two point clouds requires loss functions that define a distance between them. Desirable properties of such loss functions are positivity, definiteness, and that they satisfy the triangle inequality [Fey20]. The stability of a loss function additionally guarantees that it stays consistent under an up- and down-sampling of the point cloud. These requirements only put minor constraints

on possible loss functions, but the choice of a specific function has a strong impact on the achieved non-rigid registration results. The following examples for distances are commonly used in combination with point clouds [Fey20, GM24, KPB+22].

Chamfer Distance

The Chamfer distance [Fey20] between two point clouds X and Y is defined as the sum of the average distance between each point x in X and its nearest neighbor y in Y and vice versa. Formally defined as:

$$\text{Chamfer}(X,Y) = \frac{1}{N}\sum_{i\in N}\min_{y\in Y}\|x_i - y\|^2 + \frac{1}{M}\sum_{j\in M}\min_{x\in X}\|x - y_j\|^2$$

Due to its computational simplicity, the Chamfer distance is a popular choice for measuring the similarity between two point clouds and is often used as a loss function in machine learning tasks [FYC+22]. It is also known because of its popularity as a loss function for the well-known Iterative Closest Points (ICP) algorithm [BM92], often used in the rigid registration of point clouds.

Hausdorff Distance

The Hausdorff distance [Fey20] is defined as the maximum of the distances of each point x in X to its nearest neighbor y in Y and vice versa. It is defined as:

$$\text{Hausdorff}(X,Y) = \max\left(\max_{x\in X}\left(\min_{y\in Y}\|x-y\|^2\right), \max_{y\in Y}\left(\min_{x\in X}\|x-y\|^2\right)\right)$$

The Hausdorff distance is simple to establish and is also commonly used for comparing shapes. Unfortunately, this distance is not suitable for point-wise smooth interpolations between point clouds via gradient-based optimization, since they introduce degenerate gradient fields [Fey20], as depicted in Figure 3.2. Hausdorff-like distances are still well suited for utilization in 3D rigid registration and mesh reconstruction methods when combined with heavy regularization based on modeling priors. Like in the setting of affine shape registration, where only a low-rank affine transformation matrix is the optimization target [Fey20].

Kernel Methods

Off-grid convolutions can also be used as a way to define a distance between two point clouds by picking a symmetric kernel function $k(x, y)$ that measures the similarity of any pair of inputs $x, y \in X \times Y$. For discrete point clouds, with weights α and β and respective sizes M and N, we can use such a kernel function k to define a pairwise

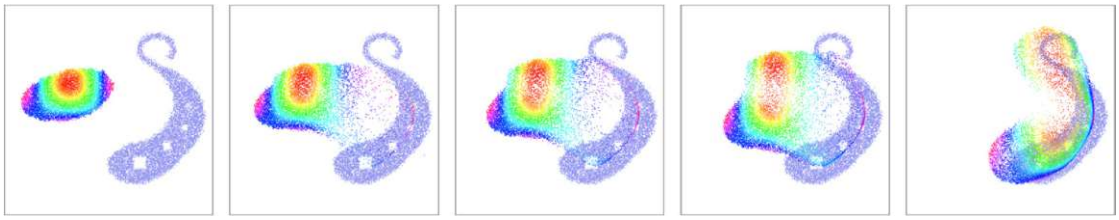


Figure 3.2: Application of a Hausdorff-distance to a toy problem from the work of Feydy et al. [Fey20]. The individual subfigures show the progression of the rainbow-colored ellipse when following a gradient descent scheme to update the positions of the individual points iteratively at the time steps 0.0, 0.25, 0.5, 1.0, and 5.0, respectively. As described in Section 3.1.2, the vector field induced by Hausdorff-like formulas cannot be relied on to produce smooth geometric interpolations.

quadratic loss function:

$$Kernel_k(\alpha,\beta) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j k(x_i,x_j) - \sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_i\beta_j k(x_i,y_j) + \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\beta_i\beta_j k(y_i,y_j)$$

This one and similar formulas find dozens of applications in physics and machine learning and are supported by an extensive body of literature [Fey20]. In the machine learning community, kernel norms are generally called Maximum Mean Discrepancies (MMD). Applying a convolution is a straightforward way to define spatial correlations, but not all kernel functions behave equally, and a combination of gradient descent and specific kernel norms can be a complete mismatch [Fey20]. An example would be the Energy Distance Kernel, which provides stable gradient vectors at low computational cost, without requiring any parameters [Fey20], but vanishing gradients lead to an extremely slow convergence of gradient descent-based schemes, as depicted in Figure 3.3. The Energy Distance Kernel is defined as:

$$k(x,y) = -\|x-y\|$$

Figure 3.3: Application of an energy distance kernel loss, as defined in Section 3.1.2, to the same toy problem from Figure 3.2 from the work of Feydy et al. [Fey20]. Although the general flow of points is monotonic and good-looking, due to vanishing gradients, some points converge extremely slowly towards their targets.

Wasserstein Distance

The Wasserstein distance is a measure of the distance between two probability distributions over a metric space [Fey20]. It is also known as the Earth Mover's Distance (EMD) since it can be seen as the minimum cost of transforming one distribution (pile of earth) into another, where the cost is defined as the amount of "mass" that needs to be moved. We can consider a point cloud as a discrete distribution with uniform weights. The Wasserstein distance is tightly connected to the theory of optimal transport, which generalizes sorting to feature spaces with a dimensionality larger than one [Fey20], which we will discuss in more detail in Section 3.2. The general definition of the Wasserstein distance between two discrete distributions is given by:

$$EMD(\alpha, \beta) = \min_{\pi \in \prod(\alpha,\beta)} \sum_{i=1}^{N} \sum_{j=1}^{M} \pi_{i,j} c\left(x_i, y_j\right) \tag{3.1}$$

where π is a matrix of size $N \times M$ with non-negative entries that are between $[0, 1]$ indicating how much the points x_i, y_j correspond. The distance function c is often picked as $\frac{1}{2}\|x - y\|^2$, leading to the so-called Wasserstein-2 distance. The equation thus reads as the weighted distance between all pairs of points, according to an optimal correspondence π. The Wasserstein distance is a powerful tool for comparing point clouds, but it is computationally expensive to compute, especially for large point clouds. The work of Feydy et al. [Fey20] uses simple regular grids to implement a multiscale solver that computes the Wasserstein distance. This hierarchical approach speeds up the computation drastically by being able to prune out pairs of clusters in the hierarchy early on, allowing the comparison of the underlying points to be omitted.

Figure 3.4: The same toy problem as in Figure 3.2 from [Fey20], when a Wasserstein-2 loss is used. Each point from the source goes towards its optimal target following a straight trajectory.

3.2 Optimal Transport (OT)

All of the previously listed distances could be used to work towards our goals of being able to compare an ensemble of point clouds. Optimal transport entails certain characteristics that can be considered advantageous when compared to Hausdorff distances and MMDs, apparent in Figure 3.2, Figure 3.3, and Figure 3.4. An extensive body of literature exists

on the topic of OT, in the context of this thesis, we mainly stuck to the explanations given in the work of Feydy et al. [Fey20] and the book from Peyré and Cuturi [PC19]. Informally, OT can be described as a generalized sorting of feature spaces with dimensions $D > 1$ [Fey20]. By leveraging the geometric structure of the data combined with gradient descent, OT can be very efficient in finding a linear assignment between pairs of point clouds, but just like sorting algorithms, this method is also highly dependent on the metric structure of the feature space. Therefore, the performance of OT can vary greatly depending on the engineered features. In the worst case, it doesn't yield more than a random sorting [Fey20]. The next section will introduce the optimal transport problem and its history, as well as computational details and incentives for why the framework of OT is attractive to employ for our use case.

3.2.1 History of Optimal Transport

The optimal transport problem was first stated by Gaspard Monge in 1781 as moving a pile of earth from one location to another with minimum effort. While Monge did not succeed in solving this problem, Leonid Kantorovich did more than 150 years later [Kan60], but computational limitations rendered the problem infeasible to compute for many years. Only recent advances and new algorithms make it possible to apply the framework of optimal transport to problems in the domains of computer graphics and vision [PC19, Fey20].

The Monge problem was formulated as an energy minimization, with the analogy of moving one pile of dirt to form another one. He considered the cost of moving the pile as the sum of the costs of moving each particle, using $c(x, y) = \|x - y\|$ as the cost of moving a particle from a location x to location y. Although the quadratic cost $c(x, y) = \|x - y\|^2$ has nicer properties for many practical applications [BD23]. The optimization of the Monge problem is performed over all possible transport maps and ensures that the mass is conserved. The transport cost $EMD(\alpha, \beta)$ gives the optimal cost of moving f towards g assuming that the total mass of f equals that of g and that these functions are never negative [BD23]. If $c(x, y) = \|x - y\|^p$ in Equation 3.1 for $p >= 1$ this yields the function $EMD_p(\alpha, \beta)$ which is called Wasserstein-p distance after Leonid Vaserštеĭn [Fey20]. The Monge problem is non-linear and not solvable in general, since it prohibits the splitting of mass.

The Kantorovich problem is a formulation of the optimal transport problem that is easier to solve and more general [BD23]. The goal is not to find a transport map that maps each particle, but a transport plan P that indicates how much mass is moved between two locations. Again an optimization is performed over all possible transport plans, with the constraints that the total mass moved from each location x corresponds to the mass $f(x)$ present at location x, that the total mass received at location y corresponds to the required mass $g(y)$, and that the mass transported and received from each location cannot be negative [BD23]. An advantage of the Kantorovich problem is that it is linear and can be solved using linear programming. This implies the existence of a dual formulation with the same optimal cost [PC19]. The dual problem can be described using

the analogy of a transport company. The company charges the consumer an amount of money for loading their truck at location x with one unit of mass and again for unloading one unit of mass off their truck at location y. The goal of the company is to maximize its profit under the constraint that the cost of loading and unloading one unit of mass is not greater than the cost $c(x, y)$ defined in the primal problem.

3.2.2 Basics of Optimal Transport

The inner workings of OT require us to take a quick detour through measure theory, which is an important subject in mathematics and therefore an extensive topic on its own. Measures provide a mathematical abstraction that encodes the notion of a weighted set [Fey20]. This can be used as a generalization of the concept of spatial distribution of mass and therefore describe geometrical attributes, like the length or the area. A measure is formally defined as a function:

$$\mu : S \subset X \to \mu(S) \in \mathbb{R} \cup \{+\infty\}$$

that assigns a non-negative mass to subsets S of X and fulfills the properties of being a σ-algebra, non-negative and additive [Fey20]. With the focus on the interface between mathematics and computer science in the context of OT, mainly discrete measures are utilized [PC19, Fey20, BD23]. Discrete or Dirac Measures are the simplest example of measures. A discrete measure α with weights $a_i \in \mathbb{R}_{>0}$ and locations $x_i \in X$ is defined as:

$$\alpha = \sum_{i=1}^{n} a_i \delta_{x_i} \tag{3.2}$$

where δ_x is the Dirac at position x, a unit of mass that is fully concentrated at the location $x \in X$ [PC19]. Simply put, such a Dirac measure in a continuous vector space can be considered as a weighted point cloud. A convenient feature of OT is that the same method works with both discrete and continuous measures.

Optimal Transport on the Line (D = 1)

A good intuition for OT can be acquired by looking at the discrete problem in dimension $D = 1$. Given Dirac measures α and β that sample univariate probability distributions with an equal number of points. So with $x_i, y_j \in \mathbb{R}^N$ and $\alpha_i = \beta_j = 1/N$ and a convex cost $c(x, y)$ we only have to find a permutation σ that assigns each point x_i to a point $y_\sigma(i)$. This corresponds to simply sorting the Dirac's on the line from left to right, associating x_k with y_k as in Figure 3.5. The essence of OT is to use this ordering of samples to pair the x_i's and the y_j's with each other. On the real line, the Wasserstein-2 loss function [Fey20] is defined as:

$$OT(\alpha, \beta) = \frac{1}{2N} \sum_{i=1}^{N} |x_i - y_{\sigma(i)}|^2 \tag{3.3}$$

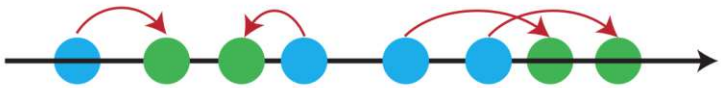


Figure 3.5: An optimal transport assignment on the real line corresponds to a permutation σ of the samples. The figure is taken from [BD23].

Optimal Transport in Higher Dimensions (D > 1)

Equation 3.3 can also be generalized to higher dimensions, where no trivial ordering is available, by modeling the sorting operation as an optimization problem on permutations of the samples [Fey20]. This follows the previously described Monge problem. The Wasserstein-2 loss between point clouds x_i and y_j in $\mathbb{R}^{(N\times D)}$ is then defined through:

$$OT(\alpha,\beta) = \min_{\sigma:[1,N]\to[1,N]} \frac{1}{2N}\sum_{i=1}^{N}\|x_i - y_\sigma(i)\|^2 \tag{3.4}$$

Kantorovich stated that the Monge assignment problem can be generalized to any pair of discrete positive measures with the same total mass [Fey20]. With the measures $\alpha = \sum_{i=1}^{N} a_i\delta_{x_i}$ and $\beta = \sum_{j=1}^{N} b_j\delta_{y_j}$ and the cost function given as $c(x,y) = \frac{1}{2}\|x-y\|^2$ we can define the optimal transport plan as the solution to the following optimization problem:

$$OT(\alpha,\beta) = \min_{(\pi_i,j)\in\mathbb{R}_{\geq 0}^{NxM}} \sum_{i=1}^{N}\sum_{j=1}^{M}\pi_{i,j}c(x_i,y_j) \tag{3.5}$$

$$\text{s.t. } \forall\ i,j,\pi_{i,j}\geq 0,\ \sum_{j=1}^{M}\pi_{i,j} = \alpha_i,\ \sum_{i=1}^{N}\pi_{i,j} = \beta_j \tag{3.6}$$

Here, we minimize over the set of non-negative transport plans $\pi_{i,j}$ whose lines sum up to α and columns sum up to β. The historical incentive behind this formulation was to describe a collection of resources α_i that are produced at factories located at x_i and need to be distributed to the cities with needs β_j at locations y_j. The matrix $c(x_i,y_j)$ describes the transportation costs per unit of mass, and the factories-by-cities transportation plan is the optimization target, as illustrated in Figure 3.6a. This definition can be directly extended to continuous measures, and can be understood as a consistent way of picking the least expensive assignment to generalize sorting to arbitrary feature spaces [Fey20].

3.2.3 Dual Problem

In the description of the Kantorovich problem in Subsection 3.2.1, the dual formulation of the linear program was mentioned using the analogy of a transport company. The

dual problem for the discrete case of the optimal transport problem can be formulated as in the work of Feydy et al. [Fey20]:

$$\max_{f_i \in \mathbb{R}^N, g_j \in \mathbb{R}^M} \sum_{i=1}^{N} f_i \alpha_i + \sum_{j=1}^{M} g_j \beta_j \text{ s.t. } \forall i, j, f_i + g_j \leq c(x_i, y_j) \tag{3.7}$$

The vectors f_i and g_j are the dual potentials, and the constraints ensure that the cost of transporting one unit of mass from x_i to y_j is not greater than the cost $c(x_i, y_j)$ defined in the primal problem. The dual problem is a maximization problem, and the optimal cost of the dual problem is equal to the optimal cost of the primal problem. The primal and the dual problem for the example of the factories-by-cities transport problem are illustrated in Figure 3.6b and Figure 3.6c. The dual problem only interacts with the sampled values through the dual potentials,

$$f_i = f(x_i) \quad \text{and} \quad g_j = g(y_j)$$

The optimal dual potentials f and g are uniquely defined up to an additive constant, so if (f, g) is an optimal solution, then $(f + k, g - k)$ is also an optimal solution for any constant $k \in \mathbb{R}$. The complementary slackness rule of linear programming also ensures that we can extend any pair of dual potentials to the whole domain $\mathbb{R}^D$ while ∇f and ∇g are well defined almost everywhere, with the following *optimality equations* [Fey20]:

$$\forall x \in \mathbb{R}^D, \ f(x) = \min[c(x, y) - g(y)] \tag{3.8}$$

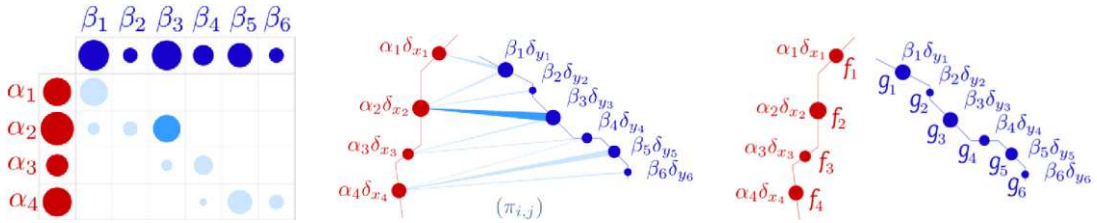
$$\forall y \in \mathbb{R}^D, \ g(y) = \min[c(x, y) - f(x)] \tag{3.9}$$

The equivalence of the primal and dual problem is known as the fundamental theorem of linear programming and led to a Nobel Prize being awarded to Leonid Kantorovich in 1975. A key feature of this duality is that the complete information about an optimal transport plan, represented as an $n \times m$ matrix, can be encapsulated within a single dual potential vector and therefore the large transportation problem can be reduced to the optimization of a single vector f_i [Fey20].

Since the individual points of a weighted point cloud can't be split up as in Figure 3.6, the *Brenier Map* is used in the work of Feydy [Fey20], which matches each source sample to a weighted barycenter of its targets. This provides a good approximation of the OT matching.

3.2.4 The Cost Function

The quadratic cost function $c(x, y) = \frac{1}{2}\|x - y\|^2$ is a common choice for the optimal transport problem, since the induced Wasserstein-2 distance has the advantageous property to faithfully generalize sorting to general features spaces and provides the most "geometric" gradients [Fey20]. Given that the Cost matrix $c(x, y)$ can be arbitrary, many other possible cost functions might be used in the optimal transport problem. Kantorovich originally studied a cost matrix that encoded the structure of the Soviet



(a) (b) (c)

Figure 3.6: Illustrations of Kantorovich's duality from [Fey20]. Figure 3.6a shows an N-by-M transport plan, e.g. factories (α) and cities (β) in the example from Section 3.2.2. Figure 3.6b shows the primal problem and corresponding cost of transport from Figure 3.6a. Figure 3.6c shows the dual formulation where, in the transport company analogy, the cost corresponds to prices for picking up mass at the factories and prices for dropping it again at the cities.

railway system [Kan60]. The members of the family of Wasserstein-p distances are the most common choice for cost functions with OT, which are defined as:

$$c(x, y) = \frac{1}{|p|}\|x - y\|^p \tag{3.10}$$

The simple case of $p = 1$ was originally used by Monge to define the earth mover's problem, but $p = -1$ or $p = 1/2$ also have their respective applications in quantum chemistry and economics [Fey20].

3.2.5 Sinkhorn Algorithm

Different techniques and algorithms have been proposed to solve the dual form of the optimal transport problem, as in Equation 3.7. Many of the proposed solvers have their origin in the field of economics and aim to find exact solutions to the linear assignment problem, but require cubic time complexity to do so. Another approach is to use the Sinkhorn algorithm as described in the work of Feydy et al. [Fey20]. The following pages are based on this work and the changes described within it to make the Sinkhorn algorithm more efficient and scalable.

A starting point for the Sinkhorn algorithm is to iterate over the two optimality equations (Equation 3.8 and Equation 3.9) and alternate between optimizing for f and g until convergence. Unfortunately, this greedy way of alternating between the two potentials generally does not converge towards a solution of the optimal transport problem, instead, it will quickly get stuck in a configuration that satisfies the optimality equations but is not an optimal solution to the primal problem. A solution closer to the true optimum can be achieved when using the *Auction algorithm* [PC19], where an additional temperature ϵ is introduced to relax the updates on f_i and g_j. This ϵ can be understood as a concave regularization of the classical dual problem. The hard constraint $f + g \leq c$ is replaced by a

soft entropic regularization penalty [PC19]. In this entropic transport plan π, points $\alpha_i \delta_{x_i}$ are send onto a fuzzy collection of targets $\beta_j \delta_{y_j}$ whose diameter is roughly proportional to a blurring scale $\sigma = \epsilon^{1/p}$ [Fey20], as depicted in Figure 3.7. Dual potentials and the entropic regularized optimal transport plan OT_ϵ are linked through the following equation:

$$\pi_{i,j} = \exp \frac{1}{\epsilon} \left[f_i + g_j - c(x_i, y_j) \right] \cdot \alpha_i \beta_j$$

that ensures that an amount of mass is only transported between source α_i and target β_j if the constraint $f + g \leq c(x_i, y_j)$ is ϵ-close to being saturated [Fey20].

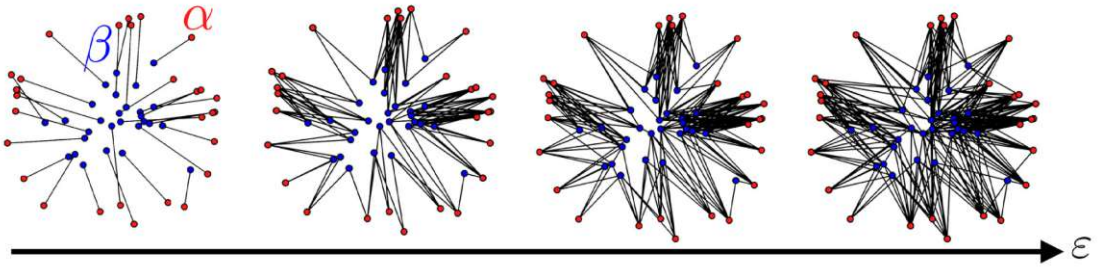


Figure 3.7: An illustration of the influence of the regularization parameter ϵ on the entropic transport plan π. Increasing ϵ leads to a blurred transport assignment between the two discrete distributions α and β that have the same number of points. The figure is taken from [PC19].

To achieve smoother updates, we can swap the min operator in the optimality equations with a *softmin* operator [PC19], which is defined as:

$$\min_\epsilon x = -\epsilon \log \sum_i e^{-x_i/\epsilon} \tag{3.11}$$

This formula converges to $\min(x)$ for any vector x when $\epsilon \to 0$, and serves as a differentiable approximation of the min function. The log-sum-exp stabilization trick can be used to avoid underflow problems for small values of ϵ [PC19]. According to that trick $\min_\epsilon x$ can be written, using $\underline{x} = \min(x)$ as:

$$\min_\epsilon x = \underline{x} - \epsilon \log \sum_i e^{-(x_i - \underline{x})/\epsilon} \tag{3.12}$$

The complete Sinkhorn algorithm is shown in Algorithm 1. The dual potentials are updated in lines 3 and 4, evaluating the softmin of the optimality equations until some tolerance is reached. This leads to smooth gradients and a stable convergence towards an ϵ-approximation of the optimal dual potentials. This evaluation only requires the computation of simple matrix-vector products, making it particularly suited to be executed on the GPU [PC19].

Algorithm 1 Sinkhorn Algorithm

$f_i, g_j \leftarrow 0^N_{\mathbb{R}}, 0^M_{\mathbb{R}}$
repeat
    $f_i \leftarrow -\epsilon \log \sum_{j=1}^{M} \beta_j \exp \frac{1}{\epsilon}[g_j - c(x_i, y_j)]$
    $g_j \leftarrow -\epsilon \log \sum_{i=1}^{N} \alpha_i \exp \frac{1}{\epsilon}[f_i - c(x_i, y_j)]$
until convergence up to a set tolerance
return f_i, g_j

Debiased Sinkhorn Divergence

When the entropic OT_ϵ is used and the cost function is defined as the Wasserstein-p distance from Equation 3.10 the results of the Sinkhorn algorithm deteriorate when $\epsilon > 0$ and the algorithm will not converge to the true optimal transport plan [Fey20]. In general, if β is a given target measure, there exists a degenerate measure α with:

$$\mathrm{OT}_\epsilon(\alpha, \beta) < \mathrm{OT}_\epsilon(\beta, \beta) \neq 0$$

Letting the temperature ϵ decrease towards 0 with the iterations can be done, but due to being slow and brittle is not a genuine solution to the problem [Fey20]. Another downside of entropy regularized distances is that the triangle inequality largely breaks [BD23].

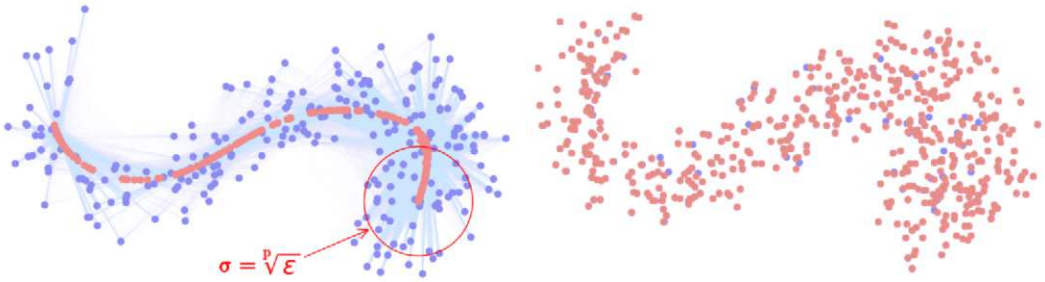
A paradigm shift in the literature on OT led to the outlook on entropic regularization changing from being conceived as a dirty hack to a feature that helps to prevent overfitting on the target sample's location [GCB+19]. To overcome the degeneracy of the Sinkhorn algorithm and receive a formula that vanishes when $\alpha = \beta$, the de-biased Sinkhorn divergence was introduced in the work of Ramdas et al. [RTC17] as:

$$\mathrm{S}_\epsilon(\alpha, \beta) = \mathrm{OT}_\epsilon(\alpha, \beta) - \frac{1}{2}\mathrm{OT}_\epsilon(\alpha, \alpha) - \frac{1}{2}\mathrm{OT}_\epsilon(\beta, \beta) \tag{3.13}$$

The incentive behind this formula is that the Sinkhorn divergence converges towards the genuine Wasserstein distance when $\epsilon \to 0$ and on the other hand, it converges towards the MMD kernel when $\epsilon \to +\infty$. The blurring parameter ϵ can be interpreted as a smooth selection possibility for the level of detail that should be preserved in the transport plan. Lowering it will result in a better approximation of the target. The Sinkhorn divergence can also be used in an unbalanced case, where the total mass of the source and target measures is not equal, by introducing a dampening coefficient [Fey20, SFV+23].

Symmetric Multiscale Sinkhorn Algorithm

The debiased Sinkhorn algorithm, depicted so far, already entails desirable properties. The work of Feydy et al. [Fey20] further improved upon that by making it symmetric, fast, scalable, and accessible to users through interpretable parameters. We now provide a summary of their extensive improvements.



(a) (b)

Figure 3.8: The difference in results between the Loss is computed using OT_ϵ in Figure 3.8a and the de-biased formulation in Figure 3.8b, showing the importance of debiasing when the blur value is larger than the average distance between samples x_i, y_j and their neighbors. From the work of Feydy et al. [Fey20].

Symmetry, where $OT(\alpha, \beta) = OT(\beta, \alpha)$, can be achieved by swapping the alternating updates of f_i and g_j to averaged updates. So that both potentials are updated at the same time by interpolating between maximization over f and g.

The convergence rate of the Sinkhorn algorithm is proportional to the ratio between the diameter of the point clouds and the blurring scale ϵ [Fey20], and can be approximated by:

$$\left(\frac{\max_{i,j} \|x_i - y_j\|}{\epsilon^{1/p}}\right)^p$$

Small values of ϵ will result in many iterations. A technique that can be employed to make the Sinkhorn algorithm faster is to utilize annealing strategies. Decreasing the temperature ϵ over the iterations will effectively decrease the step size we take each iteration, which in turn will lead to faster convergence. It also makes a lot of sense to take larger steps at the beginning of the optimization and smaller steps in the end to fine-tune the transport plan. These changes greatly enhance the accuracy of the naive Sinkhorn algorithm. However, the problems of quadratic computation times and memory requirements persist when working with large datasets with more than ~10k samples, given that the optimal transport problem takes about $O(NM)$ operations to solve, since the cost function is evaluated on all pairs of points.

Clustering of the measures allows for performing optimal transport on coarser representations at first. This makes sense because if the blurring radius $\sigma = \epsilon^{1/p}$ is large, the updates on f and g can be effectively described using coarse samples at scale σ.

Kernel Truncation can be performed subsequently on the transport plan. When (f_ϵ, g_ϵ) denotes an optimal dual pair for the coarse problem at temperature ϵ, then the influence of the measures on the transport plan is restricted to coarse points or clusters [Fey20]. This way, the so-called kernel truncation trick can be performed, where useless computations at finer scales are avoided by leveraging the information at a coarser level. Computing

point-to-point influences for the whole dataset, of which most have almost no influence on the result, can be circumvented this way.

The final symmetric de-biased multiscale Sinkhorn algorithm from [Fey20] is depicted in Algorithm 2. For the lines that correspond to steps required for debiasing, gray is used as the text color. Section 5.3 explains this Algorithm in more detail.

3.3 Prerequisites for Computational Cost Efficiency

An exhaustive comparison of two point clouds with respective sizes N and M, in its most trivial form, requires the evaluation of a full N-by-M distance matrix. Such a matrix quickly exceeds the limits of modern hardware in terms of memory requirements. Alternative representations of such a distance matrix can still allow us to process it on consumer-level hardware. Furthermore, by performing the point cloud to point cloud registration as a gradient descent problem on the coordinates, modern frameworks for efficiently computing the gradients can be leveraged. A single gradient descent step is sufficient to perfectly match the source points onto the target points [Fey20]. This yields a great increase in the computational efficiency of Algorithm 2.

3.3.1 Symbolic Tensors and KeOps

Being able to implement the features of this work is only possible due to the results of Chapters 2 and 3 of the work from Feydy et al. [Fey20]. While Chapter 3 and its main contribution, packaged conveniently for end-users within the Python package GeomLoss [FSV+19], have been mentioned before in this thesis, they could not achieve such an efficient implementation of the Sinkhorn algorithm without the underlying results of Chapter 2 [Fey20].

Summarized, it describes the utilization of fast GPU-based implementations to compute *Kernel Operations*, which are collected in a Python package named KeOps [CFG+21]. Other popular implementations for linear algebra and tensor products, like PyTorch [AYH+24], are restricted by a limitation to what full matrices can do. These matrix-vector products can be vastly outperformed by domain-specific implementations whenever the operators involved present some structure [Fey20]. A simple example for such a Kernel Operation on two point clouds X and Y with respective points x,y and sizes M,N, would be to compute the distance matrix of all pair-wise distances, simply defined as $K_{i,j} = \|x_i - y_j\|$. KeOps allows to abstract this large matrix, utilizing its mathematical structure, into two collections of vectors x and y together with the formula $k(x_i, y_j) = \|x_i - y_j\|$, which when evaluated on the two vectors produces the coefficients $K_{i,j}$. This evaluation is performed efficiently on the GPU when the coefficients are required. The representation of $K_{i,j}$ as two vectors x and y together with a formula $k(x_i, y_j)$ is referred to as a symbolic matrix.

The authors of the KeOps package compare symbolic matrices and their importance for geometric learning applications to the importance of sparse matrices for graph processing

Algorithm 2 Symmetric Debiased Multiscale Sinkhorn Algorithm

- p - Order of the cost function $c(x_i, y_j)$ 3.10
- d - Diameter
- q - Scaling Ratio
- σ - Blurring Scale
- τ - Truncation Margin
- r - Reach Scale

k, α, β, N, M ▷ Weights and coordinates of the coarsest scale k

$f_i^{\beta\to\alpha}, g_j^{\alpha\to\beta}, f_i^{\alpha\to\alpha}, g_j^{\beta\to\beta}$ ▷ Dual vectors

$M(\alpha \leftrightarrow \beta), M(\alpha \leftrightarrow \alpha), M(\beta \leftrightarrow \beta) \leftarrow 1_{R^{N\times M}}, 1_{R^{N\times N}}, 1_{R^{M\times M}}$ ▷ Masks

for all s in $\{d, (d\cdot q), (d\cdot q^2), \ldots, \sigma\}$ **do** ▷ Sinkhorn loop

  $\epsilon \leftarrow s^p$ ▷ Annealing

  $\lambda \leftarrow 1/(1+(s/r)^p)$ ▷ Dampening

  $f_i^{\beta\to\alpha} \leftarrow \frac{1}{2} f_i^{\beta\to\alpha} + \frac{1}{2}\lambda \min_{y\sim\beta,\epsilon}^{M(\alpha\leftrightarrow\beta)} \left[C\left(x_i^{(k)}, y\right) - g^{\alpha\to\beta}(y)\right]$

  $g_j^{\alpha\to\beta} \leftarrow \frac{1}{2} g_j^{\alpha\to\beta} + \frac{1}{2}\lambda \min_{x\sim\alpha,\epsilon}^{M(\alpha\leftrightarrow\beta)^T} \left[C\left(x, y_j^{(k)}\right) - f^{\beta\to\alpha}(x)\right]$

  $f_i^{\alpha\leftrightarrow\alpha} \leftarrow \frac{1}{2} f_i^{\alpha\leftrightarrow\alpha} + \frac{1}{2}\lambda \min_{x\sim\alpha,\epsilon}^{M(\alpha\leftrightarrow\alpha)} \left[C\left(x_i^{(k)}, x\right) - f^{\alpha\leftrightarrow\alpha}(x)\right]$

  $g_j^{\beta\leftrightarrow\beta} \leftarrow \frac{1}{2} g_j^{\beta\leftrightarrow\beta} + \frac{1}{2}\lambda \min_{y\sim\beta,\epsilon}^{M(\beta\leftrightarrow\beta)} \left[C\left(y, y_j^{(k)}\right) - g^{\beta\leftrightarrow\beta}(y)\right]$

  if $\epsilon < (d(1/2)^k)^p$ **then** ▷ Extrapolate to finer scale

    $M(\alpha \leftrightarrow \beta)_{i,j} \leftarrow f_i^{\beta\to\alpha} \oplus g_j^{\alpha\to\beta} \geq C\left(x_i^{(k)}, y_j^{(k)}\right) - \tau\epsilon$

    $M(\alpha \leftrightarrow \alpha)_{i,j} \leftarrow f_i^{\alpha\to\alpha} \oplus f_j^{\alpha\to\alpha} \geq C\left(x_i^{(k)}, x_j^{(k)}\right) - \tau\epsilon$

    $M(\beta \leftrightarrow \beta)_{i,j} \leftarrow g_i^{\beta\to\beta} \oplus g_j^{\beta\to\beta} \geq C\left(y_i^{(k)}, y_j^{(k)}\right) - \tau\epsilon$

    $f_i^{\beta\to\alpha} \leftarrow \lambda \min_{y\sim\beta,\epsilon}^{M(\alpha\leftrightarrow\beta)} \left[C\left(x_i^{(k+1)}, y\right) - g^{\alpha\to\beta}(y)\right]$

    $g_j^{\alpha\to\beta} \leftarrow \lambda \min_{x\sim\alpha,\epsilon}^{M(\alpha\leftrightarrow\beta)^T} \left[C\left(x, y_j^{(k+1)}\right) - f^{\beta\to\alpha}(x)\right]$

    $f_i^{\alpha\leftrightarrow\alpha} \leftarrow \lambda \min_{x\sim\alpha,\epsilon}^{M(\alpha\leftrightarrow\alpha)} \left[C\left(x_i^{(k+1)}, x\right) - f^{\alpha\leftrightarrow\alpha}(x)\right]$

    $g_j^{\beta\leftrightarrow\beta} \leftarrow \lambda \min_{y\sim\beta,\epsilon}^{M(\beta\leftrightarrow\beta)} \left[C\left(y, y_j^{(k+1)}\right) - g^{\beta\leftrightarrow\beta}(y)\right]$

    $k, \alpha, \beta, N, M \leftarrow k+1, \alpha^{k+1}, \beta^{k+1}, N^{k+1}, M^{k+1}$

  end if

end for

if r $= +\infty$ **then**

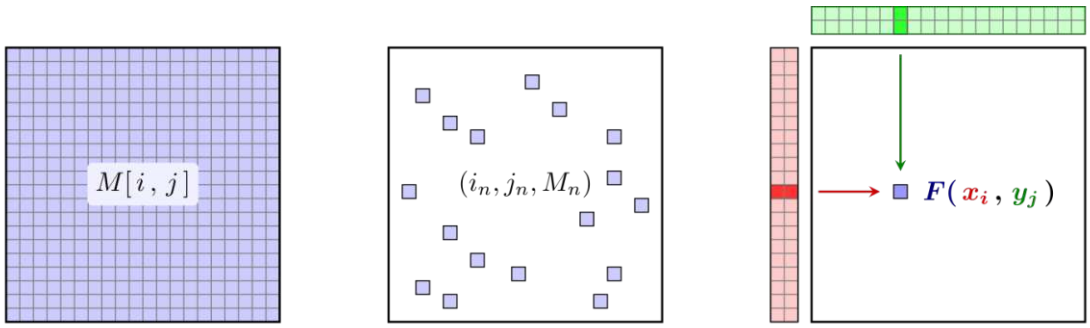
  return $f_i^{\beta\to\alpha} - f_i^{\alpha\to\alpha}, g_j^{\alpha\to\beta} - g_j^{\beta\to\beta}$ ▷ Balanced case

else

  return $\rho(e^{-f_i^{\alpha\to\alpha}/\rho} - e^{-f_i^{\beta\to\alpha}/\rho}), \rho(e^{-g_j^{\beta\to\beta}/\rho} - e^{-g_j^{\alpha\to\beta}/\rho})$ ▷ Unbalanced case

end if

[CFG+21]. A sparse matrix representation consists of a list of values M_n and a list of indices (i_n, j_n) that map the values to their entry within the matrix $M_{i,j}$. An illustration comparing regular matrices to their sparse and symbolic counterparts can be seen in Figure 3.9.



(a) Dense matrix (b) Sparse matrix (c) Symbolic matrix

Figure 3.9: Illustrations of a regular dense matrix, a sparse matrix, and a symbolic matrix, from left to right. The figure is taken from [CFG+21].

Algorithm 2 requires such Kernel Operations throughout the implementation. Therefore, depending on KeOps for the computations of large matrices results in a much more memory-efficient method. The main workhorse of KeOps is a generic reduction operator, allowing users to efficiently compute a vector-valued formula on a collection of vector sequences, alongside its derivatives concerning all variables and parameters [CFG+21]. Our goal of comparing large point clouds would not be possible using regular matrices for most computations. Using again the simple example of a distance matrix, this is easy to conclude by looking at the memory requirements of a dense matrix encoding the distance of two point clouds. A N-by-N kernel matrix stops fitting in continuous memory as soon as the size of the point clouds exceeds some memory budget-dependent threshold in the range of 10k – 40k for most contemporary consumer-grade GPUs.

3.3.2 Automatic Differentiation (AD)

Automatic differentiation (AD) is another software concept that is highly necessary for us and enables the fast computation times of GeomLoss [FSV+19]. AD refers to a family of techniques that compute derivatives through the accumulation of values during code execution to compute numerical derivative evaluations rather than derivative expressions [BPRS17]. In simple terms, this means that if we want to compute the derivative of the tensor T for some formula F, we need to save the intermediate results that are assigned to T and the respective inner operations of F. This persistence of the forward information through a program is one part of the *backpropagation* algorithm, the other part is to traverse the executed operations in reverse and add up the transition of the function values, which in its essence is an application of the chain rule [Fey20].

Due to the library PyTorch [AYH+24], using automatic differentiation in Python code

can be as simple as calling a single method, but your implementation needs to allow for that. The mathematical operations used in the program must be supported by the AD framework. The backward pass of the backpropagation algorithm usually requires run times that are approximately four to five times longer than the run time of the forward pass, which is a very time-effective way of computing the gradients [Fey20]. Additionally, the implementation in GeomLoss avoids the costly backpropagation through the whole Sinkhorn loop by utilizing the envelope theorem [FSV$^+$19]. The envelope theorem works well if users are only interested in the gradients of the Sinkhorn loss but produces incorrect results for second-order derivatives.

CHAPTER 4

Overview

The contribution of this work can be divided into two key areas. The first focuses on extending the implementation of GeomLoss [FSV+19] for faster non-rigid registration of large point clouds. The second part of our contribution is to complement this registration with a visualization pipeline that eases the comparison of many point clouds while directly displaying the underlying data. Designing a visualization pipeline that fulfills the requirements of our objective described in Section 1.2 demands multiple steps that we explain in this chapter. Additionally, we give a brief overview of the main methodology employed in our work that will be the focus in Chapter 5 and Chapter 6. Our proposed workflow from acquisition to finished animation is schematically depicted in Figure 4.1.

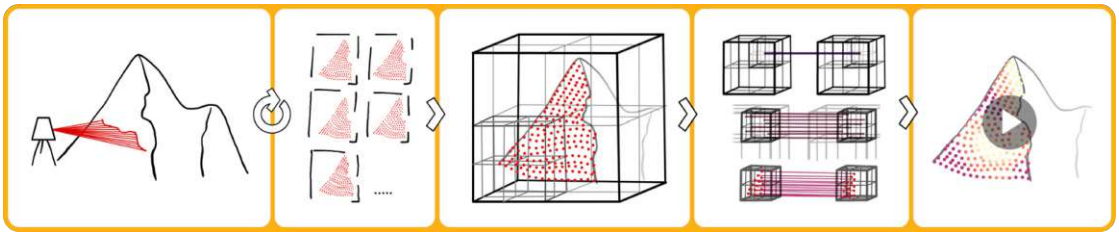


Figure 4.1: An illustration of our pipeline. The individual steps depicted represent: The repeated surveying of a mountain using a laser scanner, and the resulting timeseries for the point cloud. The octree creation from one such point cloud. The hierarchical assignment of points using optimal transport, and finally, the time series is presented in an animation highlighting areas of high variability using explicit encoding.

4.1 Requirements

For the development of our visualization pipeline, we consider and incorporate the following key requirements to ensure its effectiveness for our use cases. These requirements are established by reviewing our motivation and problem definition, and should guide us in implementing the essential elements of our comparative visualization framework. They reflect our research question that we established in the introduction, and separate it into distinct criteria that define the project scope within which we can maneuver. The intended users require insights into ensembles of digital reconstructions of spatial data. Domain experts should be able to analyze professionally collected ensembles of large point clouds and convey their findings using a captivating animated visualization. Simultaneously, the framework should be accessible to allow novice users to use the available methods with self-collected data.

Scalability (R1)

Processing a full ensemble of point clouds should be possible in our workflow, instead of the commonly employed pairwise comparison, while the application should still be capable of displaying a complex scene. Therefore, in R1, we include both scalability to points in a point cloud and the number of ensemble members we compare.

Computational Efficiency (R2)

The processing time to register the data is a strong requirement for us since the experience from the data import to the final rendering should be as seamless as possible and without unnecessary delays for the user. This would ease the workflow for expert users and research-related tasks, but also majorly improve the experience for novice users. Since modern photogrammetry solutions [SF16, SZFP16] allow for widespread access to accurate point cloud reconstruction tools from image sequences, we pay explicit attention to facilitating the workflow.

Interaction (R3)

The user should be able to explore the data effortlessly and highlight areas of significant difference effectively. The workflow should be smooth and without interruptions while the visualization is adapted to fit the requirements of the use case. Such as displaying the animation through an ensemble while simultaneously customizing the visual encoding.

Comparability (R4)

The application should enable an insightful comparison and, therefore, an intuitive interpretation of the spatial variance of the point clouds while still maintaining simplicity in the workflow. We should be able to follow intuitively how one point cloud "evolves" into the next one, without requiring expertise about the data or manual labeling.

4.2 Acquisition and Pre-processing

Different data modalities like meshes, volumes, and point clouds are commonly used to process spatial data within the visualization community. We focus on point clouds in our processing pipeline. The reason for this is that they are a simpler modality when compared to volumetric data or surface meshes. A strong benefit of relying on point clouds is that nowadays, they can be created easily without requiring any deeper knowledge of computer vision and graphics. All that is required is an image acquisition and software to solve the bundle adjustment problem [SF16]. A modern smartphone, for instance, can provide both, while some phones even include a dedicated LIDAR sensor.

As described in Chapter 2, when we have shape data that we want to compare, we would like to do so using only those characteristics that contribute to the shape itself. Like the surface of the mountain in Figure 4.1, without regard to the different positions of the laser scanner during acquisition. Similarly, if we use photogrammetry to reconstruct real-life objects, for instance, multiple statues, we want them all to be upright and face in the same direction. Therefore, we need to normalize the coordinates and remove any affine transformation that positions the data in our reference coordinate system. Which is necessary to satisfy **R1**. Terrestrial LIDAR scans are often georeferenced using information from the Global Navigation Satellite System (GNSS), making it effortless to position multiple scans from different time points in the same reference frame. This information is not provided in general, and for the scans of individual objects resulting from photogrammetry, we need to orient them manually to compare them. There is no automatic step included in our processing pipeline to remove the extrinsic pose from the data. Unfortunately, this has a big influence on the final assignment computed using OT that heavily relies on the Wasserstein-p distance from Equation 3.10.

4.3 Data Processing

After an affine registration, the point clouds need to be brought into an appropriate form as required by the interface of the multiscale Sinkhorn implementation in Algorithm 2. While in general, just the 3D coordinates of our point clouds would already be sufficient as input for the algorithm, the processing time greatly decreases when an additional clustering of the data is provided, as described in Section 3.2.5, which we prioritize according to **R2**. In our approach, we decide to utilize octrees, described in Section 3.1.1, to structure the data. The hierarchically arranged nodes of the tree can be directly employed as clusters. A custom octree is implemented, as described in Section 5.1, which we convert to the data format required by our augmented multiscale Sinkhorn implementation based on Geomloss [Fey20]. The next step in our pipeline is to perform the registration of all data ensemble members. Having extracted the hierarchical clusters, the computational performance is greatly increased, satisfying **R2**. Finally, statistical information can be extracted and inserted together with the registered models in the proper data structures for rendering. Theoretically, the clustering step could be neglected within our pipeline, and the solving of the optimal transport problem could still be

performed as a computationally cost-intensive pre-processing step, but given **R3**, we aim to implement the workflow as seamlessly as possible.

4.4 Exploration and Rendering

After all the imported point clouds within an ensemble are registered to one another, we can display the final point set being used for rendering in the custom viewer, where users can transform it to explore the ensemble. A range of different tools can be used to highlight areas of significant difference or shape characteristics. The graphical user interface (GUI) allows for an intuitive interaction with these tools as well as with the linear shape space derived from the OT solution, which contributes towards our requirements **R3** and **R4**

CHAPTER 5

Multiscale Sinkhorn Algorithm

We previously described a version of the Sinkhorn algorithm and laid the foundation for the following extension, which we add as one of the main contributions of this work. The described steps of Algorithm 2, from the work of Feydy et al. [Fey20], already add various modifications compared to the naive form in Algorithm 1, making it faster and more stable. Entropic regularization described in Subsection 3.2.5 makes the algorithm less greedy and allows for a convergence toward an optimal assignment while reducing the run-time and accuracy with respect to a tunable parameter. A softmin function, as described in Subsection 3.2.5, is used instead of a regular min function, allowing for smooth gradients in the optimization problem. The de-biased Sinkhorn divergence [GCB+19] is used to remove the entropic bias when the source measure α converges towards the target measure β, and symmetrized updates of the dual potential are employed [FSV+19]. An optimized annealing strategy leads to a fast convergence of the algorithm with a single gradient descent step required at the end, to compute the final assigned positions for the weighted Dirac masses of a source measure α. A further speed-up of the algorithm is achieved by using a two-scale approach that allows for the truncation of many non-essential comparisons when extrapolating from a coarse cluster scale to the individual points [Fey20].

These endeavors allow for the non-rigid registration of an ensemble of point clouds while the algorithm can be adjusted for either higher accuracy or faster computation. The output of the algorithm implemented in GeomLoss [FSV+19] for a pair of point clouds is a single Loss value that represents the Wasserstein-2 distance between them. The gradients of the individual coordinates with respect to the Wasserstein-2 distance indicate the linear trajectory that each point moves when morphing from the source point cloud to the target [Fey20]. An example of the morphing for three different point clouds of reconstructed lion statues can be seen in Figure 5.1. Our extension to GeomLoss enables faster run times for larger datasets. In our comparative visualization pipeline, we require a reference point cloud as a basis for the comparison that can be selected manually. This

(a) 0.0 (b) 0.25 (c) 0.5 (d) 0.75 (e) 1.0

Figure 5.1: Five frames of a morphing animation between three lion statues. The caption of each subfigure indicates the relative progression of the transformation from the first point cloud to the last one. The respective numbers of points for the three point clouds are 295774, 318401, and 260019.

reference point cloud is then registered to all other ensemble members. In the example of Figure 5.1, we morph through the three lions depicted in Figure 5.1a, Figure 5.1c, and Figure 5.1e in an order dictated by the Wasserstein-2 distance to the lion in Figure 5.1a. The influence of the parameter choice on the obtained results later in this chapter is considered at the time of Figure 5.1c. The data for the figures in this chapter are point clouds depicting stone statues of lions, which originate from Vienna and Lower Austria. The original statues have been reconstructed using photogrammetry by H. Wraunek [Wra25].

A large contribution from the work of Feydy et al. [Fey20] to the Sinkhorn algorithm is the implementation of a GPU-friendly multiscale method, where many of the point-wise comparisons, necessary with the regular single-scale methods, can be omitted. This enables optimal correspondence computations between point clouds with up to a million points within seconds. However, their implementation uses strictly a two-scale approach, one layer where the points of a point cloud are distributed into the bins of a regular grid, and then another layer of the actual points. We want to support even larger datasets with this fast correspondence computation, but as soon as we exceed this threshold of around a million points, the run time of the algorithm increases rapidly. To counteract this effect, we implement a true multiscale approach on top of the work done by Feydy et al. [Fey20], using a hierarchical data structure as the starting point for the computations.

5.1 Structuring the Data

Our contribution to the implementation of the Sinkhorn algorithm, for the most part, takes place before the actual loop that solves the optimal transport problem. We implement a simple octree data structure to be utilized as input for the Sinkhorn algorithm. The full set of points is used for the initialization of the octree so that the root node has the same extent as the point cloud. After this initialization, starting from the root, the

data is repeatedly partitioned into the eight child segments of the current tree node until we reach a node where the remaining number of points is less than some threshold, given as the Maximum Leaf Size parameter described in Subsection 5.2.6, which is then inserted into the tree as a leaf, stopping the recursive subdivision of the current branch. This threshold has to be chosen according to the size of the point cloud since the kernel truncation step described in Section 3.2.5 imposes a hard limit on the maximum number of leaves that the tree can hold. The number of points that are within the bounds of each node is computed using a bounding box query on the remaining points. An example of an octree built over the point cloud of the lion statue in Figure 5.1c can be seen in Figure 5.2.

The implementation of the kernel truncation requires, at one step, the full dense cost matrix of the current level in the hierarchy, which therefore needs to fit into the GPU memory. For the machine used for the implementation and testing in the context of this thesis, described in detail in Section 8.2, about 30% of the GPU memory is reserved for other data during the execution of the kernel truncation. The required memory for the dense cost matrix is equal to the product of the number of leaves M and N of the two currently processed octrees times 32. This results in the following condition for the number of leaves in the two octrees during testing: GB-RAM $\times 10^6 \times 0.7 \geq N \times M \times 32$. Including a limit on the tree depth to comply with this condition would be sensible. However, this is not currently enforced as it imposes constraints that we like to omit for our evaluation of the approach, so that this threshold could be exceeded for large point clouds and deep octrees. A deeper hierarchy allows for a more excessive truncation of necessary comparisons and a fast computation of the final assignment of the points, but exceeding the memory limit is computationally not feasible. This memory constraint implies that the octree has to be built according to the sizes of the point clouds and the memory at hand. Therefore, larger point clouds are required to also have larger leaves, resulting in an increased demand for computational time compared to deeper octrees, because the final assignment between the points of larger leaf nodes is more demanding, as we will see in Section 8.2.

After the initialization, the tree is traversed in a bottom-up fashion, starting from the leaf nodes. An illustration of this process is depicted in Figure 5.3. During the traversal of the tree, information on the hierarchy, metadata, points, and colors is collected. The hierarchy of the tree is a map from each non-leaf node to its eight children. The metadata contains the point count, the level in the hierarchy, bounds, center of mass, and the continuous range in the point data that belongs to the node, as well as the reference to itself in the hierarchy. This data can also be exported as a binary or text file.

The implementation of the multiscale Sinkhorn algorithm requires the ranges of points with respect to the hierarchical clusters to be sorted in memory. Therefore, we need to do one more step of reordering the octree information, where the nodes of each layer and their respective range of points are collected, to be able to forward it to the Sinkhorn loop starting in line 3 of Algorithm 2. To be precise, only the center of mass is used for each node as a coordinate for the discrete measure α defined in Equation 3.2 in the optimal

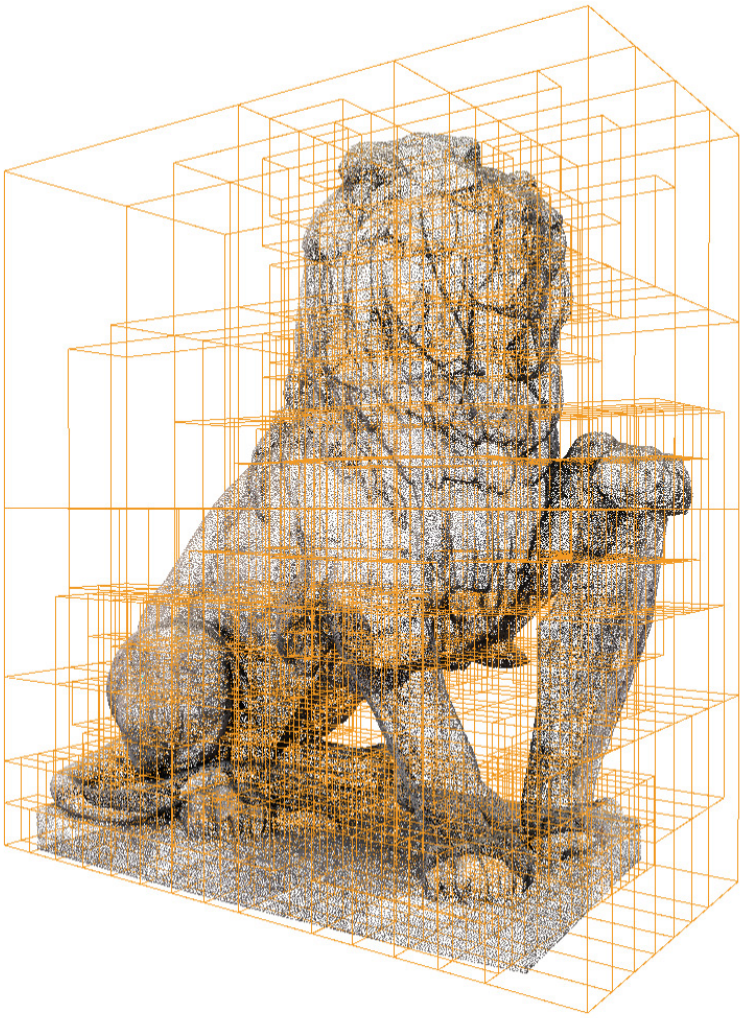


Figure 5.2: Rendering of an octree that is depicted as the orange partitions. The octree is built using the point cloud of a lion statue with 318401 points, where the maximum number of points in a leaf is 4096. We select a relatively large threshold for the maximum number of points in a leaf node for this rather small point cloud so that the resulting tree hierarchy is shallow and the rendering is less cluttered.

transport problem. The weight of the measure corresponds to the number of points within the node. The full vector of weights is normalized to sum to 1. The individual points in the point cloud are assigned a uniform weight, where the total weight for all points also sums to one.

The hierarchical layers of the octrees are later used in the Sinkhorn loop to extrapolate

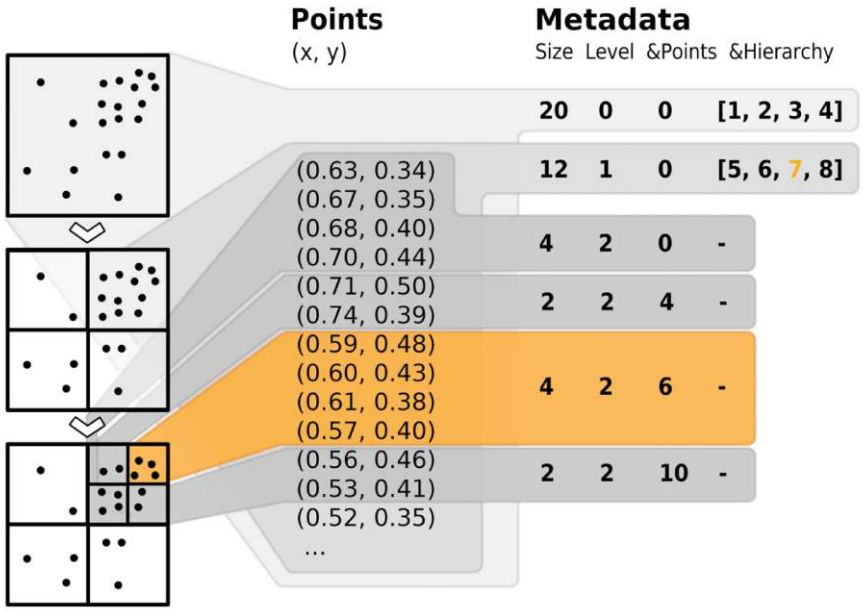


Figure 5.3: An illustration of how we build our hierarchical data structure, using 2D points and a quadtree. The first column of the figure on the left depicts the successive splitting of a quadtree. Here, the maximum number of points in a leaf is four. Next to the quadtree hierarchy, we show example coordinates of the points that belong to the children of a node in the first level of the tree. In the last column of the figure on the right, we show exemplary values for the metadata and hierarchy that are collected in a bottom-up fashion after the tree is built. In this illustration, we omit the colors as well as the values for the bounds of the node and the center of mass that are also contained in the metadata.

from a coarser to a finer scale of the discrete measures. This requires that both octrees have the same total depth, which can be easily simulated by holding back nodes or not changing the nodes at all when processing each octree layer in the reordering step. Branches that do not reach the full depth of the octree are also kept constant for successive iterations. For all other nodes, successive lists of values indicate which range of centers of mass, or actual points, belong to a node in the current layer of the octree. These centers of mass, together with individual points, make up the discrete measures that are used to calculate correspondences when solving the optimal transport problem. The cost matrices are also decomposed into this coarse-to-fine structure.

5.2 Parameters

GeomLoss introduces many tunable parameters that greatly influence the achieved registration results. In Algorithm 2, we see that six parameters are defined, which we will discuss now. We added one parameter that is relevant for the pre-processing

and steers the depth of the octree and thus also strongly influences the run time of the algorithm. The first parameter from Geomloss is the value p of the Wasserstein-p distance in Equation 3.10, which was chosen as 2 throughout this thesis to yield the popular and robust Wasserstein-2 distance. During the Sinkhorn loop, we iterate over a precomputable list of epsilon values, as can be seen in line 4 of Algorithm 2. The first and last values in the list are directly derived from the given parameters. The first value is $diameter^p$, the last value is $blur^p$ [FSV+19]. The remainder of the epsilon values in between is computed as:

1: $\epsilon_0 \leftarrow \text{diameter}^p$
2: **repeat**
3: $\quad \epsilon_i \leftarrow \exp\left(p \ln(\text{diameter}) + i \cdot p \ln(\text{scaling})\right), i \in \mathbb{N}.$
4: **until** $\epsilon_i < p \ln(\text{blur})$

5.2.1 Diameter

The Diameter d, which is the starting point for the list of epsilon values that the Sinkhorn loop iterates through in line 4 of Algorithm 2, is chosen as an upper bound on the largest distance between the centers of mass within our first layer of the octree. Since we select $diameter^p$ as the first epsilon value, it directly influences the remainder of the list of epsilon values. In general, larger values will lead to a longer list and thus more iterations of the Sinkhorn loop, but the Scaling Ratio and the Blurring Scale have to be respected as well. For the results in this thesis, the Diameter is set to $d = \sqrt{3}$.

5.2.2 Scaling Ratio

The Scaling Ratio q indicates how fast the epsilon values should decrease and is recommended to be chosen somewhere between 0.5 to 0.9 [Fey20]. Choosing smaller values for the Scaling Ratio will lead to faster computational times but less accurate results, as depicted in Figure 5.4. Choosing a very small value might create too few epsilon values required to reach the last level of the octree. We do not ensure that the chosen Scaling Ratio results in enough epsilon values. Very small ratios will also result in a faulty registration, as can be seen in Figure 5.4d. An explanation for these artifacts is that the algorithm is making too large steps when decreasing the epsilon value, so that the assignment of points can not balance out between octree nodes.

5.2.3 Blurring Scale

The Blurring Scale σ indicates the final value for the entropic regularization at the last Sinkhorn iteration, as described in Subsection 3.2.5. The smaller this value, the smaller the region for a sample that is considered for the optimal assignment, but the number of epsilon values and the iterations of the loop increase, as can be seen in Figure 5.5. Therefore, the influence of this parameter on the algorithm is similar to the Scaling Ratio, where larger values require more computational time, but also give more accurate results.

(a) q=0.9
t=3.595
iter=65

(b) q=0.7
t=1.886
iter=21

(c) q=0.5
t=1.406
iter=12

(d) q=0.3
t=0.199
iter=8

Figure 5.4: The assignment from the point cloud in Figure 5.1a to the one in Figure 5.1c using different values for the Scaling Ratio. The Blurring Scale is kept constant at 0.001, and the Reach Scale at 1.0. The caption below each subfigure shows, respectively, the used Scaling Ratio, the computational time for the registration in seconds, and the number of iterations of the Sinkhorn loop.

The Sinkhorn Algorithm 2 is practically blind to details in the target distribution that are smaller than this Blurring Scale [Fey20]. As we increase the Blurring Scale from left to right by a factor of 10 with each subfigure in Figure 5.5, fewer details of the statue are correctly preserved.

5.2.4 Truncation Margin

The Truncation Margin influences the threshold for correspondences between our octrees, which we would like to disregard at finer scales. The Truncation Margin τ steers the effective influence of the neighborhood around a sample as a multiple of the current epsilon value. If the sum of the dual potentials f_{k_i} and g_{k_j} is smaller than $c_{k_{i,j}} - \tau * \epsilon$, the correspondence between the nodes or points i and j is disregarded. A default value to use would be $\tau = 5$. A smaller value will remove more comparisons, resulting in a faster computation. Setting this value aggressively low can lead to faulty gradients and invalid states of the program, where all correspondences are truncated. One could perform sanity checks on the truncation scheme to ensure that such an invalid program state is not entered, but this would introduce a significant overhead to the algorithm [Fey20, Sch19].

5.2.5 Reach Scale

The Reach Scale r can be used to soften the marginal constraints of the transport plan π that ensure the total mass of the transport plan matches the original distributions. This allows for unbalanced optimal transport [SFV+23]. The value for the dampening is

(a) σ=0.0001 t=2.707 iter=27 (b) σ=0.001 t=1.893 iter=21 (c) σ=0.01 t=0.972 iter=14 (d) σ=0.1 t=2.295 iter=8

Figure 5.5: The same assignment as in Figure 5.4 when we only vary the Blurring Scale instead. The Scaling Ratio is kept constant at 0.7, and the Reach Scale at 1.0. The caption below each subfigure shows, respectively, the used Blurring Scale, the computational time for the registration in seconds, and the number of iterations of the Sinkhorn loop.

computed at each iteration of the Sinkhorn loop as $1/(1 + (\epsilon/r^p))$. A value of $r = +\infty$ is equivalent to a balanced optimal transport problem. For the examples in this chapter, we use the value of $r = 1.0$. Larger values could be chosen as well, but smaller values will quickly lead to degenerate results, as depicted in Figure 5.6.

5.2.6 Maximum Leaf Size

We add a parameter for the maximum size of a leaf to the implementation that defines the threshold for further subdivisions of an octree node. If the point count within the node is higher than this threshold, we split the node and create a knot. If it is below the threshold, we convert it into a leaf, as described in Section 5.1. This implicitly defines the range for the value k in Algorithm 2. For the examples in this chapter, we used a value of 500 points as a maximum within each leaf. For larger point clouds, this value has to be increased accordingly.

5.3 Sinkhorn Loop

At the core of GeomLoss is the implementation of a Sinkhorn loop from Algorithm 2. The parts of the algorithm that are colored in Grey show the variables and steps that are necessary for the debiasing of the Sinkhorn loop. In line 11, our implementation augments the condition for an extrapolation from a coarse to a finer scale. We also provide the right data structures that are assigned in lines 1 and 3 to adjust the algorithm for an octree data structure. The logic of the actual Sinkhorn loop remains unchanged.

(a) r=100 t=2.028 (b) r=10 t=1.929 (c) r=1 t=1.908 (d) r=0.1 t=1.755

Figure 5.6: The same assignment as in Figure 5.4 when we only vary the Reach Scale instead. The Scaling Ratio is kept constant at 0.7, and the Blurring Scale at 0.001. The caption below each subfigure shows, respectively, the used Reach Scale and the computational time for the registration in seconds. The number of iterations of the Sinkhorn loop was 21 for all four examples.

Before we run the iterations of the loop to compute the final loss value, we need to apply a logarithmic scale to the weights for numerical stability as described for Equation 3.12. We also initialize the dual vectors using the product of the Softmin of the cost at the current level and a dampening factor based on the Reach Scale. This product is equal to 1 for balanced OT and ≤ 1 for the unbalanced case [SFV+23].

The loop itself iterates over the exponentially decreasing epsilon values. These are computed by multiplying the current value s with the Scaling Ratio at each iteration until we reach the Blurring Scale, as in line 4 of Algorithm 2. The list of epsilon values is computed beforehand using user-given parameters. In conjunction with the epsilon values, a list of jump indices is defined, each indicating an iteration where we perform an extrapolation from a coarser to a finer scale. An extrapolation is performed if the current epsilon value is smaller than the average distance between samples and their neighbors.

At the start of the loop, in line 7 of Algorithm 2, the dampening coefficient for the unbalanced optimal transport problem is updated with the new epsilon value. In our case, the input data is pre-processed to have almost a similar number of points. With an increasing discrepancy in the number of points of both point clouds in the Algorithm 2, the problem will eventually degenerate and fail at the step of the kernel truncation.

In lines 8 – 11 of the algorithm, the dual potentials $f_i^{\beta\to\alpha}$ and $g_j^{\alpha\to\beta}$ are updated symmetrically, as well as the debiasing vectors $f_i^{\alpha\to\alpha}$ and $g_j^{\beta\to\beta}$. Line 12 shows the extrapolation to a finer scale. The following lines 13 – 15 show the masking process of the kernel truncation step, removing any comparisons where the tensor sum of the dual potentials is greater than the associated cost value minus the Truncation Margin times

epsilon. The tensor sum of two vectors f, g in $\mathbb{R}^N$ is defined as $f \oplus g = f + g^T$, and in this setting corresponds to a large N-by-N array. The Truncation Margin allows the user to influence how rigorously the potential comparisons at finer scales are removed. Next, in lines 16 – 19, the dual potentials are extrapolated to the finer scale according to the next level in the hierarchies of the octrees. Line 20 shows that the measure weights and implicit cost matrices are being updated.

Finally, after the loop has terminated, with the gradient computation of PyTorch enabled, one last iteration of the previous steps is performed. Depending on whether the setting of the problem is balanced or unbalanced, different post-processing of the values takes place in lines 24 and 26, respectively. We only care for the unbalanced case where the dual variables have to be scaled correctly and refer to the work of Feydy et al. [Fey20] and Séjourné et al. [SFV+23] for details.

5.4 Post-Processing

Our final result, after the Sinkhorn loop finishes, is a single scalar loss value representing the Wasserstein-2 distance between the pair of point clouds. Using the automatic differentiation framework from PyTorch, we can get the gradients of the loss with respect to the coordinates of the target point cloud. We subtract the gradients, scaled by the weights of the source point cloud, from the coordinates of the source point cloud to yield the registered positions of these points that resemble the target point cloud. This way, all the point clouds in the ensemble are reconstructed from a single set of points of the reference point cloud used for the registration. This also means that we do not directly get a Wasserstein-2 distance for all pairs of point clouds in the ensemble. Therefore, we compute the full Wasserstein-2 distance matrix from the optimally assigned point set in a post-processing step using Equation 3.1. We do not directly yield the color values for each point from the optimal transport solution since we only utilize the coordinates for the Sinkhorn algorithm. Also, using the gradients to compute the registered point positions, we do not directly get the explicit transport plan π that contains the information which target points correspond to each source point. When we explicitly compute π, finding the right color value would be as simple as computing a weighted mean for each row in the transport plan. We obtain the colors for the points, when fully registered to the target point cloud, by performing a query on a kd-tree that we build from the target point cloud. Each registered point gets assigned the color of the closest neighbor in the kd-tree. This query can be performed very efficiently, even for large point clouds. To not unnecessarily slow down the computation when performing an assignment using suboptimal parameters that introduce larger distances between the registered point cloud and the target, we restrict the search distance for the nearest neighbor query and assign a default color value in that case.

Alternatively, one could also attempt to solve the optimal transport plan in 6D using point coordinates and colors. This way, the optimal assignment would respect the color of each point as well. For certain scenarios, this could lead to a semantically more

meaningful assignment. Although looking at the use case of the stone lions from this chapter, the patina that formed on the stone that is captured in the surface texture could influence the registration, which we do not want.

Additionally, from all states of the registered point set, we can compute fundamental statistical measures of the ensemble. After the registration finishes, we compute the sum of the Euclidean distance between all positions in the ensemble for each point and also the variance of these positions. Other statistical moments could be computed as well, but we leave this for future investigations.

CHAPTER 6

Comparative Visualization of Point Cloud Ensembles

After computing point-wise correspondences for ensembles of point clouds, as described in Chapter 5, we need visualizations tailored to our use case. In addition to rendering each point cloud on its own, we also need to display an animation that morphs each ensemble member into the next one, as if the intermediate frames of the transformation had also been captured during the data acquisition. The attention should go to the relevant areas of high variability within the ensemble. To facilitate this, we need to supplement the visualization with the explicit encoding of hot spots in the colors of the point cloud. This should be adjustable to the current needs of the analysis and offer different options to choose from for the encoding. Furthermore, we want to show contours for each point cloud to emphasize its shape. In summary, we require real-time renderings of point clouds where each point can be animated, while the visualization has to be intuitive, interactive, and needs to be tunable to the specific requirements during the comparison of point cloud ensembles.

It is important to maintain an easily manageable interface, since we want the interaction with the application to be intuitive for the users. The exploration of the shape space should be simple and responsive while enabling browsing through ensembles of large point clouds. Our requirement R1 imposes scalability in the sense that both the size of ensembles and the size of each ensemble member can be increased and are conform with the size of contemporary datasets. At the same time, the workflow should demand only minimal waiting times and not require several minutes, as specified by our requirement R2, which is mostly implemented by the multiscale Sinkhorn algorithm described in Chapter 5. Further, the exploration of data should be effortless, as stated in requirement R3. The discrete interface and the available tools within it have to provide this ease of analysis through different workflows and use cases. This is also related to an insightful comparison and analysis of the ensemble, as described by requirement R4. The visualization should

Figure 6.1: An exemplary interpolation between two stone lion statues that have been captured by H. Wraunek [Wra25] using photogrammetry. They sit respectively on the left and right side of the Austrian National Library in Vienna. The color encoding depicts the points that travel the furthest distance during the transition from the lion on the left to the one on the right, where brighter colors encode longer distances.

directly communicate the areas where the most variance can be seen, which are therefore highly relevant for a comparison.

An example visualization for the interpolation between two real-world objects, together with explicit encoding in the form of colors and contours, is depicted in Figure 6.1. The data shown in the Figures of this chapter depicts multiple statues of lions originating from Vienna and Burgenland and have been reconstructed digitally by H. Wraunek [Wra25]. This chapter is coupled to the Graphical User Interface (GUI), of which he individual elements will be discussed in detail in Section 7.5.

6.1 Linear Shape Space

To keep the interaction with the ensemble and the navigation through the shape space as simple as possible, we decide on a scalar ordering of the individual elements, utilizing the Wasserstein-2 distance. As specified in Chapter 5, the ordering is based on a reference point cloud and its Wasserstein-2 distance to all other ensemble members. The reference point cloud is not fixed and can be changed by the user after the registration has finished. The rendered point set will still originally belong to the reference point cloud that is used during registration and will have the same number of points as before. Only the time series of further states that will be chained to the current state of this point set changes. We talk about the states of a single point set here because it is important to note that the number of points that we visualize remains fixed, only the coordinates are updated to represent the individual point clouds of an ensemble, to which the reference

point set has been registered. The initial first point cloud model in the sequence is chosen as the Fréchet mean Ψ of the ensemble E, which generalizes the notion of an average to our shape space [ZP19]. It is defined as the point cloud with the minimal average Wasserstein-2 distance to all other ensemble members, as:

$$\Psi = \arg\min_{p \in E} \sum_{i=1}^{N} EMD(p, E_i)$$

Reducing the shape space to a single dimension enables users to navigate it as effortlessly as possible. The transition from one shape to the next one in the space is given by linearly interpolating between the two states of the points. The color values are also obtained by linearly interpolating between the respective colors of the points. A linear representation of the shape space allows for the utilization of popular interaction methods like a simple slider, so that users can pick a position within the shape space to display. This position is linked to either a linear interpolation between two point clouds or a discrete member of the ensemble. Each discrete ensemble member is also linked to a Wasserstein-2 distance, inducing the sequence in which members in the shape space are ordered. An exemplary screenshot of our viewer can be seen in Figure 6.2.

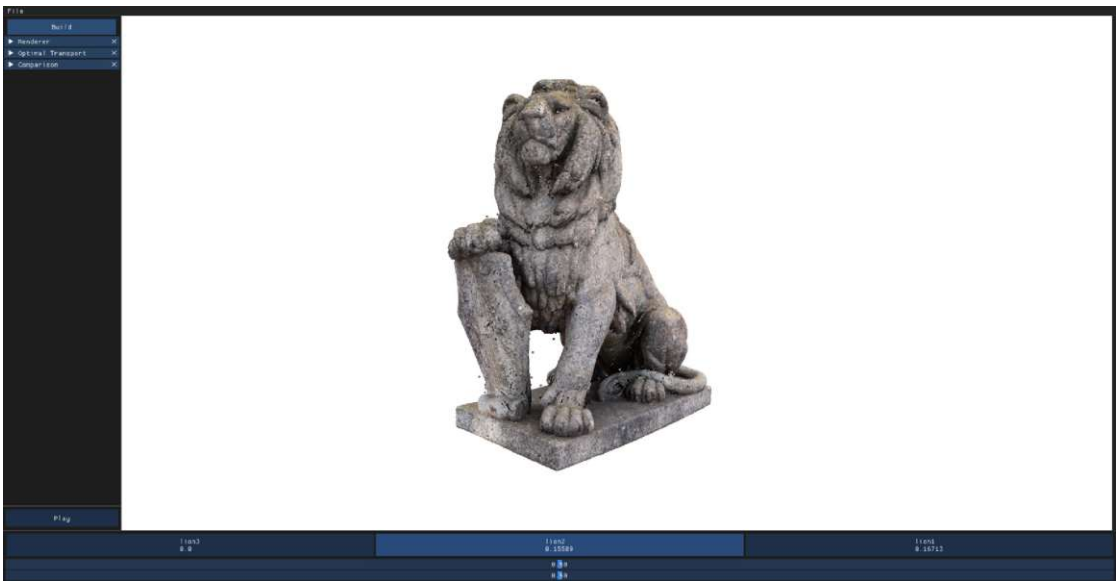


Figure 6.2: A screenshot of our application right after the Sinkhorn loop described in Chapter 5 has finished registering three point clouds depicting lion statues, of which the middle one is shown in the render window.

To facilitate the awareness of the current position within the shape space, we add visual indications within our method. The ordered shape space in our interactive visualization is encoded directly as a slider, together with labels for each ensemble member, containing a title, and the Wasserstein-2 distance to the reference member. Dragging the slider will

alter the state and result in an animation that displays the interpolation between the n states for the n loaded models. The brightness of these labels encodes the corresponding position within shape space. A regular sampling of the frames of such an animation for the example use case of eight different lion statues can be seen in Figure 6.3.

(a) 0.0 (b) 0.07 (c) 0.14 (d) 0.22 (e) 0.29

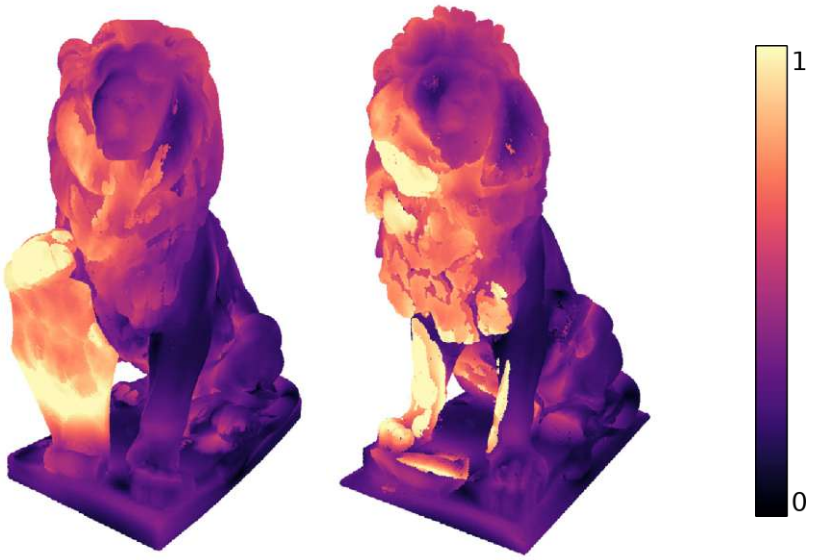
(f) 0.36 (g) 0.43 (h) 0.5 (i) 0.57 (j) 0.64

(k) 0.72 (l) 0.79 (m) 0.86 (n) 0.93 (o) 1.0

Figure 6.3: Frames from an animation through shape space at uniform sampling positions. Each second subfigure depicts an interpolation between the two lion statues next to it. The captions below each subfigure show the position in the shape space spanned by ordering all statues based on their Wasserstein-2 distance to the point cloud in Figure 6.3a.

6.2 Explicit Encoding

Following the principles of interchangeable design for comparative visualization [KCK17], we implement the traversal of shape space as an animation that morphs from one shape to the next one. Our animation conveys information about the shape variation within the ensemble, but we want to encode additional information in the texture of the rendered point cloud. This follows the principles of explicit encoding [KCK17] and guides the focus of recipients and also eases the detection of areas with high variance. Therefore, we implement measures to encode information in the texture of the point cloud and add visual indications that illustrate the shapes. So far, the color values of the point clouds encode the real-world visual impression that the sampled surface had at the time of recording, but we could also utilize the color values of the points as a way within our render window to directly encode additional information. The real-world colors are helpful to make sense of the rendered point cloud. While they often contain information about the shading, they do not directly contribute to the shape of the entity, which we are mainly interested in. Therefore, we utilize the color channel of the visualization to encode user-selectable attributes of the ensemble. This rendering of the data, together with explicitly encoded differences, gives a good intuition for the spatial variance, even when we display a static frame and not an animation, as depicted in Figure 6.4. When we use the explicit encoding of differences together with the animation, it helps to focus on regions of interest that could be overlooked with the original colors of the points.

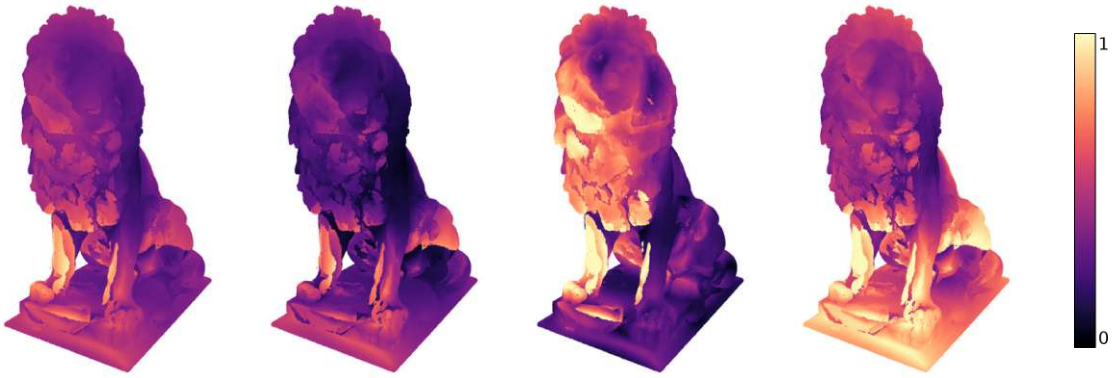


(a) (b)

Figure 6.4: An example for the explicit encoding of the per-point distance in the texture of the point cloud. Brighter values encode a larger distance that the points have to travel from the statue in Figure 6.4a to the one in Figure 6.4b. We can see that a prominent difference in the shape is the shield held by the lion statue on the left.

6.2.1 Colors

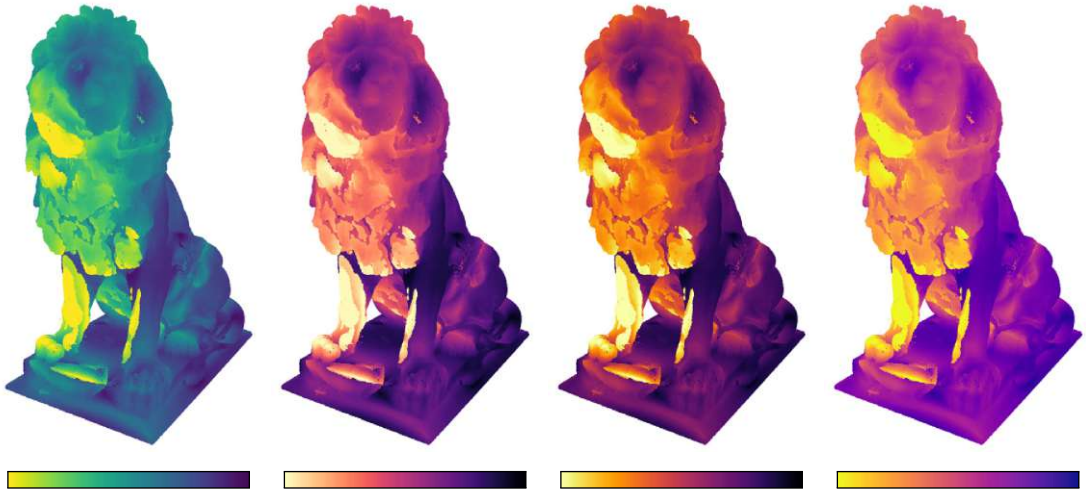
We offer four attributes that can be selected for explicit encoding in the texture of the point cloud: *Cumulative, Variance, Reference,* and *Pairwise*. All four attributes are derived from the Euclidean distance that an individual point of the moving point set travels when moving between its assigned positions in the states of shape space, which correspond to our ensemble members. *Cumulative* encodes the sum of the total distance that each point travels during the animation. *Variance* encodes the sum of positional variance for each point. *Reference* encodes the linear distance from the position within the reference model to the current position for each point. *Pairwise* encodes the linear distance from the position in the current source point cloud to the current target point cloud. Mind that the color values for *Cumulative* and *Variance* remain constant throughout the animation, whereas *Pairwise* changes with each traversed position of an ensemble member in shape space, and *Reference* changes with the current coordinates of a point. All four attributes are normalized to a range of $[0, 1]$. An example of how these different attributes look for an ensemble of three lion statues can be seen in Figure 6.5.



(a) Cumulative (b) Variance (c) Reference (d) Pairwise

(e) (f) (g) (h) (i)

Figure 6.5: Examples for the four explicit encoding attributes to choose from. The ensemble in this case consisted of the statues displayed in the second row of the figure, of which we only show the middle point cloud within shape space in the first row. Using *Reference* as an attribute gives the same result as in Figure 6.4b, just comparing the statues from Figure 6.5e and Figure 6.5g. *Pairwise* shows the distances between Figure 6.5g and Figure 6.5i.



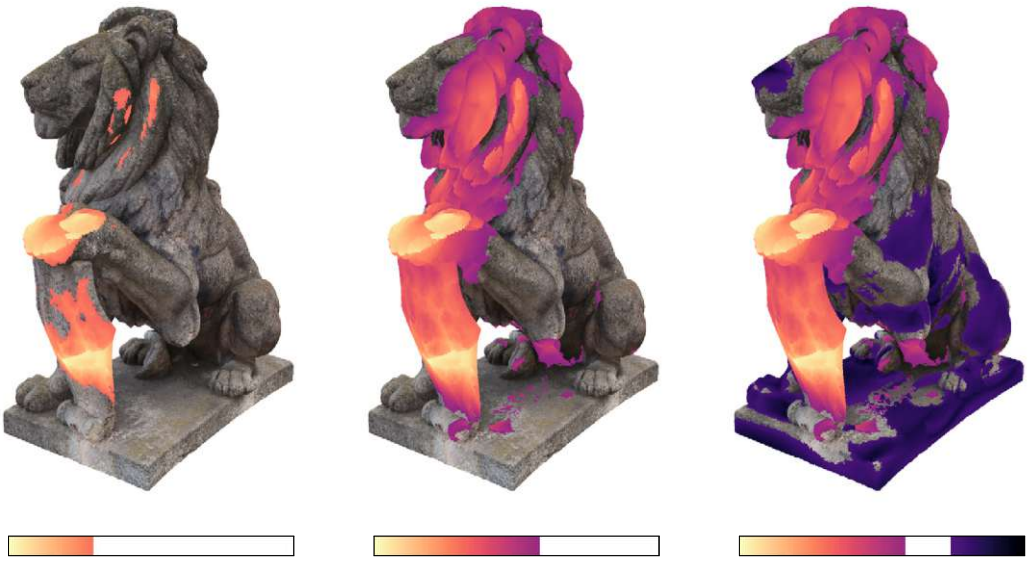
(a) Viridis (b) Magma (c) Inferno (d) Plasma

Figure 6.6: Example renderings showcasing the different colormaps that we added to the renderer, on the example from Figure 6.4b using the Reference attribute, as depicted in Figure 6.5c.

We offer the four well-known colormaps *Viridis, Plasma, Inferno*, and *Magma* to map the scalar values from the explicit encoding attribute of each point to a color value. The four colormaps displayed on a lion statue can be seen in Figure 6.6. These four colormaps are designed to be perceptually uniform so that a change from 0.1 to 0.2 has a similar perceptual effect as a change from 0.8 to 0.9. They are also sequential so that the lightness increases monotonically and are all accessible to persons with color vision deficiencies. These colormaps were originally designed to replace the old default colormap of matplotlib [Hun07], and a summary about them can be found in a blog post from the original creators [SvdW15]. Optionally, users can also choose two color values between which we simply interpolate linearly to map colors to our scalar values.

6.2.2 Filtering

It can be helpful for the analysis of differences to only display hot spots of change with different colors and otherwise leave the color of the point cloud untouched. This can draw more focus on areas with a specific variance in the ensemble while maintaining the real-life context given by the original colors. For instance, if we use a visual encoding, as depicted in Figure 6.7b, for the animation, it is very simple to follow the bright points like the shield and the paw of the lion statue. We implement a filtering functionality to enable a hybrid visualization between the explicit encoding and the real-life color values. A range to display the explicit encoding can be selected by choosing two different threshold values between 0 and 1, where the first value has to be smaller than the second one. Values outside of the range are displayed using the colormaps, and values inside use the original colors of the point cloud, as depicted in Figure 6.7.



(a) (b) (c)

Figure 6.7: Here we depict the lion statue from Figure 6.4a using different ranges to filter the explicit encoding, as indicated below each statue. For Figure 6.7a, only the upper 30% of the distance is shown. For Figure 6.7b, the upper 60% of the distance are shown, and for Figure 6.7c, the upper 60% and the lower 25% are encoded in the colors. Figure 6.7b shows where the two statues from the example in Figure 6.4 differ the most while retaining the textural features that identify the model as a stone statue of a lion.

6.2.3 Contours

Complex structures like convoluted surfaces can be difficult to see in a rendering without intricate light and shadow computations. To ease the perception of the shapes with all their overlapping structures without relying on complex shading, we added contours to the rendering. We support two different types of contours: depth contours and explicit encoding contours. Depth contours are based on the depth of the points, given as the distance to the virtual camera while rendering the frame. The explicit encoding contours are derived as isolines of the explicit encoding texture. Both types of edges are computed using 2D convolution. An example rendering of such contours alone and as an overlay for a lion statue can be seen in Figure 6.8. For each of the two types, we offer different parameters to adjust the visualization, which we will explain in detail in this section.

Depth Contours

The first type of contour is obtained by performing a 2D convolution within the depth texture. The depth texture is an attachment of the framebuffer that contains our rendered image. This depth attachment of the framebuffer holds the depth values of each pixel produced during rendering. It is used to perform depth testing, where incoming fragments

(a) (b) (c)

Figure 6.8: Example renderings of the supported contours. Figure 6.8a shows the depth contours, and Figure 6.8b the explicit encoding contours. Figure 6.8c shows the final visualization when both contour types are rendered overlaid on the explicit encoding of the cumulative distance attribute, as depicted in Figure 6.5a.

are compared to the currently held fragments and discarded if they are further away. We perform a convolution over this depth texture using a simple 3×3 Laplace filter. This filter matrix, used as a convolution kernel, computes the sum of differences of the nearest neighbor pixels and is well known in the image-processing community for its applications as an edge detection filter. Using a custom shader, we apply the convolution as a post-processing effect in the rendering pipeline. The filter is defined as:

$$H^L = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{6.1}$$

The resulting edges help with the perception of the shapes. They are especially useful when combined with the explicit encoding, since then all the cues about the shape that reside within the shading of the original colors are missing. An example of this can be seen in Figure 6.8c, where it would be more difficult to distinguish the lion's paw from the held shield without any contours.

Since we are only relying on the depth values to compute this type of contour, gaps between the points will also result in edges being drawn around these points if there is nothing immediately behind them. To give the impression of a surface without holes that allows us to compute the contours using convolution, we decide to give users the option to render the point clouds with an adaptive point size. Points that are closer to the camera are rendered with a larger size when this rendering mode is activated, as

(a) Point size: 1 - 5px (b) Point size: 2 - 10px (c) Point size: 3 - 20px

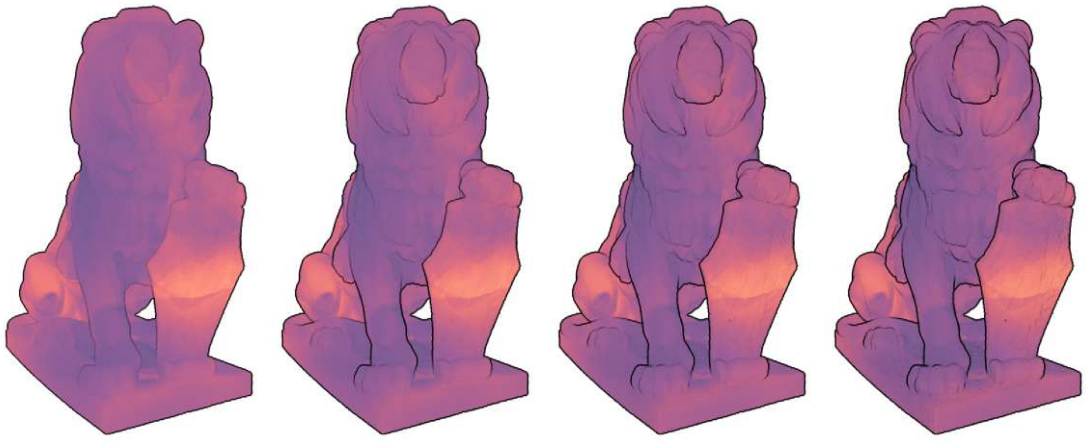
Figure 6.9: The difference between the adaptive point size and a constant rendering size is most prominent when the camera is close to the point cloud and the rendering contains a lot of depth variation. The captions below show the size range of the rendered points in pixels. In Figure 6.9a, there are a lot of gaps in the surface sampling of the lion statue, so that it is difficult to recognize the shape of the lion when we zoom in closely. For Figure 6.9b and Figure 6.9c, we use the same point cloud but render the points using larger relative sizes that decrease further away from the camera.

visible in Figure 6.9. To further improve the impression of a closed surface, we render the individual points as circles and not in the default representation of quads. Circumventing gaps between the points this way is very easy, but the point size has to be adjusted to the input data based on the point density.

Four different parameters are available to adjust the visual representation of the depth contours. The first one simply is the color of the contour, which can be selected manually, as depicted in Figure 6.12. We also provide an amplification parameter that increases the opacity of the drawn edges. For this amplification, the influence on the visualization is stronger for more subtle contours, as can be seen in Figure 6.10. We also add an option to dilate the contour using morphological operations. We implement an additional post-processing shader to perform this dilation. The results for different numbers of dilation passes can be seen in Figure 6.11. The last option is to apply a Gaussian blur to the rendering of the contour to soften the edges. We implement a separate Gaussian kernel by alternating between a 1D horizontal and a 1D vertical filter for better performance. The result for the blurred contour can be seen in Figure 6.11d.

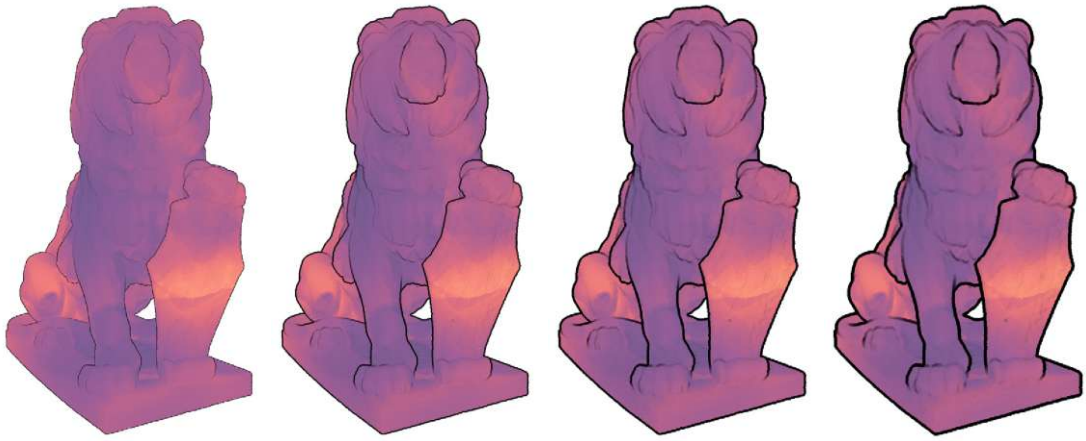
Explicit Encoding Contours

The second type of contour represents isolines of the explicit encoding textures. Similarly to the depth contours, we perform a 2D convolution on the framebuffer texture that holds the rendering of the point cloud with the explicit encoding as color values. However, the first step is to apply a Gaussian blur to the framebuffer image to reduce high frequencies and noise, using our previously described post-processing shader that utilizes a separated Gaussian filtering kernel. Then, before we perform the convolution step, the texture is thresholded to n discrete color values, where n corresponds to the number of explicit



(a) Amplification: 5 (b) Amplification: 30 (c) Amplification: 60 (d) Amplification: 100

Figure 6.10: Examples of the influence of the amplification parameter on the depth contour. In Figure 6.10a, with an amplification value of 5, we can only see the outer contours of the lion clearly, in the following Figure 6.10b, with a value of 30, and Figure 6.10c, with a value of 60, we start to see more of the inner creases and cavities as well. For Figure 6.10d, we are starting to see the general surface of the lion statue not restricted to the outer silhouette, since with such a high amplification value, most of the points have a subtle contour drawn around them.



(a) Dilation: 0 (b) Dilation: 1 (c) Dilation: 2 (d) Dilation: 3

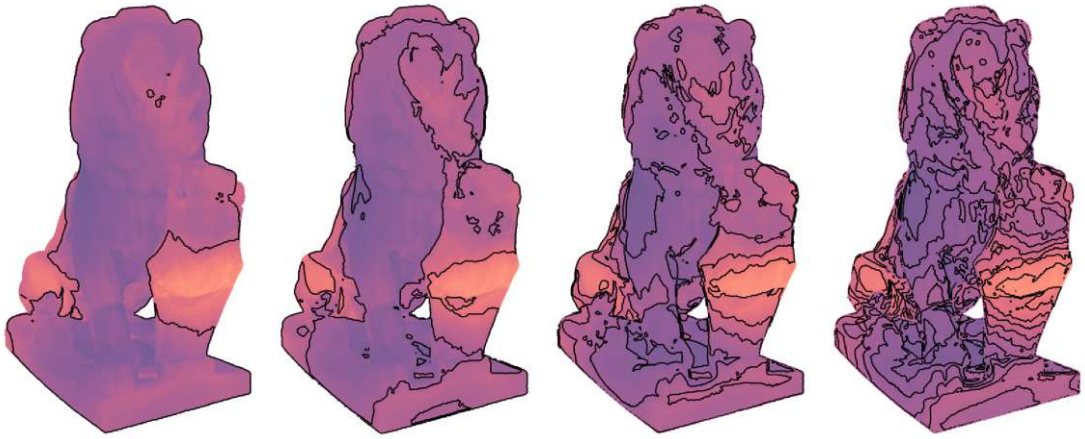
Figure 6.11: Examples of the influence of the number of dilation iterations on the depth contour. The caption shows the number of iterations performed using a circular dilation filter. For all subfigures, we used a constant amplification value of 50.

(a) (b)

Figure 6.12: The explicit encoding contour can also be used without rendering the color overlay as in Figure 6.6. Additionally, rendering both types of contours using distinctive colors will blend them additively where they overlap, as depicted in Figure 6.12b. Here, the yellow explicit encoding contour and the green depth contour will result in yellow contours where they coincide.

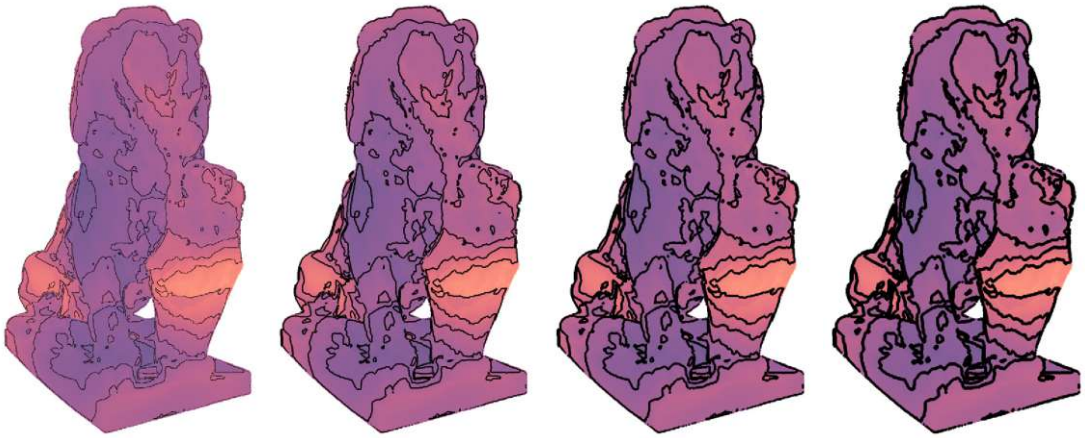
encoding contours that we will have in our final visualization. The convolution kernel is the same that we utilized for the depth contours defined in Equation 6.1. The explicit encoding contours can also be used without the actual explicit encoding texture in the visualization, as depicted in Figure 6.12a. This way, both the information of the real-world colors, as well as the variance of each point in the ensemble, can be viewed simultaneously. This requires us to always render the explicit encoding texture, even if it is not currently viewed in the render window, and vice versa for the regular texture of the point cloud. We render both textures using two separate texture attachments within our framebuffer, which we populate in a single rendering pass, one after the other. In a composite shader, we decide which information to use for the currently considered fragment, respecting the filtering range as well. If both types of contours overlap, we blend the respective colors additively, as depicted in Figure 6.12b.

For the explicit encoding contours, there are two parameters to adjust the visualization. The first one is the mentioned number of isolines that we want to draw, illustrated in Figure 6.13. A value of one for this parameter corresponds to a binary thresholding of the underlying explicit encoding texture and only one isoline between the two areas in the texture, as depicted in Figure 6.13a. This parameter has to be handled with care since an excessive number of lines will quickly clutter the visualization, as can be seen in Figure 6.13d. The second parameter is the number of dilation iterations to perform for



(a) Contours: 1 (b) Contours: 4 (c) Contours: 8 (d) Contours: 16

Figure 6.13: The caption below each subfigure shows the number of isolines of the explicit encoding texture drawn. For a larger number of lines, the visualization will quickly get cluttered. A single dilation pass was used for every subfigure.



(a) Dilation: 0 (b) Dilation: 1 (c) Dilation: 2 (d) Dilation: 3

Figure 6.14: With every subfigure, the number of dilation iterations is increased by one, starting from zero iterations in Figure 6.14a. The number of contour lines drawn is 9 for all subfigures.

the explicit encoding contour and is defined similarly as the dilation explained before for the depth contour in Section 6.2.3. Different values and their effect on the visualization can be seen in Figure 6.14.

6.3 Comparability and Interactivity

We include the option to automatically play the animation that morphs from one ensemble member to the next one, which allows us to simultaneously interact with the visualization by transforming the camera and changing all the previously described visual encodings. Adapting the rendering this way during the traversal of shape space follows the requirement of interaction that we defined in Section 4.1. The animation will play forward and then in reverse in a loop. The speed of the animation can be selected in the GUI.

The auto-play of the animation increases the interactivity of our application, but it can be hard to perceive the discrete ensemble members since they are only visible in their original form for a single frame of the animation. To ease the perception of the discrete ensemble members, we add a cubic easing function that maps from the relative animation state to the displayed state within the shape space. This way, the speed of the animation will steadily slow down once the current state gets closer to an ensemble member in shape space and then pick up speed again until the state is exactly in between two shapes, and so on for every pairwise morphing between ensemble members. The part of the animation where we interpolate between two shapes is then displayed faster, which can help with perceiving the emerging differences. In contrast to that, it can be difficult to detect small changes that happen slowly [SFR00]. An example of the influence of the easing function can be seen in Figure 6.15.

To benefit from using animations, we have to respect the principles of congruence and apprehension [TMB02, HR07]. Congruency entails that the visual elements depicted in the animation are directly tied to the information that should be conveyed, disregarding any unnecessary visual effects. This applies to our approach since we only move the points that make up the models. This depicts exactly the variance within the point clouds in the intermediate frames of our animation between two ensemble members. The apprehension principle requires the animation to be designed in a way that is fully comprehensible to recipients. We work towards this goal by giving the users full control over the progression of the animation, be it by manually advancing it or by using the play/pause functionality with variable speed. The cubic easing of the temporal progression also facilitates this and helps to ease the perception of motion from points with high variability, similarly to the explicit encoding in the colors of the points. This highlighting of certain areas helps to draw the focus of users during the transition.

We considered the progression of coordinates and colors in the animation as synchronized so far, but we can also only allow the coordinates of the points to advance in the animation and keep the colors static, or the other way around. An example where we only advance the colors can be seen in Figure 6.16. This neglect of colors can help to focus only on the shape of the ensembles. When we only advance the colors but keep the shape itself static, it can be helpful to get insights into where specific semantic features of the point cloud morph within the animation.

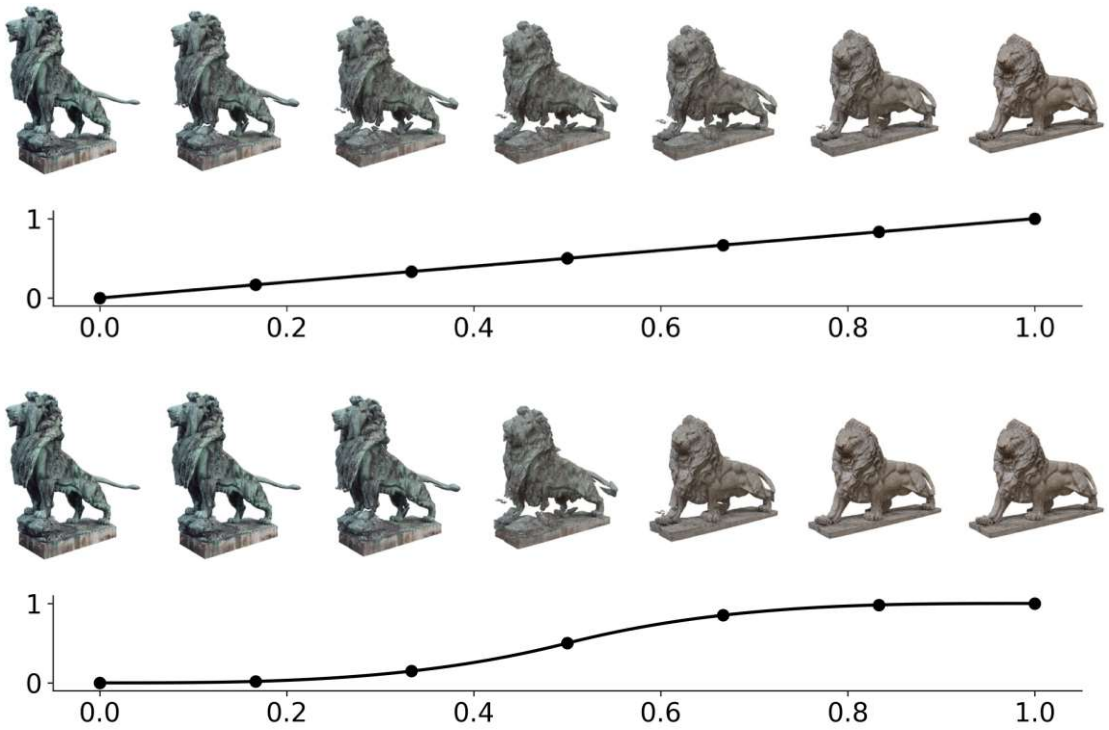


Figure 6.15: An example of seven frames of an animation between two lion statues. The first row uses no easing function, so the mapping between the time of the animation and the progression of morphing is simply linear. The second row uses a cubic easing function during the interpolation between the two states.

(a) (b) (c) (d)

Figure 6.16: In Figure 6.16a and Figure 6.16d we can see each lion statue with their original colors. In Figure 6.16b and Figure 6.16c, we swapped the colors with the ones from the respective other statue. In Figure 6.16c, we can see that the mane of the lion in Figure 6.16a morphs approximately to the position of the mane of the lion in Figure 6.16d.

CHAPTER 7

Implementation

7.1 Pre-Processing

The pre-processing of the data, used within this thesis, is done using two different applications: Blender and CloudCompare. Blender is a 3D graphics software that allows for the modeling and manipulation of spatial data. We use Blender to manually rigidly register some of the photogrammetry models from the use cases 8.1.1, 8.1.4, and 8.1.5. CloudCompare is a software for processing and comparing terrestrial point clouds, as described in Chapter 2. We use it to segment the large data instances in use case 8.1.2 into smaller submodels and also to sample some of the triangular meshes from photogrammetry, given a fixed number of sample points. Data can be imported into our application in various formats. Supported are .obj, .ply, .e57, and .laz files using the Python libraries meshio, e57, and laspy, respectively. For the mesh formats, .obj and .ply, the colors have to be exported as vertex attributes. There is no support for textures, and we remove the adjacency information of the triangles if it is provided for these files.

7.2 Octree

Our implementation of the octree data structure uses only Python and the commonly employed libraries NumPy [HMvdW+20] and PyTorch [AYH+24]. Our implementation consists of classes for the octree, the nodes, and the leaves. The data structures for the tree hierarchy and the metadata per node are returned as NumPy arrays, which allow for the pre-allocation of memory. The data structure for the points themselves is a PyTorch tensor that allows for the use of the automatic differentiation feature of the library, described in Subsection 3.3.2.

7.3 Sinkhorn Algorithm

The implementation of the Sinkhorn algorithm from the work of Feydy et al. [FSV+19] is open-source and available via a GitHub repository [Fey22], which we extend using Python. Also, their library KeOps [CFG+21] is open-source, of which we require the symbolic tensor class LazyTensor that enables speed-ups in the range of ×10-×100 for PyTorch GPU programs. The softmin operator described in Section 3.2.5 is also implemented efficiently in GeomLoss using KeOps [Fey20].

7.4 Rendering

Our renderer uses modernGL [Dom16], a Python wrapper over the core of OpenGL. The window context is created using moderngl-window [For19]. And the GUI is created using pyimgui [Jaw18], a Python wrapper over the popular dear imgui C++ library [Cor14]. The points are animated utilizing compute shaders from OpenGL. The post-processing effects described in Chapter 6 are implemented using a custom rendering pipeline with dedicated shaders for depth contours, explicit encoding contours, dilation, and the Gaussian blur. A final composite shader combines the texture from the regular rendering of the point cloud, the explicit encoding texture, and the two contour types into the final rendered image.

7.5 Graphical User Interface (GUI)

We implement a GUI as the interface to the functionality described in Chapter 5 and Chapter 6, depicted in Figure 6.2. To initialize the data, we provide a button that opens a file-picker dialog, where a user is prompted to select multiple files in the same folder that have one of the supported data types specified in Section 7.1. Afterward, the button stating "Build" in the top left corner of Figure 6.2 can be pressed to start the import of the selected files and the construction of the necessary octree and kd-tree data structures as described in Chapter 5.

We attempt to aggregate the GUI-elements meaningfully below the three groups *Renderer*, *Optimal Transport*, and *Comparison*, displayed on the left of Figure 6.2. Each group offers multiple tools to interact with the visualization. *Comparison* also offers further sub-groups of GUI elements. The *Renderer* group provides access to settings that directly influence the general aesthetic and parameters of the rendering. For instance, the background color and the rendering size of the points can be changed within this group, this can be seen in Figure 7.1a. The *Optimal Transport* group provides an interface for the parameters explained in detail in Section 5.2, as depicted in Figure 7.1b. The GUI elements that facilitate adding an explicit encoding on top of the rendered point cloud are grouped within the collapsible *Comparison* menu, shown in Figure 7.1c. This menu provides functionality to add and manipulate visual elements as explained in Chapter 6. The submenus for the *Comparison* menu are depicted in Figure 7.1d, Figure 7.1e, and

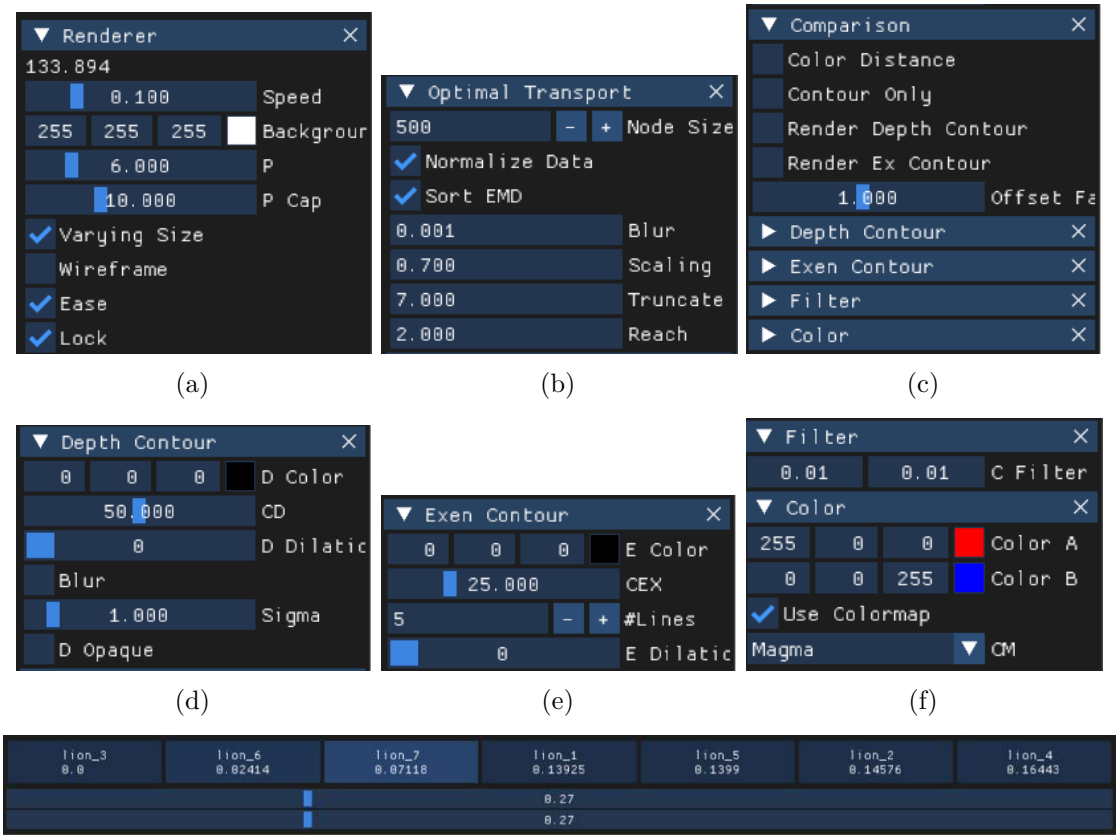


Figure 7.1: An overview of different parts of our GUI that enable control over the following functionality: Figure 7.1a depicts general settings for the renderer, like the speed of the animation during the autoplay, an option to activate the variable size of the points and associated bounds for point size, as well as options to activate the cubic easing and the synchronization of both sliders depicted in Figure 7.1g. Figure 7.1b depicts the parameters for the Sinkhorn algorithm as described in Section 5.2. Figure 7.1c shows the elements in the interface to control the visual encodings. Figure 7.1d shows the settings for the depth contours and Figure 7.1e the settings for the explicit encoding contours. Figure 7.1f depicts the GUI elements to control the colors of the explicit encoding overlay and the filtering. Figure 7.1g depicts the representation of the shape space for the example of stone lions from Figure 5.1 with the associated sliders to manually advance the current position within shape space.

Figure 7.1f. These allow for editing the contours as well as the explicit encoding within the texture of the point cloud, as explained in Section 6.2. The container depicted in Figure 7.1g at the bottom of the GUI allows us to travel through the shape space.

To navigate the shape space, we decide to use a simple slider in the GUI to control the morphing animation between states. In Figure 7.1g, two sliders are visible, one to control

(a) (b) (c)

Figure 7.2: A close-up of the slider and the labels for the loaded ensemble. Figure 7.2a depicts the color of the label and the context menu showing the relative state when we are exactly at the position of an ensemble member in the shape space. Figure 7.2b shows the same close-up when we advance the slider a bit further, the colors of the labels change accordingly to the current relative state shown in the context menu. Figure 7.2c shows the interactive reordering when dragging the label for *lion_6* over the label for *lion_4* to swap the two. Only a swap with the reference model is allowed, as long as the ordering based on the Wasserstein-2 distance is enforced.

the progress of the coordinates and one for the colors, which can be synchronized, as depicted in Figure 6.16. In the container at the bottom of the GUI that holds the two sliders, we can also see the labels of the individual ensemble members. As soon as the computation has finished, the rendering window, which can be seen in the middle of Figure 6.2, will display the Fréchet mean of the ensemble. The bottom container of the GUI now displays the labels of the loaded ensemble, sorted based on the Wasserstein-2 distance, above the slider elements. This row of labels functions as our representation of the shape space. In Figure 7.1g, these labels correspond to the seven lion statues. Below the name in each label, we can see a floating point number that indicates the Wasserstein-2 distance to the first point cloud in the sequence. An example of how the reordering of the ensemble members via drag and drop of the labels looks in the GUI can be seen in Figure 7.2c.

The state of the rendering within the shape space is controlled using the sliders. We link the slider elements to our representation of the shape space using multiple visual indications. Since we only need a single coordinate to reference a position in shape space, the textual representation of the slider in the GUI indicates the global state within shape space, visible in Figure 6.2. It can frequently be required to see the relative state between two point clouds, therefore, we added a context menu that displays this value, which can be seen in Figure 7.1g, Figure 7.2a, and Figure 7.2b. The color of the label elements also changes with respect to the current state, by interpolating between darker and brighter shades of blue, as depicted in Figure 7.2. A remaining GUI element that was not mentioned so far is the *Play* button at the bottom of the left container visible in Figure 6.2. A click of this button will automatically play the animation that morphs from one ensemble member to the next one.

CHAPTER 8

Results

In the previous chapter, we restrict ourselves to the example of stone lions to showcase the implemented functionality, but on the introductory pages of Chapter 1, more has been promised in terms of scale and flexibility regarding the input data. In this chapter, we want to delve into various possible challenges that can be overcome using our proposed application. In Section 8.1, we consider a range of different use cases with various data sets. In Section 8.2, we evaluate the performance of our approach with respect to different aspects and compare our octree-based multiscale approach to the performance of native GeomLoss [FSV+19].

8.1 Case Studies

An overview of the datasets that we use in this section can be seen in Table 8.1, containing metadata, settings, and the achieved performance. The first column of the table shows the title of the dataset that corresponds to the subsection with the same title within this section, which describes details and the origin of the use case, and depicts results. The second column of Table 8.1 shows the number of ensemble members and the approximate number of points for each of the members, where the exact number can be a few points above or below that value for each point cloud due to an inexact subsampling method. The third column shows the values for the parameters Blurring Scale, Scaling Ratio, Truncation Margin, Reach Scale, and Maximum Leaf Size that influence the runtime strongly. The used values for each dataset were determined during the experiments. Smaller values for the Blurring Scale were used if the ensemble contained small structures. The Truncation Margin, in conjunction with the Maximum Leaf Size, was used to balance the differences in run times resulting from the necessity for larger leaf sizes for some cases. The Reach Scale was selected to be 2 for all examples given that the number of points between the ensemble members does not vary much. The Scaling Ratio was chosen to be 0.7 for all examples, because this value gives a good tradeoff between accuracy and

speed. We refer to the explanations in Section 5.2 for more details. The last column of Table 8.1 shows the total time it took to finish the pre-processing and octree creation, the Sinkhorn loop, and the post-processing for the whole ensemble. Here, we also show the minimum and maximum necessary time for individual assignments between ensemble members for the Sinkhorn loop only.

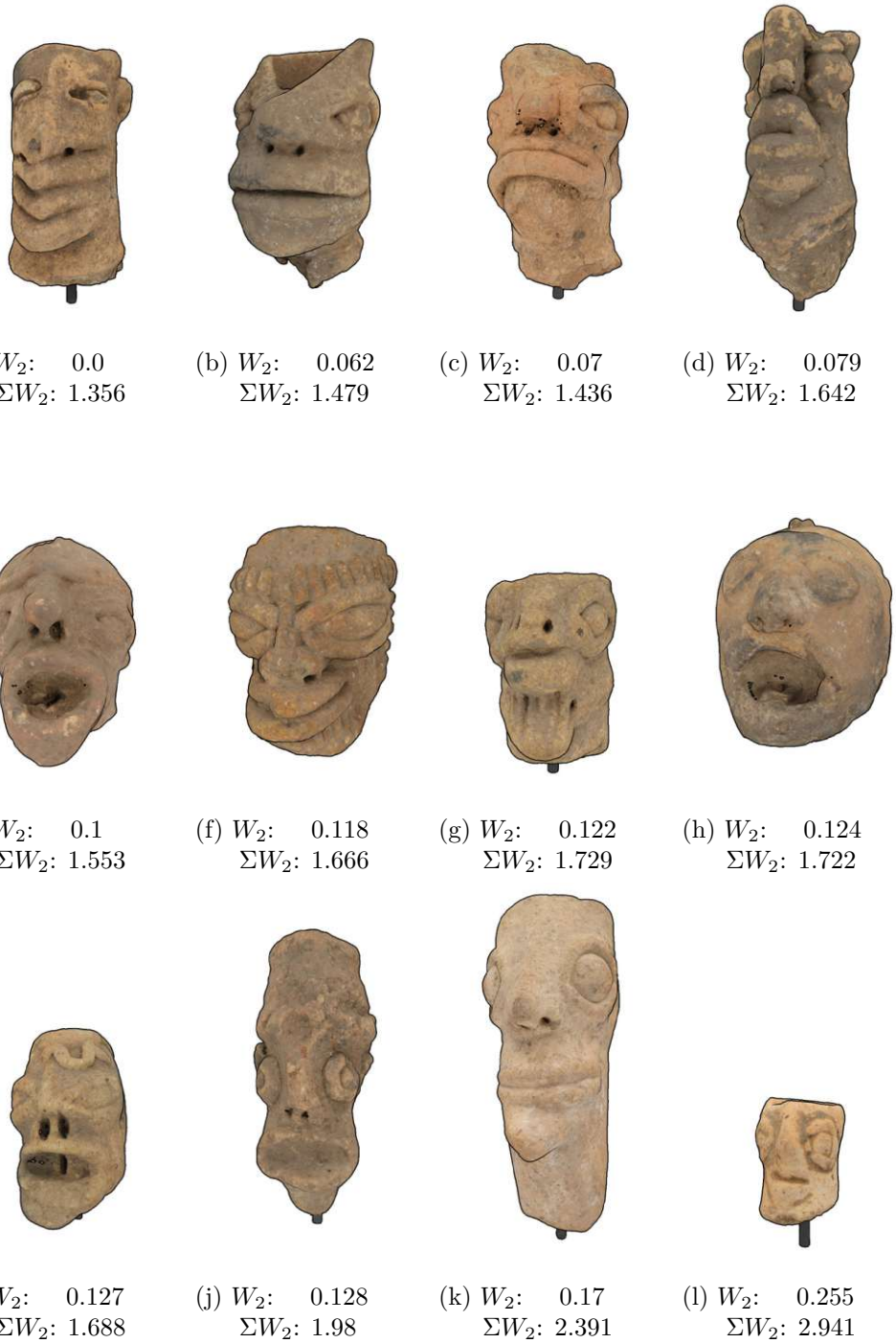
Dataset	Sizes	Parameters	Performance
Anthropomorphic Funerary Terracotta Figures	Ensemble Size: 12 Points: 500.000	blur: 0.001 truncation: 10 leave size: 100	Total: 60.324s Min: 3.069s Max: 4.746s
Belvedere Glacier	Ensemble Size: 9 Points: 3.500.000	blur: 0.001 truncation: 1 leave size: 2000	Time: 328.47s Min: 34.932s Max: 35.882s
Heidentor	Ensemble Size: 4 Points: 2.000.000	blur: 0.0001 truncation: 7 leave size: 1000	Time: 56.92s Min: 12.68s Max: 17.49s
Hominids	Ensemble Size: 10 Points: 1.000.000	blur: 0.0005 truncation: 7 leave size: 200	Time: 83.07s Min: 6.161s Max: 6.957s
Canine Skulls	Ensemble Size: 2 Points: 1.000.000	blur: 0.0001 truncation: 7 leave size: 500	Time: 7.19 OT: 4.784

Table 8.1: The datasets that we used for the evaluation, with their respective metadata and performance results.

8.1.1 Anthropomorphic Funerary Terracotta Figures

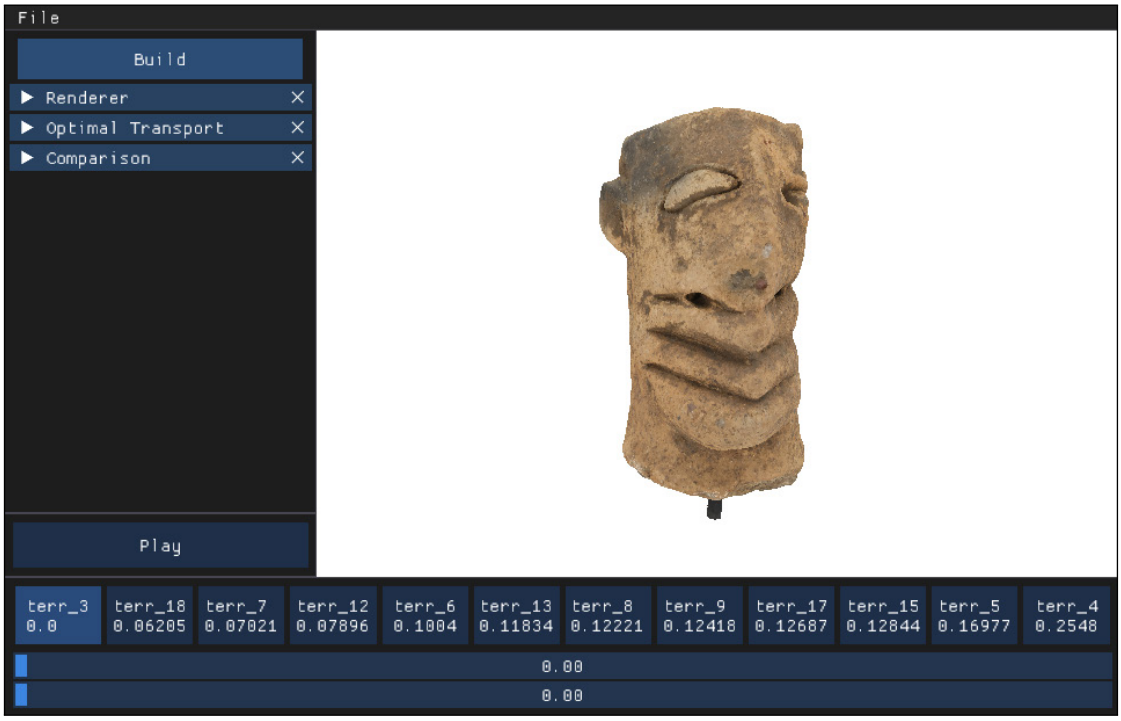
This dataset contains twelve different terracotta figures from the Komaland culture in Ghana that were crafted between the 13th and 17th centuries. The real figures are preserved in the Museum of African Art Arellano Alonos of the University of Valladolid [Ber08] and provided by Global Digital Heritage, as triangular meshes reconstructed using photogrammetry, on Sketchfab [Ske24]. An overview of the whole ensemble can be seen in Figure 8.1, where the first point cloud in the ensemble is the member with the minimal sum of Wasserstein-2 distance to all other ensemble members. The order of the ensemble is given by the Wasserstein-2 distance to the reference model in Figure 8.1a.

The general shapes within this ensemble are relatively simple compared to our other use cases. Here we are interested in establishing an order for the data, and an overview that lets us observe the changes from one to the next member slowly, to improve the understanding of what might have been the process of creation for these sculptures. The Fréchet mean of the ensemble will be shown in the render window once the twelve different point clouds are processed. We can see that for this experiment in Figure 8.2a, where we can also see the automatic sorting that has been created. The individual ensemble



(a) W_2: 0.0 ΣW_2: 1.356
(b) W_2: 0.062 ΣW_2: 1.479
(c) W_2: 0.07 ΣW_2: 1.436
(d) W_2: 0.079 ΣW_2: 1.642
(e) W_2: 0.1 ΣW_2: 1.553
(f) W_2: 0.118 ΣW_2: 1.666
(g) W_2: 0.122 ΣW_2: 1.729
(h) W_2: 0.124 ΣW_2: 1.722
(i) W_2: 0.127 ΣW_2: 1.688
(j) W_2: 0.128 ΣW_2: 1.98
(k) W_2: 0.17 ΣW_2: 2.391
(l) W_2: 0.255 ΣW_2: 2.941

Figure 8.1: Overview of the Anthropomorphic Funerary Terracotta Figures dataset from Subsection 8.1.1 rendered using the depth contours. The caption below each subfigure shows the following two distances: W_2 stands for the Wasserstein-2 distance to the reference point cloud in Figure 8.1a, ΣW_2 stands for the sum of Wasserstein-2 distances from this member to all other members in the ensemble. The camera angle does not change between subfigures, therefore, the displayed positioning shows how the individual members are oriented for our approach.



(a)

(b)

Figure 8.2: A screenshot of the application after finishing the Sinkhorn loop for the ensemble in Figure 8.1. The sequence of the files during the import is given in the label names. The automatic reordering reshuffles the ensemble based on the Wasserstein-2 distance that is visible under the names in each label. When we drag the label "terr_4" in Figure 8.2a from the last position in the shape space representation to the first one, the order of the ensemble will be changed as depicted. This yields us a shape space that goes from smaller members to medium-sized and spherical ones, then to cylindrical, and at last to the conical and large members of the ensemble.

(a) (b) (c) (d) (e) (f)

Figure 8.3: A top-down view of the point clouds in Figure 8.1a and Figure 8.1c using the depth contours to depict the cavities that have been modeled on the top of some of the sculptures from Figure 8.1.

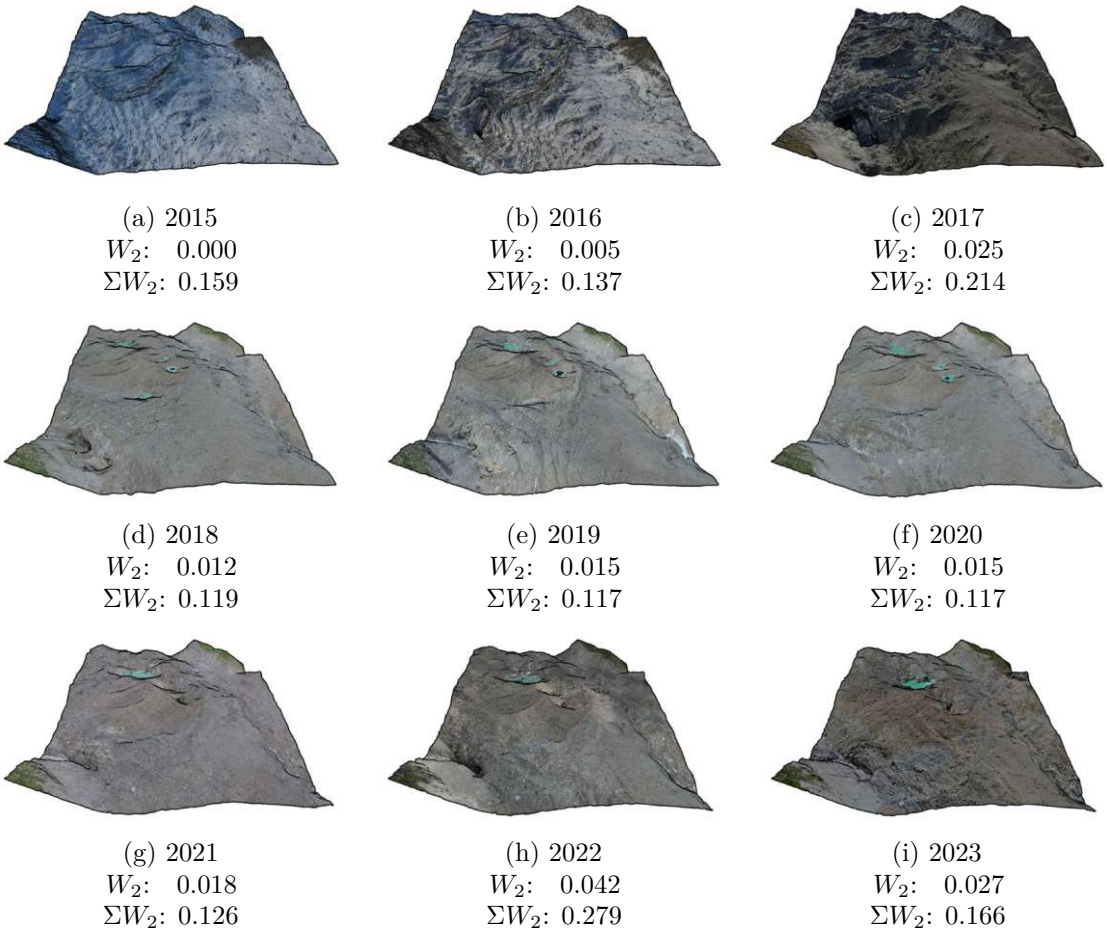
members vary a lot in shape. Some exhibit a rather round general characteristic, others an elongated one, and two members are very small compared to the rest of the ensemble, which explains why the model with the label "terr_3", visible in Figure 8.1a, is selected as a mean shape. It is of medium size and rather cylindrical, and therefore close to the large oblong shapes, but also the very small members in the ensemble. The overview in Figure 8.1 also shows the sequence for the ensemble that is automatically established. For this example, it is quite easy to spot that the first few members also exhibit a relatively cylindrical shape close to the reference in Figure 8.1a. The further we go to the other side of the shape space, we can see that the members have increasingly differing shapes, which peaks with the least two members. Figure 8.1k is much larger than the rest of the ensemble and slightly conical, widening upwards. In contrast to that, the member in Figure 8.1l is by far the smallest ensemble member, which has the largest distance to our reference point cloud. This reference can always be changed by the users to another manually selected ensemble member, which will update the sequence in which the shape space is displayed.

We could also decide to use the point cloud from Figure 8.1l as a reference, which will lead to the reordering visible in Figure 8.2b. That way, the shape space will be structured to go from the smaller and rounder shapes to the more oblong and larger members of the ensemble. This allows for an automatic and intuitive structure of the shape space and the derived animation when traversing it using the slider or the autoplay functionality.

The traversal of the shape space in a smooth animation can help to better follow the local changes that the shapes exhibit when morphing from one to the next, as compared to having hard transitions between the shapes. It can still be difficult to see subtle differences in the general shape. The depth contours can ease the perception of these more subtle variations, as depicted in Figure 8.3, where we inspect the concave top of two of the ensemble members using only the contours.

8.1.2 Belvedere Glacier

This debris-covered glacier is located on the east face of Monte Rosa in the Anzasca Valley of the Italian Alps. Since 2015, the glacier has been surveyed annually using Unmanned Aerial Vehicle (UAV) based photogrammetry. It is of particular interest since over the



(a) 2015
W_2: 0.000
ΣW_2: 0.159

(b) 2016
W_2: 0.005
ΣW_2: 0.137

(c) 2017
W_2: 0.025
ΣW_2: 0.214

(d) 2018
W_2: 0.012
ΣW_2: 0.119

(e) 2019
W_2: 0.015
ΣW_2: 0.117

(f) 2020
W_2: 0.015
ΣW_2: 0.117

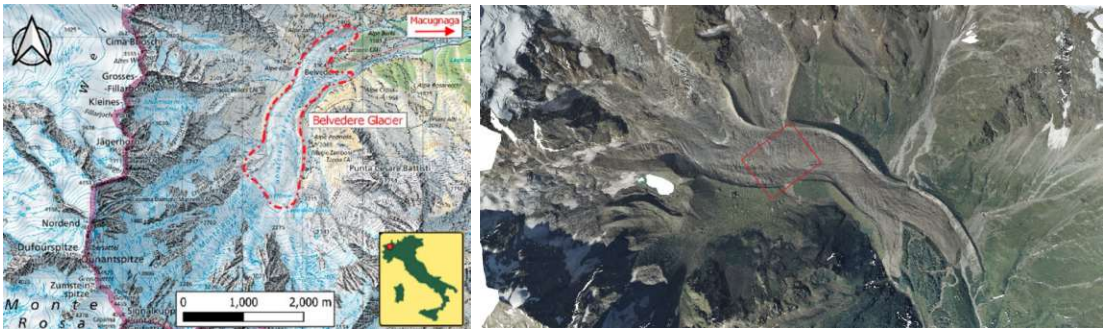
(g) 2021
W_2: 0.018
ΣW_2: 0.126

(h) 2022
W_2: 0.042
ΣW_2: 0.279

(i) 2023
W_2: 0.027
ΣW_2: 0.166

Figure 8.4: Overview of the Belvedere Glacier dataset that contains reconstructions of the mountain surface created each year from 2015 – 2023. We use a subsample out of the full point cloud from each year, as depicted in Figure 8.5b, which is taken around the supraglacial lakes. We subsample each of the segmentations to approximately 3.500.000 points, which is equivalent to the number of points that the sparsest point cloud from the year 2016 contains within the selected rectangular area. The slope downwards of the glacier extends from the right corner of the front-facing edge of the subfigures. The Wasserstein-2 distance to the reference point cloud from 2015 is again noted in each caption as W_2. We can see that an ordering based on this distance would result in a non-chronological sequence. Therefore, the difference of the landscape, to the reference point cloud in Figure 8.4a, is not steadily increasing over the years. Each ensemble member is rendered using the depth contours.

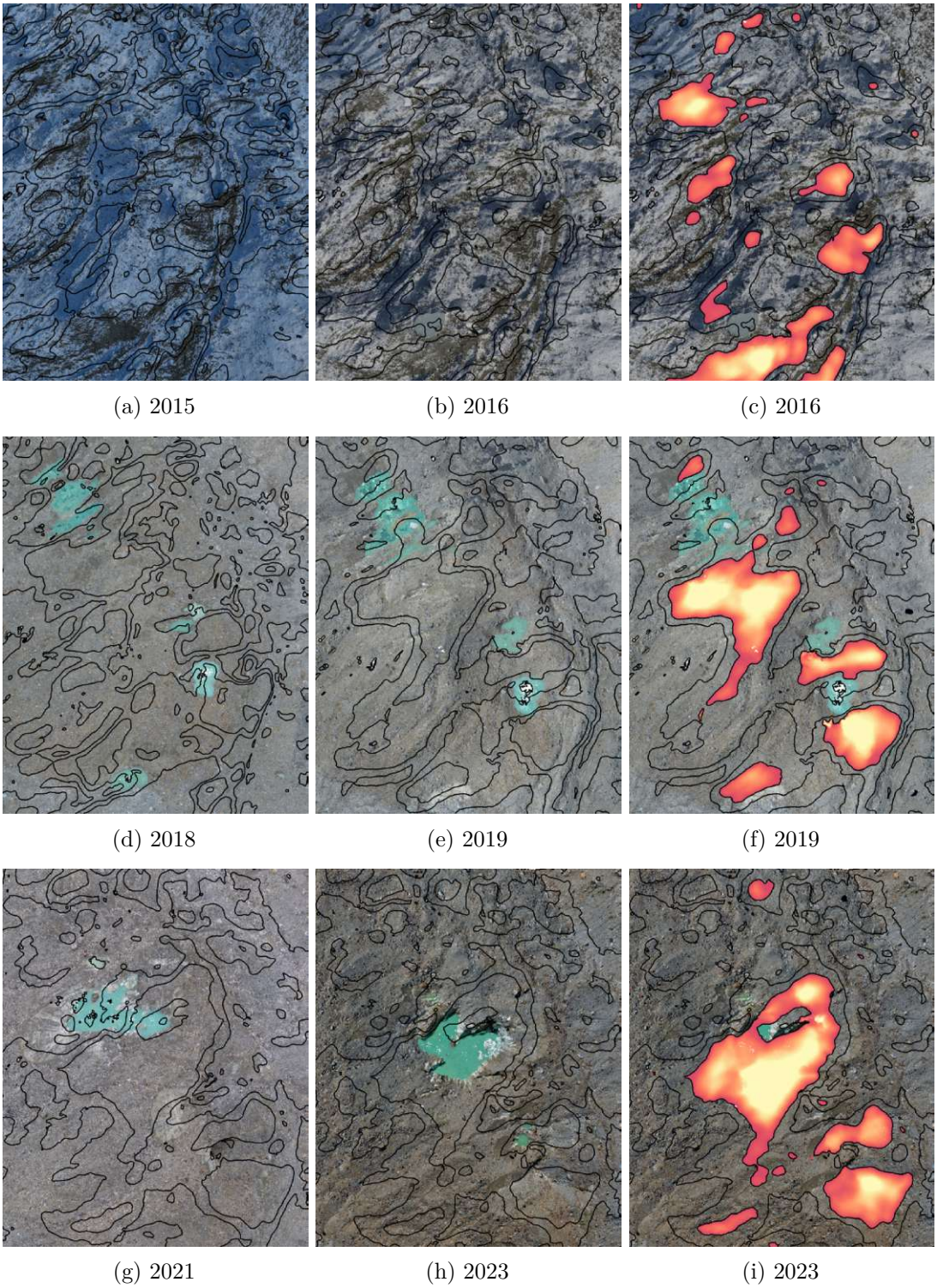
last century, the Belvedere glacier experienced extraordinary dynamics, such as strong movements in the surface and the formation of a supraglacial lake that threatened the nearby community of Macugnaga [FCF+24], which is located only 2km to the west from the approximate foot of the glacier [BRS+24]. Multiple publications focus on different aspects of the data acquisition, data processing, and the dynamics of the glacier, finding that between $2 \times 10^6 m^3$ and $3.5 \times 10^6 m^3$ of ice volume were lost every year from 2015 to 2020 [IBC+22]. A dataset has been published accompanying the publications of Ioli et al. [DGIP21, IBC+22, IDG+24]. The dataset contains point clouds from 2001 to 2023. The earlier reconstructions are obtained from digitalized historical aerial images, the newer reconstructions from unmanned aerial vehicle (UAV) surveys [DGIP21], allowing for detailed point clouds of the face of the glacier for every year dating back decades. In Figure 8.4 we depict the subset that we used for our experiments, dating from 2015 to 2023. We neglect the older reconstructions since the point clouds are far less dense. We do not use the point clouds to their full extent, but only small segmentations. In Figure 8.5b we depict the selection that we used as a red rectangle over the reconstruction from 2009, which shows parts of the east face of Monte Rosa and the Belvedere Glacier. The reconstructions after 2009 do not cover such a large extent of the landscape, and rather follow the boundaries of the stone glacier closely, which is visible in the middle of Figure 8.5a.



(a) (b)

Figure 8.5: In Figure 8.5a, we can see the exact location of the Belvedere Glacier within the Italian Alps close to the Swiss border. The map is taken from [DGIP21]. Figure 8.5b shows a reconstruction of the Belvedere Glacier based on historic aerial images from 2009. The red rectangle shows the area that we use for our experiments within Subsection 8.1.2.

The fast vanishing of glacial ice, due to global warming, leads to rapid changes in the surfaces and the occurrence of surge-type events, as well as the manifestation of the mentioned supraglacial lakes [BRS+24]. These lakes can hold vast amounts of water, and due to the ongoing glacial retreat, they can outburst and lead to floods with immense destructive power, threatening the integrity of the glacial surface and downstream populations. The Belvedere Glacier features a famous representative of this type of lake, the so-called *Effimero* lake came to its notoriety from the large extent during its first appearance, holding an estimate of 3 million cubic meters of water [BRS+24]. The lake



(a) 2015 (b) 2016 (c) 2016

(d) 2018 (e) 2019 (f) 2019

(g) 2021 (h) 2023 (i) 2023

Figure 8.6: At closer inspection of the Effimero lake, we can uncover the dynamics of the glacier surface in the immediate vicinity over the years. The explicit encoding contours show the pairwise changes from the current to the next year. Except for Figure 8.6h and Figure 8.6i, where the changes from 2021 to 2023 are depicted. We added a copy of the second column with additional explicit encoding in the last column of the Figure. This encodes the areas that deform the most and correspond to the positions of supraglacial lakes in the following year.

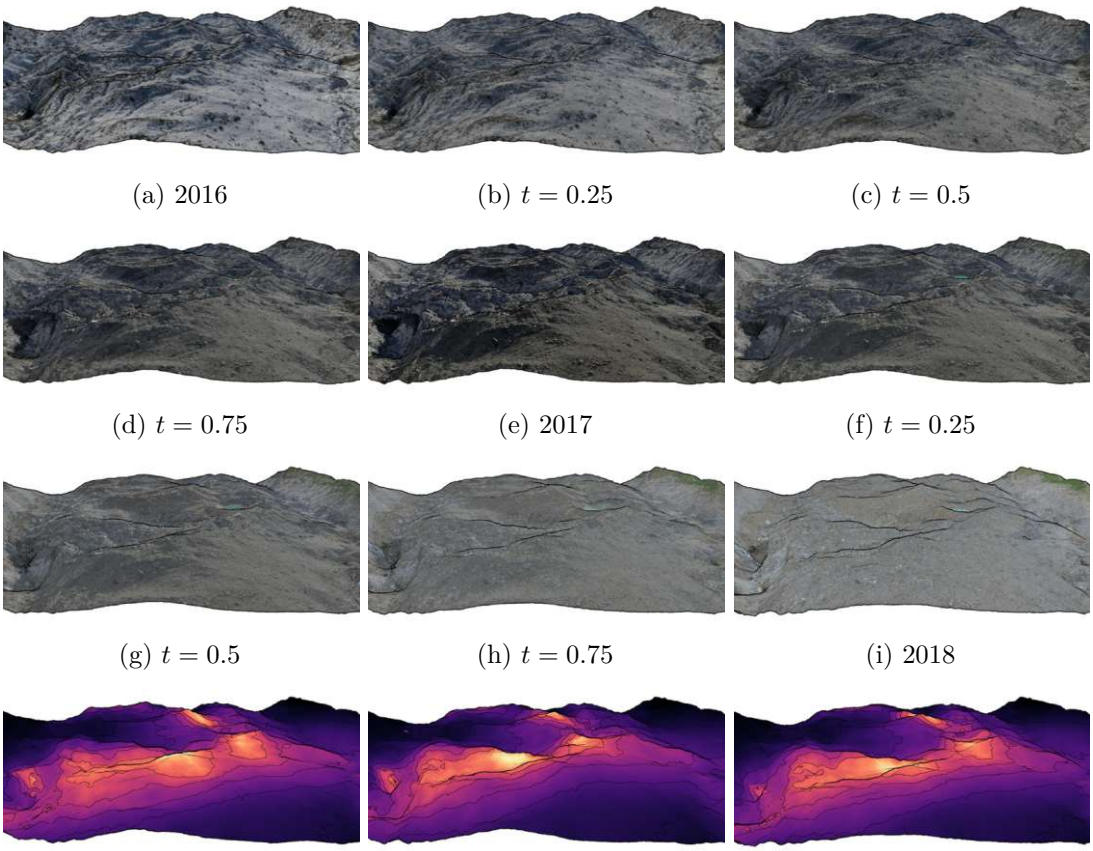
had to be monitored and regulated, and it vanished in the following years. In Figure 8.5b, depicting the glacier in 2009, we can see that there are only traces of lake *Effimero* upwards the glacier from the red rectangle, none of which hold any water. The change of the supraglacial lake has been studied by many researchers, including a recent work of Brodský et al. [BRS+24]. They conclude that while Lake Effismero is stable in its position, it varies largely in size over the years, with surface size variations from 428m^2 to 100.000m^2 at its largest in 2001. In Figure 8.4, we can see that the position and extent of the supraglacial lakes vary to some degree. When we take a closer look at the surrounding areas, depicted in Figure 8.6, we can see that a single basin of the lake emerges in the year 2016, visible in Figure 8.6b. The following years have a drastically different impression compared to these first two, since the acquisition month for the recording was changed from October to July. We can see that four smaller basins formed in Figure 8.6d, three of which are still filled with water in the following year. In Figure 8.6f we can see the areas most affected by glacier retreat in the next two years. In 2021, depicted in Figure 8.6g, we see that the lower two lakes have disappeared, and what seems to be a crevasse remains. The larger lake, further upwards on the surface of the glacier, seems to have flooded a larger basin that grew beneath it.

Modern surveying methods make it possible to reconstruct full 3D models even from historical data gathered long ago. Given such reconstructions for a sequence of different time points of a model, our approach allows us to create an intuitive and appealing animation that shows the transformation over time. This animation can be used to communicate the morphological changes of a glacier without the need to use any additional explanation to a broader audience. A few frames of such an animation can be seen in Figure 8.7, where the surface of the glacier is rendered from a front-facing view that makes advantage of the contour lines. Although it is difficult to catch the impression of an animation in a few static frames, you can easily see the drastic changes that the landscape exhibits. The explicit encoding can give additional insights and guide the view towards areas of interest, as depicted in Figure 8.7j.

8.1.3 Heidentor Monument

To conserve cultural heritage within the European Union, the "Twin it!" campaign was started in 2023 and asked all Member states to submit at least one 3D digital reconstruction of an important monument or building within that country. Austria submitted a reconstruction of the *Heidentor*, or “Pagans’ Gate”, an approximately 1660-year-old monument located within Petronell-Carnuntum in Lower Austria [kul24]. It was constructed as a four-gated arched monument during Roman times, presumably under Emperor Constantius II (351-361 AD). The monument had a square footprint with a side length of 14.5m. Nowadays, only two pillars and one arch of the original monument remain. An overview of the different 3D reconstructions, provided by [Wra25], can be seen in Figure 8.8.

The reconstructions of the early 19th century, in Figure 8.8b, and the one depicting the Heidentor at the late 19th century, in Figure 8.8c, are modeled based on paintings



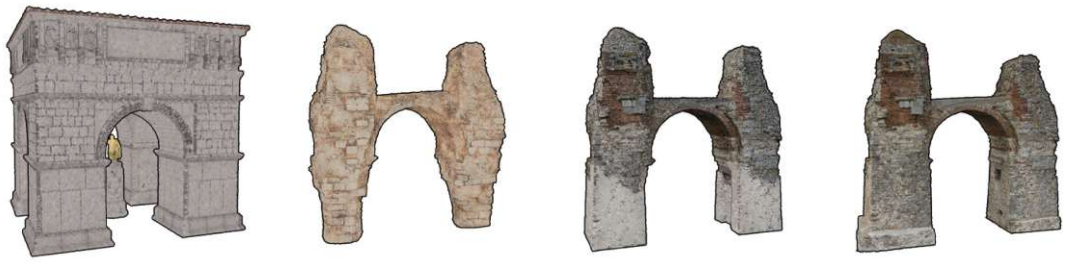
(a) 2016 (b) $t = 0.25$ (c) $t = 0.5$

(d) $t = 0.75$ (e) 2017 (f) $t = 0.25$

(g) $t = 0.5$ (h) $t = 0.75$ (i) 2018

(j) 2016 (k) 2017 (l) 2018

Figure 8.7: Frames of an animation through shape space. The discrete ensemble members can be seen along the diagonal of the figure, with 2016 in Figure 8.7a, 2017 in Figure 8.7e, and 2018 in Figure 8.7i and in the last row as well. Mind that for the years 2016 and 2017 the recording happened in October, and for 2018 the data was recorded in July. Each subfigure is rendered using the depth contours. We emphasize the areas of the largest variance using explicit encoding together with contours in the last row of the Figure.



(a) 351-361
W_2: 0.391
ΣW_2: 1.151

(b) 1837
W_2: 0.065
ΣW_2: 0.484

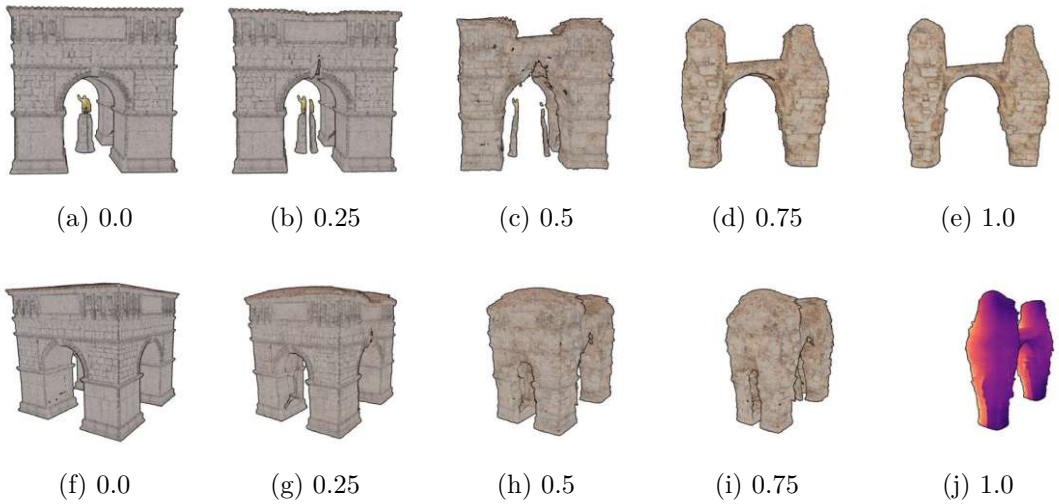
(c) 1869
W_2: 0.026
ΣW_2: 0.456

(d) 2024
W_2: 0.000
ΣW_2: 0.483

Figure 8.8: An overview of the Heidentor ensemble reconstructed from H. Wraunek [Wra25] for the "Twin it!" campaign [kul24]. The Wasserstein-2 distance to the current state of the Heidentor in Figure 8.8d is stated as W_2 in the caption of every subfigure. The sum of Wasserstein-2 distances to all ensemble members is given as ΣW_2. Each ensemble member is rendered using the depth contours. The reconstruction of the original Heidentor in Figure 8.8a is based on studies of numerous experts [kul24]. An early detailed study of the Heidentor can be found in the watercolor by Rudolf von Alt from 1837 [kul24], which was used to reconstruct the model depicted in Figure 8.8b. Detailed studies and lithographs can be found from the years when the Heidentor was majorly restored around 1869, the derived 3D model is depicted in Figure 8.8c. The last Figure 8.8d depicts the current state of the Heidentor in 2024, captured using a drone and photogrammetry.

and photographs. The reconstruction of the original Heidentor, in Figure 8.8a, is based on numerous historical studies of the monument. Especially, the roof construction is surrounded by a high amount of uncertainty, because none of it has survived the centuries. This example of a typical quadrifrons, a monument with double passages on four pillars, often features a special figure at the center of the structure, which was added to the historical reconstruction visible in Figure 8.9a. Since this figure is not contained in the remainder of the ensemble, the points that contribute to it have to dissolve within the masonry during the morphing animation, as depicted in Figure 8.9b and Figure 8.9c. Since the full reconstruction of all four pillars covers much larger extents than the reconstruction of the later years, the explicit encoding of the *Cumulative* attribute is reduced to a linear gradient as visible in Figure 8.9j.

The Heidentor has been renovated several times since the 19th century. The current aesthetic of the monument was mainly shaped by renovation works in 1907. Smaller renovations happened in 1957 and after 1998 [kul24]. The monument changed significantly in the 15th and 16th centuries, when a shortage of building materials led to stone theft. Large stones were blasted out using black powder. This shaped the Heidentor forever, and is visible in the remaining pillars in Figure 8.10. For Figure 8.10, we removed the full reconstruction of the Heidentor from the ensemble. This way, the explicit encoding



(a) 0.0 (b) 0.25 (c) 0.5 (d) 0.75 (e) 1.0

(f) 0.0 (g) 0.25 (h) 0.5 (i) 0.75 (j) 1.0

Figure 8.9: Frames of an animation that transforms the original Roman monument to a reconstruction modeled from a watercolor of the Heidentor by Rudolf Alt from 1837 [kul24]. If the ensemble contains a member that differs greatly in aspect ratio from the other members, the explicit encoding overlay showing the *Cumulative* attribute will mostly capture the distance to this one member that exhibits strong variance, as depicted in Figure 8.9j.

(a) 0.0 (b) 0.25 (c) 0.5 (d) 0.75 (e) 1.0

Figure 8.10: A sequence of frames from an animation through shape space using only the reconstructions from the early 19th century to today. We have explicitly encoded the total distance traveled during the animation, which quickly shows the area where the monument has changed the most, i.e., the lower parts of the pillars.

can be used to show the areas within the remaining two pillars where stones have been taken, as depicted in Figure 8.10a, Figure 8.10b, and Figure 8.10c.

8.1.4 Hominids

The Paleontological Institute of Cornell University in Ithaca, New York, created artificial casts of different members of the hominid family. These recreations of the skulls from the relatives of the Homo Sapiens have then been digitally reconstructed as triangular meshes using photogrammetry. The data was obtained from Sketchfab [Ske21a]. To

(a) Homo sapiens
W_2: 0.000
ΣW_2: 0.513

(b) Homo floresiensis
W_2: 0.029
ΣW_2: 0.562

(c) Homo habilis
W_2: 0.039
ΣW_2: 0.454

(d) Homo neanderthalensis
W_2: 0.046
ΣW_2: 0.535

(e) Australopithecus africanus
W_2: 0.055
ΣW_2: 0.536

(f) Homo heidelbergensis
W_2: 0.056
ΣW_2: 0.566

(g) Ardipithecus ramidus
W_2: 0.057
ΣW_2: 0.518

(h) Paranthropus boisei
W_2: 0.066
ΣW_2: 0.603

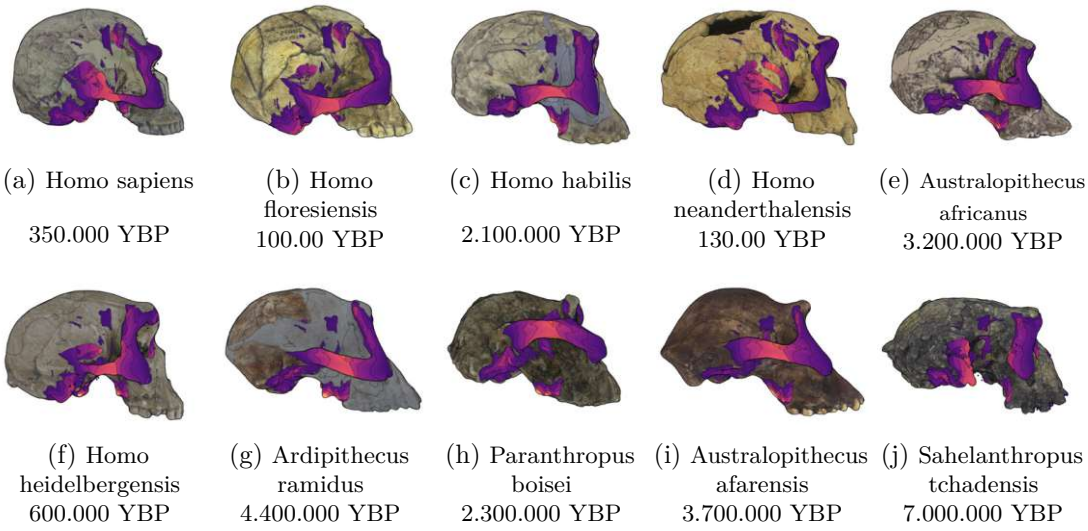
(i) Australopithecus afarensis
W_2: 0.07
ΣW_2: 0.571

(j) Sahelanthropus tchadensis
W_2: 0.092
ΣW_2: 0.755

Figure 8.11: An overview of the hominids ensemble taken from [Ske21a] rendered using the depth contours. The Wasserstein-2 distance to the skull of the Homo sapiens in Figure 8.11a, is stated as W_2 in the caption of every subfigure. The sum of Wasserstein-2 distances to all ensemble members is given as ΣW_2. The ordering generally groups members of the genus homo. The cast for the Homo heidelbergensis, depicted in Figure 8.11f, has a large cavity on the non-visible side of the skull, distorting its position in the sequence.

better understand the evolutionary differences within the family, we can import densely sampled point clouds of the original meshes. We need to manually choose an appropriate extrinsic pose for these models. All ensemble members are normalized concerning the axis of largest extent. This way we omit the influence of the size on the comparison, which was not truthfully captured in the data. An overview of the ten different point clouds can be seen in Figure 8.11. The names of all the hominids contained in the ensemble can be seen in the captions of Figure 8.11 and Figure 8.12. The reference is chosen to be the Homo sapiens depicted in Figure 8.11a. The Fréchet mean on the ensemble is given by the Homo habilis depicted in Figure 8.11c, which is one of the oldest member in the family of the genus homo. The earliest estimated time for these hominids, based on fossil evidence [FL13], can be seen in the captions of Figure 8.12.

We attempt to establish an ordering of the different hominid species by using their morphological similarity, based on the Wasserstein-2 distance. Our approach groups most of the representatives of the genus homo together except for the Homo heidelbergensis. The reason for this is that the cast is incomplete on the right side of the skull, where it



(a) Homo sapiens
350.000 YBP

(b) Homo floresiensis
100.00 YBP

(c) Homo habilis
2.100.000 YBP

(d) Homo neanderthalensis
130.00 YBP

(e) Australopithecus africanus
3.200.000 YBP

(f) Homo heidelbergensis
600.000 YBP

(g) Ardipithecus ramidus
4.400.000 YBP

(h) Paranthropus boisei
2.300.000 YBP

(i) Australopithecus afarensis
3.700.000 YBP

(j) Sahelanthropus tchadensis
7.000.000 YBP

Figure 8.12: The ensemble of hominids in the same sequence as in Figure 8.11 rendered using both types of contours. The explicit encoding shows large differences in the temporal and zygomatic bones, due to differences in diet, but also cognitive and locomotor shifts between these members of the family of hominids. The caption states the earliest approximated period that these hominids arose, indicated as Years Before Present (YBP). The reference model used for the computation of the differences was the Homo sapiens depicted in Figure 8.12a.

has a large cavity, distorting the automatic ordering. Having a truthful affine registration for this ensemble could improve the automatic ordering. Still, the result of the sorting does not stray far from an ordering based on the estimated distance of the evolutionary relationships of these hominid species to the Homo sapiens.

The variation of shape of the bones that make up the skull contains information about different evolutionary trends due to cognitive and dietary changes, as well as locomotor shifts [Lie11]. Using the explicit encoding of variance within our application, clearly shows that the largest variety within the ensemble of hominids can be found at the temporal and zygomatic bones, visible in Figure 8.12. These bones anchor various muscles that are necessary, for instance, to move the mandible to chew, but also for different forms of social signaling. Additionally, the changes in brain size and the general structure of the head also influenced the form and size of the temporal and zygomatic bones [Lie11]. The family of so called Robust australopiths exhibit heavily build skulls that can produce larger bite forces necessary to chew much more and tougher food, like grass and other unprocessed plants, as compared to the Homo sapiens [Lie11]. Examples are the Paranthropus boisei, depicted in Figure 8.12h, whose skull differs the most to the one from the Homo sapiens according to our results.

8.1.5 Canine Skulls

The last example within our use cases is the smallest ensemble that we include, which is just a pair of point clouds. The Edna Lawrence Nature Lab at Rhode Island School of Design collects and digitizes all sorts of interesting specimens from biology. Within their collection on Sketchfab are many 3D reconstructions of skulls from various animals [Ske21b]. The comparison of animal skulls is commonly done to find distinguishing features between members of a common genus. We have mentioned the work of Herman et al. [HSSK16] that analyzes rodent skulls to uncover different characteristics for different families of rodents. We only pick two of the many reconstructions available, the skull of a medium-sized domestic dog and the skull of a bulldog, to highlight the major differences between these two closely related canines. We depict four frames of the transformation between the two skulls from different viewpoints in Figure 8.13, with a partial explicit encoding and contour lines. The same animation frames are shown in Figure 8.14 with and without explicit encoding. Between the two, the deformation of the incisive bone at the very front of the snout, the nasal bone at the top, and the larger maxilla bone, which forms the side of the snout, is evident in this comparison.

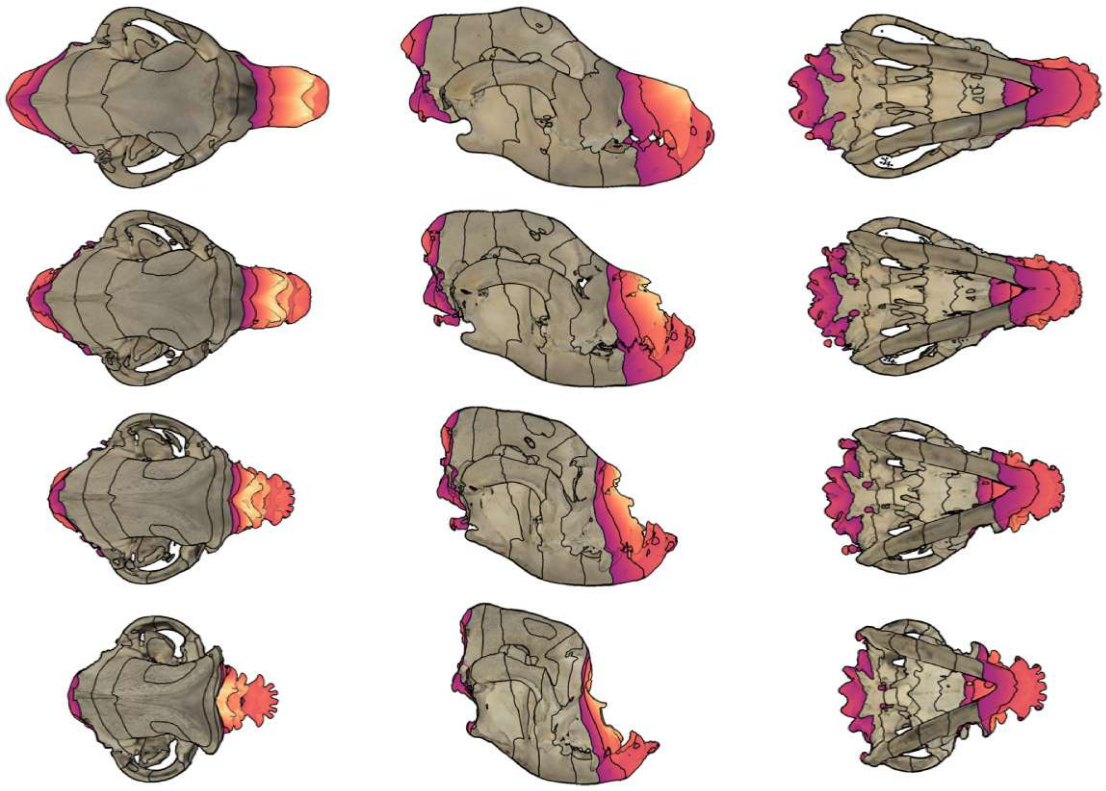


Figure 8.13: A transformation between a domestic dog and a bulldog skull viewed from the top, right, and bottom, respectively. We only encode the distance in the color for those points that move the furthest and add depth contours to the rendering.

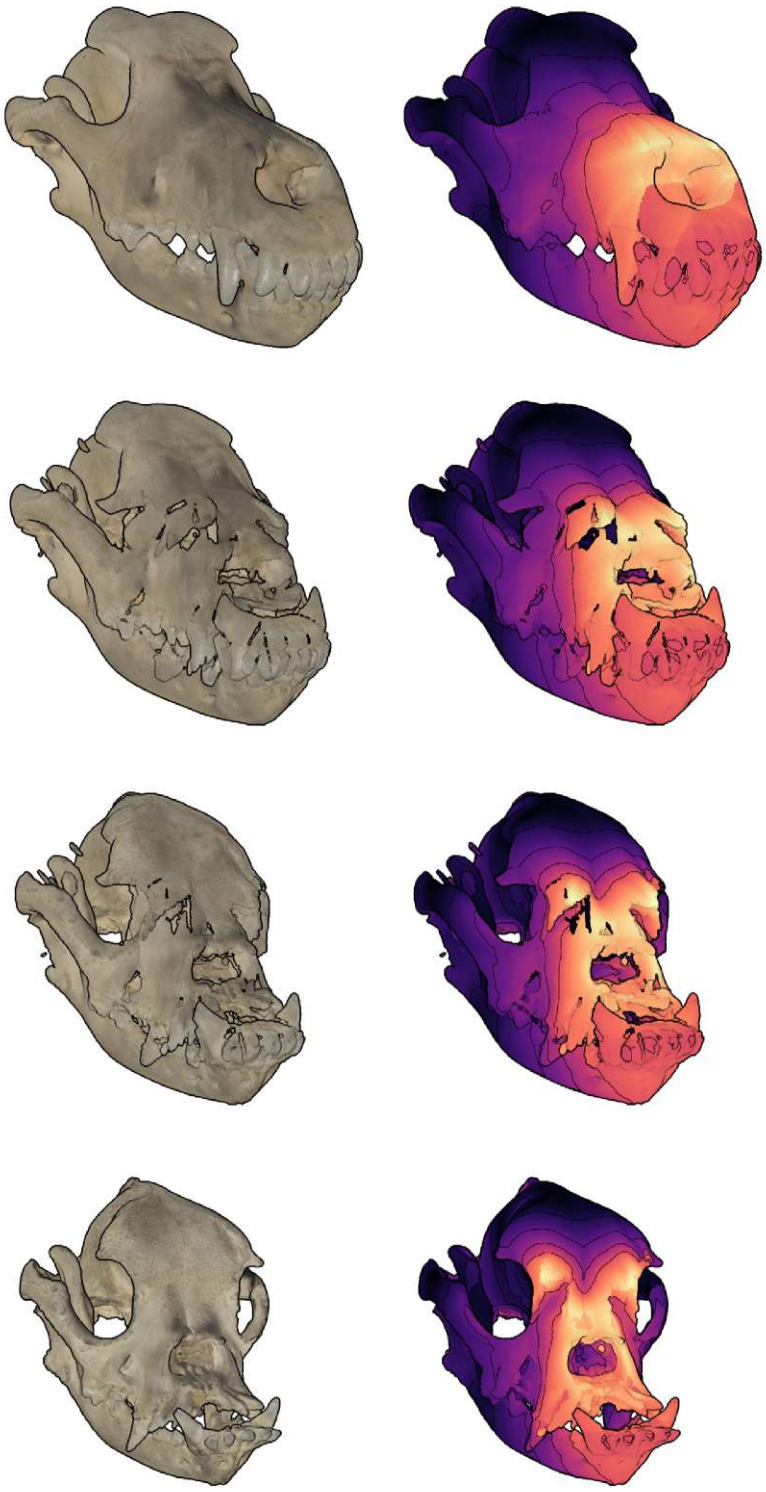


Figure 8.14: The first row of the Figure depicts the skull of a domestic dog, the last row the skull of a bulldog, acquired from the RISD Nature Lab [Ske21b]. In between are the interpolated shapes at a third and two-thirds of the morphing. In the first column, only depth contours are used to enhance the visualization. The second column uses explicit encoding together with contours. The colors depict the distances between the two skulls, where brighter values encode greater differences.

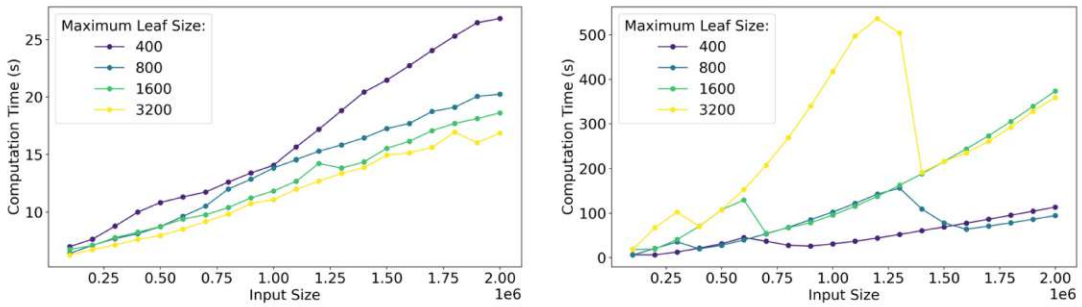
8.2 Performance

We use various modified copies of the same ensemble to evaluate the behavior of the algorithm when increasing either the size of the input point clouds or the number of members within the ensemble. Using a random sub-sampling of the original point clouds, we create multiple copies of the ensemble, using different sampling rates to yield more and less dense versions of each point cloud. We implemented an automatic evaluation that iterates over the different versions of the ensemble and computes the optimal assignment for increasing subsets of the data within the folder until all point clouds are used. The results are saved as a `.csv` file and further visualized within a line plot, as depicted in Figure 8.15. We do not need any window context and rendering capabilities, which frees up a bit of computational resources. If not stated otherwise, we use the same parameters depicted in Table 8.1 as reference for the respective datasets.

The hardware setup that we use for the development and testing is a laptop with a mobile Nvidia RTX 3070 GPU and an 11th-generation Intel i7-11800H CPU. The system memory is 32 GiB with a 3200MHz clock speed. The operating system is Ubuntu 22.04.5.

Before we start with a general evaluation of the performance of our approach, we need to take a close look at the Maximum Leaf Size parameter, since it greatly influences the run time of the algorithm. We evaluate the influence of the parameter choice for both the pre-processing and the optimal transport problem by repeatedly computing the assignment using different point cloud sizes and values for the Maximum Leaf Size. In Figure 8.15a, we see the time needed to import the data and build the octree. The general trend shown in the data is that the smaller we pick the threshold to stop the descent along a branch of the tree, the longer the octree initialization will take, which is to be expected. The influence of the number of points within each point cloud shows a linear trend. The time needed to compute the optimal assignment exhibits more involved dynamics, as depicted in Figure 8.15b. The plot shows that faster processing can be achieved when selecting a more finely divided octree. We can also see that for the typical sizes of the point clouds that we utilize within our approach, more finely divided octrees will lead to much faster performance. As long as we do not exceed the maximum octree depth concerning the computational limits of the kernel truncation, described in Subsection 5.2.4. A value of 200 for this example ensemble would already be too small when used in combination with the largest file sizes.

The unsteady behavior at different sizes for all of the runs can be explained by the combination of point cloud size and branching criterion for the tree, leading to deeper octrees. For instance, the most prominent discontinuity point in Figure 8.15b is roughly at 1.3 million points and a Maximum Leaf Size of 3200, encoded in the yellow line. Here, the depth increases from 5 levels within the octree to 6. The additional step of kernel truncation leads to the algorithm taking a third of the time while using larger point clouds as compared to the run before. This emphasizes the effectiveness of our approach. A similar effect can also be seen for the smaller and more practical values for the Maximum Leaf Size at different sizes of the input point clouds.



(a) Pre-Processing (b) Sinkhorn Loop

Figure 8.15: Multiple evaluations of the optimal transport problem for copies of the hominids dataset from Subsection 8.1.4 with varying sizes. The y-axis shows the necessary computation time in seconds. The x-axis shows the number of points for the point clouds. Each line shows the results for a different value for the Maximum Leaf Size parameter. The size of the ensemble was 9, and all other parameters were similar to the choices written in the row for the Hominids dataset in Table 8.1.

The influence that the depth of the octrees has on the computational efficiency is highlighted in Figure 8.16, where we use pairs of input point clouds that are an order of magnitude larger, sampled from the dataset described in Subsection 8.1.2. Here, the light green curve depicts the time that is necessary for pre-processing with respect to the size of the input. The dark green curve shows the time that is required to iterate through the Sinkhorn loop, we can see similar unsteady behavior as in Figure 8.15b. The purple curve encodes the average depth of the two octrees at the given input size, for which the second y-axis on the right shows the range. The floating point values for the octree show cases where only one tree grew deeper. The Maximum Leaf Size is chosen as 2000 for all runs. Smaller values were not possible due to the constraint described in Subsection 5.2.4 in combination with large input sizes.

We also test the influence of the ensemble size with respect to the individual point cloud size using the data from Subsection 8.1.4. We iteratively recompute the octree structures and the Sinkhorn loop with increased point cloud size and an increased number of ensemble members, always adding one point cloud until we reach nine ensemble members for the given size group. In Figure 8.17, we visualize the time it takes for the pre-processing and the Sinkhorn loop to finish, concerning the number of ensemble members and the individual size. In general, we see a proportional increase in required processing time with respect to both the number of ensemble members and the size of the input point clouds. We can also observe a dip for each plotted curve at an input size of 400.000 again because the octrees grew deeper, from 6 levels to 7 for the following input sizes with the used Maximum Leaf Size of 200.

As a last performance test, we evaluate our approach in comparison to the runtime that we achieve using the original GeomLoss implementation with two scales and a regular

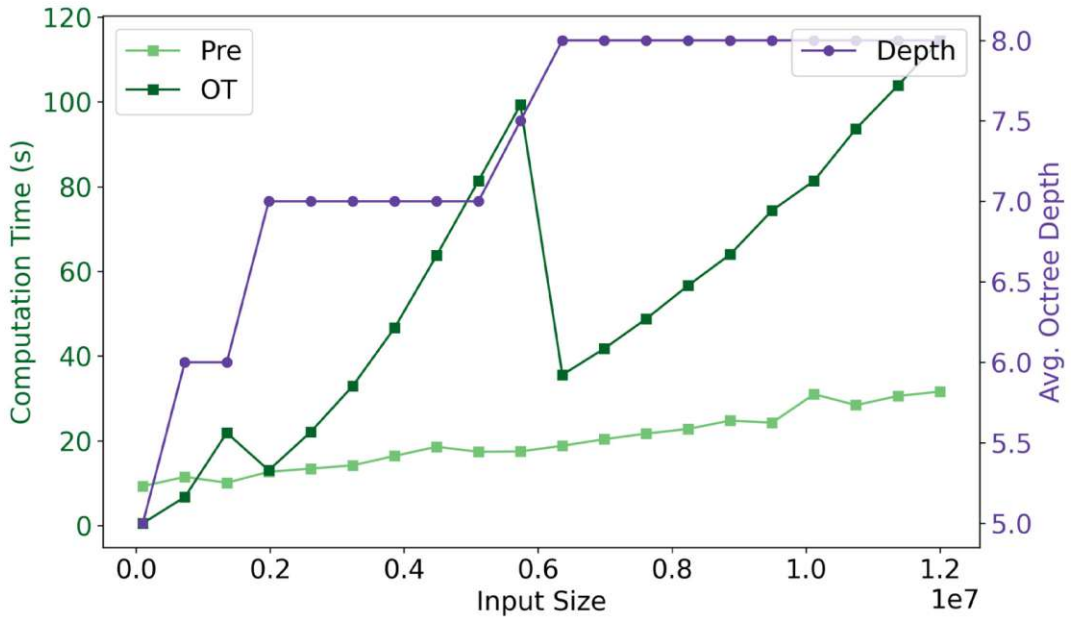


Figure 8.16: Here we show the runtime required to align a pair of point clouds from the Belvedere Glacier dataset described in Subsection 8.1.2. The pair is sampled much denser than what is described in Table 8.1 to allow for input sizes from 100.000 to 12.000.000 points. In total, we perform 20 runs uniformly distributed within that range of sizes. The plot features two different y-axes, the green one on the left to show the required computational time and the purple one on the right to show the average depth of the two octrees. We split the time necessary for the pre-processing and the Sinkhorn algorithm into two lines. The one colored in a brighter shade of green depicts the time necessary for the pre-processing, and the one colored in a darker shade of green depicts the computation time that was necessary to solve the optimal transport problem. The Maximum Leaf Size was selected to be 2000 for all runs.

grid to distribute the points within. For this test, we omit the times necessary for any file imports and pre-processing, as well as the runtime of the kd-tree queries to acquire the color values for the aligned point clouds. We use again the Hominids dataset from Subsection 8.1.4, but only align two point clouds from the ensemble. The parameters were chosen as depicted in Table 8.1, except for the Truncation Margin, since this parameter can influence the results for both implementations a lot.

The results can be seen in Figure 8.18, since the time required by the two approaches differs by up to two orders of magnitude, we encode the y-axis using a logarithmic scale. The set of green curves with dots as glyphs encodes the run time for the original GeomLoss implementation concerning the input size. The set of purple curves with squares as glyphs encodes the time required by our approach. The three different curves

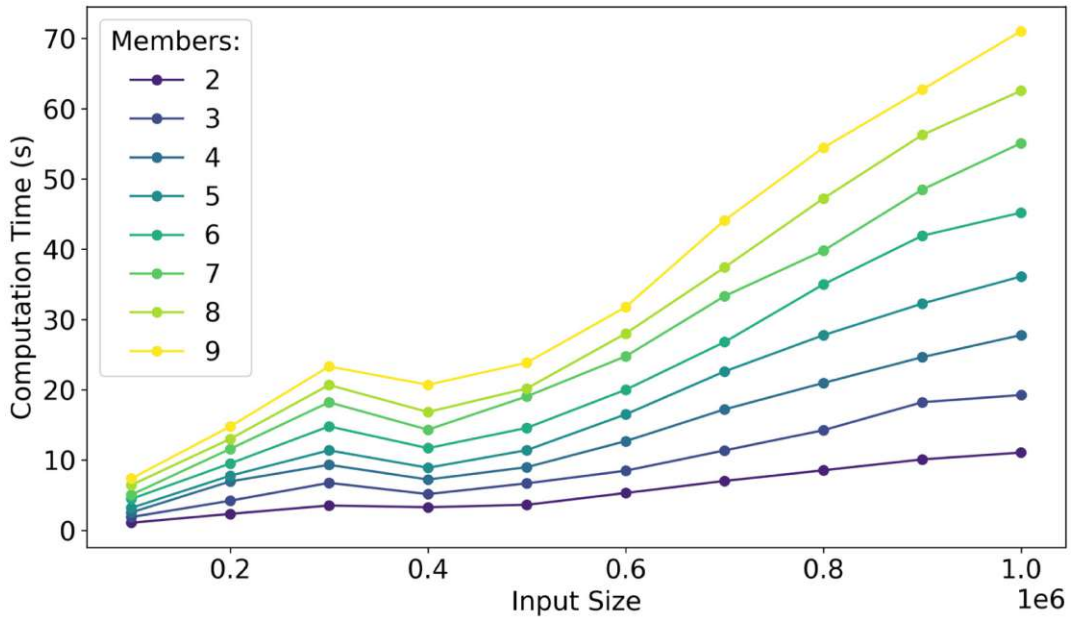


Figure 8.17: The effect of the input size and the ensemble size on the required runtime. Both variables appear to influence the necessary runtime linearly. Except for the range between 400.000 and 600.000, which is influenced by the depth of the octrees. The Maximum Leaf Size was selected as 200 for all runs.

in both the red and blue sets represent different values used for the Truncation Margin parameter. The influence of the Truncation Margin can be seen when comparing the runtime required for the largest input size, which increased by roughly 200% from 54 seconds to 167 seconds when going from $\tau = 0.1$ to $\tau = 7.0$. In Figure 8.18 we can see the influence of the octree depth again, first in the range between 200.000 and 400.000 points, where the depth increases from 6 to 7, and later in the range between 1.200.000 and 1.400.000 points, where the depth rises from 7 to 8. To emphasize the difference between using just the two scales and the full octrees, which vary from 6 to 8 scales of the input measures in this case, we annotated the fastest runtime for the largest input size in the plot. To align a pair of point clouds with 1.5 million points, our approach requires 3.16 seconds, where the original implementation using 2 scales requires 55.52 seconds.

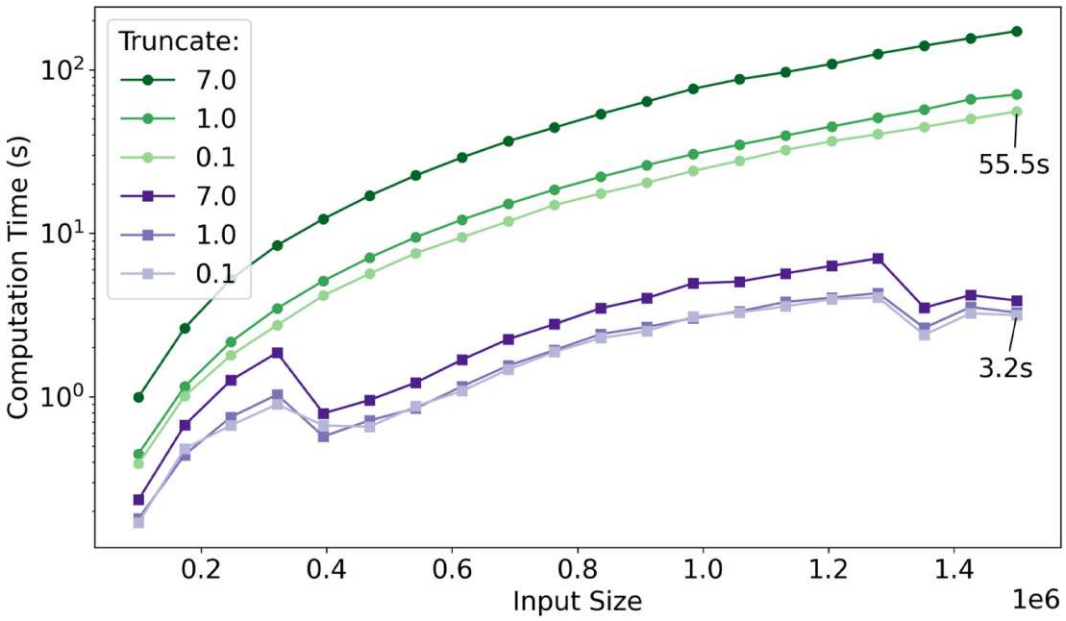


Figure 8.18: A comparison between the runtimes required by our approach in shades of purple and the original Geomloss implementation that uses two scales of the input point clouds in shades of green. Each curve encodes a different value that was used for the Truncation Margin parameter, which can be seen in the legend for the plot. The y-axis scales logarithmically so that the purple curves can still be distinguished. The Maximum Leaf Size is 200 for all runs. The times are measured without including computations for the colors or any pre-processing.

CHAPTER 9

Conclusion

We propose an approach to automatically sort ensembles of point clouds into a linear representation of a shape space, along with tools that support an intuitive visual exploration of the space. We build our sorting approach upon a multiscale optimal transport solver to find a set of coordinate states that contain the information about the whole ensemble in a minimal representation. An octree data structure is used as the foundation for the fast multiscale optimal transport algorithm and supplies the necessary structuring of the input point clouds. Our fast GPU implementation enables us to display the whole ensemble in a smooth animation, while the traversal of the ensemble is being guided by an automatically defined sequence. Explicit encoding of ensemble metrics that are derived from the optimal assignment of a single point set to all ensemble members allows for further insights into the general variation within the data. Adjustable features of our visualization can be activated within our viewer to aid the perception of the characteristics of each point cloud and facilitate comparison across large ensembles.

The overarching research question of this work, as formulated in the introduction, is: *How can a scalable and computationally efficient comparative visualization framework enable interactive exploration and insightful comparison of large spatial data ensembles, while effectively highlighting areas of significant difference through intuitive animations?* In this work, we have answered this question by implementing a comparative visualization framework that facilitates the interactive analysis of a wide range of different point cloud ensembles, as we have exemplified in our results showcasing the incorporated visual encodings that emphasize the variation within each ensemble.

Our approach to computing correspondences of discrete spatial regions within the data is based on a state-of-the-art solver for optimal transport problems. This allows us to align multiple point clouds within seconds. The fast computation is fueled by utilizing hierarchical data structures and progressively solving the optimal transport problem with increasing granularity. As a result, we can greatly decrease the necessary computational

time to solve the optimal transport problem as compared to the native implementation, and allow for a continuous workflow even with large datasets.

By utilizing multiscale optimal transport, we are also not hindered by the lack of explicit point-to-point correspondences. Our datasets do not need to contain the same number of points and can facilitate different densities in similar areas throughout the ensemble. Our approach, in contrast to algorithms that are built onto nearest neighbor queries, also supports applications to general inhomogeneous ensembles of shapes that do not share a common theme.

The use of interchangeable visualization enables us to display the ensemble in a way that is invariant to the number of members, allowing us to convey the data consistently. The transition between ensemble members is displayed in a smooth animation, which facilitates gaining insights based on all the members. The users can decide on how many point clouds should be considered, allowing for more flexibility compared to traditional pairwise comparison. This scalability enables us to support a broader spectrum of applications.

Effective highlighting of areas using explicit encoding in the colors of the points allows tracking of certain changes throughout the animation. This facilitates observing the evolution of regions of interest across all the ensemble members. Adding contours compensates for the loss of context that the original texture incorporates, and eases the perception of the shape for each point cloud.

The animation is fully controllable by the users and can be advanced manually or automatically. The automatic animation through the shape space frees up other ways of interaction with the visualization, like rotations and zoom, as well as tuning the explicit encoding of ensemble metrics. To enable an intuitive investigation of the data, we incorporate the option to automatically establish a linear order of the individual ensemble members that is based on the similarities between all the shapes.

The mentioned advantages of our approach can make the comparison of spatial data accessible to a broader public without any expertise in the visualization domain. Showing ensemble variance as motion while interacting with the visualization enhances intuitiveness for recipients. The interchangeable design of the visualization further facilitates a direct display of differences that does not require any prior knowledge of the visual encoding.

9.1 Limitations

Given the advances in technology, contemporary point cloud datasets can easily scale beyond sizes that would fit into the main memory of modern desktop machines and contain billions of points. Airborne laser scans of entire countries are openly accessible, a popular example is the AHN dataset [Rij25] that covers the entire Netherlands. Out-of-core methods that we mentioned in Section 3.1.1 allow us to only load the parts of a much larger dataset into memory that are currently required. Our approach does not support out-of-core techniques and reaches its limit with input data that contains about

50 million points, dependent on the Maximum Leaf Size as described in Subsection 5.2.6. An important problem to overcome is the kernel truncation of the Sinkhorn algorithm, which at one step requires us to build a dense cost matrix of the problem.

Our current implementation does not support any form of multiprocessing, so when the data is imported and the *Build* button is clicked, the application will freeze until the Sinkhorn loop is done with aligning the whole ensemble. It would be a convenient extension to already display finished ensemble members while the remaining ones are processed in the background. Taking this thought one step further, it would be nice to get progressive updates of the Sinkhorn loop for every layer of our hierarchy so that they are drawn to the render window immediately. This would be especially handy for the aforementioned large datasets that require out-of-core techniques.

The entropic regularization of the Sinkhorn loop incorporates a small error in the assignment that is currently not regarded in any way. Giving some feedback to the users that states, for instance, the probability of inaccuracies would make the results more relevant for applications that require high confidence in the comparison.

Our workflow only allows for point clouds as input data, but similar tasks are frequently done using volumetric data or triangular meshes. Supporting volumetric data as well in a unified approach could open up new possibilities and applications, but it is not straightforward to incorporate. When we try to follow our current approach, one problem is that volumetric data requires the computation of the full transport plan to know how much mass is transported from each voxel to another voxel, which would require extremely long run times and unrealistic memory requirements. Without this information, the morphing between data instances would be reduced to a simple cross-dissolving between registered volumes. For use cases with triangular meshes, it is difficult to incorporate the adjacency information of the triangles using our current pipeline. It would be necessary to regularize the assignment in a way that ensures neighborhood information remains the same, or do an expensive re-computation of the graph structure for each change of coordinates. Also, the implementation of our renderer relies on rasterization of point primitives, and we would need to implement new rendering logic to support triangular meshes or Direct Volume Rendering. In general, the rendering pipeline is currently not optimized for the best possible performance using modern best practices. Level-of-detail techniques that omit points that do not contribute much to the rendering would be one possible way to increase the performance when rendering large point clouds. Further performance improvements could also be achieved by using a different Graphics API. We also did not implement lighting and shadow computation that could, at a low overhead, greatly improve the visual appeal of our renderings. Further, incorporating an Order Independent Transparency (OIT) technique that allows us to utilize the alpha channel of the visualization could be useful, for instance, in combination with the filtering functionality. Our octree implementation does not make use of GPU processing capabilities. Having an interlinked pipeline that, in a sense, seamlessly transports the data through our processing steps, from octree, over optimal transport, to rendering it, could lead to much faster total processing times.

The representation of shape space as depicted in Figure 7.1g is limited to displaying the individual GUI elements next to each other, resulting in a suboptimal representation for large ensembles. The approach could be adapted to support large datasets by building a hierarchy of similar groups of shapes, which is then also displayed hierarchically using mean shapes for groups. Alternatively, as a simpler solution to the scalability problem, we could allow the GUI container of the labels to extend beyond the visible borders of the application using a horizontal scrollbar.

9.2 Future Work

Multiple directions for future work are related to open research problems. We have already mentioned out-of-core techniques and the possibility of utilizing much larger datasets within a reasonable processing time. Improving the approach to enable the processing of larger data at a faster runtime would open up new possibilities like visualizing the progression of the mentioned AHN [Rij25] dataset over time, which is an ambitious but worthwhile undertaking.

In Section 9.1, we mentioned that it would be convenient for users to have progressive updates of the Sinkhorn loop in the render window. This way, there would be immediate visual feedback to users, bringing us closer to a real-time implementation of non-rigid registration. In this context, it would be interesting to investigate the so-called layered octrees, where the points do not reside only in the leaves, but equally sized portions of the total points are also contained in the inner nodes. This type of data structure is often used to facilitate level of detail rendering, where, depending on the current view, different subsets of all points are displayed. Our approach can not directly incorporate this alternative octree data structure, but investigating how it could be utilized would be relevant for the support of point clouds that require such a level of detail structure to be viewed interactively.

Besides the octree, other hierarchical structuring approaches could be considered as well for the implementation of the multiscale Sinkhorn approach. We considered general clustering techniques that would allow for irregular boundaries between subsets of point clouds. This is especially interesting in combination with semantic feature extraction. Looking at the examples of lion statues in Chapter 6, it would make sense for a more intricate comparison to only match between semantically similar areas, for instance, from the head of the lion in one statue again to the head of the lion in the other statue. The extraction of such semantic clusters is a complicated problem on its own, as are the challenges of balancing the total points in cases that are not semantically equivalent. For instance, we have one lion that holds an object that is not apparent in the other statue, like the shield in example Figure 6.4. Nevertheless, a semantically aware optimal transport problem opens up the possibilities for much more involved correspondence matching and assignment, as shown in the work of Shen et al. [SFL+21]. Also, more general concepts for the clusters could extend the possibilities, like the local orientation and curvature of a surface.

In Section 8.2 we have seen that the chosen parameters for the optimal transport problem heavily influence the performance. Finding the best possible values is difficult and depends on the input size and spatial distribution of points. Therefore, supporting the user with the selection of parameters by evaluating the input and either recommending or directly using an optimal set of parameters could greatly ease the use of our application. An optimal selection for all parameters requires an in-depth study of the parameter space concerning many different input ensembles. A simple starting point would be to only regard the creation of the octrees and their depth at first, where optimal parameters depend mostly on the known size of the input point clouds, but can already make a lot of difference for the performance and stability of the approach.

While the automatic order that we establish for an ensemble eases a first general comparison of the point clouds and helps with acquiring an overview of the data, there are use cases where additional freedom in the traversal of shape space could be useful. While we do compute an all-to-all comparison for the input, we only allow a linear traversal of the shape space and thus pairwise interpolation. Enabling an exploration of the shape space, where morphing from one ensemble member to all other members is possible, would help with gaining additional insights about the data. Further, the visualization of a barycenter of the aligned point sets could show the mean shapes of different subsets of the ensemble and allow for even more freedom in the analysis of shape space.

A popular use case for optimal transport in state-of-the-art literature is to utilize the Wasserstein distance as a training loss for deep learning applications [KTT+24], specifically in the generative AI subfield. Popular datasets in the field, like ShapeNet [CFG+15], consist of small shapes with a few thousand points. Investigating generative networks concerning larger training data, which could be possible due to the good performance of the multiscale Sinkhorn algorithm, can allow for interesting use cases like the extrapolation of the glacier time series in Subsection 8.1.2.

Another popular application for optimal transport is style transfer for images [BD23]. This could also be utilized in our case to transfer not only the shape but also the color information, sustaining a similar visual impression throughout the animation. If we do not have color information for all ensemble members, we could find colors where they are missing based on the information that we have within the ensemble.

We also mentioned in Section 9.1 that support for other data types, like volumetric data or triangular meshes, is not easy to incorporate using our current processing pipeline. Still, a lot of scientific domains rely on these data types, and a general comparative visualization solution that scales across types would greatly enhance the possible use cases. Triangular meshes would allow us to render closed surfaces, which we currently emulate using the adaptive point size.

A technique that has attracted a lot of attention recently is Gaussian Splatting [KKLD23]. Here, an image bundle is fed into an optimization loop that tries to recreate a 3D representation of the photographed scene using 3D Gaussian splats with anisotropic appearance. This means that ellipsoids, which can change their color according to the

viewing direction, are rendered to recreate a scene for the use case of novel view synthesis. This would be interesting to combine with our approach, so that we would require multiple image sequences as input that show the same scene at different times. From that, we could create a 3D representation using Gaussian splats that can then be adjusted to depict the scene at different time points in a smooth animation.

While there are limitations to our approach that need to be addressed in future work, our method fulfills our established requirements and goals and exceeds our expectations with regard to computational efficiency. We implement a fast alignment of multiple large point clouds to enable an insightful comparison and intuitive interpretation of the spatial variance. Our proposed workflow allows for a simple exploration of the data that highlights areas of significant difference effectively. There exist many exciting opportunities to continue this work, and our approach delivers a strong foundation to build upon for the comparison of vast spatial datasets.

Overview of Generative AI Tools Used

As an automatic grammar and spell checker, Grammarly was used. For the translation of individual words from German to English and vice versa, DeepL was used. To create suggestions for synonyms for individual words, as well as for prompts regarding the LaTeX API, to find commands for mathematical formulas and the formatting of figures, ChatGPT was used.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AKS+19] Aleksandr Amirkhanov, Ilona Kosiuk, Peter Szmolyan, Artem Amirkhanov, Gabriel Mistelbauer, M. Eduard Gröller, and Renata G. Raidou. Manylands: A journey across 4d phase space of trajectories. *Computer Graphics Forum*, 38(7):191–202, 2019.

[AYH+24] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, volume 2, page 929–947. Association for Computing Machinery, 2024.

[BBP09] Stef Busking, Charl P. Botha, and Frits H. Post. Direct visualization of deformation in volumes. *Computer Graphics Forum*, 28(3):799–806, 2009.

[BBP10] Stef Busking, Charl P. Botha, and Frits H. Post. Dynamic multi-view exploration of shape spaces. *Computer Graphics Forum*, 29(3):973–982, 2010.

[BD23] Nicolas Bonneel and Julie Digne. A survey of optimal transport for computer graphics and computer vision. *Computer Graphics Forum*, 42(2):439–460, 2023.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[Ber08] Oliva Cachafeiro Bernal. *African sculpture in terracotta and stone: Alberto Jiménez-Arellano Alonso Foundation Collection*. Alberto Jiménez-Arellano Alonso Foundation and University of Valladolid, 2008.

[BM92] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[BMTY05] M Faisal Beg, Michael I Miller, Alain Trouvé, and Laurent Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *International Journal of Computer Vision*, 61:139–157, 2005.

[BPRS17] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, January 2017.

[BRS+24] Lukáš Brodský, Samo Rusnák, Susanne Schmidt, Vít Vilímek, Roberto Azzoni, Marcus Nüsser, Gianluca Tronti, Jan Kropáček, and Aayushi Pandey. Interannual spatio-temporal evolution of the supraglacial lakes on the belvedere glacier between 2000 and 2023. *AUC GEOGRAPHICA*, 59:1–16, 12 2024.

[CEW17] Joshua Cates, Shireen Elhabian, and Ross Whitaker. Chapter 10 - shapeworks: Particle-based shape correspondence and visualization software. In *Statistical Shape and Deformation Analysis*, pages 257–298. Academic Press, 2017.

[CFG+15] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015.

[CFG+21] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. Kernel operations on the gpu, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6, 2021.

[Cor14] Omar Cornut. Dear imgui, a bloat-free graphical user interface library for c++. `https://github.com/ocornut/imgui`, 2014. Accessed 2025-04-16.

[DGIP21] Carlo Iapige De Gaetani, Francesco Ioli, and Livio Pinto. Aerial and uav images for photogrammetric analysis of belvedere glacier evolution in the period 1977–2019. *Remote Sensing*, 13(18), 2021.

[Dom16] Szabolcs Dombi. Moderngl, high performance python bindings for opengl 3.3+. `https://github.com/moderngl/moderngl`, 2016. Accessed 2025-04-16.

[DYDZ22] Bailin Deng, Yuxin Yao, Roberto M Dyke, and Juyong Zhang. A survey of non-rigid 3d registration. In *Computer Graphics Forum*, volume 41, pages 559–589. Wiley Online Library, 2022.

[EL22] Marina Evers and Lars Linsen. Multi-dimensional parameter-space partitioning of spatio-temporal simulation ensembles. *Computers & Graphics*, 104:140–151, 2022.

[FCF+24] Ioli F., De Gaetani C., Barbieri F., Gaspari F., Pinto L., and Rossi L. Belvedere glacier long-term monitoring open data (2.0). `https://doi.org/10.5281/zenodo.10817029`, 2024. Accessed 2025-04-19.

[FCVP17] Jean Feydy, Benjamin Charlier, François-Xavier Vialard, and Gabriel Peyré. Optimal transport for diffeomorphic registration. In *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017: 20th International Conference, Quebec City, QC, Canada, September 11-13, 2017, Proceedings, Part I*, page 291–299. Springer-Verlag, 2017.

[FE17a] S. Frey and T. Ertl. Fast flow-based distance quantification and interpolation for high-resolution density distributions. In *Proceedings of the European Association for Computer Graphics: Short Papers*, page 37–40. Eurographics Association, 2017.

[FE17b] Steffen Frey and Thomas Ertl. Progressive direct volume-to-volume transformation. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):921–930, 2017.

[Fey20] Jean Feydy. Geometric data analysis, beyond convolutions. *Applied Mathematics*, 3, 2020.

[Fey22] Jean Feydy. Geomloss. `https://github.com/jeanfeydy/geomloss`, 2022. Accessed 2025-04-16.

[FKRW17] Florian Ferstl, Mathias Kanzler, Marc Rautenhaus, and Rüdiger Westermann. Time-hierarchical clustering and visualization of weather forecast ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):831–840, 2017.

[FL13] Robert Andrew Foley and Roger Lewin. *Principles of Human Evolution*. Wiley-Blackwell, 2nd edition, 2013.

[FMCM+21] Katarína Furmanová, Ludvig P. Muren, Oscar Casares-Magaz, Vitali Moiseenko, John P. Einck, Sara Pilskog, and Renata G. Raidou. Previs: Predictive visual analytics of anatomical variability for radiotherapy decision support. *Computers & Graphics*, 97:126–138, 2021.

[For19] Einar Forselv. moderngl-window, a cross-platform windowing/utility library for moderngl. `https://github.com/moderngl/moderngl-window`, 2019. Accessed 2025-04-16.

[FSV+19] Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouve, and Gabriel Peyré. Interpolating between optimal transport and mmd using sinkhorn divergences. In *The 22nd international conference on artificial intelligence and statistics*, pages 2681–2690, 2019.

[FYC+22] Ben Fei, Weidong Yang, Wen-Ming Chen, Zhijun Li, Yikang Li, Tao Ma, Xing Hu, and Lipeng Ma. Comprehensive review of deep learning-based 3d point cloud completion processing and analysis. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):22862–22883, 2022.

[GCB+19] Aude Genevay, Lénaic Chizat, Francis Bach, Marco Cuturi, and Gabriel Peyré. Sample complexity of sinkhorn divergences. In *The 22nd international conference on artificial intelligence and statistics*, pages 1574–1583. PMLR, 2019.

[GM98] Ulf Grenander and Michael I. Miller. Computational anatomy: An emerging discipline. *Quarterly of Applied Mathematics*, 56(4):617–694, 1998.

[GM24] Daniel Girardeau-Montaut. Cloudcompare. `http://www.cloudcompare.org`, 2024. Version 2.13.2.

[Gol11] Albert Goldbeter. A model for the dynamics of bipolar disorders. *Progress in Biophysics and Molecular Biology*, 105(1):119–127, 2011.

[GVOC18] Xuefeng Guan, Peter Van Oosterom, and Bo Cheng. A parallel n-dimensional space-filling curve library and its application in massive point cloud management. *ISPRS International Journal of Geo-Information*, 7(8), 2018.

[HK15] Max Hermann and Reinhard Klein. A visual analytics perspective on shape analysis: State of the art and future prospects. *Computers & Graphics*, 53:63–71, 2015.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[HR07] Jeffrey Heer and George Robertson. Animated transitions in statistical data graphics. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1240–1247, 2007.

[HSSK16] Max Hermann, Anja C. Schunke, Thomas Schultz, and Reinhard Klein. Accurate interactive visualization of large deformations and variability in biomedical image ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):708–717, 2016.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[IBC+22] Francesco Ioli, Alberto Bianchi, Alberto Cina, Carlo De Michele, Paolo Maschio, Daniele Passoni, and Livio Pinto. Mid-term monitoring of glacier's variations with uavs: The example of the belvedere glacier. *Remote Sensing*, 14(1), 2022.

[IDG+24] Francesco Ioli, Niccolò Dematteis, Daniele Giordan, Francesco Nex, and Livio Pinto. Deep learning low-cost photogrammetry for 4d short-term glacier dynamics monitoring. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 92(6):657–678, Dec 2024.

[Jaw18] Michał Jaworski. pyimgui, python bindings for the dear imgui c++ library. `https://github.com/pyimgui/pyimgui`, 2018. Accessed 2025-04-16.

[Kan60] Leonid V Kantorovich. Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422, 1960.

[KCK17] Kyungyoon Kim, John V. Carlis, and Daniel F. Keefe. Comparison techniques utilized in spatial 3d and 4d data visualizations: A survey and future directions. *Computers & Graphics*, 67:138–147, 2017.

[KKLD23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

[KMP07] Martin Kilian, Niloy J. Mitra, and Helmut Pottmann. Geometric modeling in shape space. *ACM Trans. Graph.*, 26(3):64–es, July 2007.

[KPB+22] Abderrazzaq Kharroubi, Florent Poux, Zouhair Ballouch, Rafika Hajji, and Roland Billen. Three dimensional change detection using point clouds: A review. *Geomatics*, 2(4):457–485, 2022.

[Kra07] Karl Kraus. *Photogrammetry: Geometry from Images and Laser Scans*. De Gruyter, 2007.

[KTT+24] Abdelwahed Khamis, Russell Tsuchida, Mohamed Tarek, Vivien Rolland, and Lars Petersson. Scalable optimal transport methods in machine learning: A contemporary survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20, 2024.

[kul24] kulturpool. Heidentor. `https://kulturpool.at`, 2024. Accessed 2025-04-19, Relevant sections: "The European Twin It Campaign", "The Heidentor of Petronell-Carnuntum", "Albertina object DG1936".

[LAA+13] Tatiana Landesberger, Gennady Andrienko, Natalia Andrienko, Sebastian Bremm, Matthias Kirschner, Stefan Wesarg, and Arjan Kuijper. Opening up the "black box" of medical image segmentation with statistical shape models. *Vis. Comput.*, 29(9):893–905, September 2013.

[Lie11] Daniel E. Lieberman. *The Evolution of the Human Head*. Harvard University Press, 2011.

[LLPY07] Claes Lundström, Patric Ljung, Anders Persson, and Anders Ynnerman. Uncertainty visualization in medical volume rendering using probabilistic animation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1648–1655, 2007.

[MS13] Ashley H. McKeown and Ryan W. Schmidt. Chapter 12 - geometric morphometrics. In Elizabeth A. DiGangi and Megan K. Moore, editors, *Research Methods in Human Skeletal Biology*, pages 325–359. Academic Press, 2013.

[Mun15] T. Munzner. Section 7.2 - Eyes Beat Memory. In *Visualization Analysis and Design*, AK Peters Visualization Series, pages 119–121. CRC Press, 2015.

[OALRRRSS23] Carlos J. Ogayar-Anguita, Alfonso López-Ruiz, Antonio J. Rueda-Ruiz, and Rafael J. Segura-Sánchez. Nested spatial data structures for optimal indexing of lidar data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 195:287–297, 2023.

[OZI+22] Benjamin A. Orkild, Brian Zenger, Krithika Iyer, Lindsay C. Rupp, Majd M Ibrahim, Atefeh G. Khashani, Maura D. Perez, Markus D. Foote, Jake A. Bergquist, Alan K. Morris, Jiwon J. Kim, Benjamin A. Steinberg, Craig Selzman, Mark B. Ratcliffe, Rob S. MacLeod, Shireen Elhabian, and Ashley E. Morgan. All roads lead to rome: Diverse etiologies of tricuspid regurgitation create a predictable constellation of right ventricular shape changes. *Frontiers in Physiology*, 13, 2022.

[PBM20] Gilles Puy, Alexandre Boulch, and Renaud Marlet. Flot: Scene flow on point clouds guided by optimal transport. In *European conference on computer vision*, pages 527–544. Springer, 2020.

[PC19] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[PSCC18] Gianpaolo Palma, Manuele Sabbadin, Massimiliano Corsini, and Paolo Cignoni. Enhanced visualization of detected 3d geometric differences. *Computer Graphics Forum*, 37(1):159–171, 2018.

[RBGR18] Oliver Reiter, Marcel Breeuwer, Meister Eduard Gröller, and Renata Georgia Raidou. Comparative visual analysis of pelvic organ segmentations. In *Computer Graphics Forum (Proceedings of EuroVis 2018)*, pages 37–41. The Eurographics Association, 2018.

[RCMA+18] R.G. Raidou, O. Casares-Magaz, A. Amirkhanov, V. Moiseenko, L.P. Muren, J.P. Einck, A. Vilanova, and M.E. Gröller. Bladder runner: Visual analytics for the exploration of rt-induced bladder toxicity in a cohort study. *Computer Graphics Forum*, 37(3):205–216, 2018.

[Rij25] Rijkswaterstaat. Actueel hoogtebestand nederland. `https://www.ahn.nl`, 2025. Accessed: 2025-04-12.

[Roh90] F. James Rohlf. Morphometrics. *Annual Review of Ecology and Systematics*, 21:299–316, 1990.

[RTC17] Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2), 2017.

[SAR23] Marwin Schindler, Aleksandr Amirkhanov, and Renata Raidou. Smoke surfaces of 4d biological dynamical systems. In *VCBM 2023: Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 93–97. The Eurographics Association, September 2023.

[Sch19] Bernhard Schmitzer. Stabilized sparse scaling algorithms for entropy regularized transport problems. *SIAM Journal on Scientific Computing*, 41(3):A1443–A1481, 2019.

[SF16] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[SFL+21] Zhengyang Shen, Jean Feydy, Peirong Liu, Ariel Hernán Curiale, Rubén San José Estépar, Raúl San José Estépar, and Marc Niethammer. Accurate point cloud registration with robust optimal transport. In *Neural Information Processing Systems*, 2021.

[SFR00] D J Simons, S L Franconeri, and R L Reimer. Change blindness in the absence of a visual disruption. *Perception*, 29(10):1143–1154, 2000.

[SFV+23] Thibault Séjourné, Jean Feydy, François-Xavier Vialard, Alain Trouvé, and Gabriel Peyré. Sinkhorn divergences for unbalanced optimal transport, 2023.

[Ske21a] Sketchfab. Digital atlas of ancient life. `https://skfb.ly/owLJn`, 2021. Accessed 2025-04-19.

[Ske21b] Sketchfab. Risd nature lab. `https://skfb.ly/oOWLA`, 2021. Accessed 2025-04-19.

[Ske24] Sketchfab. Global digital heritage and gdh-afrika. `https://sketchfab.com/GlobalDigitalHeritage`, 2024. Accessed 2025-04-19.

[SOW20] Markus Schütz, Stefan Ohrhallinger, and Michael Wimmer. Fast out-of-core octree generation for massive point clouds. *Computer Graphics Forum*, 39(7):1–13, nov 2020.

[SPA+14] Johanna Schmidt, Reinhold Preiner, Thomas Auzinger, Michael Wimmer, M. Eduard Gröller, and Stefan Bruckner. Ymca — your mesh comparison application. In *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 153–162, 2014.

[SvdW15] Nathaniel Smith and Stefan van der Walt. A better default colormap for matplotlib. `https://bids.github.io/colormap/`, 2015. Accessed: 2025-04-10.

[SZFP16] Johannes L. Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *Computer Vision – ECCV 2016*, pages 501–518. Springer International Publishing, 2016.

[TMB02] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: can it facilitate? *International journal of human-computer studies*, 57(4):247–262, 2002.

[VdMH08] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[VEFC17] Claudio Vanneschi, Matthew Eyre, Mirko Francioni, and John Coggan. The use of remote sensing techniques for monitoring and characterization of slope instability. *Procedia Engineering*, 191:150–157, 2017. ISRM European Rock Mechanics Symposium EUROCK 2017.

[VM10] G. Vosselman and Hans-Gerd Maas, editors. *Airborne and terrestrial laser scanning*. Whittles Publishing, 2010.

[WFG+19] Johannes Weissenböck, Bernhard Fröhler, Eduard Gröller, Johann Kastner, and Christoph Heinzl. Dynamic volume lines: Visual comparison of 3d volumes through space-filling curves. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1040–1049, 2019.

[WHLS19] Junpeng Wang, Subhashis Hazarika, Cheng Li, and Han-Wei Shen. Visualization and visual analysis of ensemble data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 25(9):2853–2872, 2019.

[Wra25] Harald Wraunek. Wien-3d, niederösterreich-3d, burgenland-3d. `https://wien.denkmal-3d.at/`, `https://noe-3d.at`, `https://burgenland.denkmal-3d.at/`, 2025. Accessed 2025-04-19.

[You10] L. Younes. *Shapes and Diffeomorphisms*. Applied Mathematical Sciences. Springer Berlin Heidelberg, 2010.

[ZF03] Barbara Zitová and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, 2003.

[ZP19] Yoav Zemel and Victor M. Panaretos. Fréchet means and Procrustes analysis in Wasserstein space. *Bernoulli*, 25(2):932 – 976, 2019.