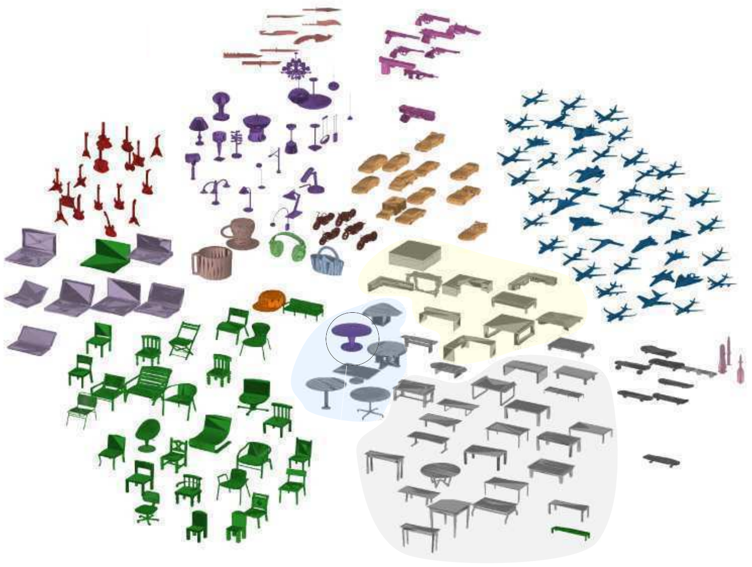
ANALYSIS AND EXPLORATION OF LARGE 3D SHAPE DATABASES



XINGYU CHEN

Cover: Projection of a 3D shape database using features extracted by deep learning.

Analysis and Exploration of Large 3D Shape Databases

Xingyu Chen
PhD Thesis

university of groningen

Analysis and Exploration of Large 3D Shape Databases

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. C. Wijmenga
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Monday 14 June 2021 at 12.45 hours

by

Xingyu Chen

born on July 23rd, 1993
in Hunan, China

ABSTRACT

Many applications generate digital descriptions of 3D shapes. As more methods emerge for the acquisition, creation, and processing of such content, so do grow the size and complexity of collections of 3D shapes, generically known as shape databases. Exploring the wealth of content present in such databases is an increasingly difficult, and important, problem, especially for the case of unannotated databases with limited search functionality.

We identified three key problems in this field: exploration, examination, and analysis of 3D shape data. This thesis discusses these problems and proposes several solutions to each of them, as follows. First, we propose methods for creating visual overviews of large 3D shape collections based on feature engineering and also on deep learning features. Our methods can automatically organize hundreds of thousands of 3D shapes by their similarity and also support incremental updates of shape databases. Secondly, we propose a novel method for the examination of individual shapes, with the aim of specifying rotations of 3D shapes, using axes inferred from the visible shape structure extracted using silhouette skeletons. An executed user study shows that, when combined with traditional viewpoint specification mechanisms, our method reduces task completion times and increases user satisfaction, while not introducing additional costs. Finally, we present a method for analyzing families of structurally related shapes by computing consistent curve skeletons from the given families to induce semantic information on skeleton branches. The computed so-called co-skeletons also increase the robustness of curve skeleton computation, lifting the skeleton simplification power from individual shapes to shape families.

SAMENVATTING

Veel toepassingen genereren beschrijvingen van 3D vormen. Als meer methoden beschikbaar komen voor de acquisitie, creatie en verwerking van dergelijke informatie zo groeien ook de grootte en complexiteit van verzamelingen van 3D vormen, algemeen bekend als vormdatabases. Het exploreren van de inhoud van dergelijke databases is een steeds moeilijker en tegelijkertijd belangrijk probleem, in het bijzonder voor het geval van databases zonder annotaties en met beperkte zoekfuncties.

We identificeren drie markante problemen in deze context: exploratie, examinatie, en analyse van 3D vormgegevens. Dit proefschrift discussieert deze problemen en stelt verscheidene oplossingen voor elk daarvan zoals volgt. Ten eerste worden methodes voorgesteld voor het creëren van visuele samenvattingen van grote 3D vormverzamelingen met behulp van *feature engineering* en ook het diep leren van features. Deze methodes kunnen honderdduizenden 3D vormen automatisch organiseren volgens hun similariteit en ook staan incrementele veranderingen van de vormdatabase toe. Ten tweede wordt een nieuwe methode gepresenteerd voor het examineren van individuele vormen, door de specificatie van rotaties van deze vormen langs assen die berekend worden vanuit de zichtbare vormstructuur door midden van silhouetskeletten. Een gebruikersstudie laat zien dat onze methode, wanneer gecombineerd met klassieke kijkpuntspecificatiemechanismen, de nodige tijd om een taak te voltooien reduceert en de gebruikerstevredenheid bevordert zonder extra kosten te introduceren. Ten slotte wordt een methode gepresenteerd voor de analyse van families van door hun structuur gerelateerde vormen, die consistente curveskeletten berekent voor een gehele familie om semantische informatie te induceren langs de skelettakken. De zogenaamde co-skeletten die worden berekend doen ook de robuustheid van curveskeletberekening toenemen, zodat de versimpelingskracht van skeletten voor individuele vormen wordt nu toegekend aan gehele vormfamilies.

PUBLICATIONS

This thesis is the result of the following publications:

- Visual Exploration of 3D Shape Databases Via Feature Selection [28]
- Co-skeletons: Consistent curve skeletons for shape families [176]
- Interactive Axis-based 3D Rotation Specification using Image Skeletons [186]

This thesis is also based on the following papers which are currently under review:

- Scalable Visual Exploration of 3D Shape Databases via Feature Synthesis and Selection [29]
- Skeleton-and-Trackball Interactive Rotation Specification for 3D Scenes [185]

The following publications constitute prior work of the author. While not explicitly included in this thesis, the material discussed in these publications has outlined the necessity for better tools for exploration and examination of large and complex collections of multi-media items. This thesis further examines this problem for the specific context of 3D shape databases.

- A fuzzy ontology for geography knowledge of China's College Entrance Examination [27]
- Classification of medical consultation text using mobile agent system based on Naïve Bayes classifier [26]
- Question answering over knowledgebase with attention-based LSTM networks and knowledge embeddings [22]
- Tree-LSTM Guided attention pooling of DCNN for semantic sentence modeling [21]

CONTENTS

1 INTRODUCTION

The world we live in consists of three-dimensional shapes; they are the basic elements of our life. We see, touch, and interact with them at every moment. Although we know how to interact with 3D shapes intuitively, the details of their structure, topology, and properties are still worth for us to study. Apart from their ubiquity in the real world, 3D shapes are also an essential ingredient of the digital ones. In recent years, significant advances in data storage, computational speed, cloud computing, and Internet speed have made it possible to build more and more applications that revolve — metaphorically but also literally — around 3D scenes. As a consequence, 3D shapes are ubiquitous in many application domains. For example, 3D video games are now the main type of computer games; they are seen by numerous users as more vivid and attractive than 2D or text-based games. The best-selling video games on personal computers, mobile phones, and consoles last year are all 3D games [73]. Another case in point is the spread of virtual reality (VR) [12, 188]. People can experience a completely different world from the real one with a VR headset. This technique is very useful in gaming, film, and housing industries, and its advent has spawned an increased need for the creation and management of 3D shapes.

3D content creation and acquisition technologies have made big progress in the last decades. As a consequence of the increasing number of 3D shapes being created, manipulated, and exchanged, *databases* (also called collections or repositories) of 3D shapes have emerged [150, 165]. In the beginning, such databases were quite small collections of tens to hundreds of shapes, typically dedicated to a single application and used by a few specialists. Over time, these have evolved into large databases of hundreds of thousands of shapes, representing objects of different types, collected from various sources, stored in various formats, and used by thousands or even millions of people with different training and interests. Apart from the explicit creation of 3D shapes by digital modeling and acquisition (scanning), large collections of 3D shapes are created implicitly in medical science, for instance when extracting various anatomical structures from 3D Computer Tomography (CT) or Magnetic Resonance Imaging (MRI) scans [79, 105].

Regardless of their origin and creation process, 3D shapes typically come with additional data attributes, such as type of object being represented (*e.g.*, cars, planes, furniture, natural shapes, specific kinds of anatomical tumors), size of object, quality of the representation (*e.g.* resolution of a mesh), and provenance (author of the data). When available, such attributes can be also stored into shape databases, alongside the actual geometry that describes the shapes [126]. As such, a shape database is essentially a large and complex multidimensional dataset, where the observation, or sample, is the shape plus all its measured or associated data; and each measurement type represents a separate dimension.

Without any doubt, the emergence of (public) large 3D shape databases represents a major help for all stakeholders interested in creating applications that revolve around 3D content. On the other hand, the size and complexity nature of such databases also creates new problems and challenges. As such, the problems of *analyzing* and *exploring* such 3D shape databases have grown into a research field of its own. To illustrate this, we outline next several instances of the above joint analysis-and-exploration problems.

Search and exploration: A typical starting problem for users facing such 3D shape databases is to *find* the shapes they are interested in among the many ones present in the database. After all, the main purpose of such databases is precisely to facilitate *reuse* of 3D shapes, and for this to occur, one needs to find the shapes one can reuse. Current 3D shape database systems often offer search mechanisms for users to search shapes in their collections. Many databases also provide navigation systems to help users exploring their collections. Each of these two main methods has its strengths and weaknesses, as follows.

A *search* system will list the objects it finds (retrieves) that are related to the search information (query specification) supplied by its users. Typical search information includes keywords, sketches [37], and sample shapes [150]. Every time when a user submits a search request, the system queries the database to find – or retrieve – the shapes that meet the search condition within a certain tolerance, and then presents these results to the user. If the system cannot find any matching shape, it tells the user this search request failed and asks the user to try to search with other information. Relaxing the search tolerance increases the likelihood of returning more results to the user, but also the likelihood that some of these returned results will not be relevant. In general, search methods are efficient when users know what they are looking for, know how to describe their requirements, have an idea what shapes a database contains, and have a reasonable understanding of how the search is actually executed. Simply put, when users are familiar with the target databases and their underlying structure and search mechanisms, a search system is a very efficient tool for finding the shapes one is looking for.

Different from the search system, *exploration* methods are more suitable for non-targeted search. Such methods come into play in situations when the users do not know their desired shapes exactly; know what they are looking for but not how to specify the query via the available search mechanisms; observe that targeted search does not effectively return what they are looking for; or, more generally even, are interested to browse a shape database to get familiar with its contents, without having a specific target object in mind. As such, exploration differs from search or querying. In both cases, the process is supposed to deliver some set of shapes deemed of interest by the user. Still, search has a clear way of deciding what types of shapes are to be returned in a query, based on a set of explicit search criteria. In contrast, exploration does not typically use concrete criteria to say, beforehand, whether a shape is or is not of interest. The set of shapes found of interest that are returned from exploration is, often, a process of the insights discovered or found *during* exploration itself by the user. Exploration mechanisms for 3D shape databases are typically created around thumbnail galleries and hierarchies [125]. Users can explore the database, which often use these two methods — hierarchies and thumbnails — combined, like browsing photos in their computer file system or navigating the products on a web shopping portal. However, free exploration becomes difficult when 3D shape databases become large, that is, contain thousands of shapes or more. Hierarchical organization helps this, but it also introduces a limitation, as all available content has to be organized along the lines of a single, or a few, predefined hierarchies. The analogy with a file system holds here: It is difficult for users to create a comprehensive *overview* of all their (tens of) thousands of files [16].

Both search and exploration are important and useful tools for effectively and efficiently (re)using the available content in 3D shape databases. They are also complementary mechanisms, used in different situations, as outlined above. However, we note that, whereas 3D shape *retrieval* has been extensively explored [90, 133, 137, 150, 165], the area of 3D shape database *exploration*, albeit older, has received relatively less attention. Easily creating an overview of a large, possibly un-annotated and unstructured, 3D shape database, that groups shapes together by similarity in ways that enable users to understand at a glance what the database contains, is still challenging.

Shape examination: At a lower level of detail following the exploration of an entire 3D shape database, one is interested in studying a single (or a few) 3D shapes. This serves multiple purposes, *e.g.*, finding whether the shape corresponds to the one that was searched for in the database by examining the shape's fine-grained details; and finding aspects related to the quality of the shape representation, such as 3D mesh quality (or lack thereof). Understanding these aspects in detail is next important for the decision whether the shape is indeed directly suited for further use in a specific application; if it needs preprocessing before such usage, *e.g.*, mesh repairing to improve grading or closing holes; or whether the shape is actually unsuitable for that application and a new search in the 3D database must be executed. More generally put, the examination follows the exploration stage in 3D shape databases much like in other contexts in the information visualization arena, following Shneiderman's mantra [135] "Overview first, zoom and filter, then details-on-demand". In our context, the overview stage relates to the exploration; the zoom and filter to targeted search; and the details-on-demand to the examination stage, respectively.

Shape examination takes usually the form of viewing the shape – or few selected shapes – from various viewpoints and with various rendering modes. Since the advent of 3D graphics, many methods have been proposed for manipulating the viewpoint to explore virtual worlds, such as the virtual trackball and its enhancements [56]. However, manipulating 3D content using a 2D screen — the most typical setting — is still difficult. In this context, we ask ourselves if we can improve this process by providing virtual viewpoint manipulation tools that exploit information present in the 3D shapes. Also, it is important to note that shape examination takes place usually *after* a search and/or exploration process was first executed to find the (small) set of candidates of interest to examine.

Shape analysis: Once one has decided that a shape (or a small set of shapes) found in a 3D database is suitable for the application at hand, typically by examining them in detail as outlined above, the shape typically undergoes some form of *processing*. This can take many forms. For instance, one can *analyze* the shape by extracting metrics relevant for understanding the shape quantitatively or for comparing various shapes, especially important in engineering and medical contexts [83]; or one can *process* the shape to remove noise, improve its meshing quality, or otherwise obtain different shapes from a base one [172]. In contrast to all operations listed above – search, exploration, and examination – analysis is the only operation that can actually *modify* a shape. Analysis is most closely connected to examination, as one needs to study the shape in detail, both before and after the analysis.

A particularly interesting topic in this context is the *joint* processing of sets of related shapes, such as obtained as result of a 3D shape database query. Since such shapes are related — by their common aspects that have been used by the query to retrieve them in the first place — it is arguable that such commonality can be also exploited when processing them. Simply put, we would, in that case, process the *set*, or family, of shapes, rather than each shape individually. This can have several advantages. For example, for delicate operations that strongly depend on the quality of a 3D shape — and which may fail or produce otherwise low-quality results from a single poor-quality shape — we can use the redundancy and variability present in a shape collection to make them more robust. Moreover, the results of these operations next describe the entire collection rather than individual shapes. This can help further the processing of large shape collections in terms of computational scalability. However, shape retrieval (searching) and shape analysis (processing) are typically treated separately in classical 3D pipelines.

We believe that a joint approach is of added value and we aim to explore such approaches next.

Summarizing the above, we can now introduce our main research question:

How to help users in exploring, examining, and analyzing shapes and their families present in large 3D collections?

Note that, in practice (and thus in solving our research question), the three above-mentioned operations — exploration, examination, and analysis — are typically executed several times each, and in various orders. For example, one can do a quick exploration (or search) to find shapes of interest in a database; then, examine these in detail, and finally decide to select a few for further analysis. However, one can also integrate the analysis step into the search process, by *e.g.* extracting shape features that are used by the search mechanism in contexts such as content-based shape retrieval. As such, in the following, we do not assume a particular order in which these three tasks are needed to be executed.

We approach our research question along the three dimensions outlined above — search and exploration, shape examination, and shape analysis — by various parts of our research. We present these next along the structure of our thesis, as follows.

Chapter 1, the current chapter, presents the research object of our work, which includes exploration of 3D shape databases, 3D shape examination, and 3D shape analysis.

Chapter 2 presents the related work of our thesis, which includes research concerning 3D shape database exploration, 3D shape examination, and 3D shape analysis using skeletons.

Chapter 3 presents our solution to build an *exploration* system for large 3D shape databases. We introduce here several 3D shape properties that can be also used in 3D shape analysis such as discussed later on in Chapter 6. We then propose several methods that use these properties to create summarizations of 3D shape database for exploration. Our proposal allows one to easily create visual overviews of 3D shape databases where structurally similar objects are placed close to each other. Additionally, our proposal allows the user to control the way that such overviews are generated in a visual analytics manner, that is, by novel mechanisms for interactively exploring and selecting the way in which shape properties, computed from their actual descriptions, influence the creation of the overview.

Chapter 4 presents an improved solution for the *exploration* problem studied in Chapter 3. Rather than precompute a set of engineered feature descriptors, as the approach in Chapter 3 does, we now use a deep learning set-up for reusing feature vectors computed during the training of a 3D shape classifier. Our set-up uses a recent deep learning method for constructing 2D projections with attractive scalability, out-of-sample, and stability properties and combines it in a novel way with another recent deep learning method designed for feature extraction for classification. We show how visual overviews can be created in a significantly more computationally scalable way from high-dimensional feature vectors using our combined deep learning approach. We explore several architectures and training modes for our approach. Our proposal is demonstrated in terms of scalability, ease of use, and robustness on large-scale 3D shape databases.

Chapter 5 turns our focus to the *examination* part of our research question. We show here how 3D rotations can be easily specified for arbitrary 3D shapes using a single click operation in a classical 2D projection view. In contrast to other 3D rotation specifications, we use information available from the shape's projection, computed and extracted using the so-called shape skeleton. This way, rotation axes automatically 'latch' to the visible parts of the examined 3D shapes. To our knowledge, this is the first time that 2D skeletons have been used to assist the interactive creation of rotation axes for 3D shapes. We include a user study that compares our interactive rotation specification technique with classical virtual trackball rotation, showing that our technique augments the added value of virtual trackball.

Chapter 6 turns to the third and last part of our research question, namely shape *analysis*. As Chapter 5 shows that 2D skeletons are useful descriptors for the examination of 3D shapes, we now consider the computation of 3D curve skeletons for the same shapes. In line with the search paradigm outlined earlier in this chapter, we ask ourselves whether one can compute such 3D curve skeletons jointly for a set of similar shapes, such as delivered by the result of a query on a 3D shape database. We present a method that computes such novel joint descriptors which we call co-skeletons. We show that computing co-skeletons, as opposed to individual 3D curve skeletons for each shape in a collection, comes with added value in terms of capturing the essence of the shapes present in the collection and, at the same time, being more robust to small-scale noise or variability present in the individual shapes. Also, we show the added value of co-skeletons for applications such as shape co-deformation and co-segmentation.

Chapter 7 concludes this thesis by summarizing our contributions and also sketching possible directions for future work.

2 RELATED WORK

As outlined in Chapter 1, analyzing and understanding large 3D shape databases and 3D shapes is an important topic with many applications in a wide range of disciplines. Given this, it is not surprising that related work spans a wide set of sub-disciplines in computer science, ranging from machine learning and geometry processing to computer graphics, human-computer interaction, and information visualization. Given the sheer size of these fields, we do not aim here to provide a complete overview of related work to shape processing, exploration, and examination. Rather, we focus our discussion of related work to the most important classes of techniques in these respective fields, which also relate to our current work. Additional related work will be introduced in the next chapters in the context of the more specific technical topics discussed there individually. As such, this chapter should be seen as a reading guideline that introduces the reader to the more specific technical information discussed in the following chapters.

This chapter is structured as follows. Section 2.1 outlines fundamental concepts related to 3D shape representation. Section 2.2 presents related work in the direction of the exploration of 3D shape collections. Section 2.3 presents work related to the task of examining individual 3D shapes in interactive settings. Finally, Section 2.4 introduces notations and work related to the context of shape descriptors, which are key to our own work described in the next chapters.

2.1 SHAPE REPRESENTATION

3D shapes can be stored in computer systems in numerous formats. Such as polygonal mesh, point cloud, CGS (Constructive Solid Geometry) [122], octrees [115], splines, etc. By using these formats, we can display 3D shapes on screens. When doing 3D shape analysis tasks, each of them has its advantages. For instance, when computing the volume or the surface area of a recorded 3D shape, the polygonal mesh is more convenient than CSG; when doing 3D shape segmentation, CSG is more convenient than the polygonal mesh. In this thesis, we mainly use the polygonal mesh and point cloud to represent 3D shapes.

Polygonal mesh: Encoding the surface geometry of 3D models by their approximation surfaces is a very popular way to store 3D models in computer systems. The polygonal mesh method covers a 3D model's surface by a mesh that consists of many small polygons. The vertices of the polygons and optionally the outward normal vector of the vertices are recorded. As such, a 3D shape can be represented as its surface mesh m which is the combination of a group of polygons. Each polygon can be defined by its vertices. As such, the polygonal mesh of a 3D shape can be represented as $m = (V = [v_i], F = [f_i])$, which is an array of vertices $v_i \in \mathbb{R}^3$ and a collection of (polygonal) faces f_i, typically an array of indices into the vertex array.

The polygonal mesh approximates the real surface geometry of 3D models. Therefore, meshes are not precise representations of 3D shapes. When the polygons get smaller and finer, the mesh is more precise to approximate the model. On the other hand, the smaller polygons size means the larger quantity of polygons — a larger number of vertices and faces. This will take more storage space and will take more time for rendering or analysis tasks.

Point clouds: Sampling points on the surface of 3D models is another method we apply to represent shapes. A 3D shape is represented by a point cloud $P = \{x_i\} \in \mathbb{R}^3$ which is a set of points in space. Point clouds can be directly rendered and inspected [128], and they can be converted into surface meshes. It is a very simple and powerful way to represent shapes. Recently, it received a lot of attention since it can be easily used in various deep learning methods [53].

2.2 3D SHAPE COLLECTION EXPLORATION

As outlined in Chapter 1, exploring 3D shape collections can be structured, from a task perspective, into targeted exploration and free exploration. Targeted exploration corresponds to the goal of finding shapes that match specific characteristics of interest to the user. Free exploration, or browsing, corresponds to the goal of finding what a certain 3D shape database contains in general and/or finding how such a database is organized. From the perspective of techniques used, exploration of 3D shape databases can be structured along three modalities, as follows.

Keyword search uses words to search for shapes whose annotation data — also called metadata — contains those words. From all exploration mechanisms, keyword search is the simplest to support, and therefore the oldest and most widespread form of searching for 3D content, present in many shape databases, such as TurboSquid [164] and Aim@Shape [1], to mention just a few. Such databases allow providers to upload models with associated keywords for subsequent search. However, keyword lists are only weakly structured, possibly containing redundant or vague keywords, potentially added this way to increase exposure rate. Besides general-purpose databases of this type, more specialized ones exist, such as containing 3D shapes related to space exploration [104]. Overall, keyword search is popular and widely supported, but works best for targeted searches performed by users aware of a database's organization, requires a good annotation with specific keywords, and is less effective for the task of free exploration or browsing. Given these limitations of keyword search, but also the fact that many solutions are already established for this type of exploration, we will not focus further in our work on this modality.

Hierarchical exploration systems organize shapes along with different criteria, following an existing taxonomy of the targeted 3D shape universe at hand. Such systems support exploration (apart from the keyword search) by allowing users to browse the hierarchy, with shapes or shape categories depicted by thumbnails, much like when exploring a file system. Examples of such systems are the Princeton Shape Benchmark [134], Aim@Shape [1], or the ITI 3D search engine [66] that allows browsing multiple hierarchically-organized shape databases. Hierarchy browsing supports browsing better than keyword-based search. Yet, it typically only allows examining a single path (shape subset) at a time, and thus cannot provide a rich global overview of an entire database. Moreover, its effectiveness relies on the provided hierarchy, which may or may not match the way users see the grouping of shapes. At a larger scale, defining a good hierarchy is challenging: if a 3D shape database evolves freely, it may need to accommodate, in the future, shapes which do not easily fit within the existing hierarchy. While any given hierarchy can be refined, this can be a costly procedure, especially if levels close to the hierarchy root need to be edited and existing shapes in the database require re-distribution in the edited hierarchy. Additionally, as for keyword search, hierarchical exploration is well known and many solutions exist for 3D shape databases to this end [125, 161]. As such, we do not explore this direction further in our work.

Content based shape retrieval (CBSR) allows users to search for shapes similar to a given query shape, and therefore depend far less on an upfront organization of the database in terms of suitable keywords or hierarchies and/or on the user's familiarity with these. Good surveys of CBSR methods are provided by [17, 150]. These methods essentially extract a high-dimensional descriptor from the query and database shapes, and then search and retrieve the most similar shapes to the query based on a suitable distance metric in descriptor space. Many types of descriptors and distance metrics have been proposed, as follows. Global descriptors, such as shape elongation, eccentricity, and compactness, are simple, yet crude ways to discriminate between highly different shapes. Local descriptors, such as saliency, shape thickness, and shape contexts capture more fine-grained shape details [129, 131, 138, 151]. Topological descriptors, such as based on curve skeletons [69] or surface skeletons [47] capture the part-whole shape structure. Finally, view-based descriptors capture the appearance of the shape from multiple viewpoints [32, 133]. Kalogerakis *et al.* [74] provide a tool to compute several types of shape features. Apart from such hand-engineered descriptors, deep learning has proved effective in automatically extracting low-dimensional representations of shape with high accuracy for query tasks [147] and also for related classification tasks [116]. CBSR frees the user from the burden of specifying keywords or choosing explicit navigation paths in a hierarchy to examine a shape database. Additionally, CBSR assists in finding the most similar shapes to a given prototype (query). However, CBSR does not readily support the task of general-purpose exploration of a shape database, *e.g.*, seeing how *all* the shapes within it are organized in terms of similarity.

Summarizing the above, keyword search, hierarchical exploration, and CBSR offer largely complementary mechanisms for exploring a shape database, and can be readily combined in a 3D database exploration system. However, as outlined, none of these methods offer a compact, complete, and detailed overview of an entire database. Moreover, such mechanisms do not explain *why* a set of shapes are deemed similar. In earlier work, Rauber *et al.* [117] have used interactive feature selection to improve image classification, which is related, but not the same, to our goal of *exploring* data collections. Such functionalities are essential in contexts where users do not know precisely what they are looking for and would like to understand the information contained in a database before proceeding to more specific queries.

Free exploration of high-dimensional data spaces — such as our 3D shape databases can be seen — are, however, not new in information visualization. For instance, treemaps provide highly scalable approaches to explore large hierarchies up to hundreds of thousands of elements [16]. Closer to our goal and interests, so-called dimensionality reduction, or projection, methods, allow data scientists to create overviews of datasets containing millions of samples each having up to thousands of attributes or data dimensions [42]. Such dimensions correspond precisely to the features extracted in CBSR. Both these classes of techniques ideally match our goal of free exploration. We elaborate on these topics, also introducing more specific related work, in the contexts of Chapters 3 and 4.

2.3 3D SHAPE EXAMINATION

The second step of exploration, following after targeted search or free browsing, is to examine a (typically small) subset of shapes of interest. These are either the results of a query (in targeted search) or a subset of shapes deemed to be of interest obtained from free exploration, *e.g.*, the contents of a sub-hierarchy in the case of data that is presented hierarchically. Once such a small subset of shapes of interest — in the limit, a single shape — is obtained, by any means deemed suitable, the shapes in question are examined one by one and in detail.

Several aspects are important during the detailed examination step. Following the traditional computer graphics pipeline [63], we mention below two sub-steps in this examination process:

- *viewing:* This step corresponds to selecting a suitable viewpoint, including eye position, viewing vector, up vector, projection transformation, and viewport sizes;
- *presentation:* This step corresponds to selecting a suitable lighting model, as well as layers of (material) properties that are used to display the 3D shape in a realistic way, *e.g.* using textures, simplified way, *e.g.* using simple Gouraud shading [63], or color-coding various properties of the shape such as surface Gaussian curvature [152, 155].

Presentation modalities are further the object of computer graphics and rendering research which is out of our scope. We focus next on the *viewing* step. Within this step, one typically needs to specify various combinations of translation (panning), scaling (zooming), and rotation transformations. Whereas translation and scaling are relatively easy to specify by interactive means such as keyboard and mouse or similar interaction devices, rotation is more challenging. The issue, in our context, is that specifying a general 3D rotation by using such tools typically involves specifying six parameters corresponding *e.g.* to a rotation axis given by a location and a direction in 3D, and an angle. Note that this specification involves a practical trade-off: Specifying a 3D axis requires, formally speaking, only 4 degrees of freedom; thus, adding a rotation angle around it brings one to the need of specifying 5 degrees of freedom. However, existing 3D interaction tools often prefer to allow users to specify 3D rotation axes using more parameters. For instance, specifying a 3D axis can be done by selecting two 3D points, which amounts to specifying six parameters. This makes the specification arguably more natural but increases the number of parameters.

Rotation specification: To address the above challenges, many techniques have been proposed to ease the specification of 3D rotations. The trackball metaphor [23] is one of the oldest and likely most popular techniques. Given a 3D center-of-rotation $\mathbf{x}$, the scene is rotated around an axis passing through $\mathbf{x}$ and determined by the projections on a hemisphere centered at $\mathbf{x}$ of the 2D screen-space locations $\mathbf{p}_1$ and $\mathbf{p}_2$ corresponding to a (mouse) pointer motion. The rotation angle α is controlled by the amount of pointer motion. While simple to implement and use, trackball rotation does not allow precise control of the actual axis around which one rotates, as this axis constantly changes while the user moves the pointer [6, 187]. Several usability studies of trackball and alternative 3D rotation mechanisms explain these limitations in detail [49, 59, 68, 109]. Several refinements of the original trackball [23] were proposed to address these [64, 136]. In particular, Henriksen *et al.* [56] formally analyze the trackball's principle and its limitations and also propose improvements which address some, but not all, limitations. At the other extreme, world-coordinate-axis rotations allow rotating a 3D scene around the x, y, or z axes [67, 187]. The rotation axis and rotation angle are chosen by simple click-and-drag gestures in the viewport. This works best when the scene is already *pre-aligned* with a world axis so that rotating around that axis yields meaningful viewpoints.

Pre-alignment of 3D models is a common preprocessing stage in visualization [19]. Principal Component Analysis (PCA) does this by computing a 3D shape's eigenvectors $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, ordered by their eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$, so that the coordinate system $\{\mathbf{e}_i\}$ is right-handed. Next, the shape is aligned with the viewing coordinate system (x, y, z) by a simple 3D rotation around the shape's barycenter [77, 150]. Yet, prealignment is not effective when the scene does not have a clear main axis (λ_1 close to λ_2) or when the major eigenvector does not match the rotation axis desired by the user.

3D rotations can be specified by classical (mouse and keyboard) [187] but also touch interfaces. Yu *et al.* [182] present a direct-touch exploration technique for 3D scenes called Frame Interaction with 3D space (FI3D). Guo *et al.* [51] extend FI3D with constrained rotation, trackball rotation, and rotation around a user-defined center. [181] used trackball interaction to control rotation around two world axes by mapping it to single-touch interaction. Hancock *et al.* [54, 55] use two or three touch input to manipulate 3D shapes on touch tables and, in this context, highlighted the challenge of specifying 3D rotations. All above works stress the need for *simple* rotation-specification mechanisms using a minimal number of touch points and/or keyboard controls.

More related work related to interactive examination of 3D shapes is discussed in Chapter 5, which also introduces our contributions to addressing this examination task.

2.4 3D SHAPE ANALYSIS

Shape descriptors are a generic term used for quantities computed from 3D shapes which serve, next, a wide range of applications such as detection, registration, recognition, classification, and retrieval of 3D objects. In our work, we involve shape descriptors for all our three tasks, as follows:

- *exploration:* We use shape descriptors for capturing the similarity of shapes in a 3D database. This allows us to create overviews for free exploration, as discussed next in Chapters 3 and 4;

- *examination:* Separately, we use shape descriptors to capture the salient visible aspects of a 3D shape from a given viewpoint. This, next, allows us to infer suitable 3D axes that match the respective shape view, which we further use to construct 3D rotations for shape examination. This use of shape descriptors is covered in Chapter 5.
- *analysis:* Finally, we use shape descriptors to capture the salient structural properties of a set of similar 3D shapes. This allows us to robustly compute a simplified description of the entire set of shapes, which we call a co-skeleton. Co-skeletons and their associated descriptors are discussed in Chapter 6.

From a technical point of view, we use two types of shape descriptors to support our exploration, examination, and analysis goals. These two types are medial descriptors and histogram-based descriptors. We introduce them next.

2.4.1 *Medial descriptors*

Also known as skeletons — a term that we prefer next for being shorter — medial descriptors have been used for decades to capture the symmetry structure of shapes [14, 139]. For a shape $\Omega \subset \mathbb{R}^n$, $n \in \{2, 3\}$ with boundary $\partial\Omega$, its skeleton is defined as

$$S_\Omega = \{\mathbf{x} \in \Omega | \exists \mathbf{f}_1 \in \partial\Omega, \mathbf{f}_2 \in \partial\Omega : \mathbf{f}_1 \neq \mathbf{f}_2 \wedge ||\mathbf{x} - \mathbf{f}_1|| = ||\mathbf{x} - \mathbf{f}_2|| = DT_\Omega(\mathbf{x}\}, \tag{2.1}$$

where $\mathbf{f}_i$ are called the *feature points* [103] of skeletal point $\mathbf{x}$ and DT_Ω is the distance transform [31, 124] of skeletal point $\mathbf{x}$, defined as

$$DT_\Omega(\mathbf{x} \in \Omega) = \min_{\mathbf{y} \in \partial\Omega} ||\mathbf{x} - \mathbf{y}||. \tag{2.2}$$

These feature points define the so-called *feature transform* [57, 149]

$$FT_\Omega(\mathbf{x} \in \Omega) = \arg\min_{\mathbf{y} \in \partial\Omega} ||\mathbf{x} - \mathbf{y}||, \tag{2.3}$$

which gives, for each point $\mathbf{x}$ in a shape Ω, its set of feature points on $\partial\Omega$, or contact points with $\partial\Omega$ of the maximally inscribed disk in Ω centered at $\mathbf{x}$.

The above definitions for skeletons, feature transforms, and distance transforms are generic for any embedding dimensionality of shapes. In particular, they hold identically for 2D shapes and 3D shapes. However, in practice, skeletons are computed differently for 2D and 3D shapes for both practical and algorithmic reasons, as follows.

2D skeletons: Many methods compute skeletons of 2D shapes, which are described as either polyline contours [108] or binary images [31, 44, 45, 158]. State-of-the-art methods *regularize* the skeleton by removing its so-called spurious branches caused by small noise perturbations of the boundary $\partial\Omega$, which bring no added value, but only complicate further usage of the skeleton. Regularization typically defines a so-called *importance* $\rho(\mathbf{x}) \in \mathbb{R}^+ | \mathbf{x} \in S_\Omega$ which is low on noise branches and high elsewhere on S_Ω. Several authors [31, 44, 45, 108, 158] set ρ to the length of the shortest path along $\partial\Omega$ between the two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$. Upper thresholding ρ by a sufficiently high value removes noise branches. Importance regularization can be efficiently implemented on the GPU [41] using fast distance transform computation [18]. Overall, 2D skeletonization can be seen, both from a practical and a theoretical perspective, as a solved problem. The theory of 2D skeletonization is described in detail by Siddiqi and Pizer [139]; their work, actually, also covers 3D skeletonization, but to a lesser extent.

From a practical perspective, current 2D skeletonization algorithms can handle 2D binary images of resolutions of thousands pixels squared in milliseconds, and produce pixel-thin, centered, multiscale skeletons for arbitrarily noisy 2D shapes [41]. In our work, we use such state-of-the-art 2D skeletonization methods to compute 2D skeletons of silhouettes (projections) of 3D shapes in real time for interactive examination, as detailed next in Chapter 5.

3D skeletons: In 3D, two skeleton types exist [149]: *Surface skeletons*, defined by Eqn. 2.1 for $\Omega \subset \mathbb{R}^3$, consist of complex intersecting manifolds with boundary, and hence are hard to compute and utilize [149]. *Curve skeletons* are curve-sets in $\mathbb{R}^3$ that locally capture the tubular symmetry of shapes [30]. They are structurally much simpler than surface skeletons and enable many applications such as shape segmentation [123] and animation [13]. Yet, they still cannot be computed in real time, and require a well-cured definition of Ω as a watertight, non-self-intersecting, fine mesh [143] or a high-resolution voxel volume [45, 118].

Kustra *et al.* [80] and Livesu *et al.* [95] address the above challenges of 3D curve-skeleton computation by using an *image based* approach. They compute an approximate 3D curve skeleton from 2D skeletons extracted from multiple 2D views of a shape. While far simpler and also more robust than true 3D skeleton extraction, such methods need hundreds of views and cannot be run at interactive rates. Our work for interactive 3D shape examination in Chapter 5 also uses an image-space skeleton computation, but uses different, simpler, heuristics than [80, 95] to estimate 3D depth and a single view, thereby achieving the speed required for interactivity. Separately, our work on shape analysis in Chapter 4 uses 3D curve skeletons computed by the methods described in [5] and [148], which, following a recent survey [149] and also additional quantitative and qualitative evaluations [143, 144], are found to excel in terms of genericity, computational scalability, robustness to noise and type of 3D shape, and quality of the resulting 3D curve skeletons.

2.4.2 *Histogram-based descriptors*

Many types of shape descriptors exist in the 3D retrieval literature. Following key surveys in this area [126, 150], such descriptors can be classified as either *local* or *global*. To explain the difference, let $\mathcal{S}$ be a set of 3D shapes $\Omega \in \mathcal{S}$ under study, such as a 3D shape database. Local descriptors essentially are functions $f : \partial\Omega \rightarrow \mathbb{R}^k$, that is, they compute a k-dimensional value for every location (or neighborhood) $\mathbf{x} \in \partial\Omega$ on the shape surface. Global descriptors, in contrast, are functions $f : \mathcal{S} \rightarrow \mathbb{R}^k$, that is, they compute a single k-dimensional value for a given shape Ω.

For descriptors to be effective in shape analysis or retrieval tasks, they should be *invariant* to changes of the shape which are deemed uninteresting for the application at hand. Such changes regard orientation, size, meshing resolution, and location (in the embedding space). Global descriptors can be computed to be invariant to such changes either natively (by definition) — *e.g.*, consider the volume or surface area of a shape — or by suitable alignment transformation. Local descriptors, however, are by nature dependent on the location $\mathbf{x} \in \partial\Omega$ where they are defined; *e.g.*, consider the Gaussian curvature for a given surface $\partial\Omega$. Since different shapes to be compared typically have different resolution – and, even if they had the same resolution, one could not guarantee a one-to-one correspondence of their sample points $\mathbf{x} \in \partial\Omega$ — local descriptors need to be made invariant, that is, converted into suitable global descriptors.

An established way to handle this local-to-global descriptor conversion is to use *histograms*. Simply put, a histogram descriptor h_f takes a local descriptor f and bins its range over $\partial\Omega$ into a suitable set of bins, and next computes the frequency of values of f over each such bin. Since normalized by the sample count — typically this being either the face count or vertex count of a shape surface $\partial\Omega$ — histograms are invariant to aspects such as resolution and sampling order. Additional simple transformations such as translation of barycenter to the origin, unit-box scaling, and usage of principal component orientations [150] can be used to make such descriptors also invariant to location, scale, and orientation. Next, by using a fixed bin count n, histogram descriptors effectively reduce a shape to an n-dimensional scalar feature vector, or a kn dimensional vector if the local descriptor was already a k-dimensional function. Examples of such histogram descriptors that we use in our work are shape contexts [8] and fast point feature histograms (FPFH) [129]. We note that, besides surface local features defined on $\partial\Omega$, also features extracted from the shape's 3D curve skeleton, such as the local diameter of skeletal cuts, can be used to create such global descriptors via histograms [46, 47]. We use histogram-based descriptors further in our work on shape database exploration (Chapter 3) and also for shape analysis (Chapter 6).

3 VISUAL EXPLORATION OF 3D SHAPE DATABASES VIA FEATURE SELECTION

In this chapter, we use shape properties for constructing effective visual representations of 3D shape databases as projections of multidimensional feature vectors extracted from their shapes. We present several methods to construct effective projections in which different-class shapes are well separated from each other. First, we propose a greedy heuristic for searching for near-optimal projections in the space of feature combinations. Next, we show how human insight can improve the quality of the constructed projections by iteratively identifying and selecting a small subset of features that are responsible for characterizing different classes. Our methods allow users to construct high-quality projections with low effort, to explain these projections in terms of the contribution of different features, and to identify both useful features and features that work adversely for the separation task. We demonstrate our approach on a real-world 3D shape database.

3.1 INTRODUCTION

Recent developments in 3D content creation and 3D content acquisition technologies, including modeling and authoring tools and 3D scanning techniques, have led to a rapid increase in the number and complexity of available 3D models. Such models are typically stored in so-called *shape databases* [66, 130]. Such databases offer various mechanisms enabling users to browse or search them to locate models of interest for a specific application at hand.

As shape databases increase, so does the difficulty that users have in locating models of interest therein [150]. Typical mechanisms offered to support this task include searching by *keywords*, browsing the database along with one or a few predefined hierarchies, or content-based shape retrieval (CBSR). While efficient for certain scenarios, all these mechanisms have limitations: Keyword search assumes good-quality labeling of shapes with relevant keywords, and also that the user is familiar with relevant search terms. Hierarchy browsing is most effective when the organization of shapes follows the way the user wants to explore them. Finally, CBSR works well when the user aims to search for shapes similar to an existing query shape.

Besides the above targeted use-cases, more generic ones involve users who simply want to *explore* the *entire* database to see what it contains. This is relevant in cases where users want to first get a good overview of what a database contains before deciding to invest more effort into exploring or using it; and also in cases where users do not have specific searches in mind. Existing mechanisms offered for the above scenarios are linear in nature, showing either a small part of the database at a single time and/or asking the user to perform lengthy navigations to create a mental map of the database itself, much like when navigating a web domain.

We address this task by a different, visual, approach. We construct a compact and scalable *overview* of an entire shape database, with shapes organized by similarity. We offer *details-on-demand* mechanisms to enable users to control the separation quality of the similar-shape groups in the visual overview; understand what makes a set of shapes similar (or two or more sets of shapes different); and find features that have high, respectively little, value for the shape classification task. Our approach is simple to use; requires no prior knowledge of the organization of a shape database; nor a prior organization or labeling of the database; handles any type of 3D shape represented by a polygon mesh; and scales visually and computationally to real-world large shape databases. Additionally, our proposal is useful for both end-users (who aim to explore a shape database) and technical users (who aim to engineer features to query or classify shapes in such databases).

This chapter is structured as follows. Section 3.2 outlines related work in exploring 3D shape databases. Section 3.3 details our pipeline that consists of shape normalization, feature extraction, and dimensionality reduction. Section 3.4 presents our automatic and user-driven methods for constructing high-quality projections for exploring shape databases and demonstrates these on a real-world shape database. Section 3.6 concludes this chapter.

3.2 RELATED WORK

CBSR, already covered in Section 2.2, and multidimensional projections (MPs) are realted to our 3D shape database exploration solution.

Multidimensional projections: also called dimensionality-reduction techniques, are the instrument of choice for reducing the number of dimensions of a dataset so that important data structures (*e.g.* clusters, correlations, outliers) are still preserved [40, 145]. MPs are used both for data preprocessing, *e.g.* to reduce the number of features that a classifier will next use; but also for visual exploration: Indeed, when the number of target dimensions is low (2 or 3), the initial dataset is reduced to a 2D or 3D scatterplot, which can be directly visualized to perceive the data structure [142].

Finding 'good' projections for a given dataset (or more exactly, a family of datasets generated by a specific problem) is an open problem in data science, for two key reasons. First, virtually any projection algorithm will have to drop information as it reduces the number of dimensions from hundreds or even more to just a few. Secondly, there are tens of different MP algorithms, which have in turn many parameters. Exploring the entire space of projection possibilities is not feasible, even for a single dataset. To evaluate how useful a certain projection is, a variety of quality measures have been proposed [99, 100].

Several projection method families have emerged, aiming to optimize different types of quality metrics [89], as follows.

Affine and Projective methods are a family of multivariate embeddings including RadViz [33, 34, 60, 107] and Star Coordinates [75, 76]. They are generally simple to implement and fast to compute but have typically poor preservation of the high-dimensional data structure (defined *e.g.* in terms of inter-point distances or point neighborhoods).

Orthographic Projections, such as the multivariate Orthographic Star Coordinates [88], generalize the concept of bivariate orthographic projections, such as scatterplots. They prevent distortions better by maintaining a set of orthography-preserving constraints.

Distance-based projection techniques aim to preserve the inter-point distances as they map points from the high dimensional space to the low (2D or 3D) dimensional space. For instance, Multidimensional scaling (MDS) [162] preserves distances between the data records under projection via the spectrum of a data-dependent centered distance matrix.

PCA-based techniques also belong to this family. They are very simple and fast to compute but fail to preserve data structure when the high-dimensional points are not spread on, or close to, a hyperplane. With Glimmer [65], a high-performance approach for multilevel MDS on graphic processing units is known. The large amount of distance information required to build up a projection can be reduced by part linear multidimensional projection (PLMP) [112] to a small number of pairwise distances between some so-called representative data samples, which substantially increases the performance of the projection process. Local affine multivariate projection (LAMP) [71] provides a local data projection technique by minimizing the distances of the projected data points with the aid of (interactively) initialized seed or control points in the visualization space. LAMP is particularly interesting in applications where one does not precisely know the quality or meaningfulness of the original dimensions; in such cases, one can 'rearrange' the projection by moving the control points so that the emerging patterns (*e.g.* clusters) better match the user's perception of similarity between items.

Recently, t-SNE [97] has gained wide popularity. Its two main advantages are that it only needs a pairwise distance (similarity) matrix, rather than actual dimensions of points; and that if the data contains well-separated clusters, such clusters become very apparent in the (2D) projection. However, t-SNE is a quite slow method, and also sensitive to its parameter setting. Finally, UMAP [102] applies the theories of fuzzy mathematics and Riemannian geometry to build a solution that can generate similar quality results as t-SNE with faster speed.

In this chapter, we use t-SNE and UMAP for multidimensional projection. While they can create good results, they are quite slow when the aim data size is really large. So we apply NNProj, a neural network method, for multidimensional projection in Chapter 4.

3.3 PROPOSED METHOD

To support the overall exploration of 3D shape databases, we propose to *augment* existing mechanisms (keyword search, hierarchies, and CBSR) by a visual navigation approach. Our approach allows users to see a complete overview of an entire database and the way shapes are organized in terms of similarity. Next, it allows selecting specific shapes or shape properties and finding similar shapes (from the perspective of one or several such properties), and also finding out how properties discriminate between different shapes. We now detail our approach.

3.3.1 *Overview*

We start by introducing some notations. A mesh $m = (V = [\mathbf{x}_i], F = [f_j])$ is a collection of vertices $\mathbf{x}_i \subset \mathbb{R}^3$ and faces f_j, assumed to be triangles for simplicity. A shape database is a set of shapes $M = \{m_k\}$. No restrictions are placed here, *i.e.*, shapes can be of different kinds, sampling resolutions, and require no extra organization or annotations, *e.g.*, classes or hierarchies.

Our key idea is to present a visual *overview* of M in which every shape m_k is represented by a thumbnail rendering thereof, and visual distances between two shapes m_i and m_j reflect their similarity. The visual overview is interactively linked with detail views in which users can explore specific shape details. The combination of overview and details, following Shneiderman's visual exploration mantra [135], enables both free and targeted exploration of the shape database along the use-cases outlined in Section 3.2.

We create our overview-and-detail visual exploration as follows. First, we preprocess all meshes in M to normalize them in terms of sampling resolution and size. Secondly, we extract local features from all meshes $m \in M$ (Section 3.3.3). These features capture the respective shapes at a fine level of detail. Next, we aggregate local features into fixed-length feature vectors (Section 3.3.4). Finally, we use a dimensionality-reduction algorithm to project the shapes, represented by their feature vectors, onto 2D screen space (Section 3.3.5). We describe all these steps next.

3.3.2 *Preprocessing*

Since we do not pose any constraints on the shapes in M, these can come with virtually any sampling resolution, orientation, and at any scale. Such variations are known to pose problems when computing virtually any type of shape descriptor [17]. Hence, as a first step, we normalize all shapes $m \in M$ by first remeshing them, with a target edge-length of 1% of m's bounding-box diagonal. Next, we translate and scale the remeshed shapes to fit the $[-1, 1]^3$ cube.

3.3.3 *Local feature extraction*

To characterize shapes, we extract several so-called *local features* from each. Such features describe the shape at or in the neighborhood of every vertex $\mathbf{x}_i \in m$ and are therefore good at capturing local characteristics. We compute seven local feature types, as follows.

Gaussian Curvature (Gc): Gaussian curvature describes the overall non-flatness of a shape close to a given point. For every vertex $\mathbf{x} \in m$ we compute its Gaussian curvature as

$$Gc(\mathbf{x}) = 2\pi - \sum_{f \in F(\mathbf{x})} \theta_{\mathbf{x},f}, \tag{3.1}$$

where $F(\mathbf{x})$ is the set of faces in F incident with $\mathbf{x}$ and $\theta_{\mathbf{x},f}$ is the angle in face f at vertex $\mathbf{x}$.

Average Geodesic Distance (Agd): We estimate the geodesic distance $d(\mathbf{x}, \mathbf{y})$ between a pair of vertices $\mathbf{x}$ and $\mathbf{y}$ of m as the geometric length of the shortest path in the edge connectivity graph of m between $\mathbf{x}$ and $\mathbf{y}$. This distance can be easily and efficiently estimated using Dijkstra's shortest-path algorithm with A* heuristics and edge weights equal to edge lengths. More accurate estimations of the geodesic distance between two points on a polygonal mesh exist, including computing the distance field (or transform) $DT(\mathbf{x})$ of $\mathbf{x}$ over F and tracing a streamline in $-\nabla DT(\mathbf{x})$ from $\mathbf{x}$ until it reaches $\mathbf{y}$ [113]; GPU minimization of cut-length using pivoting slice planes passing through $\mathbf{x}$ and $\mathbf{y}$ [70]; or hybrid search techniques [166]. While more accurate than the Dijkstra approach we use, these methods are *considerably* more complex to implement, slower to run, and require careful tuning and/or specialized platforms (GPU support). For a detailed comparison of geodesic estimation methods on polygonal meshes, we refer to [70]. More importantly, we do not use the *individual* geodesic lengths, but aggregate them into per-shape feature vectors (Section 3.3.4). As such, high geodesic estimation precision is less important.

Given the above, we estimate the average geodesic distance of a vertex $\mathbf{x}$ as

$$Agd(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in V} d(\mathbf{x},\mathbf{y})}{|V|}. \tag{3.2}$$

Normal Diameter (Nd): We first estimate the surface normal at a vertex $\mathbf{x}$ as

$$\mathbf{n}(\mathbf{x}) = \sum_{f\in F(\mathbf{x})} \mathbf{n}(f)\theta_{\mathbf{x},f}, \tag{3.3}$$

where $\mathbf{n}(f)$ is the outward normal of face f. Given the above, let $\mathbf{r}$ be a ray starting at $\mathbf{x}$ and advancing in the direction $-\mathbf{n}(\mathbf{x})$. The normal diameter $Nd(\mathbf{x})$ is then the distance along $\mathbf{r}$ from $\mathbf{x}$ to the closest face $f \in F \setminus F(\mathbf{x})$.

Normal Angle (Na) and Point Angle (Pa): These features describe how vertices $\mathbf{x} \in V$ are spread around the shape itself. In detail, let $\mathbf{e}_1$ be the dominant eigenvector of the shape covariance matrix given by all vertices V. As known, $\mathbf{e}_1$ gives the direction in which the shape spreads the most. Next, for every vertex $\mathbf{x} \in V$, we define the normal angle $Na(\mathbf{x})$ as the angle (dot product) between $\mathbf{e}_1$ and the surface normal $\mathbf{n}(\mathbf{x})$; and the Point Angle $Pa(\mathbf{x})$ as the angle (dot product) between $\mathbf{e}_1$ and the vector $\mathbf{c}-\mathbf{x}$, where $\mathbf{c}$ is the barycenter of m.

Shape Context (Sc): The shape context descriptor is a 2D histogram that characterizes how vertices of a shape are 'seen' in terms of distance and orientation from a given vertex of that shape [8]. For a vertex $\mathbf{x} \in V$, the shape context describes the number of vertices in V that are within a given distance range and direction range to $\mathbf{x}$. To compute Sc, we first build a local coordinate system at every vertex $\mathbf{x}$, using the eigenvectors of the shape covariance matrix in the neighborhood of $\mathbf{x}$. This ensures that this coordinate system is aligned with the shape locally — one of its axes will be the normal $\mathbf{n}(\mathbf{x})$, whereas the two other ones are tangent to the surface of m at $\mathbf{x}$. Next, we discretize the orientations around $\mathbf{x}$ into the eight octants of the local coordinate system, and distances using a set of bins (distance ranges) (t_i, t_{i+1}) defined by a distance-set $T = \{0, t_1, t_2, \ldots, t_n, 1\}, n \in \mathbb{N}_+$. In practice, we use $T = [0, 0.1, 0.3, 1]$. Hence, for each vertex $\mathbf{x}$, we get a shape context vector with $8 \times 3 = 24$ elements.

Point Feature Histogram (PFH): PFH [129] is a complex descriptor that captures the local geometry in the vicinity of a vertex. Given a pair of vertices $\mathbf{y}$ and $\bar{\mathbf{y}}$, one first defines a local coordinate frame $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ as

$$\begin{aligned} \mathbf{u} &= \mathbf{m}, \\ \mathbf{v} &= (\bar{\mathbf{y}} - \mathbf{y}) \times \mathbf{u}, \\ \mathbf{w} &= \mathbf{u} \times \mathbf{v}, \end{aligned} \tag{3.4}$$

where $\mathbf{m}$ is the vertex normal at $\mathbf{y}$ (Figure 3.1). Next, the variation of the shape geometry between points $\mathbf{y}$ and $\bar{\mathbf{y}}$ is captured by three polar coordinates

$$\begin{aligned} \alpha &= \mathbf{v} \cdot \bar{\mathbf{m}}, \\ \phi &= \mathbf{u} \cdot \frac{\bar{\mathbf{y}} - \mathbf{y}}{\|\bar{\mathbf{y}} - \mathbf{y}\|}, \\ \theta &= \operatorname{arctan2}(\mathbf{w} \cdot \bar{\mathbf{m}}, \mathbf{u} \cdot \bar{\mathbf{m}}), \end{aligned} \tag{3.5}$$

where $\bar{\mathbf{m}}$ is the vertex normal at $\bar{\mathbf{y}}$. Next, three histograms are built to capture the distributions of α, ϕ, θ for a given vertex $\mathbf{x}$ by considering all pairs $(\mathbf{y}, \bar{\mathbf{y}}) \in N_{\mathbf{x},k} \times N_{\mathbf{x},k}$ in the k-nearest neighbors $N_{\mathbf{x},k}$ of $\mathbf{x}$. In practice, we set $k = 30$ and use 5 bins for each histogram. This delivers a PFH feature vector of $5^3 = 125$ entries.

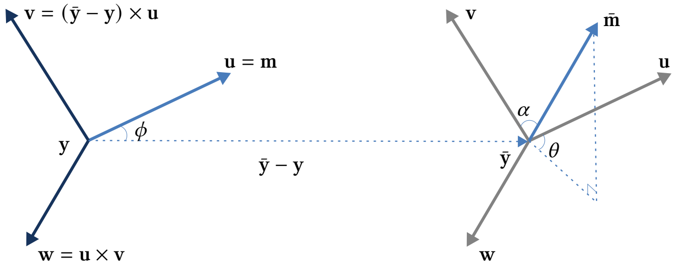

$\mathbf{v} = (\bar{\mathbf{y}} - \mathbf{y}) \times \mathbf{u}$
$\mathbf{u} = \mathbf{m}$
ϕ
$\mathbf{y}$
$\bar{\mathbf{y}} - \mathbf{y}$
$\mathbf{w} = \mathbf{u} \times \mathbf{v}$
$\mathbf{v}$
$\bar{\mathbf{m}}$
$\mathbf{u}$
α
$\bar{\mathbf{y}}$
θ
$\mathbf{w}$

Figure 3.1: PFH descriptor computation [138].

Fast Point Feature Histogram (FPFH): While PFH models a neighborhood $N_{\mathbf{x},k}$ by all its point-pairs, the Simplified Point Feature Histogram (SPFH) models $N_{\mathbf{x},k}$ by the characteristics of the pairs $(\mathbf{x}, \mathbf{y} \in N_{\mathbf{x},k})$. We proceed analogously to binning the α, ϕ, θ distributions in three histograms of 11 bins each, obtaining a feature vector of $3 \times 11 = 33$ elements. With this vector, we finally compute the FPFH value of a vertex $\mathbf{x}$ following [129] as the distance-weighted average of the SPFH values over the neighborhood $N_{\mathbf{x},k}$ as

$$FPFH(\mathbf{x}) = SPFH(\mathbf{x}) + \frac{1}{k} \sum_{\mathbf{y} \in N_{\mathbf{x},k}} \frac{SPFH(\mathbf{y})}{\|\mathbf{x} - \mathbf{y}\|}. \tag{3.6}$$

3.3.4 *Feature vector computation*

The features described in Section 3.3.3 are *local*, *i.e.*, they take different values for every mesh vertex $\mathbf{x} \in V$. To be able to compare meshes to each other, we need to reduce these to same-length *global* descriptors. For this, we use a simple histogram-based solution that aggregates the values of every local descriptor, at all vertices of a mesh, into a fixed-length (10 bin) histogram. Note that some descriptors are by definition high-dimensional — for instance, the shape context Sc has 24 dimensions. Hence, for a d-dimensional descriptor, we compute a histogram having $10d$ bins. Table 1 shows the local features, their dimensionality, and the number of bins used to quantize each. To summarizing, we reduce every shape m to an 1870-dimensional feature vector $\mathcal{F}$.

Table 1: Local features, their dimensionalities and binning.

Name	**Dimensionality**	**Bins**
Gaussian curvature (Gc)	1	10
Average geodesic distance (Agd)	1	10
Normal diameter (Nd)	1	10
Normal angle (Na)	1	10
Point angle (Pa)	1	10
Shape context (Sc)	24	240
Point Feature Histogram (PFH)	125	1250
Fast Point Feature Histogram (FPFH)	33	330
Total		1870

3.3.5 *Dimensionality reduction*

So far, we have reduced a shape database M to a set of $|M|$ 1870-dimensional feature vectors. We next create a visual representation of the shape database by projecting all these vectors onto 2D using the well-known t-SNE dimensionality reduction method [97]. Simply put, t-SNE constructs a 2D scatterplot $P(M) = \{P(m_k)\}$, where every shape $m_k \in M$ is represented by a point $P(m) \in \mathbb{R}^2$, so that the distances between scatterplot points reflect (encode) the similarities of their feature vectors.

An important concern when proposing such a representation is to gauge its *quality*. To do this, we use the classes (labels) of the shapes. For a database where each shape m has a categorical label $c(m) \in C$, where C is a set of categories (*e.g.*, keywords describing the different shapes in a database), we define the neighborhood hit $NH(m)$ as the proportion of the k-nearest neighbors of $P(m)$ that have the same label $c(m)$ as m itself [111]. In practice, we set $k = 10$, following related applications that gauge projection quality [111]. With this, we can next define the neighborhood hit of an entire class $c \in C$ as

$$NH_c(c) = \frac{\sum_{m \in M : c(m)=c} NH(m)}{|m \in M : c(m) = c|}. \tag{3.7}$$

Finally, at the highest aggregation level, we define the neighborhood hit for an entire scatterplot $P(M)$ for a shape database M as

$$NH_s(M) = \frac{\sum_{m \in M} NH(m)}{|M|}. \tag{3.8}$$

The above two NH metrics describe how a mesh (point in the 2D projection scatterplot) is separated from points of different kinds: NH_c shows whether a group of points representing same-class meshes is well-separated in a scatterplot (something we desire since we want next to use the scatterplot to answer the question "How many shape classes are in a database, and how similar are they to each other?"). NH_s shows how well a whole scatterplot can represent an entire shape database. Both NH metrics range between 0 and 1, with higher values indicating better separation, which is preferred.

3.4 APPLICATIONS

We next demonstrate our visual exploration approach on a subset of the Princeton Shape Benchmark [134] having 280 meshes from 14 classes, with 20 meshes from each class. As outlined earlier, these meshes are not labeled, hierarchically organized, or otherwise preprocessed.

3.4.1 *Optimal scatterplot creation*

The projection scatterplot (Section 3.3.5) is the central view that shows an entire shape database. Hence, creating a good scatterplot is important for all exploration tasks addressed next. In this section, we explore the following questions:

Q1: How can we create a good projection scatterplot?

Q2: Which features are best for grouping similar shapes (and, conversely, separating different shapes) in the scatterplot?

Q3: Which is the minimal set of features required to generate a good-quality scatterplot?

Concerning Q1, we could directly create and examine a t-SNE projection of the whole shape database as encoded by the 1870-dimensional feature vectors we extracted (Section 3.3.4). However, using t-SNE is not always easy, especially for high-dimensional data: This method maps

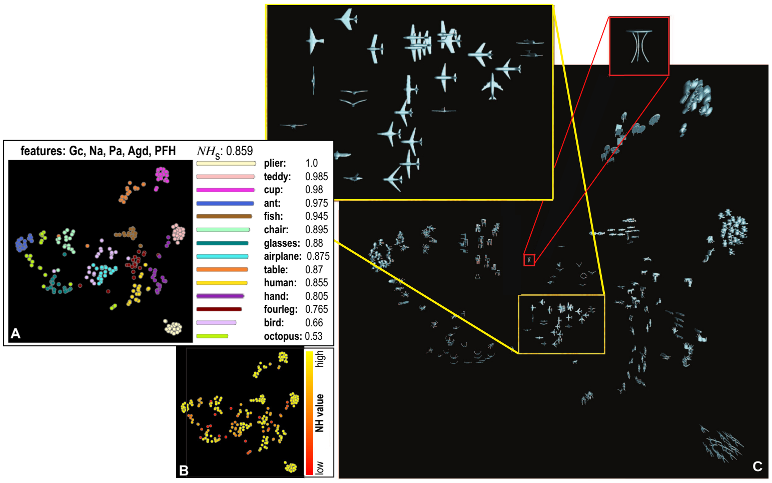


Figure 3.2: Three views of the optimal projection scatterplot for the Princeton Shape Database, depicting classes and their NH_c values and the overall plot quality NH_s (A), per-shape NH values (B), and actual shape thumbnails (C).

similarities (of feature vectors) *non-linearly* to 2D distances; also, tuning t-SNE's parameters to yield a good embedding of high-dimensional feature vectors is notoriously hard [170]. Hence, we first explore the simpler solution of projecting only *subsets* of all extracted features. As we have 8 feature types (Table 1), a natural idea is to try all combinations of groups of feature types. This yields $2^8 - 1 = 255$ possible projection scatterplots. Following the scagnostics idea [163, 171], we compute all these scatterplots (using t-SNE) and select the one having the highest quality, measured by its NH_s value.

Figure 3.2 shows three views of the optimal projection scatterplot, as follows. Image A shows the scatterplot with points (shapes $m \in M$) colored by their class value $c(m)$. The text atop this image indicates the feature subset leading to this optimal scatterplot (highest $NH_s = 0.859$ value), namely (Gc, Na, Pa, Agd, PFH). The bar chart in image A shows the NH_c values for all classes, with high values (well separated classes) at the top. From this, we can see that *pliers* are perfectly separated from all other classes ($NH_{pliers} = 1$), while *octopus* is least well separated ($NH_{octopus} = 0.53$). Image B shows the optimal scatterplot colored by NH values for all shapes, ranging between red (low NH) to yellow (high NH). Red points in this image show shapes which are not projected well — that is, placed close to shapes having different classes. Finally, image C shows the optimal scatterplot with shapes depicted by thumbnails. From this image, we find that shapes of the classes pliers, teddies, cups, ants, and fishes are well projected. However, birds are mixed with airplanes; and fourlegs are mixed with humans and hands. The octopus class is visually split in the projection into several parts. While this optimal scatterplot is not perfect, it is *still* formally the best one we can create given the combinations of our 8 available features. Indeed, while class separation is not perfect, closely-projected shapes are still quite similar. For instance, ants are surrounded by octopuses, which is arguably logical, since both shape types have many thin and spread legs. Similarly, airplanes and birds are close to each other; indeed, both have wings and are relatively flat.

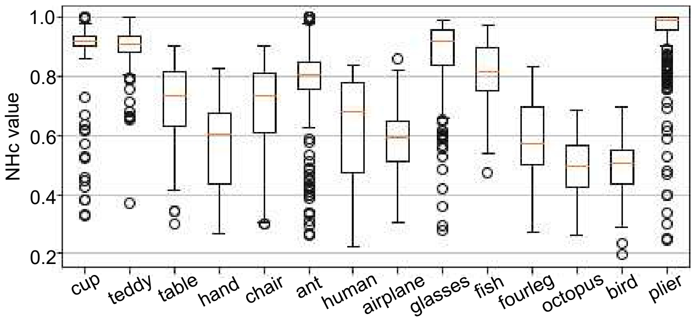


Figure 3.3: NH_c statistics for the 14 classes in the shape database for all 255 projection scatterplots.

Figure 3.3 shows boxplot statistics of the NH_c values for the 14 shape classes in our database for the 255 created projection scatterplots. Ideally, we would like to see that every class has a very high NH_c value. However, we see that birds and octopuses have quite low NH_c values, confirming the insights obtained earlier by visually examining the projection (Figure 3.2).

Finally, Figure 3.4 shows a complete view of how features affect the quality of the produced projections. The bar chart shows the NH_s values of all possible 255 projections created by combining our 8 feature types, sorted in increasing values from left to right. The matrix plot below the bar chart shows which features (color-coded according to the legend at the right) are used by which projection. Projections using more than one feature have blue bars; projections that use exactly one feature have their bars colored by the respective feature. This plot gives us several insights: (1) The quality difference between the best and worst projections is significant (NH_s 0.831 *vs* 0.38), with better projections using more features than poor ones. (2) Some features are really instrumental in achieving high quality, *e.g.* PFH (orange) and FPFH (purple), which appear consistently in the right of the matrix plot in Figure 3.4, whereas other features are actually adversely affecting quality, *e.g.* Sc (green) which appears in the left of the matrix plot. This indicates either that Sc is not a useful feature for discriminating classes in this database or that it is poorly evaluated, *e.g.*, by an insufficiently dense sampling. (3) Overall, the right of the matrix is more full than its left part, which

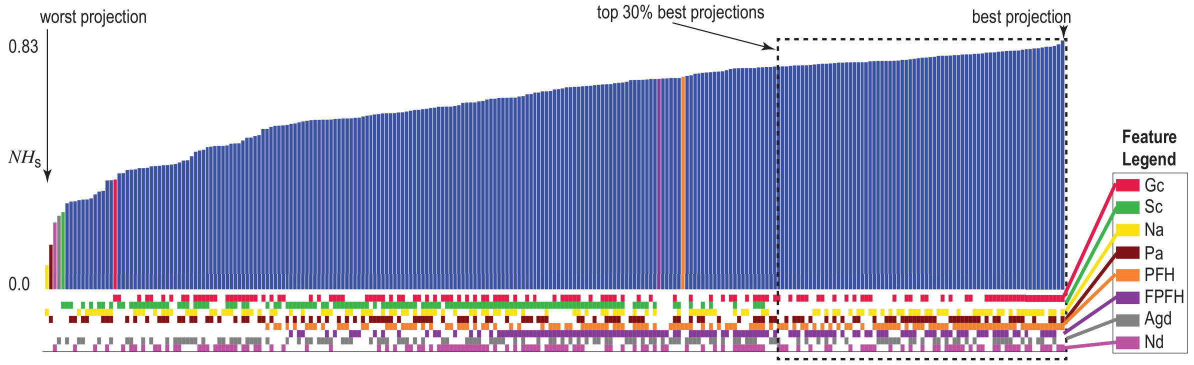


Figure 3.4: A bar chart showing the NH_s scores of 255 projections, sorted on increasing value (best projections to the right, worst ones to the left). The color blocks under a bar show which features are used for that projection (the feature color legend on the right). Bars which are not blue only use one feature, whose identity colors the bar. Scanning the color matrix below the bars row-wise tells us which projections use which features. We see that PFH (orange) and FPFH (purple) are good features since their blocks are close to the right. Conversely, Sc (green) is not a very useful feature since its blocks are spread to the left.

means that using more features produces better segregating projections, although the relation is not monotonic. (4) The highest-quality projections (roughly, the rightmost third of the bar chart) consistently use the same mix of features (Gc, Na, Pa, PFH, FPFH, Agd, Nd). (5) The patterns in the matrix plot of different features look different, which means there are no redundant features in the considered set.

3.4.2 *Fast computation of near-optimal projection scatterplot*

Computing all possible scatterplots given a feature vector $\mathcal{F}$ to find the optimal one is expensive, especially when the set $\mathcal{F}$ is large. We next propose a greedy algorithm to accelerate this task (Alg. 1). The parameter s gives the maximum size of the feature-set to search for. For every search iteration, $\binom{s}{|\mathcal{F}|}$ feature combinations are examined, and the best one, in terms of the realized NH_s value, is retained. Better solutions are obtained for larger s values, at the expense of longer search times. When $s = |\mathcal{F}|$, Alg. 1 compares all possible $2^{|\mathcal{F}|}$ feature combinations. From our tests, a quite good solution in terms of NH_s value can be found by setting $s = 1$. For this setting, the time complexity of our algorithm is $O(|\mathcal{F}|^2)$.

Table 2 shows the results of our greedy algorithm, executed 5 times, to account for the stochastic nature of the t-SNE projection. For every round, we indicate the time taken by exhaustive search *vs* our greedy search, and also the number of t-SNE projections being evaluated. We see that our algorithm yields practically the same NH_s quality as the exhaustive search, but is roughly 5 times faster.

Algorithm 1 Computing near-optimal feature sets.

```
Input: Set of features $\mathcal{F}$; maximal size $s$, $1 \leq s \leq |\mathcal{F}|$, of feature-set to search for
Output: Near-optimal feature set $\mathcal{C}$
 1: $\mathcal{C} := \emptyset$
 2: $\mathcal{C}_{new} := \emptyset$
 3: repeat
 4:     $\mathcal{C} := \mathcal{C}_{new}$
 5:     for each $\mathcal{F}_{sub} \subseteq \mathcal{F}, |\mathcal{F}_{sub}| \leq s$ do
 6:         $\mathcal{C}_{temp} := (\mathcal{C} \cup \mathcal{F}_{sub}) - (\mathcal{C} \cap \mathcal{F}_{sub})$
 7:         if $NH_s(\mathcal{C}_{temp}) > NH_s(\mathcal{C}_{new})$ then
 8:             $\mathcal{C}_{new} := \mathcal{C}_{temp}$
 9:         end if
10:     end for
11: until ($\mathcal{C}_{new} = \mathcal{C}$)
12: return $\mathcal{C}$
```

Table 2: Performance of the greedy algorithm.

Round	Search method	NH_s	Time (secs)	t-SNE runs
1	Exhaustive	0.831	459.74	255
	Greedy	0.831	103.49	56
2	Exhaustive	0.830	452.12	255
	Greedy	0.830	84.98	48
3	Exhaustive	0.829	453.70	255
	Greedy	0.820	70.24	40
4	Exhaustive	0.832	445.47	255
	Greedy	0.832	111.71	64
5	Exhaustive	0.824	447.66	255
	Greedy	0.824	97.39	55

3.4.3 *User-driven projection engineering*

Section 3.4.2 showed how we can automatically select features from the eight existing feature classes (Table 1) to create a projection scatterplot which best separates shapes from different classes. However, using this automatic approach has some disadvantages: (1) It is *expensive*, even when using the proposed greedy algorithm for feature selection. (2) It is too *coarse-grained*: All features of the same *type*, *e.g.*, the 24 shape-context *Sc* features are either selected all, or ignored, when constructing the projection. (3) It is too *simplistic*: There are cases when, for instance, we want to optimize for separation of certain classes more, based on problem-specific constraints. Hence, *user input* in deciding which feature combination leads to the optimal projection is crucial.

We next address question Q2, rephrased as: How can we pick 'good' feature-bins (from the total set of 1870 bins) that separate classes in the way we desire *in a specific context*? For this, we propose an interactive tool based on feature scoring (Figure 3.5) which contains several views (1–6) that allow the user to explore the effect of features on separating classes of shapes in the database, and also select subsets of features that lead to a desired, better, class separation. We explain these views via an overview-and-details-on-demand workflow, as follows.

Model and feature selector (1): The user starts by selecting the shape classes and feature types of interest in this view. This allows them to specify if they are interested in separating specific classes (which are then to be selected) or, alternatively, interested in separating equally well all classes from each other (in which case, all classes should be selected). For instance, from our earlier experiments discussed in Section 3.4.1, we saw that birds are hard to separate from airplanes. The user can then select only these two classes in view (1) to explore how to increase their separation. Separately, one can select the feature

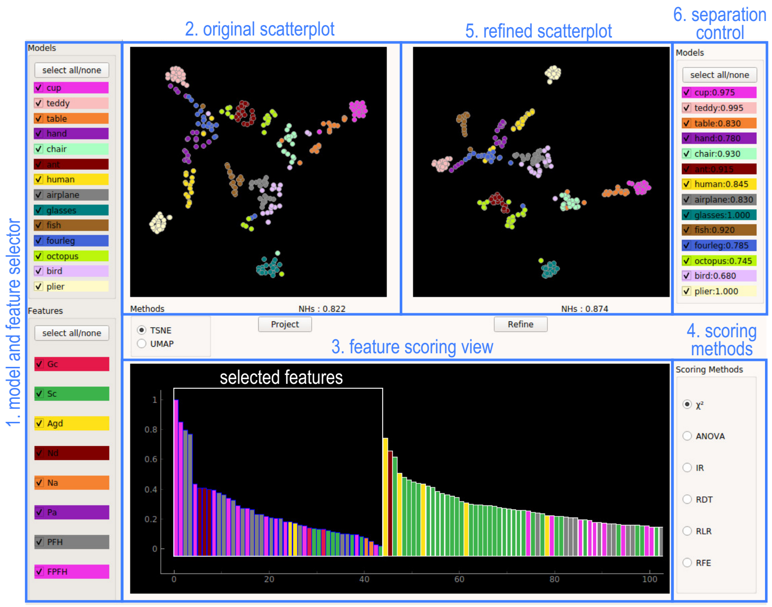


Figure 3.5: A user-driven projection engineering tool and its six views (Section 3.4.3).

types (of the 8 computed ones) to use for creating the scatterplot. This is useful to examine, or debug, the effect of a specific feature type. Classes are categorically color-coded, and the same colors are used in the scatterplots (2, 5). Similarly, feature types are categorically color-coded with the same colors used in the feature scoring view (3).

Original scatterplot (2): This view shows a scatterplot using all shape classes *vs* all feature types chosen in the selector (1). It acts as a starting point for the exploration, which can next be refined to *e.g.* produce better separation of desired classes or instances (shapes) using the feature scoring views (3, 4) discussed next. Scatterplots can be computed either with the t-SNE or UMAP [102] projection methods. t-SNE spreads similar points better over the available 2D space but takes longer to compute. UMAP creates denser clusters separated by more whitespace but is faster to execute. For a trade-off of these two techniques, we refer to a recent survey [42].

Feature scoring views (3, 4): Each bar in the barchart (3) shows the *discriminative score* of every element f_i of the 1870-dimensional feature vector, *i.e.*, how much f_i contributes to separating class c_i from a few or from all other classes $c_j \neq c_i$ selected for exploration in view (1), depending on the separation control (6, discussed later). Colors identify to which feature types the elements f_i belong. For instance, the several purple bars in Figure 3.5(3) correspond to the 330 bins that the FPFH feature (colored purple in Figure 3.5(1)) has. Scores are computed with six scoring methods [117]: chi-squared, one-way ANOVA, Randomized Decision Trees (RDT), Randomized Linear Regression (RLR), iterative relief (IR), and Recursive Feature Elimination (RFE), which can be chosen by the user in panel (4). The barchart supports two tasks: First, it shows how the many bins that each feature is represented by contributes to the separation power of that feature. Secondly, it allows fine-grained examination of the effect of each such bin on the class separation: Users can freely *select* specific bins (from the 1870 available ones) to create a new projection. The selected bins are displayed with a blue border and listed, in decreasing score order, before the unselected ones, in the barchart. The new projection created by the user-selected bins is shown in view (5).

Refined scatterplot (5): This scatterplot shows instances from the classes selected in view (1), projected according to the specific feature-bins selected in the barchart (3). This is thus a *refined view* of the original projection (1). By comparing the refined scatterplot with the original one, one can thus see how a fine-grained selection of every single of the 1870 feature-vector components can improve the projection or parts thereof. In other words, obtaining an optimal projection is achieved in two steps: First, one can select entire features (in view 1). This corresponds to considering or ignoring entire *features* that capture different aspects of shape. Upon obtaining a suitable projection, one can refine it by selecting or deselecting individual bins for the selected features. This corresponds to considering or ignoring *ranges* of the values of the features under exploration.

Separation control (6): As mentioned, feature scoring measures how well selected features separate a class c_i from one or several classes $c_j \neq c_i$. The view (6) allows controlling this. The view shows all shape classes c_i in the database. If all classes are selected in view (6), scoring will measure how well a class c_i is separated from *each* of the other classes $c_j \neq c_i$. If only one class c_i is selected in (6), then scoring will measure the separation of c_i from $\cup_{j \neq i} c_j$. This way, one can flexibly measure the separation of arbitrary *groups* of classes rather than only the separation of individual classes themselves.

3.4.4 *Use cases*

We demonstrate the added value of our user-driven projection engineering by answering several practical questions, as follows.

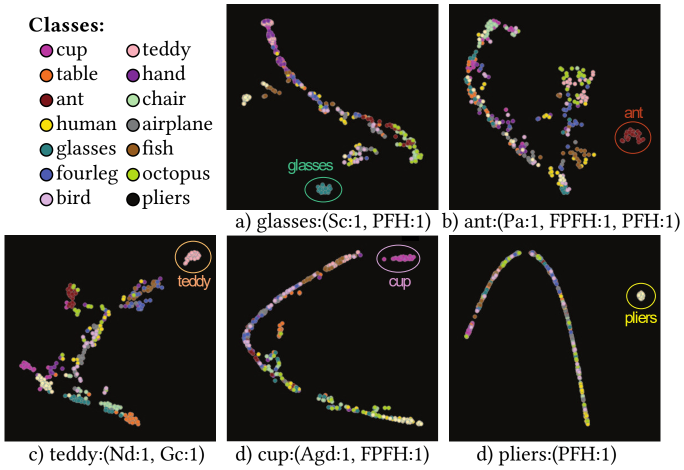


Figure 3.6: Finding the minimal number of feature-bins able to separate five shape classes from the rest of the database. Notation *name:i* indicates that *i* bins of feature *name* are used.

A. What is the minimal number of features, and which are these, that are sufficient to separate a given class of shapes from all others (Q3, Section 3.4.1)?
Figure 3.6 illustrates this use-case for the classes glasses (a), ant (b), teddy (c), cup (d), and pliers (e). For each class, we select the respective class in the model selector (Figure 3.5(1)) and use next the feature scoring view (Figure 3.5(3)) and separation control (Figure 3.5(6)) to find the feature values (bins of the 1870-dimensional feature vector) that best separate this class from the remaining ones. We assess separation both visually, using the refined projection (Figure 3.5(5)) and its corresponding NH_s score. We find, this way, that these classes can be separated very well from the rest of the database by a *maximally three*, and sometimes just one, feature bin(s) of the 1870 computed ones, as indicated in Figure 3.6. This is, we believe, a quite powerful (and novel) result as it indicates that very little computational effort is needed for classifying shapes in the Princeton Shape Database (and, by extension, in other similar databases). In turn, this can considerably increase the scalability of applications such as shape retrieval and classification.

B. How can we explain the discriminatory power of the features found in use-case A?

The computed feature scoring and the clear separation shown in the refined projection scatterplots (Figure 3.6) are, in principle, enough to let us choose the minimal set of feature-bins needed to separate a class from all others. However, it is useful to double-check and explain

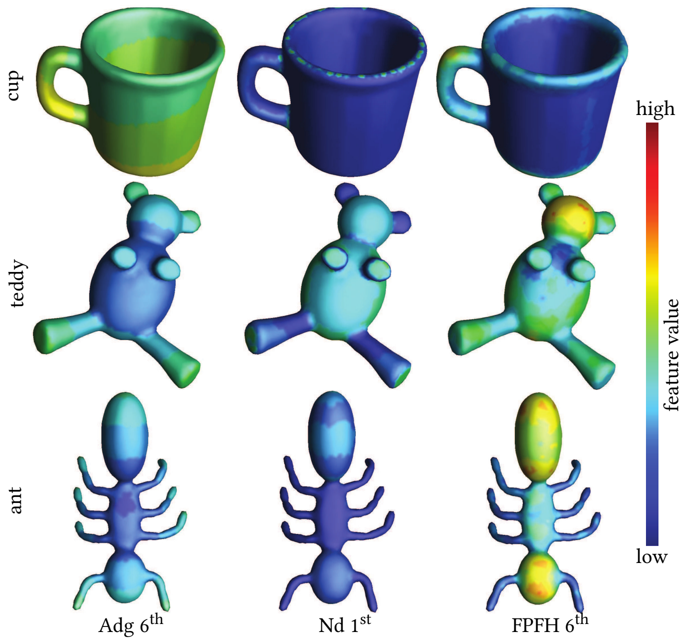


Figure 3.7: Feature-bins (Agd 6th bin, Nd 1st bin, and FPFH 6th bin) mapped on three shapes. This shows how these specific feature bins can effectively separate these shapes.

this discriminatory power, to ensure that the features found this way indeed reflect meaningful different properties of the respective shape classes. For this, we choose shapes from the analyzed classes in use-case A and color them by the values of the features found in the same step to be strongly characteristic of specific classes. Figure 3.7 shows this for three such shapes and feature-bins, respectively. As visible, the three feature-bins take indeed different values for the three shapes. Atop of this insight (which we already knew from the analysis shown in Figure 3.6), we also see now that the Nd feature has indeed low values on thin shape parts and large values on thick ones, respectively. Similarly, we see that the Agd is large for shape protrusions (*e.g.* ant and teddy legs) but is small for central shape parts (*e.g.* teddy rump). The FPFH feature is harder to interpret visually; still, we can see how it gets high values on roughly round shape parts (teddy head and ant's first and last segments) and low values elsewhere.

C. How can we create a good scatterplot which separates well all classes?

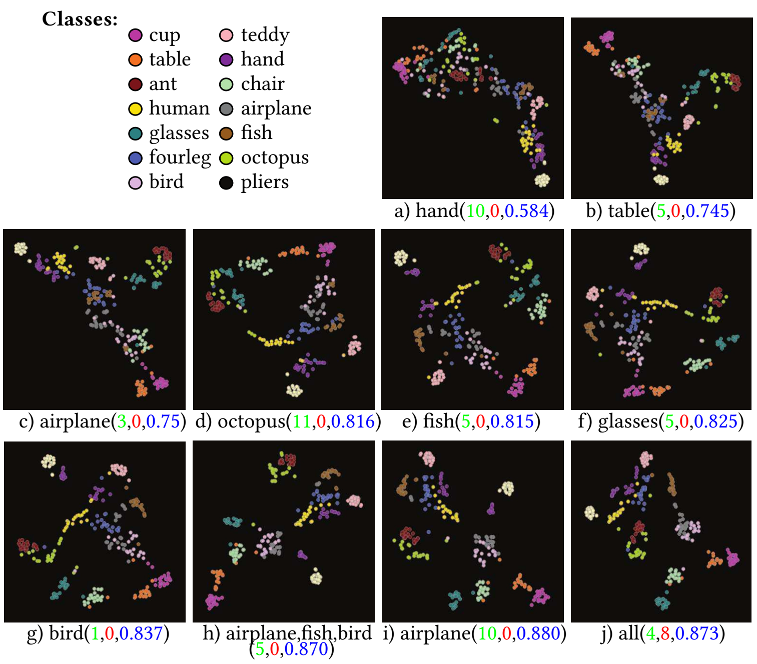


Figure 3.8: Incremental creation of high-quality projection scatterplot that separates all classes well. In each step, a few feature-bins (having high scores, count indicated in green) are selected to separate one or several classes from the rest, and a few feature-bins (having low scores, count indicated in red) are removed from the selection. NH_s at each step are rendered blue.

Figure 3.8 shows an example workflow for this task. We start here with a scatterplot that uses all 1870 features. Next, we search, using the feature scoring view (Figure 3.5(3)), for the feature-bins that are most discriminatory, *i.e.,* have the highest scores, for each of the classes in our database, starting with the hand class (we could start from any other class). As we progress investigating subsequent classes, we add feature-bins which are discriminatory for these newly visited classes. In the end, when we have considered all classes, we also remove features, from the already added ones, which have low scores (that is, bring the least discrimination value or even work adversely for this task). The entire process can be done in one to two minutes. The different images in Figure 3.8 show us how the quality (NH_s value) of the scatterplot almost monotonically improves as we add more feature-bins by considering new classes. Note that, in this process, we may visit a certain class several times (*e.g.* airplane), as features that score high for it may appear several times during the exploration as we add other classes. The final result (Figure 3.8j) contains all 14 classes, has a value $NH_s = 0.873$, and is obtained with a total set of 51 feature-bins. Note that this final NH_s value is *higher* than the one we found by the exhaustive search ($NH_s = 0.831$, Table 1). Indeed, our manual search is more fine-grained, as it allows us to consider individual *feature-bins* (of 1870 in total), whereas the automatic search only considered entire *features* (of the 8 in total). Also, note that obtaining this result by exhaustive search would be prohibitively expensive, as this would involve searching all 2^{1870} combinations.

3.5 DISCUSSION

We discuss next several aspects of our method, as follows:

Dimensionality reduction: Currently, we directly reduce the dimensionality of our 1870-dimensional feature vector to 2D using either t-SNE or UMAP. However, this may be too hard a task for these projection methods to perform, while at the same time preserving neighborhoods. Another approach is to use more dedicated methods, such as autoencoders, to reduce dimensionality to a lower value, and then project this low-dimensional dataset to 2D using t-SNE or UMAP. However, it has to be checked whether this approach can yield final 2D scatterplots that better preserve the structure of the shape database. Concerning the choice of projection techniques, we used t-SNE and UMAP as these techniques are known for their ability to separate well, in the visual space, clusters of similar observations, as opposed to other projection techniques [42].

Scalability: Our method depends on two key parameters in this respect, namely the number of *shapes* in the database to be explored, and the number of *features* which are extracted from each shape. From a computational viewpoint, feature extraction can be done offline, as shapes are changed and/or new shapes are added to the database. Since, typically, shape databases do not change with a high frequency, such an offline extraction can be done without impeding the performance of the end-user. Moreover, features can be extracted in parallel, both among themselves and over different shapes. We compute the t-SNE projection using the *scikit-learn* implementation, which projects several hundreds of instances in a few seconds; the UMAP implementation, provided by the authors [102], works in real-time for this dataset size. If needed, other, faster projections can be used [114]. From a visualization viewpoint, the scatterplot, barchart, and matrix plot metaphors we use scale well to hundreds of thousands of points (shapes) and tens of features.

Evaluation: One important aspect concerning our proposal is evaluating its effectiveness for different types of tasks and users. In detail, we identify *end-users*, for whom tasks involve getting an overview of a shape database, finding similar groups of shapes, finding which features make two shape groups similar (or different), and finding outlier shapes; and *technical users*, for whom tasks involve selecting a small set of features able to create effective visualizations for the first user group. We consider such evaluations to be part of future work.

3.6 CONCLUSION

We have presented an interactive visual analytics system for exploring 3D shape databases for CBSR applications. After reducing shapes to high-dimensional feature vectors following standard feature extraction, we visualize the similarity structure of a database by using dimensionality reduction. To this end, we offer several mechanisms for creating projections in which different shape classes are separated well from each other. First, we use a scagnostics approach to generate near-optimal projections based on maximizing the quality of resulting projections, using a greedy heuristic to optimize the search for suitable feature-sets. Next, we propose a visual analytics approach to enable users to select a small feature subset for separating specific classes, generate high-separation projections for all classes, and gauge the separation power (thus, added-value) of all available features. We show that this visual analytics approach allows generating projections with better separation quality than automatic approaches, and also helps finding both discriminating features (to be used in a CBSR system) and confusing features (of little value for such systems). Our approach can be applied to any 3D shape database and feature-set, allowing CBSR engineers to streamline the process of designing and selecting effective features for shape classification and retrieval. We demonstrate our work on a real-world 3D shape database.

Several extensions of this work are possible, as follows. First and foremost, performing a user study to gauge how well our approach can support exploration tasks of typical end-users, is an important addition. Secondly, since our approach is generic, it could be used to optimize feature selection in other applications beyond CBSR, *e.g.*, in image classification. More specifically, in the next chapter, we show how we can increase the computational scalability, ease of use with little or no user effort, and robustness to small shape changes of the overviews introduced here for database exploration.

4 SCALABLE VISUAL EXPLORATION OF 3D SHAPE DATABASES

In Chapter 3, we have presented an interactive visual analytics system for exploring 3D shape databases, by using feature selection based on discriminative score of features and dimensionality reduction (DR) techniques. As stated there, we provide several methods, including automatical and user-driven methods, to create visual overviews for 3D shape databases exploration.

However although visually effective, the approach in Chapter 3 has a few limitations: (1) The hand-engineered features it uses may not always best capture shape similarity, and also are delicate to compute for poor-quality (non-watertight, self-intersecting, and/or variable-resolution meshes). (2) The feature extraction and dimensionality reduction (DR) steps are computationally quite slow, and cannot scale to real-world databases containing tens of thousands of shapes or more; (3) The underlying DR technique used, t-SNE, is non-deterministic, meaning that overviews created from the same database (let alone databases where a few shapes change) will be different, which makes it difficult for users to maintain their mental map. In this chapter, to overcome these limitations, we extract features from 3D shapes using deep learning. Separately, we use deep learning for the deterministic computation of DR projections. Together, this dual usage of deep learning overcomes the two aforementioned limitations of the method in Chapter 3. As an extra added value point, the approach presented in this chapter for creating 3D shape database overviews is also simpler to use, requiring little to no user effort.

This chapter is structured as follows. Section 4.1 describes related work in deep learning techniques. Section 4.2 presents the deep-learning methods we used for the feature extraction and DR projection. Section 4.3 discusses our proposal. Section 4.4 concludes the chapter.

4.1 RELATED WORK

Machine learning techniques have been used for 3D shape analysis tasks in recent years. Various deep networks have been proposed. View-based methods [24, 146, 179, 180] project 3D shape in to 2D images from different angles and use standard 2D conventional neural networks (CNN) to compute features. The learned 2D features then are used to compute a global feature by pooling layers for classification. 3D CNN methods [101, 167] treat 3D shapes as structured 3D grids and use 3D CNN kernels for 3D shape classification and segmentation. In recent years, PointNet had been proposed to classify 3D shapes which represented as point clouds. PointNet [116] uses a multi-layer perceptron (MLP) to learn the features of each point, and uses a max pooling layer to compute the global feature.

These methods achieve high accuracy on 3D shape classification tasks. They all compute latent global features for 3D shapes, which are used as the input of fully connected layers in their network structures. Although these latent global features are not easy to understand for us human beings, they are good representatives for 3D shapes in our work to substitute traditional shape features. They can be computed fast when we have the trained network; they are stable since many databases do not change very quickly; and they have a powerful ability for classification task, which will be good for our projection destination. Hence we apply PointNet as the representative method to extract 3D shape features in this chapter.

Another machine learning method we use in this chapter is NNproj [43], which uses neural networks to learn the traditional multidimensional projection methods (Section 4.2). By learning the projected position of the high dimensional data generated by providing methods, NNproj can put objects at desired positions as long as the data distribution is similar to training data. Though the results of NNproj are not as good as its learned methods, such as t-SNE, they are much faster. NNproj is the only one we know in this class of methods. We use it in this chapter to project the latent feature learned by PointNet.

4.2 FEATURE LEARNING METHOD

Our proposal in Chapter 3 so far showed how we can construct good-quality projections for exploring 3D shape databases. However, our solution has several limitations:

- **input quality:** Computing the features in Section 3.3.3 involves many constraints. For instance, computing *Agd* requires the meshes to have a single connected component; computing *Gc* requires meshes to be manifold and water-tight. Overall, poor-quality meshes (containing self-intersections, holes, and/or non-uniform sampling) cause serious problems for feature computation;

- **user effort:** The feature selection process (Section 3.4.1), although able to lead to good-quality projections, is time consuming for the user and involves a non-negligible amount of trial and error;

- **replicability:** The used projections (t-SNE and UMAP) are non-parametric. That is, projecting the same (let alone slightly changed) shape database will lead to different scatterplots, thereby not helping users to maintain their mental map of the database;

- **scalability:** For large databases (more than a few hundred shapes), the feature extraction and projection takes considerable amounts of time;
- **ease of use:** Implementing and setting up the extraction of hand-engineered features (Section 3.3.3) is a highly involved process.

We address all the above issues jointly by using a deep learning approach for both feature extraction (also next feature learning, following deep learning terminology) and projection. Concretely, we adopt PointNet [116] for feature extraction and NNproj [43] for projection. We next outline this approach and its components.

PointNet is a deep-learning model used to classify 3D shapes represented as point clouds with high accuracy [116] (see Figure 4.1, blue part). For our visualization goals, and as a replacement of the hand-engineered features described in Section 3.3.3, we use the latent features extracted by PointNet (see Figure 4.1, yellow part), after training it for its original classification task using the labels present in the shape database. NNproj is also using deep learning to create high-quality DR projections of arbitrary high-dimensional data [43]. It is trained by providing it with several 2D scatterplots of corresponding feature vectors, created by any desired DR technique, *e.g.*, t-SNE. We use NNproj to replace t-SNE and UMAP in our visualization construction.

Our framework proposes three pipelines (P1, P2, P3) to create shape-database overviews, as follows. The legend in Figure 4.1 shows which models (networks) are part of the training, respectively, inference, of each pipeline. P1 includes PointNet feature-extraction followed by standard t-SNE projection thereof. P1 can already create overviews, but these do not support incremental updating, since t-SNE is non-parametric. Hence, we use P1 mainly for training P2 and P3, as outlined next. P2 runs P1, then trains NNproj to imitate the thus-constructed t-SNE projections, and then uses the trained NNproj instead of t-SNE to create the final projection. P3 drops the classification part of PointNet and trains the joint PointNet-NNProj network. To train P3, we use projections for the training-set shapes created with P1 or P2. In other words: The three pipelines are not different solutions for the same end goal. Rather, P1 is a lower-level pipeline, needed to train PointNet for feature extraction; while P2 and P3 are, functionally identical pipelines that users can choose to use for the final projection construction. The difference between P2 and P3 is simply whether feature extraction and projection are learned separately (P2) or jointly (P3).

For training all models described above, we use the ShapeNet [130] database, which has 14921 shapes from 16 classes. We divide it into a training set (12137 shapes) and a test set (2784 shapes). As this database

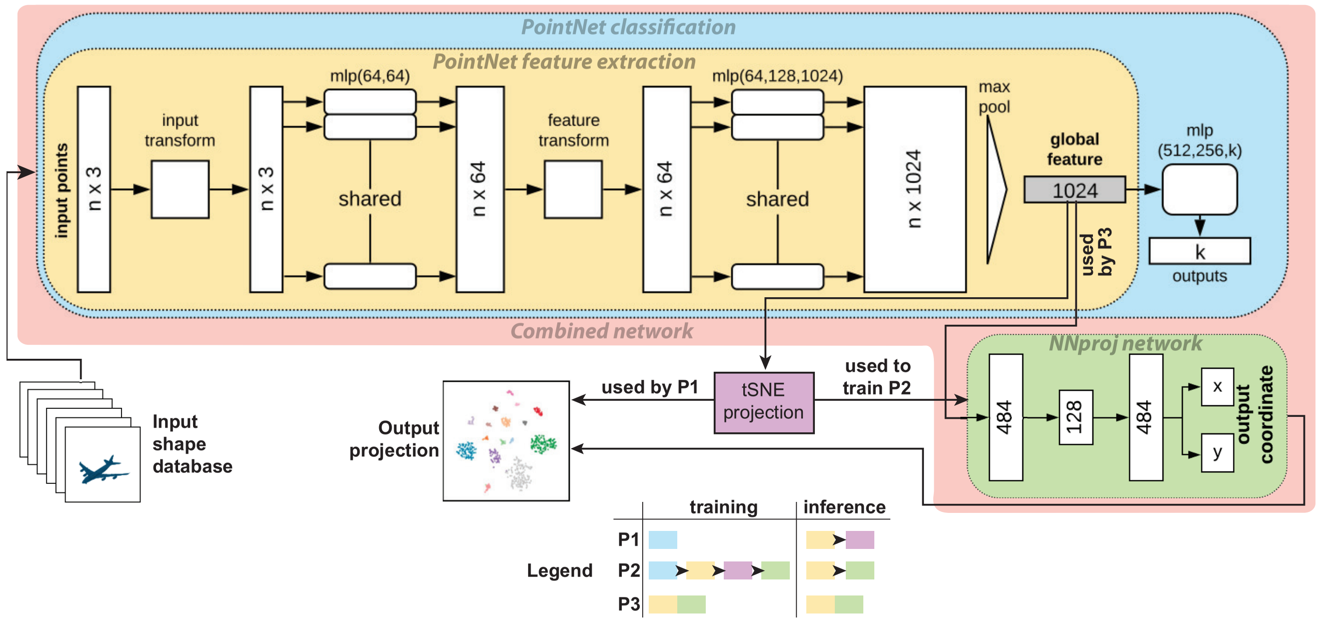


Figure 4.1: Architecture of proposed networks P1, P2, P3.

is quite large, and we have several models to train, we conducted experiments to find how large the training sets of all our deep learning models need to be for sufficient projection accuracy. Specifically, these are:

- NP: the number of shapes for training PointNet,
- NN: the number of feature vectors to train NNProj, and
- NC: the number of shapes for training P3.

We tested 13 values for each of NP, NN, and NC, ranging from 320 to 12137. All networks were trained with 250 epochs and early stopping.

4.2.1 *Experiments and results*

We now present the results of our three pipelines (P1, P2, P3) introduced above and how these depend on the sizes of their respective training sets NP, NN, and NC. We evaluate results both qualitatively (by examining the output projections) and quantitatively, by the NH metric (Section 3.3). Since our scatterplots are now larger than those discussed in Section 3.4, we use now a correspondingly larger value $k = 20$ to compute NH. For simplicity of exposition, we next compute and discuss only the per-projection neighborhood hit (Eqn. 3.8), denoted next as NH. We structure our evaluation along several points, as follows.

Results: Figure 4.2 shows the overview projection created by P2, with shape icons added to a subset of the database shapes, to limit occlusion. The full projection is shown in the top-left inset as a scatterplot colored by class labels. We see that the projection matches our overall expectations: Shapes from different classes are separated well, and similar shapes are close to each other. As with any feature extractor, including the original PointNet, some anomalies exist however. For example, we see a green chair model (A) surrounded by laptop shapes; and a purple lamp model (B) surrounded by table models. This clearly happens since these two shapes are geometrically very similar to the respective classes. Separately, the overview helps us seeing structure *within* classes. For instance, the table class appears to be visually split into four-legged (FL), round (RO), and bureaus or desks (BU). Note that this information is not available in the original labels of the shape database; it is only the projection that helped us find it.

How much data (NP) is needed to train PointNet? Table 3 shows the test accuracy AC of PointNet for different training-set sizes NP. As NP increases, AC also increases until reaching a local maximum ($AC = 97.1\%$) for $NP = 6000$ shapes. The global maximum $AC = 97.6\%$

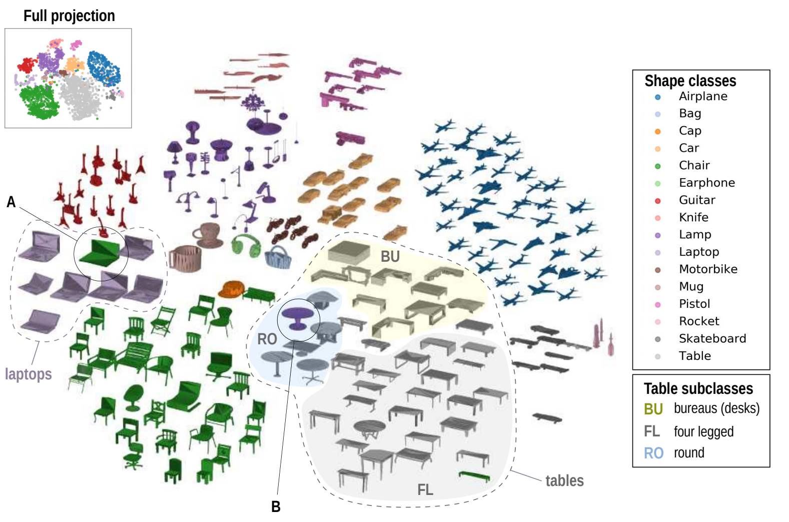


Figure 4.2: Overview projection created by pipeline P2.

Table 3: PointNet training accuracy (AC) and NH values for pipeline P1 when training ($P1Train$) and testing ($P1Test$). The two color legends at the bottom show accuracy (green shades) and NH values (yellow-red) respectively in this table and the following ones.

NP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
AC	0.825	0.935	0.954	0.958	0.962	0.963	0.971	0.968	0.971	0.975	0.972	0.972	0.976
P1Train	0.934	0.976	0.985	0.997	0.996	0.99	0.998	0.985	0.992	0.993	0.992	0.991	0.987
P1Test	0.848	0.915	0.931	0.938	0.942	0.934	0.949	0.932	0.945	0.945	0.94	0.943	0.945

low AC	0.825	0.842	0.859	0.875	0.892	0.909	0.926	0.942	0.959	0.976	high AC
low NH	0.703	0.736	0.769	0.801	0.834	0.867	0.9	0.932	0.965	0.998	high NH

is achieved, as expected, when using all $NP = 12137$ shapes in the training set. We also see that for $NP = 2000$ we already get a very good accuracy $AC = 95.4\%$, sufficient for our visualization goals. With the trained PointNet, we next extract features and project them using t-SNE (pipeline P1). Table 3, row $P1Train$ shows the NH projection quality metric for P1 on its training sets of various sizes NP. Next, row $P1Test$ shows the same NH metric, this time for the test set. While NH is slightly higher for the training set (as expected), the NH values for the test set are also quite high, indicating that P1 produces good quality projections.

How much data (NP, NN) is needed to train P2? Training P2 requires training PointNet with NP shapes and next training NNproj with NN feature vectors (Figure 4.1). So, P2's quality depends on both NP and NN. Table 4 (a–c) shows this dependency. In detail: Table 4(a) shows the NH value of t-SNE when projecting different sizes NN of feature vector sets extracted by PointNet. We notice that there are some NH fluctuations on the second row where $NN = 320$. However, when

NP and *NN* are both greater than 1000, the t-SNE projections all yield good *NH* values (above 94%). The highest *NH* value (99.8%) appears for $NP = NN = 6000$. We also see that the colors in the upper-right triangle half of Table 4(a) are darker than in the lower-left triangle half: *NH* is slightly higher when $NP \geq NN$. Indeed, when $NP \geq NN$, the input data of t-SNE is a subset of PointNet's training data. In contrast, when $NP < NN$, t-SNE runs with some shapes that are not in PointNet's training set.

After creating the ground-truth scatterplots by t-SNE, we use them to train NNProj. After this, P2 is ready to be used. Table 4(b) shows the *NH* of P2 trained with different *NN* and *NP* values, when projecting NNproj's training-data. The values in Table 4(b) are very close to their counterparts in Table 4(a), being roughly 0.1% to 2% lower. This means

Table 4: *NH* projection quality for (a) ground-truth of pipeline P2; (b) P2 training; (c) P2 testing; and (d) P3, for different values of the respective training-set sizes *NP*, *NN*, and *NC*. Color mapping follows the one in Table 3

(a) P2 ground-truth

NN\NP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
320	0.934	0.867	0.889	0.923	0.905	0.824	0.932	0.785	0.866	0.873	0.858	0.843	0.812
1000	0.888	0.976	0.977	0.991	0.994	0.961	0.997	0.942	0.974	0.961	0.969	0.952	0.957
2000	0.897	0.96	0.985	0.996	0.997	0.982	0.996	0.97	0.985	0.982	0.983	0.978	0.981
3000	0.9	0.954	0.974	0.997	0.996	0.986	0.997	0.977	0.99	0.987	0.988	0.982	0.983
4000	0.901	0.953	0.969	0.99	0.996	0.989	0.998	0.981	0.99	0.989	0.989	0.985	0.983
5000	0.905	0.953	0.965	0.982	0.989	0.99	0.997	0.981	0.992	0.991	0.99	0.987	0.985
6000	0.907	0.953	0.963	0.978	0.983	0.985	0.998	0.984	0.992	0.99	0.991	0.987	0.984
7000	0.907	0.954	0.961	0.976	0.982	0.981	0.993	0.985	0.993	0.991	0.992	0.99	0.983
8000	0.911	0.952	0.961	0.974	0.98	0.979	0.988	0.982	0.992	0.991	0.991	0.989	0.982
9000	0.912	0.952	0.963	0.975	0.98	0.977	0.987	0.98	0.99	0.993	0.992	0.989	0.983
10000	0.915	0.953	0.963	0.974	0.979	0.977	0.985	0.981	0.989	0.991	0.992	0.991	0.985
11000	0.919	0.955	0.966	0.976	0.98	0.978	0.986	0.982	0.99	0.991	0.992	0.991	0.986
12137	0.916	0.955	0.967	0.975	0.978	0.976	0.985	0.981	0.989	0.988	0.991	0.99	0.987

(b) P2 training

NN\NP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
320	0.935	0.865	0.882	0.923	0.906	0.81	0.932	0.782	0.858	0.872	0.858	0.843	0.812
1000	0.886	0.972	0.971	0.991	0.992	0.958	0.996	0.936	0.973	0.963	0.963	0.939	0.951
2000	0.893	0.956	0.983	0.996	0.996	0.964	0.994	0.962	0.978	0.978	0.981	0.971	0.976
3000	0.892	0.951	0.973	0.997	0.992	0.982	0.997	0.97	0.99	0.984	0.982	0.977	0.969
4000	0.894	0.947	0.965	0.991	0.995	0.986	0.998	0.978	0.984	0.984	0.986	0.979	0.974
5000	0.897	0.95	0.961	0.978	0.987	0.984	0.996	0.975	0.985	0.986	0.988	0.984	0.98
6000	0.893	0.948	0.956	0.976	0.979	0.981	0.996	0.978	0.988	0.986	0.984	0.982	0.974
7000	0.897	0.945	0.955	0.973	0.979	0.978	0.992	0.978	0.986	0.988	0.989	0.98	0.978
8000	0.9	0.943	0.952	0.967	0.974	0.975	0.986	0.974	0.987	0.988	0.988	0.982	0.976
9000	0.897	0.943	0.951	0.97	0.968	0.968	0.983	0.968	0.987	0.986	0.988	0.98	0.977
10000	0.897	0.939	0.952	0.963	0.971	0.973	0.982	0.963	0.985	0.982	0.987	0.983	0.978
11000	0.896	0.94	0.955	0.966	0.966	0.968	0.978	0.974	0.983	0.986	0.984	0.981	0.975
12137	0.893	0.939	0.952	0.967	0.972	0.965	0.981	0.968	0.985	0.982	0.982	0.975	0.974

(c) P2 testing

NN\NP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
320	0.766	0.829	0.857	0.89	0.858	0.847	0.905	0.812	0.877	0.901	0.88	0.866	0.854
1000	0.809	0.877	0.886	0.9	0.919	0.911	0.927	0.898	0.918	0.914	0.924	0.914	0.924
2000	0.813	0.878	0.888	0.909	0.923	0.911	0.923	0.899	0.916	0.924	0.919	0.916	0.923
3000	0.816	0.887	0.897	0.904	0.909	0.922	0.93	0.908	0.925	0.919	0.928	0.92	0.916
4000	0.823	0.897	0.897	0.905	0.912	0.919	0.925	0.916	0.924	0.917	0.926	0.921	0.912
5000	0.827	0.897	0.897	0.905	0.918	0.919	0.93	0.905	0.92	0.923	0.929	0.925	0.924
6000	0.83	0.897	0.903	0.908	0.91	0.917	0.924	0.907	0.922	0.917	0.916	0.923	0.909
7000	0.83	0.896	0.904	0.908	0.922	0.925	0.929	0.906	0.917	0.923	0.92	0.923	0.917
8000	0.835	0.893	0.908	0.906	0.914	0.916	0.932	0.914	0.917	0.928	0.918	0.917	0.926
9000	0.833	0.895	0.905	0.916	0.913	0.916	0.925	0.911	0.929	0.929	0.927	0.917	0.919
10000	0.836	0.885	0.898	0.901	0.915	0.916	0.925	0.906	0.919	0.925	0.913	0.92	0.918
11000	0.84	0.89	0.911	0.911	0.898	0.905	0.919	0.913	0.919	0.923	0.922	0.931	0.915
12137	0.825	0.895	0.894	0.904	0.92	0.914	0.935	0.912	0.926	0.922	0.915	0.916	0.916

(d) P3

NC	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
P1Train	0.934	0.976	0.985	0.997	0.996	0.99	0.998	0.985	0.992	0.993	0.992	0.991	0.987
P1Test	0.848	0.915	0.931	0.938	0.942	0.934	0.949	0.932	0.945	0.945	0.94	0.943	0.945
P2Train	0.935	0.972	0.983	0.997	0.995	0.984	0.996	0.978	0.987	0.986	0.987	0.981	0.974
P2Test	0.766	0.877	0.888	0.904	0.912	0.919	0.924	0.906	0.917	0.929	0.913	0.931	0.916
P3Train	0.927	0.963	0.973	0.993	0.993	0.979	0.99	0.975	0.981	0.988	0.986	0.982	0.979
P3Test	0.703	0.887	0.879	0.912	0.913	0.921	0.909	0.922	0.918	0.93	0.921	0.92	0.919

NH 0.703 0.736 0.769 0.801 0.834 0.867 0.9 0.932 0.965 0.998

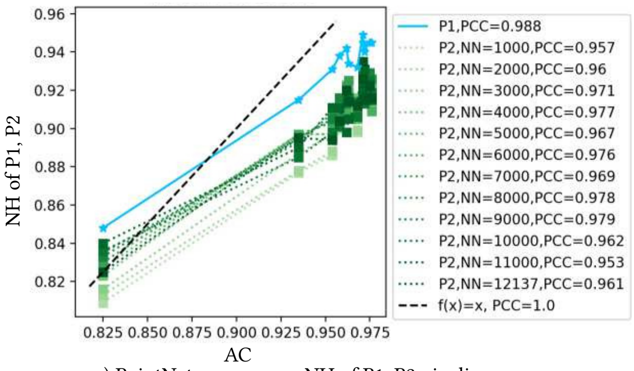
that NNproj was trained successfully, so P2 can project well its training data.

Table 4(c) shows NH values when using P2 to project test data (2784 shapes). Although the NH values are slightly lower than those in Tabs. 4(a) and (b), all of them, except the first one ($NP = 320$, $NN = 320$) outperform those delivered by our earlier feature engineering. Also, we see that $NN = 2000$ and $NP = 3000$ are already enough to deliver sufficiently high NH values, thus, high-quality projections. The overall highest NH is obtained for $NP = 6000$, same as in Tabs. 4(a, b). An interesting phenomenon happens when NP is small (320 or 1000). In this case, we train PointNet with few shapes. We can see that this does not yield high NH values. However, when next using more shapes to train NNproj (NN increases), NH also increases. That is, we can use a small labeled dataset to train PointNet, and then use a larger *unlabeled* dataset to improve P2's performance.

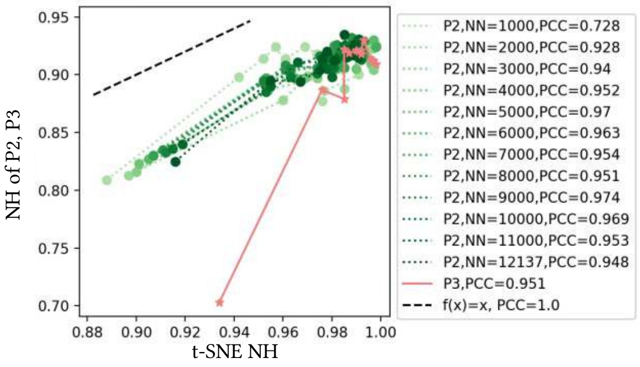
How much data (NC) is enough to train P3? Table 4(d) shows the NH results of P3, trained using P1 for different training set sizes NC, and compares them with those of P1 and P2. To ease comparison, the values in rows $P2Train$ and $P2Test$ in Table 4(d) come from the diagonals of Tabs. 4(b, c) for $NC = NN = NP$. Rows $P1Train$ and $P1Test$ show the NH values of P1 projecting its training, respectively test, data. Rows $P3Train$ and $P3Test$ show the NH values for P3 on training, respectively test, data. From Table 4(d), we see that P3 performs similarly to P2 on both training and test data, with good NH values when having at least $NC = 3000$ training shapes.

How does PointNet's accuracy influence P1 and P2? As explained, PointNet was originally designed for *classification*. However, we use here PointNet's feature vectors for *projection*. So, it is interesting to see if the classification-related accuracy (AC) and projection-related quality (NH) are correlated. Separately, we ask ourselves if there is a relationship between the NH of ground truth t-SNE and the NH of projections created with P2 and P3. To explore this, we draw scatterplots of these values and compute their Pearson Correlation Coefficients (PCC). Figure 4.3(a) shows the AC *vs* P1 NH and AC *vs* P2 NH scatterplots. The plot contains 13 point-groups, one for P1 (blue), and the other 12 for an NN setting of P2 each (green color-coded on NN). These point-groups are visually indicated by different colors and also connected by lines for easing reading. The dotted line shows ideal correlation, for reference. We see that the PCCs of all these lines — each representing an instance of P1 or P2 — are close to 1, so P1 and P2's NH metric directly correlates with PointNet's accuracy.

Next, Figure 4.3(b) shows scatterplots of P2 and P3's NH *vs* t-SNE's NH values. The plot contains 13 point-groups, one for P3 (red), and the other 12 for a NN setting of P2 each (green color-coded on NN). The



a) PointNet accuracy vs NH of P1, P2 pipelines



b) t-SNE NH quality vs NH of P2, P3 pipelines

Figure 4.3: a) Correlation of PointNet's accuracy *AC* with the *NH* quality of pipelines P1 and P2. b) Correlation of t-SNE's *NH* quality with the *NH* quality of pipelines P2 and P3.

PCCs of these lines are also close to 1, so P1 and P2's *NH* quality is directly correlated with the ground-truth (t-SNE)'s quality. A similar correlation — albeit for a different deep learning model for performing projections — was mentioned in [43], but not formally assessed by means of PCC. The data for P3 (red line) may seem at first sight far worse than that for P2 (green lines). However, this is due to a single point for the lowest t-SNE *NH* value. For all other values, the red line is practically in the same area as the green lines, telling that P2 and P3 are very similar from the perspective of t-SNE *vs* deep-learning-network produced *NH* values.

How can we use classification accuracy to interpret projections? In P1 and P2, we trained PointNet for classification. Besides a feature vector, this delivers a *confidence* value for the classification. We can use this value for our visual exploration goal, as follows. Figure 4.4 (top) shows the training-set projected by P2, with point luminances encoding

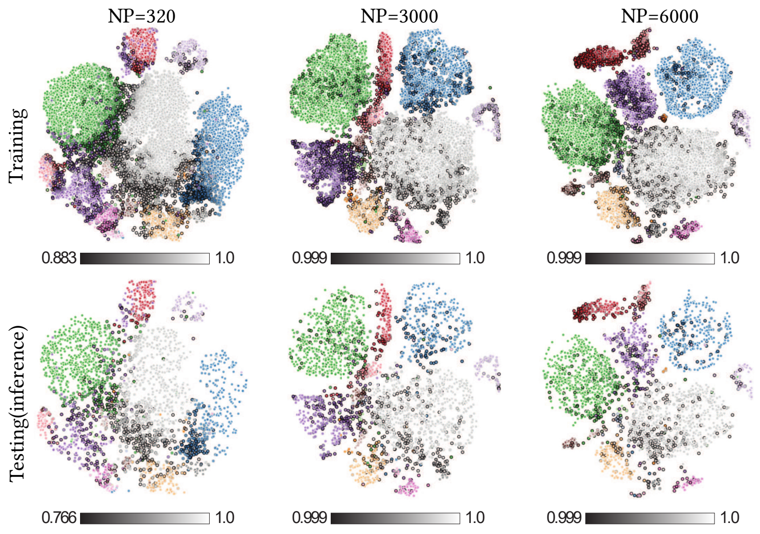


Figure 4.4: Classification confidence values (dark=low, bright=high confidence), during training (top) and testing (bottom) for the P2 pipeline trained with three NP values.

their classification confidences (dark=low, bright=high confidence). We immediately see that confidence is high within same-class clusters and low on the cluster borders. Also, as we increase the number of training samples NP for the PointNet classifier, we see how the confidence nears 1.0 for most samples; although, the lowest-confidence ones still remain on cluster borders. Figure 4.4 (bottom) shows the same visualization for a test-set projected by the trained P2. For a small test set $NP = 300$, we see that confidence is significantly lower than on the training set. For $NP \geq 3000$, test set confidence is basically the same (nearly one) as training set confidence. Importantly, we see that confidence is relatively lowest on the cluster borders also for the test data. We can use these visualizations (on the test data) to assess how confident we are that the shape database projection indeed faithfully reflects the similarities of the underlying shapes. As expected, shapes close to cluster borders are harder to classify, thus, have less discriminant feature vectors and are in turn harder to project well. Upon seeing such images, users can decide to *e.g.* further explore additional information concerning shapes having low classification confidence. This type of insight is crucial when interpreting projections as it is well known that such methods cannot always place all their input data correctly [42, 99, 106].

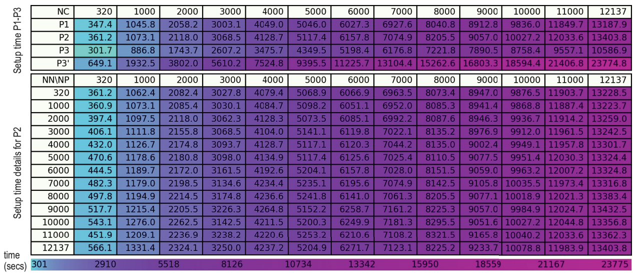
4.2.2 *Computational performance*

We discuss next the computational performance of our three pipelines P1–P3. For this, we split effort into *setup* effort, *i.e.*, the time needed to perform all operations required to have the pipeline ready for inference; and *inference* effort, *i.e.*, the time a pipeline needs to create the projection of a shape database.

Setup time: Table 5 (top) shows the setup time for all three pipelines. Columns N indicate the training set size for the three pipelines, *i.e.*, NP for PointNet, NN for NNproj, and NC for P3, respectively. Row $P1$ shows the setup time for P1, identical to PointNet's training time. Row $P2$ shows the setup time for P2 when $NN = NP$ (Table 5 (bottom) gives more detailed information, see next). The setup time for P2 includes training PointNet, feature extraction, ground truth generation (t-SNE), and training NNproj. Comparing the first two rows in Table 5 (top), we see that training PointNet is dominating the setup of P2. Row $P3$ shows the setup (training) time of P3 when we already have a ground truth projection. We see that training P3 is slightly faster than training P1 since P3's network is slightly simpler. Finally, row $P3'$ shows the setup time for P3 when we use P1 to create the ground truth needed for training it. Table 5 (bottom) shows the setup time of P2 for all combinations of NP and NN in our experiments. Values in this table increase rapidly with NP and slightly with NN. That is, training NNProj is negligible compared to training PointNet.

Inference time: Figure 4.5 shows the projection (inference) time for three pipelines as a function of how many shapes they need to project. We see that all three pipelines have linear time complexity. P1 and P2 are really close and they are about 5 times faster than P1. The high relative cost of P1, and its deviations from a perfect line, are explained by the fact that P1 uses t-SNE, whose cost (a) is high and (b) varies depending on its stochastic initialization, this explaining the wiggles in the blue

Table 5: Setup time, pipelines P1–P3 (top). Setup time of P2 as function of NP and NN (bottom).


Setup time P1-P3
Setup time details for P2
time (secs)
301
2910
5518
8126
10734
13342
15950
18559
21167
23775

NC	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
P1	347.4	1045.8	2058.2	3003.1	4049.0	5046.0	6027.3	6927.6	8040.8	8912.8	9836.0	11849.7	13187.9
P2	361.2	1073.1	2118.0	3068.5	4128.7	5117.4	6157.8	7074.9	8205.5	9057.0	10027.2	12033.6	13403.8
P3	301.7	886.8	1743.7	2607.2	3475.7	4349.5	5198.4	6176.8	7221.8	7890.5	8758.4	9557.1	10586.9
P3'	649.1	1932.5	3802.0	5610.2	7524.8	9395.5	11225.7	13104.4	15262.6	16803.3	18594.4	21406.8	23774.8

NN\NP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
320	361.2	1062.4	2082.4	3027.8	4079.4	5068.9	6066.9	6963.5	8073.4	8947.0	9876.5	11903.7	13228.5
1000	360.9	1073.1	2085.4	3030.1	4084.7	5098.2	6051.1	6952.0	8085.3	8941.4	9868.8	11887.4	13223.7
2000	397.4	1097.5	2118.0	3062.3	4128.3	5073.5	6085.1	6992.2	8087.6	8946.3	9936.7	11914.2	13259.0
3000	406.1	1111.8	2155.8	3068.5	4104.0	5141.1	6119.8	7022.1	8135.2	8976.9	9912.0	11961.5	13242.5
4000	432.0	1126.7	2174.8	3093.7	4128.7	5117.1	6120.3	7044.2	8135.0	9002.4	9949.1	11957.8	13301.7
5000	470.6	1178.6	2180.8	3098.0	4134.9	5117.4	6125.6	7025.4	8110.5	9077.5	9951.4	12030.3	13324.4
6000	444.5	1189.7	2172.0	3161.5	4192.6	5204.1	6157.8	7028.0	8151.5	9059.0	9963.2	12007.2	13324.8
7000	482.3	1179.0	2198.5	3134.6	4234.4	5235.1	6195.6	7074.9	8142.5	9105.8	10035.5	11973.4	13316.8
8000	497.8	1194.9	2214.5	3174.8	4236.6	5241.8	6141.0	7061.3	8205.5	9077.1	10018.9	12021.3	13383.4
9000	517.7	1215.4	2205.5	3226.3	4264.8	5152.2	6258.7	7161.2	8225.3	9057.0	9984.9	12024.7	13432.5
10000	543.1	1276.0	2262.5	3142.5	4211.5	5200.3	6249.9	7181.3	8295.5	9051.6	10027.2	12044.8	13356.8
11000	451.9	1209.1	2236.9	3238.2	4220.6	5253.2	6210.6	7108.2	8321.5	9165.8	10040.2	12033.6	13362.3
12137	566.1	1331.4	2324.1	3250.0	4237.2	5204.9	6271.7	7123.1	8225.2	9233.7	10078.8	11983.9	13403.8

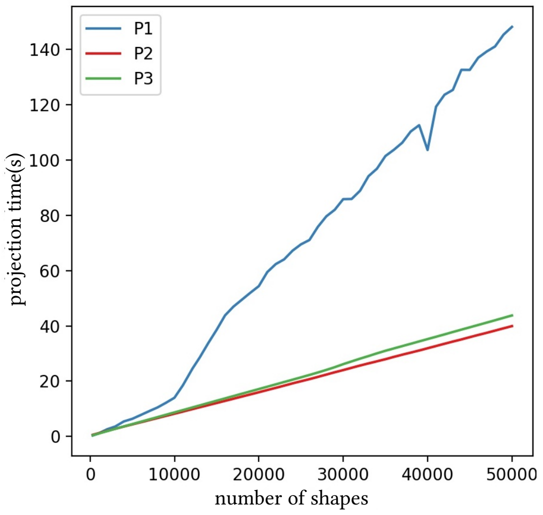


Figure 4.5: Projection time comparison for pipelines P1, P2, P3.

line in Figure 4.5 (for a related analysis, see [42]). In contrast, P2 and P3 show a perfect linear relation with the shape database size, as these are purely deep-learning model executions.

These three pipelines are all much faster than our feature selection method presented in Chapter 3 which takes about 127 hours to extract the 8 features we listed in Table 1 for 320 shapes.

4.3 DISCUSSION

We discuss next several aspects of our proposal, as follows.

Feature selection *vs* feature learning: We have presented, in this and the previous chapter, two approaches for creating shape-database projections: *selecting* features from a pre-computed set based on feature engineering (Section 3.3) *vs* using an automatically *learned* feature vector using deep learning (Section 4.2). We call these next the feature selection (FS) and the feature learning (FL) approaches. Let us discuss and contrast these two approaches.

Both approaches share the same aims, listed as Q1–Q3 in Section 3.4.1. However, from this viewpoint, each approach has its own advantages and limitations. As mentioned at the beginning of Section 4.2, FS has some clear limitations with respect to input shapes, user effort, replicability, scalability, and ease of use; thus, it does not fully address Q1. The FL approach scores very highly on all these points: It accepts any point cloud shape as input, so has no constraints on mesh quality (input shape independence); it works fully automatically, not requiring any specific user input (low user effort); it creates projections deterministically, thus stably upon small-to-medium input changes (replicability); it scales linearly with the input size, being 4 orders of magnitude faster than FS; and it is very easy to deploy, being based on standard deep learning libraries [110]. Also, FL addresses Q2 (Section 3.4.1), *i.e.*, which is the minimal feature-set needed for a good projection, in a different way than FS: Its deep learning approach does not care about feature *selection*, but rather *synthesizes* features which are best for good projection creation. The results in Section 4.2.1 show quite clearly that FL can create high-quality projections this way, without having to worry about feature selection. In contrast, FS *must* consider feature selection, since it is by construction restricted to a fixed number of predefined features.

However, FS has two interrelated advantages over FL: First, it allows users to see and select how features affect the projection (Q2, Section 3.4.1). The bar chart view (Figure 3.4) allows users to find and select features to optimize the projection quality. Secondly, the FL feature scoring view (Figure 3.5) interacts with the projection view to enable users to select specific features that explain the similarity of shapes in a group and/or the separation of several groups. Using these views with the FL features is not straightforward, since NNproj is trained on an *entire* feature set and would need re-training if this set changes. Also, the FL features are *abstract*, *i.e.*, they do not have a concrete meaning for users, thereby making reasoning about them extremely challenging.

Selecting the best feature learning approach: We have studied and presented two approaches for jointly doing feature learning and projection, called P2 and P3. Which one is best? Our results (Section 4.2.1) show that P2 and P3 produce very similar results, quality-wise, given the same (amounts of) training data. P3 exhibits slightly higher quality than P2, which makes sense, as P3 trains jointly for both feature extraction and projection. Separately, the results in Section 4.2.1 show very little variation in the performance of all pipelines as a function of their training set sizes. In practice, as we discussed there, setting NP, NN, and NC around 3000 shapes gives good results for all pipelines, with only minimal improvements obtained when tweaking these training-set size values.

Learning projections: Currently, we train NNproj with all the 1024 PointNet features to create projections. We could reduce this dimensionality to a lower value using an intermediate autoencoder stage, or alternatively using a feature-selection optimization technique as presented in Section 3.4.1. This would possibly make NNproj's task of learning projections easier in terms of training set size required to obtain a certain class separation (NH) and/or epochs needed for convergence. Concerning the choice of projection techniques, we used t-SNE to train NNproj. However, learning other projection techniques such as UMAP may lead to ultimately better, easier to interpret, projections.

Scalability: Both the FS and FL approaches depend linearly on the number of *shapes* in the database to be explored and the number of *features* which are extracted from each shape. Yet, as explained already, FS is 4 orders of magnitudes slower. For handling real-world databases of tens of thousands of shapes, like ShapeNet, the FL approach is clearly more suitable. Note that both FS and FL approaches can be applied offline, *i.e.*, when shapes are changed and/or new shapes are added to the database.

4.4 CONCLUSION

We started this chapter by stating our aim of improving our approaches presented in Chapter 3, that is, to construct a visual overview of large 3D shape databases with a stable, robust, easy to use, and high-quality pipeline. Speed wise, we believe to have largely reached our goals. The deep learning pipelines presented in this chapter are computationally very scalable during the inference process. Their training stage is also faster than the feature engineering methods discussed in Chapter 3. Putting it together, that is, adding training and inference time, the pipeline presented in this chapter is faster than the functionally comparable one presented in Chapter 3. Quality-wise, the computed neighborhood hit (NH) values indicate that our deep learning pipelines create good quality projections, the differences with respect to the similar quality metrics computed on ground-truth t-SNE projections being very small and, we argue, not affecting the visual quality of our deep-learned projections or their practical usage in applications. Separately, following the earlier analyses of NNproj [43], our own experience during the evaluations presented here, and the deterministic way in which neural networks of the NNproj architecture operate, our visualizations should be stable to additions, modifications, and removal of shapes in a database. This directly corresponds to the out of sample capability of NNproj. Last but not least, our deep learning approach is easy to use as it requires no user intervention during inference and only a minimal set-up of the hyperparameters during the training stage.

While the above results strongly support our claims on added value, our current approach can be further improved, as it still has several limitations. Given that the proposed deep learning pipelines are black-box methods, when their results are not good, it is not evident what the user should do to correct or improve that situation. For example, when a shape gets projected in an ‘odd’ position, *e.g.*, far away from similar shapes, it is hard to know how to change the training process so as to ‘move’ the respective shape closer to similar ones in the projection.

Conversely, when the resulting projections appear satisfactory, which is desirable, it is not easy to find the hard evidence to prove why this process worked satisfactorily. This, in turn, means that it is hard to give strong guarantees about the generalization power of the trained neural networks, or, how they would project new 3D shapes which are, potentially, quite different from the ones seen during training. These are well-known problems that our approach shares with other machine learning models. Apart from these issues, an unexplored, but potentially interesting direction, is to extend our approach to other media types besides 3D shapes. Indeed, nothing specific exists in our pipelines presented in this chapter that ties them to use only 3D shapes. Having a replacement network for PointNet which classifies other media types than 3D shapes would allow us to directly generalize our exploration views to such other content types, like image, video, or audio databases.

5 SKELETON-AND-TRACKBALL ROTATION FOR 3D SCENES

In Chapters 3 and 4 we described several methods for generating presentations of large 3D shape databases. Here, we move our focus onto our second goal: examination of individual shapes. This chapter introduces a method for specifying rotations of 3D shapes, which plays an important role in the general task of examining 3D shapes. Finding a good way to examine a 3D model can be difficult. An effective and convenient mechanism to do this will save a lot of effort when manipulating 3D scenes and trying to find the right angles for visual examination. To this end, a great many methods have been proposed [6, 19, 23, 51, 67, 150], with many of these methods implemented in various 3D modeling systems. However, all these methods have various limitations in the sense of the types of shapes they can handle, with or without preprocessing; ease of use and learning; and effectiveness in the type of rotations they can generate.

We aim to enlarge the palette of techniques available to users for specifying 3D rotations, and to do this, we exploit information readily available in the shape itself; or more precisely, in the shape's projection on the screen. We present a new technique for specifying rotations of 3D shapes around axes inferred from the local shape structure. We compare our method with classical trackball rotation, both in isolation and in combination, in a controlled user study. The results show that, when combined with trackball, skeleton-based rotation reduces task completion times and increases user satisfaction, while not introducing additional costs, being thus an interesting addition to the palette of 3D manipulation tools.

5.1 INTRODUCTION

Interactive exploration and navigation of 3D scenes is essential in many applications such as CAD/CAM modeling, computer games, and data visualization [67]. 3D rotations are an important interaction tool, as they allow examining scenes from various viewpoints. Two main 3D rotation types exist: rotation around a *center* and rotation around an *axis*. Rotation around a center can be easily specified via classical mouse-and-keyboard [187] or touch interfaces [182] by well-known metaphors such as the virtual trackball [56]. Axis rotation is easy to specify if the axis matches one of the world-coordinate axes. Rotations around arbitrary axes are much harder to specify, as this requires a total of 5 degrees of freedom (4 for the axis and one for the rotation angle around the axis).

Moreover, as explained in Section 2.3, several tools require more parameters to be specified to derive such rotations, to make the specification process more natural.

For certain tasks, users do not need to rotate around *any* 3D axis. Consider examining a (complex) 3D shape such as a statue: We can argue that a natural way to display this shape is with the statue's head upwards; and a good way to explore the shape from all viewpoints is to rotate it around its vertical symmetry axis while keeping its upwards orientation fixed.

Several methods support this exploration scenario by aligning the shape's main symmetry axis with one of the world coordinate axes and then using a simple-to-specify rotation around this world axis [36]. This falls short when (a) the studied shape does not admit a *global* symmetry axis, although its parts may have local symmetry axes; (b) computing such (local or global) symmetry axes is not simple; or (c) we do not want to first align the shape with a world axis.

To address the above, we propose a novel interaction mechanism based on local symmetry axes: The user points at a region of interest (part) of the viewed 3D shape, from which a local symmetry axis is computed. Next, one can rotate the shape around this axis with an interactively specified angle. This method allows an easy selection of parts and automatic computation of their approximate 3D symmetry axes, both done using the shape silhouette's 2D skeleton. The method handles any 3D scene, *e.g.*, polygon mesh or polygon soup, point-based or splat-based rendering, or a combination thereof, without preprocessing; and works at interactive rates for scenes of hundreds of thousands of primitives.

The skeleton-based rotation is not to be seen as a replacement, but a *complement*, of classical trackball rotation. Yet, what this precisely means, *i.e.*, how the two rotation mechanisms perform when used in practice, either separately or jointly, is an open question; also, a formal evaluation of the effectiveness of skeleton-based rotation is an important open research question.

Hence, we present the design and execution of a controlled user study aimed at gauging the added value of skeleton-based rotation when used against, but also combined with, trackball rotation. The results of our study show that, when used together with trackball rotation, skeleton-based rotation brings in added value, therefore being a good complement, and not a replacement, of trackball rotation. The structure of this chapter is as follows. Section 5.2 presents related work on skeleton computation. Section 5.3 details the skeleton-based rotation method. Section 5.4 presents a formative evaluation aimed at finding out how the skeleton-based rotation is received by users. Section 5.5 presents an in-depth quantitative and qualitative user study that studies the hypotheses outlined by the formative study. Section 5.6 discusses the skeleton-based rotation and our findings regarding its best ways of use. Section 5.7 concludes the chapter.

5.2 RELATED WORK

We have shown the related work in interactive rotation specification and skeleton computation in Chapter 2. Here we repeat some definitions of medial descriptors, skeletons, for helping some readers who jump into this chapter directly and also refresh the memory of other readers.

Skeletons have been used for decades to capture the symmetry structure of shapes [14, 139]. Recalling the notations introduced in Section 2.4.1 for ease of reading, for shapes $\Omega \subset \mathbb{R}^n$, $n \in \{2, 3\}$ with boundary $\partial\Omega$, skeletons are defined as

$$S_\Omega = \{\mathbf{x} \in \Omega | \exists \mathbf{f}_1 \in \partial\Omega, \mathbf{f}_2 \in \partial\Omega : \mathbf{f}_1 \neq \mathbf{f}_2 \wedge ||\mathbf{x}-\mathbf{f}_1|| = ||\mathbf{x}-\mathbf{f}_2|| = DT_\Omega(\mathbf{x}\}, \tag{5.1}$$

where $\mathbf{f}_i$ are called the *feature points* [103] of skeletal point $\mathbf{x}$ and DT_Ω is the distance transform [31, 124] of skeletal point $\mathbf{x}$, defined as

$$DT_\Omega(\mathbf{x} \in \Omega) = \min_{\mathbf{y} \in \partial\Omega} ||\mathbf{x} - \mathbf{y}||. \tag{5.2}$$

These feature points define the so-called *feature transform* [57, 149]

$$FT_\Omega(\mathbf{x} \in \Omega) = \arg\min_{\mathbf{y} \in \partial\Omega} ||\mathbf{x} - \mathbf{y}||, \tag{5.3}$$

which gives, for each point $\mathbf{x}$ in a shape Ω, its set of feature points on $\partial\Omega$, or contact points with $\partial\Omega$ of the maximally inscribed disk in Ω centered at $\mathbf{x}$.

To compute skeletons of 2D shapes, state-of-the-art methods *regularize* the skeleton by removing its spurious branches caused by small noise perturbations of the boundary $\partial\Omega$. Regularization typically defines a so-called *importance* $\rho(\mathbf{x}) \in \mathbb{R}^+ | \mathbf{x} \in S_\Omega$ which is low on noise branches and high elsewhere on S_Ω. Several authors [31, 44, 45, 108, 158] set ρ to the length of the shortest path along $\partial\Omega$ between the two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{x}$. Upper thresholding ρ by a sufficiently high value removes noise branches.

In 3D, *curve skeletons* are curve-sets in $\mathbb{R}^3$ that locally capture the tubular symmetry of shapes [30]. Curve skeletons still cannot be computed in real time, and require a well-cured definition of Ω as a watertight, non-self-intersecting, fine mesh [143] or a high-resolution voxel volume [45, 118].

Our proposal also uses an image-space skeleton computation to estimate 3D depth, and a single view, thereby achieving the speed required for interactivity.

5.3 PROPOSED METHOD

We construct a 3D rotation in five steps (Figure 5.1). We start by loading the scene of interest — any arbitrary collection of 3D primitives, with no constraints on topology or sampling resolution — into the viewer (a). Next, the user can employ any mechanisms offered by the viewer, *e.g.* trackball rotation, zoom, or pan, to choose a *viewpoint of interest*, from which the scene shows a detail around which one would like to further rotate to explore the scene. In our example, such a viewpoint (b) shows the horse's rump, around which — for the sake of illustration — we want to rotate to examine the horse from different angles.

5.3.1 *Rotation axis computation*

From the above-mentioned initial viewpoint, we compute the rotation axis by performing three image-space operations, denoted as A, B, and C next.

A. Silhouette extraction: This is the first operation in Figure 5.1, step (d). We render the shape with Z buffering on and using the *GL_LESS* OpenGL depth-test. Let Ω_{near} be the resulting Z buffer. We next find the silhouette Ω of the shape as all pixels that have a value in Ω_{near} different from the default (the latter being 1 for standard OpenGL settings).

B. Skeleton computation: We next compute the silhouette skeleton S_Ω (Eqn. 5.1) by the method in [158] (Figure 5.1, step (d)). To eliminate spurious skeletal branches caused by small-scale noise along $\partial\Omega$, we regularize S_Ω by the salience-based metric in [153]. This regularization works as follows — see also the sketch in Figure 5.2c. For every point $\mathbf{x} \in S_\Omega$ of the full skeleton delivered by Eqn. 5.1, we first compute the importance ρ [158], *i.e.*, the shortest path along $\partial\Omega$ between the two feature points of $\mathbf{x}$ (see also Section 5.2). This path is marked red in Figure 5.2c. As shown in [45, 149, 158], and outlined in Section 5.2, ρ monotonically increases along skeletal branches from their endpoints to the skeleton center, and equals, for a skeleton point $\mathbf{x}$, the amount of boundary which is captured (described) by $\mathbf{x}$.

We next define the *salience* of skeletal point $\mathbf{x}$ as

$$\sigma(\mathbf{x}) = \frac{\rho(\mathbf{x})}{DT_\Omega(\mathbf{x})}, \tag{5.4}$$

that is, the importance ρ normalized by the skeletal point's distance to boundary. As shown in [153], σ is *overall high* on skeleton branches caused by important (salient) cusps of $\partial\Omega$ and *overall low* on skeleton branches caused by small-scale details (noise cusps) along $\partial\Omega$. Figure 5.2c shows this for a small cusp on the boundary of a 2D silhouette of a noisy 3D dino shape. As we advance in this image along the black

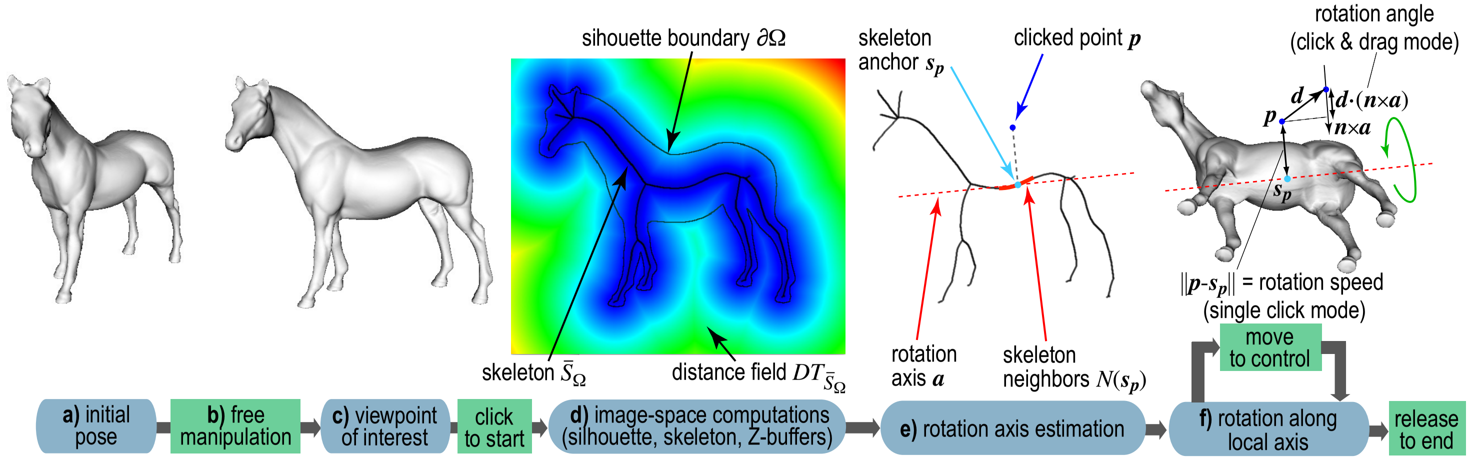

sihouette boundary $\partial\Omega$
skeleton anchor $\boldsymbol{s_p}$
clicked point $\boldsymbol{p}$
rotation angle (click & drag mode)
$\boldsymbol{p}$
$\boldsymbol{d}$
$\boldsymbol{d}\cdot(\boldsymbol{n}\times\boldsymbol{a})$
$\boldsymbol{n}\times\boldsymbol{a}$
$\boldsymbol{s_p}$
$\|\boldsymbol{p}$-$\boldsymbol{s_p}\|$ = rotation speed (single click mode)
skeleton $\bar{S}_\Omega$
distance field $DT_{\bar{S}_\Omega}$
rotation axis $\boldsymbol{a}$
skeleton neighbors $N(\boldsymbol{s_p})$
move to control
a) initial pose
b) free manipulation
c) viewpoint of interest
click to start
d) image-space computations (silhouette, skeleton, Z-buffers)
e) rotation axis estimation
f) rotation along local axis
release to end

Figure 5.1: Skeleton-based rotation pipeline with tool states (blue) and user actions (green).

skeleton branch into the shape's rump (going below the grey area in the picture), ρ stays constant, but the distance to boundary DT_Ω increases, causing σ to decrease. Hence, we can regularize S_Ω simply by removing all its pixels having a salience value lower than a fixed threshold σ_0. Following [153], we set $\sigma_0 = 1$. Figure 5.2 illustrates this regularization by showing the raw skeleton S_Ω and its regularized version

$$\overline{S}_\Omega = \{\mathbf{x} \in S_\Omega | \sigma(\mathbf{x}) \geq \sigma_0\} \tag{5.5}$$

for the noisy dino shape. Salience regularization (Figure 5.2b) removes all spurious branches created by boundary noise, but leaves the main skeleton branches, corresponding to the animal's limbs, rump, and tail, intact. Images (d–g) in the figure show the silhouette Ω, importance ρ, distance transform DT_Ω, and salience σ for a zoom-in area around the shape's head, for better insight. Looking carefully at image (e), we see that ρ has non-zero values also outside the main skeleton branch corresponding to the animal's neck, visible as light-blue pixels. While such details may look insignificant, they are crucial: Thresholding ρ by too low values — the alternative regularization to our proposal — keeps many spurious skeletal branches, see the red inset in Figure 5.2a. In contrast, σ is practically zero outside the neck branch (Figure 5.2g). So, thresholding σ by $\sigma_0 = 1$ yields a clean skeleton, see the red inset in Figure 5.2b. Salience regularization is simple and automatic to use, requiring no free parameters, and hence preferable to ρ regularization — which requires a careful setting of the threshold for ρ — or to any other skeleton regularization we are aware of. For further details on salience regularization, we refer to [153] and also its public implementation [154].

C. Rotation axis computation: This is step (e) in Figure 5.1. Let $\mathbf{p}$ be the pixel under the user-controlled pointer (blue in Figure 5.1e). We first find the closest skeleton point $\mathbf{s}_p = \operatorname{argmin}_{\mathbf{y} \in \overline{S}_\Omega} \|\mathbf{p} - \mathbf{y}\|$ by evaluating the feature transform (Eqn. 5.3) $FT_{\overline{S}_\Omega}(\mathbf{p})$ of the regularized skeleton $\overline{S}_\Omega$ at $\mathbf{p}$. Figure 5.1d shows the related distance transform $DT_{\overline{S}_\Omega}$. In our case, $\mathbf{s}_p$ is a point on the horse's rump skeleton (cyan in Figure 5.1e). Next, we find the neighbor points $N(\mathbf{s}_p)$ of $\mathbf{s}_p$ by searching depth-first from $\mathbf{s}_p$ along the pixel connectivity-graph of $\overline{S}_\Omega$ up to a fixed maximal distance set to 10% of the viewport size. $N(\mathbf{s}_p)$ contains skeletal points along a single branch in $\overline{S}_\Omega$, or a few connected branches, if $\mathbf{s}_p$ is close to a skeleton junction. In our case, $N(\mathbf{s}_p)$ contains a fragment of the horse's rump skeleton (red in Figure 5.1e). For each $\mathbf{q} \in N(\mathbf{s}_p)$, we set the depth $\mathbf{q}_z$ as the average of $\Omega_{far}(\mathbf{q})$ and $\Omega_{near}(\mathbf{q})$. Here, Ω_{near} is the Z buffer of the scene rendered as described in step A above; and Ω_{far} is the Z buffer of the scene rendered as before, but with front-face culling on, *i.e.*, the depth of the *nearest* backfacing polygons to the view plane.

Figure 5.3 shows how this works. The user clicks above the horse's rump and drags the pointer upwards (a). Image (b) shows the resulting

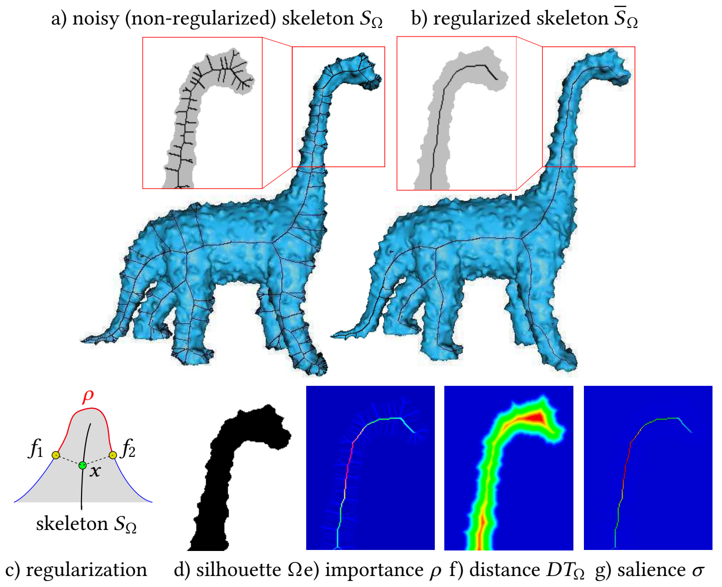


Figure 5.2: Raw skeleton S_Ω with (a) noise-induced branches and (b) salience-based regularized skeleton $\overline{S}_\Omega$. c) Principle of salience regularization. (d–g) Details of silhouette, importance, distance transform, and salience values for the noisy dino's head.

rotation. As visible in the inset in (a), the rotation axis (red) is centered inside the rump, as its depth $\mathbf{q}_z$ is the average of the near and far rump faces. To better understand this, the image left to Figure 5.3a shows the horse rendered transparently, seen from above. The depth values in Ω_{near} and Ω_{far} are shown in green, respectively blue. The skeleton depth values (red) are the average of these. Note that, when the rotation ends, the new silhouette skeleton does *not* match the rotation axis — see inset in (b). This is normal and expected. If the user wants to start a new rotation from (b), then the 2D skeleton from this image will be used to compute a new, matching, rotation axis.

Next, we consider a case of overlapping shape parts (Figure 5.3c). The user clicks left to the horse's left front leg, which overlaps the right-front one, and drags the pointer to the right. Image (d) shows the resulting rotation. The rotation axis (red) is centered inside the left front leg. In this case, $\Omega_{far}(\mathbf{q})$ contains the Z values of the backfacing part of the left front leg, so $(\Omega_{near}(\mathbf{q}) + \Omega_{far}(\mathbf{q}))/2$ yields a value roughly halfway this leg along the Z axis. The image left to Figure 5.3c clarifies this by showing the horse from above and the respective depth values in Ω_{near} (green) and Ω_{far} (blue).

Separately, we handle non-watertight surfaces as follows: If $\Omega_{far}(\mathbf{q})$ contains the default Z value (one), this means there's no backfacing sur-

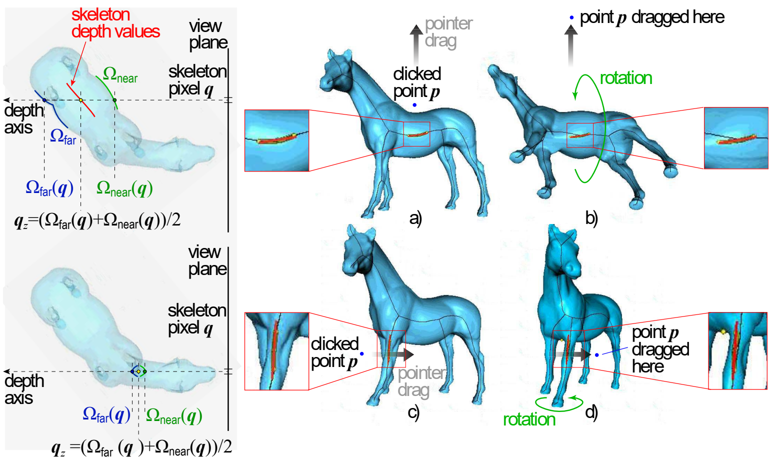


Figure 5.3: Depth estimation of rotation axis for (a, b) non-overlapping part and (c, d) overlapping parts. In both cases, the rotation axis (red) is nicely centered in the shape. See Section 5.3.1.

face under a given pixel $\mathbf{q}$, so the scene is not watertight at $\mathbf{q}$. We then set $\mathbf{q}_z$ to $\Omega_{near}(\mathbf{q})$.

We now have a set $N_{3D} = \{(\mathbf{q} \in N(\mathbf{s}_p), \mathbf{q}_z)\}$ of 3D points that approximate the 3D curve skeleton of our shape close to the pointer location $\mathbf{p}$. We set the 3D rotation axis $\mathbf{a}$ to the line passing through the average point of N_{3D} and oriented along the largest eigenvector of N_{3D}'s covariance matrix (Figure 5.1e, red dotted line).

5.3.2 *Controlling the rotation*

We propose three interactive mechanisms to control the rotation (Figure 5.1, step (f)):

- **Indication:** As the user moves the pointer $\mathbf{p}$, we continuously update the display of $\mathbf{a}$. This shows along which axis the scene *would* rotate if the user initiated rotation from $\mathbf{p}$. If $\mathbf{a}$ is found suitable, one can start rotating by a click following one of the two modes listed next; else one can move the pointer $\mathbf{p}$ to find a more suitable axis;

- **Single click:** In this mode, we compute a rotation speed σ equal to the distance $\|\mathbf{p} - \mathbf{s}_p\|$ and a rotation direction δ (clockwise or anticlockwise) given by the sign of the cross-product $(\mathbf{s}_p - \mathbf{p}) \times \mathbf{n}$, where $\mathbf{n}$ is the viewplane normal. We next continuously rotate (spin) the shape around a with the speed σ in direction δ;

- **Click and drag:** Let $\mathbf{d}$ be the drag vector created by the user as she moves the pointer $\mathbf{p}$ from the current to the next place in the viewport with the control, *e.g.* mouse button, pressed. We rotate the scene around $\mathbf{a}$ with an angle equal to $\mathbf{d} \cdot (\mathbf{n} \times \mathbf{a})$ (Figure 5.1e).

We stop rotation when the user releases the control (mouse button). In single-click mode, clicking closer to the shape rotates slowly, allowing one to examine the shape in detail. Clicking farther rotates quicker to *e.g.* explore the shape from the opposite side. The rotation direction is given by the *side* of the skeleton where we click: To change from clockwise to counterclockwise rotation in Figure 5.1, we only need to click below, rather than above, the horse's rump. In click-and-drag mode, the rotation speed and direction is given by the drag vector $\mathbf{d}$: Values $\mathbf{d}$ orthogonal to the rotation axis $\mathbf{a}$ create corresponding rotations clockwise or anticlockwise around $\mathbf{a}$; values $\mathbf{d}$ along $\mathbf{a}$ yield no rotation. This matches the intuition that, to rotate along an axis, we need to move the pointer *across* that axis.

The skeleton-based construction of the rotation axis is key to the effectiveness of our approach: If the shape exhibits some elongated structure in the current view (*e.g.* rump or legs in Figure 5.1c), this structure yields a skeleton branch. Clicking closer to this structure than to other structures in the same view — *e.g.*, clicking closer to the rump than to the horse's legs or neck — selects the respective skeleton branch to rotate around. This way, the 3D rotation uses the 'natural' structure of the viewed shape. We argue that this makes sense in an exploratory scenario, since, during rotation, the shape parts we rotate around stay *fixed* in the view as if one 'turns around' them. The entire method requires a *single click* and, optionally, a pointer drag motion to execute. This makes our method simpler than other 3D rotation methods for freely specifiable 3D axes, and also applicable to contexts where no second button or modifier keys are available, *e.g.*, touch screens.

5.3.3 *Improvements of basic method*

We next present three improvements of the local-axis rotation mechanism described above.

Zoom level: A first issue regards computing the scene's 2D silhouette Ω (Section 5.3.1A). For this to work correctly, the entire scene must be visible in the current viewport. If this is not the case, the silhouette boundary $\partial\Omega$ will contain parts of the viewport borders. Figure 5.4a shows this for a zoomed-in view of the horse model, with the above-mentioned border parts marked purple. This leads to branches in the skeleton S_Ω that do not provide meaningful rotation axes. We prevent this to occur by requiring that the entire scene is visible in the viewport before initiating the rotation-axis computation. If this is not the case, we do not allow the skeleton-based rotation to proceed but map the

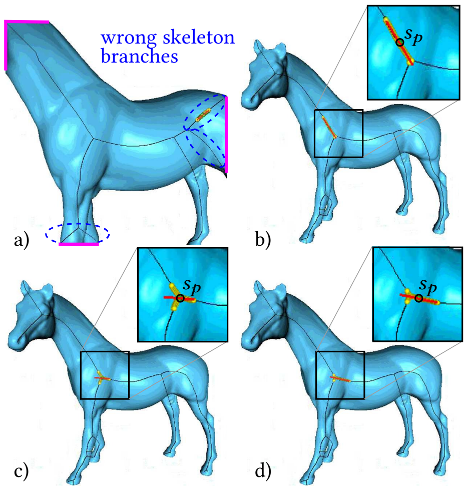


Figure 5.4: Two problems of estimating rotation axes from skeletons. (a) Zoomed-in scene. Anchor points close to (c), respectively farther from (b, d) a skeleton junction. See Section 5.3.3.

user's interaction to the standard trackball-based rotation.

Skeleton junctions: If the user selects $\mathbf{p}$ so that the skeleton anchor $\mathbf{s}_p$ is too close to a skeleton junction, then the neighbor-set $N(\mathbf{s}_p)$ will contain points belonging to more than two branches. Estimating a line from such a point set (Section 5.3.1C) is unreliable, leading to possibly meaningless rotation axes. Figures 5.4b–d illustrates the problem. The corresponding skeleton points $N(\mathbf{s}_p)$ used to estimate the axis are shown in yellow, and the resulting axes in red. When $\mathbf{s}_p$ is far from the junction (Figures 5.4b, d), $N(\mathbf{s}_p)$ contains mainly points from a *single* skeleton branch, so the estimated rotation axes are reliable. When $\mathbf{s}_p$ is very close to a junction (Figure 5.4c), $N(\mathbf{s}_p)$ contains points from all three meeting skeletal branches, so, as the user moves the pointer $\mathbf{p}$, the estimated axis 'flips' abruptly and can even assume orientations that do not match any skeleton branch.

We measure the *reliability* of the axis $\mathbf{a}$ by the anisotropy ratio $\gamma = \lambda_1/\lambda_3$ of the largest to smallest eigenvalue of N_{3D}'s covariance matrix. Other anisotropy metrics can be used equally well [38]. High γ values indicate elongated structures N_{3D}, from which we can reliably compute rotation axes. Low values, empirically detected as $\gamma < 5$, indicate problems to find a reliable rotation axis. When this occurs, we prevent executing the axis-based rotation.

Selection distance: A third issue concerns the position of the point $\mathbf{p}$ that starts the rotation: If one clicks too far from the silhouette Ω, the rotation axis $\mathbf{a}$ may not match what one expects. To address this,

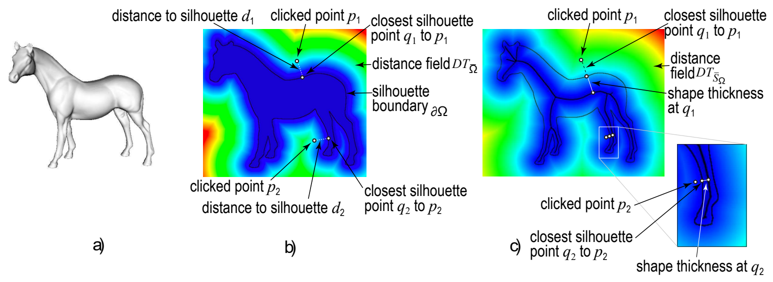

distance to silhouette d_1
clicked point p_1
closest silhouette point q_1 to p_1
distance field DT_{Ω}
silhouette boundary $\partial\Omega$
clicked point p_2
distance to silhouette d_2
closest silhouette point q_2 to p_2
a)
b)
clicked point p_1
closest silhouette point q_1 to p_1
distance field $DT_{\overline{S_\Omega}}$
shape thickness at q_1
clicked point p_2
c)
closest silhouette point q_2 to p_2
shape thickness at q_2

Figure 5.5: Improvements of axis-based rotation method. (a) A view of the shape to be rotated. (b) Fixed maximum-distance setting for two clicked points $\mathbf{p}_1$ and $\mathbf{p}_2$. (c) Thickness-based maximum-distance setting for two clicked points $\mathbf{p}_1$ and $\mathbf{p}_2$.

we forbid the rotation when the distance d from $\mathbf{p}$ to Ω exceeds a given upper limit d_{max}. That is, if the user clicks too far from any silhouette in the viewport, the rotation mechanism does not start. This signals to the user that, to initiate the rotation, she needs to click closer to a silhouette. We compute d as $DT_{\overline{\Omega}}(\mathbf{p})$, where $\overline{\Omega}$ is the viewpoint area outside Ω, *i.e.*, all viewport pixels where Ω_{near} equals the default Z buffer value (see Section 5.3.1A).

We studied two methods for estimating d_{max} (see Figure 5.5). First, we set d_{max} to a fixed value, in practice 10% of the viewport size. Using a constant d_{max} is however not optimal: We found that, when we want to rotate around *thick* shape parts, *e.g.* the horse's rump in Figure 5.5b, it is intuitive to select $\mathbf{p}$ even quite far away from the silhouette. This is the case of point $\mathbf{p}_1$ in Figure 5.5b. In contrast, when we want to rotate around *thin* parts, such as the horse's legs, it is not intuitive to initiate the rotation by clicking too far away from these parts. This is the situation of point $\mathbf{p}_2$ in Figure 5.5b. Hence, d_{max} depends on the scale of the shape part we want to rotate around; selecting large parts can be done by clicking farther away from them than selecting small parts.

We model this by setting d_{max} to the local *shape thickness* (Figure 5.5c). We estimate thickness as follows: We find the closest point on the silhouette boundary $\partial\Omega$ to the clicked point $\mathbf{p}$ as $\mathbf{q} = FT_{\overline{\Omega}}(\mathbf{p})$. The shape thickness at $\mathbf{q}$ is the distance to the skeleton, *i.e.*, $DT_{\overline{S}_\Omega}(\mathbf{q})$. This is the 2D equivalent of the more general 3D-shape-thickness estimation proposed in [157]. In Figure 5.5c, the point $\mathbf{p}_1$ is the farthest clickable point around $\mathbf{q}_1$ to the silhouette that allows starting a rotation around the rump. If we click further from the silhouette than the distance d_{max} from $\mathbf{p}_1$ to $\mathbf{q}_1$, no rotation is done. For the leg part, the farthest clickable point around $\mathbf{q}_2$ must, however, be much closer to the silhouette (Figure 5.5c), since here the local shape thickness (distance d_{max} from $\mathbf{p}_2$ to $\mathbf{q}_2$) is smaller.

5.4 FORMATIVE EVALUATION

To evaluate our method, we conducted first a *formative* evaluation. In this evaluation, only the authors of this work and a few other researchers, familiar with 3D interactive data visualization, were involved. This evaluation aimed at (a) verifying how the skeleton-based rotation practically works on a number of different 3D shapes; and (b) eliciting preliminary observations from the subjects to construct next a more in-depth evaluation study. We next present the results of this first evaluation phase. Section 5.5 details the second-phase evaluation designed using these findings.

Figure 5.6 shows our 3D skeleton-based rotation applied to two 3D mesh models. For extra insights, we recommend watching the demonstration videos [159]. First, we consider a 3D mesh model of a human hand which is not watertight (open at the wrist). We start from a poor viewpoint from which we cannot easily examine the shape (a). We click close to the thumb (b) and drag to rotate around it (b–e), yielding a better viewpoint (e). Next, we want to rotate around the shape to see the other face, but keeping the shape roughly in place. Using a trackball or world-coordinate axis rotation cannot easily achieve this. We click on a point close to the shape-part we want to keep *fixed* during rotation (f), near the wrist, and start rotation. Images (g–j) show the resulting rotation.

Figure 5.6(k–ad) show a more complex ship shape. This mesh contains multiple self-intersecting and/or disconnected parts, some very thin (sails, mast, ropes) [81]. Computing a 3D skeleton for this shape is extremely hard or even impossible, as Eqn. 5.1 requires a watertight, non-self-intersecting, connected shape boundary $\partial\Omega$. Our method does not suffer from this, since we compute the skeleton of the *2D silhouette* of the shape. We start again from a poor viewing angle (k). Next, we click close to the back mast to rotate around it, showing the ship from various angles (l–o). Images (p–u) show a different rotation, this time around an axis found by clicking close to the front sail, which allows us to see the ship from the front. Note how the 2D skeleton has changed after this rotation — compare images (p) with (v). This allows us to select a new rotation axis by clicking on the main sail, to see the ship's stern from below (w–z). Finally, we click on the ship's rump (aa) to rotate the ship and make it vertical (ab-ad). The entire process of three rotations took around 20 seconds.

Figure 5.7 shows a different dataset type: a 3D point cloud that models a collision simulation between the Milky Way and the nearby Andromeda Galaxy [35, 39]. Its 160K points describe the positions of the stars and dark matter in the simulation. Image (a) uses volume rendering to show the complex structure of the cloud, for illustration purposes. We do not use this rendering, but rather render the cloud in our pipeline using 3D spherical splats (b). Image (c) shows the cloud, rendered with

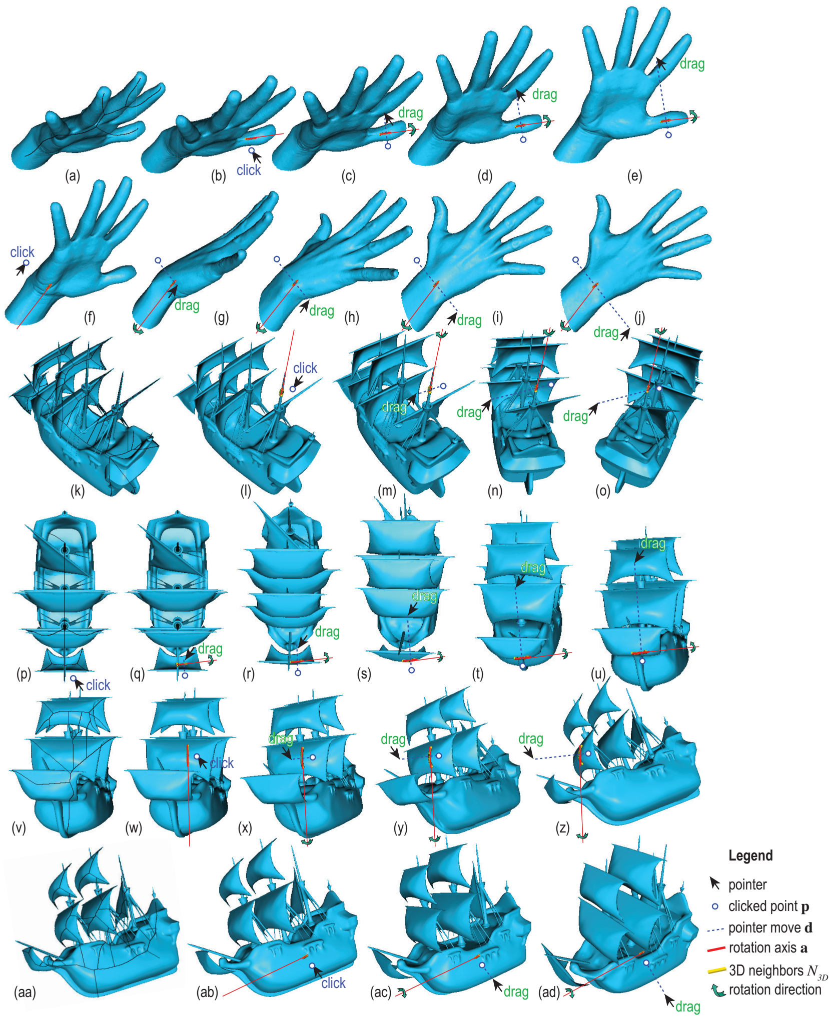


Figure 5.6: Examples of two rotations (a–e), (f–j) for the hand shape and four rotations (k–o), (p–u), (v–z), (aa–ad) for the ship model.

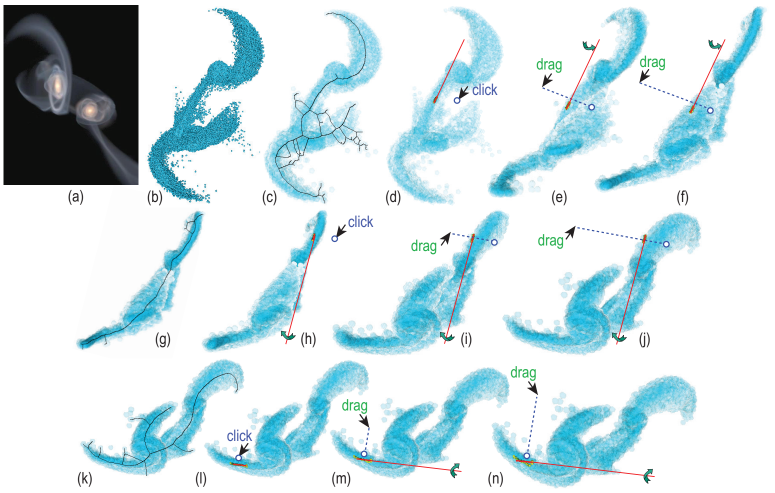


Figure 5.7: Exploration of astronomical point cloud dataset. (a) Volume-rendered overview [39]. Rotations around three 3D axes (b–f), (g–j), (k–n).

half-transparent splats, so that opacity reflects local point density. Since we render a 3D sphere around each point, this results in a front and back buffer Ω_{near} and Ω_{far}, just as when rendering a 3D polygonal model. From these, we can compute the 2D skeleton of the cloud's silhouette, as shown in the figure. Images (d–f) show a rotation around the central tubular structure of the cloud, which reveals that the could is relatively flat when seen from the last viewpoint (f). Image (g) shows the new 2D skeleton corresponding to the viewpoint after this rotation. We next click close to the upper high-density structure (f) and rotate around it. Images (h–j) reveal a spiral-like structure present in the lower part of the cloud, which was not visible earlier. To explore this structure better, we next click on its local symmetry axis (l) and rotate around it. Images (l–n) reveal now better this structure. As for the earlier examples, executing these three rotations took roughly 15 seconds. Scientists involved with studied this dataset for roughly a decade appreciated positively the ease of use of the skeleton-based rotation as compared to the standard trackball and multi-touch gestures.

We gathered several insights during our formative evaluation by free-form discussions with the participants, that is, without following a strict evaluation protocol based on tasks and quantitative responses. We summarize below the most important ones:

- Skeleton rotation works quite well for relatively *small* changes of viewpoint; more involved changes require decomposing the desired rotation into a set of small-size changes and careful selection of their respective rotation axes;

- Skeleton rotation seems to be most effective for *precise* rotations, in contrast to typical trackball usage, which works well for larger, but less precise, viewpoint changes;
- All participants stated that they *believe* that skeletons allow them to perform certain types of rotation easier than if they had used the trackball for the same tasks. However, they all mentioned that they do not feel that skeletons can *replace* a trackball. Rather, they believe that a free combination of both to be most effective. Since they could only use the skeleton rotation (in our evaluation), they do not know whether (or when) this tool works better than a trackball;
- All participants agreed that *measuring* the added-value of skeleton rotation is very important for its adoption.

5.5 DETAILED EVALUATION — USER STUDY

The formative evaluation (Section 5.4) outlined that there is perceived added-value in the skeleton rotation tool, but this value needs to be actually measured before users would consider adopting the tool — either standalone or in combination with trackball. To deepen our understanding of how skeleton-based rotation works, and to answer the above questions, we designed and conducted a more extensive user evaluation. We next describe the design, execution, and analysis of the results of this evaluation.

5.5.1 *Evaluation design*

Tool: To assess how the skeleton rotation modality compares with the trackball modality, we designed an experiment supported by an interactive *tool*. The tool has two windows: The *target* window shows a 3D shape viewed from a viewpoint (pose) that is preselected by the evaluation designer. No interaction is allowed in this window. The *source* window shows the same shape, which can be freely manipulated by the user via the skeleton (S), the trackball (T), or both tools (B), activated via the left, respectively right, mouse buttons. Both windows have the same resolution (512^2 pixels), use the same lighting and rendering parameters, and have a fixed position on the computer screen, to simplify usage during the experiment that invokes multiple runs of the tool. Besides rotation, the tool also allows panning and zooming. We also added an option to automatically zoom out to show the full extent of a shape. This eliminates the issues described in Section 5.3.3, *i.e.*, manipulations that move part of the shape outside the window. When in S mode, the tool shows the silhouette skeleton (black), nearest skeleton points (yellow), and estimated rotation axis (red) as explained earlier in Section 5.3.1 and

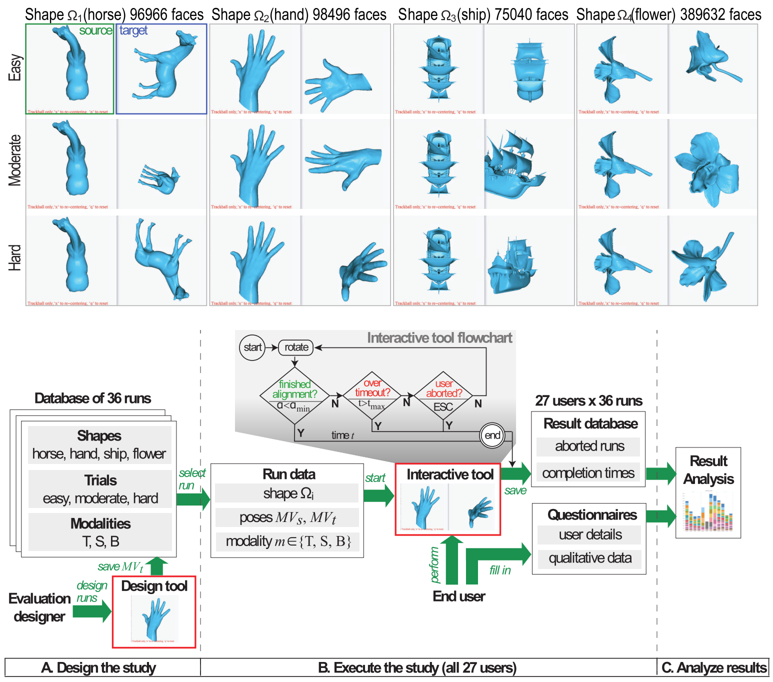


Figure 5.8: Top: User evaluation showing the 12 trials for one modality (Section 5.5.1). Each trial consists of a source window in which the user interacts to align the shape to match the target window. Bottom: Execution of end-to-end user evaluation. The use of our interactive tool in both design and evaluation modes is shown in red (Section 5.5.2).

shown *e.g.* in Figure 5.3. The user can interactively tune the simplification level of the skeleton via the '+' and '-' keys, to show more or fewer branches from which to select a suitable rotation axis (*cf* Figure 5.2).

Figure 5.8 (central inset) shows a flowchart of the tool's operation, which we detail next. Participants are asked to use the tool with each modality in turn (S, T, B) to align the source with the target. The tool continuously computes, after each motion of the mouse pointer, the value

$$\alpha = \arccos\left(\frac{Tr(MV_s \cdot MV_t^T) - 1}{2}\right), \tag{5.6}$$

where MV_s and MV_t are the 3×3 OpenGL rotation matrices (ignoring, thus, translation and scaling) corresponding to the pose of the shape in the source and the target, respectively; Tr is the matrix trace operator; and T denotes matrix transposition. The value $\alpha \in [0, 180]$ is the smallest rotation (around any axis) needed to obtain the target pose from the source pose [10]. Note that Eqn. 5.6 is sensitive to mirroring, which is desired, since rotations *cannot* cause mirroring. Alignment is considered *completed* when $\alpha < \alpha_{min}$; in practice, we set $\alpha_{min} = 15$ degrees. Also, note that Eqn. 5.6 only checks for *rotation*, and not scaling or panning, differences. This makes sense, since the tested modalities S, T, B control rotation only; scaling (zooming) and panning, though allowed to help users to inspect shapes, are not part of our evaluation, and perform identically with S, T, and B. During manipulation, the tool continuously displays the current value of α. This shows users how far away they are from the target rotation MV_t, thus, from completing a task. This feedback is useful when visual comparison of the source and target poses is hard to do.

Shapes: We use the alignment tool to evaluate the performance of the S, T, and B modalities on $N = 4$ *shapes* Ω_i, $1 \leq i \leq N$, shown in Figure 5.8 (top). Shapes were selected so as to be familiar, have a structure that exposes potential local-rotation axes, and have geometric complexity ranging from simple (horse, hand) to complex (ship). The flower shape is of lower complexity than the ship; however, its manifold structure makes it particularly hard to understand and manipulate, since it looks quite similar from many viewpoints. All shapes use identical material properties and no opacity or textures, to favor uniform evaluation. We excluded the more complicated point-cloud shape (Figure 5.7) used during formative evaluation (Section 5.4) since no more than five of our recruited subjects had the technical background needed to understand what such data means in the first place.

Task difficulties: For each shape, we use three target poses MV_t to capture three levels of difficulty of the alignment task:

- *Easy:* Alignment can be done by typically one or two manipulations, such as a rotation around one of the x or y window axes, or a rotation around a clearly-visible symmetry axis of the shape). For example, the blue-framed target in Figure 5.8 can be obtained from the green-framed pose (left to it) by a single counterclockwise rotation of the horse with 90 degrees around the y axis or, alternatively, the rump's skeleton;

- *Hard:* Alignment requires multiple rotations around many different rotation axes; it is not easy to see, from the source and target, which would be these axes;

- *Intermediate:* Alignment difficulty is gauged as between the above two extremes.

We call next the combination of shape Ω_i and start-and-end pose (MV_s, MV_t) a *trial*. Using multiple-difficulty trials aims to model tasks of different complexity. Trial difficulty was assessed by one of the authors (who also designed the actual poses MV_t) and agreed upon by the others by independent testing. We verified that all three modalities could accomplish all trials within a time t lower than a predefined timeout $t_{max} = 120$ seconds.

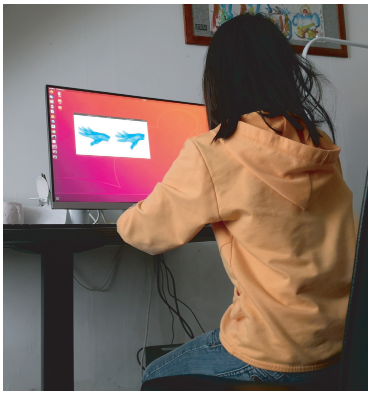
Figure 5.8 (top) shows the source (left window in each window-pair) and target (right window in the same pair) windows for the 12 trials spanning the 4 shapes using the T modality. Source windows show the currently-enabled modality in red text, to remind users how they can interact. We see, for instance, that the *easy* trial would require, in S mode, a simple 90-degree rotation around the y axis (in T mode) for the ship model, or around the main skeleton branch passing through the horse's rump for the horse model. In contrast, the *hard* task requires several incremental rotations for all modalities. The 12 trials use *identical* initial poses MV_s and target poses MV_t. That is, the user is asked to perform, for each shape, the same alignment $MV_s \longrightarrow MV_t$ using all three modalities, thus ensuring that only the target pose (endpoint of manipulation) and, of course, the used modality, affect the measured execution time.

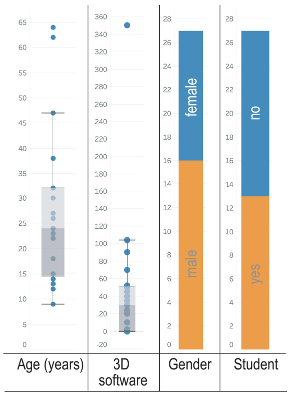
In total, we thus execute 12 trials $\times 3$ modalities = 36 *runs*. For each run, we record the time needed for the user to complete it. If the user fails to perform the alignment within the allowed timeout, the run is considered *failed* and the user moves automatically to the next run. Users can at any time (a) abort a run by pressing 'ESC' to move to the next run; this helped impatient users who did not grasp how to perform a given alignment task and did not want to wait until the timeout; (b) abort the entire evaluation, if something goes entirely wrong; and (c) reset the viewpoint to the initial one (MV_s), to 'undo' all manipulations performed so far if these are deemed unproductive.

Pose design: The different target poses MV_t were designed in advance by us by using the S and T tools — intermixed — to freely change the shape's pose until obtaining the desired target poses, and stored, as explained, as 3×3 OpenGL rotation matrices.

5.5.2 *Evaluation execution*

Subjects: Twenty-seven persons took part in the evaluation. they self-report ages of 9 to 64 years (median: 24, average: 26.9); and gender being male (16) and female (11), see Figure 5.9b. To gauge their experience with 3D manipulation, we asked them to report how many times a year they used 3D games and/or 3D design software. Both categories are reported in Figure 5.9b as '3D software usage'. Results show a median of 30 times, with the minimum being zero (never) and the maximum being basically every day. From these data, we conclude that most participants should have a good practical mastery of 3D manipulation. From the 27 participants, 13 were students in fields as diverse as Computer Science, social science, medicine, economy and society, and mathematics; the other 14 were primary or secondary school pupils (6) or employed in various liberal professions (8). All participants


a)



b)

Figure 5.9: a) Setup employed during the user evaluation. b) Self-reported characteristics of the experiment participants. See Section 5.5.2.

reported no color blindness issues. All except one were right-handed. They all reside in the Netherlands or Belgium. Communication during the training and experiment was done in the native language of each participant by a (near-)native speaker. For participants with limited English proficiency, all English material (tutorial, questionnaires) was transcribed by the trainer.

Workflow: Participants followed the evaluation workflow showed in Figure 5.8 (bottom). First, we created the information needed to execute the 36 runs (Figure 5.8 (bottom, A)), as explained in Section 5.5.1. Next, participants were given access, prior to the actual experiment, to a web tutorial which describes both S and T tools in general, and also allows users to practice with these tools by running the actual application to execute some simple alignment tasks. No statistics were collected from this intake phase. After intake, users asked if they felt interested in, and able to follow, the tutorial. This intake acted as a simple filter to separate users with interest in the evaluation (and potential ability to do it) from the rest, so as to minimize subsequent effort. Seven persons dropped from the process due to lack of general computer skills (1 user), one too young (6 years), one too old (82 years), and four due to technical problems related to remote-deployment of the tool. These persons are not included in any of the statistics further on nor in Figure 5.9b.

Next, a trainer (role filled by different co-authors) took part in a *controlled session* where they explained to either individual participants or, when social distancing rules due to the Corona pandemic were not applicable, to groups of participants how the tool works and also illustrated it live. The aim of this phase was to refine the knowledge disseminated by the web tutorial and confirm that participants understood well the evaluation process and tooling. Participants and trainers used Linux-based

PCs (16 to 32GB RAM) with recent NVidia cards, wide screens, and a classical two-button mouse. To maximize focus on the experiment, no application was run on screen during the evaluation besides the two-window tool described in Section 5.5.1. Training took both the in-person form (with trainer and user(s) physically together), and via TeamViewer or Skype screen sharing when social distancing rules mandated separation. Training took between 20 and 40 minutes per user and was done until users told that they were confident to use the tool to manipulate both a simple model and a complex one via all three modalities (S, T, B). During this phase, we also verified that the tool runs at real-time framerates on the users' computers so as to eliminate confusing effects due to interaction lag; and that the users did not experience any difficulty in using the keyboard shortcuts outlined in Section 5.5.1.

After training, and confirmation by participants that they understand the evaluation tool and tasks to be done, participants started executing the 36 runs (Figure 5.8 (bottom, B). They could pause between runs as desired but not change the orders of the runs. Figure 5.9a shows the setup used during the evaluation by one of the actual participants; notice the two-window interaction tool on the screen. In the end, the results of all 36 runs — that is, either completion time or run failure (either by time-out or user abortion) — were saved in a database with no mention of the user identity. Next, users completed a questionnaire covering both personal and self-assessment data and answers to questions concerning the usability of the tool. Both types of results (timing data and questionnaires) were further analyzed (Figure 5.8 (bottom, C)), as described in Section 5.5.3.

5.5.3 *Analysis of results*

We next present both a quantitative analysis of the timing results and an analysis of the qualitative data collected via questionnaires.

5.5.3.1 *Analysis of timing results*

A most relevant question is: How did performance (measured in completion time and/or the number of aborted runs) depend on the interaction *modality* and *shape*? Figure 5.10a shows the average completion time, for the *successful* runs, aggregated (overall users) per modality and next per shape. User identities are categorically color-coded for ease of reading the figure. Median and interquartile ranges for each modality are shown by black lines, respectively gray bands. We see that the S modality is significantly slower than T and S. However, the B modality is faster than T, both as median and interquartile range, and also for each specific shape. This is an interesting observation, as it suggests that, in B mode, users did gain time by using S only for some specific manipulations for which T was hard to use. A likely explanation for this is that

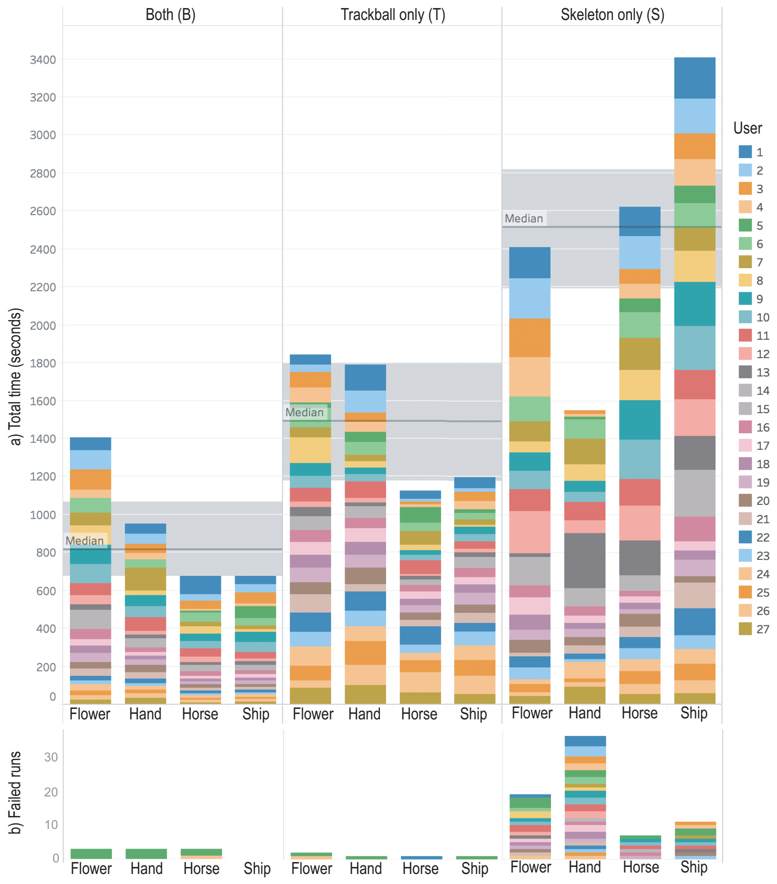


Figure 5.10: Completion time (a) and number of failed runs (b) per modality and shape, all users. See Section 5.5.2.

the B modality was always used last during the trials. Hence, when in B mode, users could discover the situations when S outperformed T, and switch to S in those cases to gain time. We analyze this hypothesis further below.

Figure 5.10b shows the number of *failed* runs per modality, shape, and user. These are the largest for the S modality. This tells again that S cannot be used *alone* as a general-purpose manipulation tool. If we combine this insight with the total times per shape (Figure 5.10a), we see that the perceived difficulty of the task varies significantly over both shapes and modalities: T and S behave quite similarly, with *horse* and *ship* being easier to handle and *flower* being the hardest. In contrast, *hand* seems to be the hardest to handle by the S modality, as it has most aborted runs. Upon a closer analysis, we found that the pose used by the 'hard' trial for *hand* (see the respective image in Figure 5.8 (top)) is quite easy to achieve with T (and thus also B), but quite difficult to obtain using S, since it implies, at several points, performing a rotation around an axis orthogonal to the hand's palm, for which no skeleton line exists in the silhouette. The second-hardest shape for S is *ship*. Analyzing the users' detailed feedback showed us that *ship*'s complex geometry produces a wealth of potential rotation axes with quite different angles, which makes the users' choice (of the optimal rotation axis) hard. This happens far less for the other simpler-structure shapes. Separately, Figure 5.10b shows that the number of aborted runs in B mode is far lower than that in S mode, being practically the same as for T mode. This, and the fact that B mode is fastest, reinforces our hypothesis that users employ the S tool in B mode only for very *specific* manipulations and revert to T for all other operations. Hence, S works best as a complement, not a replacement, of T.

Figure 5.11a introduces additional information in the analysis by showing how the average times vary over the three task *difficulty* levels (easy, moderate, hard, see Section 5.5.1). For all shapes and modalities, the task labeled easy by us is, indeed, completed the fastest. The other two difficulty levels are, however, not significantly different in execution times. We also see that effort (time) is distributed relatively uniformly overall difficulty levels for all shapes and modalities. This indicates that there is no 'outlier' task or shape in our experiment that would strongly bias our evaluation's insights.

Finally, we examine the data from a *user-centric* perspective. Figure 5.11b shows the total time per user, split per modality, with the fastest users at the right and the slowest at the left. We see a quite large spread in performance, the fastest user being roughly 2.5 times faster than the slowest one. We see that the T modality does not explain the big speed difference — the red bars' sizes do not correlate with the total time. In contrast, the blue bars show an increase when scanning the chart right-to-left, at the 8^{th} leftmost bar — meaning that the 8 slowest users needed clearly more time to use the B modality as opposed to the remaining 19 users. Scanning the graph right-to-left along its orange bars shows a *strongly* increasing bar-size. That is, the main factor differentiating slow from fast users is their skill in using the *S tool*. We hypothesized that this skill has to do with the users' familiarity with 3D manipulation tools. To examine this, we show a scatterplot of the average time per user (all trials, all shapes) *vs* the user's self-reported number of days per year that one uses 3D computer games or 3D creation software (Figure 5.11c). All points in the plot reside in the lower range of the y axis, *i.e.*, all users report under 100 days/year of 3D tool usage, except user 12 who indicated 3D gaming daily. The computed correlation line shown in the figure ($R^2 = 0.0022$, $p = 0.813$) indicates a negligible inverse correlation of average time with 3D software usage. Hence, our hypothesis is not confirmed. The question what determines the variability in users' average completion times is still open.

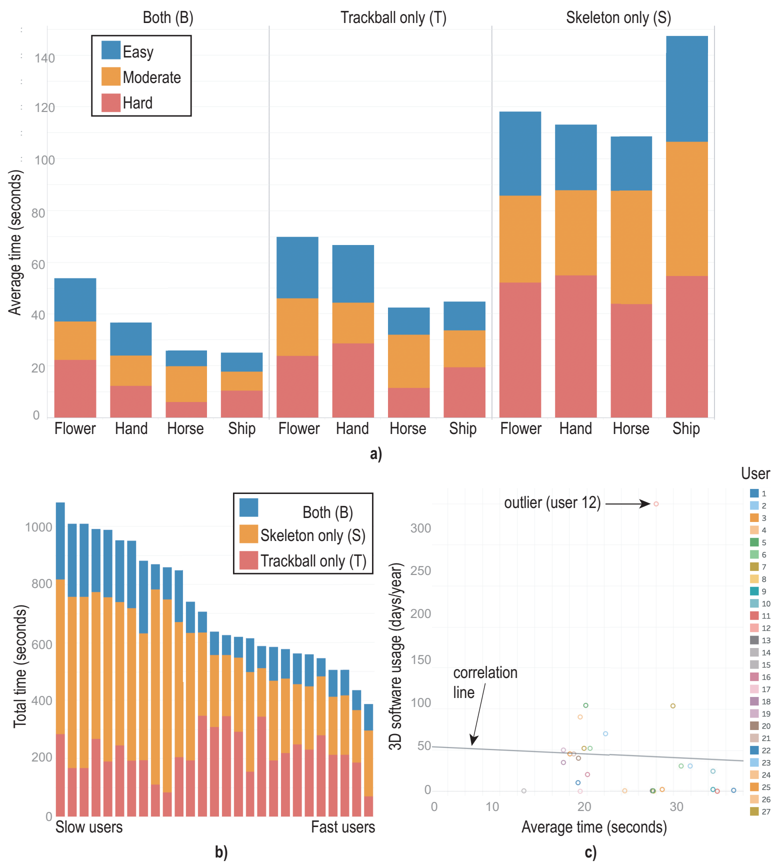


Figure 5.11: a) Average completion time per difficulty levels, modality, and shape. b) Total time for all users, from slowest to fastest, split per modality. c) Correlation of average time (all runs) with users' frequency of 3D software usage. See Section 5.5.2.

5.5.3.2 *Analysis of questionnaire results*

As mentioned at the beginning of Section 5.5.2, users completed a questionnaire following the experiment. They were asked to answer 13 questions concerning their experience with each of the three modalities (T, S, B) using a 7-point Likert scale S (1=strongly disagree, 2=disagree, 3=disagree somewhat, 4=no opinion, 5=agree somewhat, 6=agree, 7=strongly agree). An extra question (Q14) asked which of the three modalities users prefer *overall*. Figure 5.12 (bottom) shows these 14 questions. Here, 'tool' refers to the modality being evaluated. Following earlier studies that highlight that user *satisfaction* is not the same as user *efficiency* or *effectiveness* when using interactive tools [49, 109], we included questions that aim to cover all these aspects. Users could also input free text to comment on their perceived advantages and limitations of all three modalities or any other remarks.

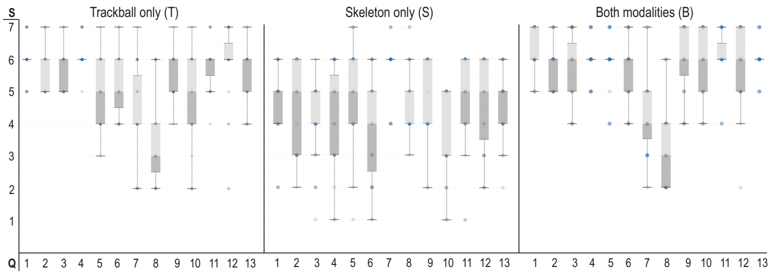
Figure 5.12(top) shows the aggregated answers for Q1–Q13 for each of the three modalities with box-and-whisker plots (box shows the interquartile range; whiskers show data within 1.5 times this range). We see that the S modality ranks, overall, worse than the T modality, except for accuracy (Q5). Accuracy (Q5) can be explained by the fact that users need to control a *single* degree of freedom with S — the rotation angle — but two degrees of freedom with T. In other words, once a suitable rotation axis is chosen, S allows one to precisely specify the rotation angle around this axis. We also see that S helps completing the task less often than T (Q10), which matches the failure rates shown in Figure 5.10b. However, the B modality ranks in nearly all aspects better than both T and S. This supports our hypothesis that S best *complements*, rather than replaces, T. An interesting finding are the scores for Q8 and Q6, which show that B was perceived as less tiring to use, and needing fewer steps to accomplish the task respectively, than both T and S. This matches the results in Figure 5.10a that show that B is faster than both T and S — thus, arguably less tiring to use. For Q14, 22 of the 27 users stated that they prefer B overall, while the remaining 5 users preferred T, with none mentioning S as the highest-preference tool. As for the previous findings, this strongly supports our hypothesis that the S and T modalities work best when combined.

From the free text that captures the user's comments on the perceived advantages and limitations of all three modalities, we could distil several salient points. For space constraints, we list only a few below:

- **Trackball (T):** Several users praised T for being "easy to use". However, users also complained about trackball being imprecise for performing fine adjustments;

- **Skeleton (S):** This modality was mentioned as better than the other two by only a few users, and specifically for the horse, hand, and flower models, because of their clear and simple skeletons, which allow one to intuitively rotate the shape around its parts ("easy to turn the hand around a finger"'; "easy to turn the horse around a leg"; "S helps to turn the flower around its stem"). However, several users mentioned advantages of S when used in combination with T. These are discussed below;

- **Both (B):** Overall, this modality received the most positive comments. It was deemed the "most accurate"; and "feeling quick to use when we have two methods [to choose from]". Specifically, users noted that B is "good for doing final adjustments / fine tuning the alignment" and that "S helps T to get the desired result easily" and "I started with T and used S for final touches". One user also commented: "I work as a graphic designer with a lot



Q1	The tool met my needs for performing the alignment task
Q2	The tool worked as expected (after following the training)
Q3	The tool helped me be more effective than the two other tools
Q4	The tool was easy to use
Q5	The tool was accurate
Q6	The tool requires the fewest steps (compared to the other two) to accomplish my goals
Q7	I felt that I have to think carefully to get a good result with this tool
Q8	The tool was tiring to use
Q9	Both occasional and regular users would like the tool
Q10	I can use the tool successfully every time
Q11	I learned to use the tool quickly
Q12	I easily remember how to use the tool
Q13	I am satisfied with the tool
Q14	Which tool (T, S, or B) do you overall prefer?

Figure 5.12: Results of 13-point user questionnaire for the three modalities. Questions are shown below the charts. See Section 5.5.3.2.

of 3D tools; I see how S helps me by providing a lot of control when rotating, and I would love to have this tool along with my other manipulation tools in my software [...] but I would not use it standalone".

Summarizing the above, we see that our initial hypothesis that the S modality helps (complements) T for precision tasks is largely supported by user experience.

5.6 DISCUSSION

We next discuss our proposed skeleton-based rotation method along a number of dimensions.

5.6.1 *Technical aspects*

The skeleton-based rotation method presented in Section 5.3 has the following main features:

Genericity: We handle 3D meshes, polygon soups, and point clouds; our only requirement is that these generate fragments with a depth value. This contrasts using 3D curve skeletons for interaction, which heavily constrain the input scene quality, and cannot be computed in real time, as already mentioned. Also, the skeleton tool can be directly combined (used alongside) any other interaction tool, such as trackball, with no constraints.

Reversibility: Since 3D rotation axes are computed from 2D silhouette skeletons, rotations are not, strictly speaking, invertible: Rotating from a viewpoint $\mathbf{v}_1$ with an angle α around a 3D local axis $\mathbf{a}_1$ computed from the silhouette Ω_1 leads to a viewpoint $\mathbf{v}_2$ in which, from the corresponding silhouette Ω_2, a different axis $\mathbf{a}_2 \neq \mathbf{a}_1$ can be computed. This is however a problem only if the user *releases* the pointer (mouse) button to end the rotation; if the button is not released, the computation of a new axis $\mathbf{a}_2$ is not started, so moving the pointer back will reverse the rotation.

Scalability: Our method uses OpenGL 1.1 (primitive rendering and Z-buffer reading) plus the 2D image-based skeletonization method in [158] used to compute the skeleton S_Ω, its regularization $\overline{S}_\Omega$, and the feature transform $FT_{\overline{S}_\Omega}$. We implemented skeletonization in NVidia's CUDA and C++ to handle scenes of hundreds of thousands of polygons rendered at 1000^2 pixel resolution in a few milliseconds on a consumer-grade GPU, *e.g.* GTX 660. The skeletonization computational complexity is linear in the number of silhouette pixels, *i.e.*, $O(|\Omega|)$. This is due to the fact that the underlying distance transform used has the same linear complexity. For details on this, we refer to the original algorithm [18]. The separate code of this skeletonization method is available at [156]. Implementing the two improvements presented in Section 5.3 is also computationally efficient: The skeleton's distance transform $DT_{\overline{S}_\Omega}$ is already computed during the rotation axis estimation (Section 5.3.1C). The distance $DT_{\overline{\Omega}}$ and feature transforms $FT_{\overline{\Omega}}$ require one extra skeletonization pass of the background image $\overline{\Omega}$. All in all, our method delivers interaction at over 100 frames-per-second on the aforementioned consumer-grade GPU. The full code of our skeleton-and-trackball manipulation tool (Section 5.5.1) is provided online [159].

Novelty: To our knowledge, this is the first time when 2D image-based skeletons have been used to perform interactive manipulations of 3D shapes. Compared to similar view-based reconstructions of 3D curve skeletons from their 2D silhouettes [80, 95], our method requires a *single* viewpoint to compute an approximate 3D curve skeleton and is two to three orders of magnitude faster.

5.6.2 *Usability and applicability*

The evaluation described in Section 5.5 confirmed the insights elicited from the earlier formative study (Section 5.4), *i.e.* that skeleton rotation is best for precise, small-scale, final alignment touches; and that skeleton rotation best works as a *complement*, and not replacement, of trackball rotation. The latter point was supported by all types of data from our evaluation — task timing, scores assigned by users to evaluation questions, and free-form text feedback. The same data shows that users rank the combined modality (B) as better than both S and T modalities taken separately. The user scores also show that, overall, the combined modality is easy to learn and use (Figure 5.12, Q2-4-8-11-12). Put together, all the above support our claim of added value for the skeleton-based rotation technique.

Besides the above results, the user study also unveiled several questions which we cannot fully answer:

User performance: There is a large variability of user performance, measured as task success rates and completion times (see Figure 5.11b and related text). We cannot explain this variability by differences in the experiment setup, previous user familiarity with 3D manipulation, amount of training with the evaluated tool, or other measured factors. This variability may be due to user characteristics which the self-reported variables (Figure 5.9b and related text) do not capture; to the high heterogeneity of the user population; but also due to dependent variables which we did not measure, *e.g.*, how often did users use the skeleton simplification level (Section 5.5.1) to produce suitable skeletons for generating rotation axes. Repeating the experiment with a more homogeneous population and more measured variables would help answering this question.

Applicability: An important limitation of our study is that, for the B modality, we did not measure how (much) the task was completed using each *separate* modality, *i.e.*, S and T. The formative study (Section 5.4), textual user feedback for the controlled experiment, and our observation of the users during the experiment jointly show that, in most cases involving moderate or hard tasks, trackball was *first* used to obtain a viewpoint roughly close to the target one, which was next fine-tuned using skeleton. This is fully in line with our initial design ideas (see Figure 5.1 and related text) and also with earlier findings on what trackball best works for [56, 68, 109]. However, understanding more precisely which are the rotation types that skeleton best supports would greatly help to improve the combined modality by *e.g.* suggesting this modality to the user when it appears fit, and/or conversely, blocking this modality when it does not match the task at hand.

Study limitations: Besides the above-mentioned aspects, our study (Section 5.5) has further limitations: It uses only four shapes that cannot capture the rich distributions of 3D shapes that need manipulation. Also, it only covers the task of rotating from an initial pose to a given final pose. Yet, manipulation is also used for free exploration and/or design actions which do not require reaching a predefined pose. It is unclear how to *quantitatively* measure the added value of interaction tools in such contexts, beyond qualitative user-satisfaction questionnaires [56]. Also, we cannot exclude learning effects between the trials that address the same task with different modalities. Finally, what is the exact added-value of all the rotation-specification improvements (Section 5.3.3) was not currently measured. Exploring all these directions is left to future work.

5.7 CONCLUSION

We proposed a method for specifying interactive rotations of 3D scenes around local rotation axes using image skeletons. We compute such axes from the skeleton of the 2D silhouette of the rendered scene, enhanced with depth information from the rendered Z buffer. Specifying such rotation axes requires a single click-and-drag gesture and no additional parameter settings. Our method is simple to implement, using distance and feature transforms provided by modern 2D skeletonization algorithms; handles 3D scenes consisting of arbitrarily complex polygon meshes (not necessarily watertight, connected, and/or of good quality) or 3D point clouds; can be integrated in any 3D viewing system that allows access to the Z buffer; and works at interactive frame-rates even for scenes of hundreds of thousands of primitives.

We measured the added value of the proposed rotation technique by a formative study (to elicit main concerns from users) followed by a controlled user study. Results showed that, when combined with trackball rotation, our method leads to better results (in terms of task completion times) and higher user satisfaction than trackball rotation alone. Also, our method is easy to learn and does not carry a significant learning or execution cost for the users, thereby not increasing the costs of using standard trackball rotation.

Several future work directions are possible. More cues can be used to infer more accurate 3D curve skeletons from image data, such as shading and depth gradients, leading to more precise rotation axes. Such data-driven cues could be also used to better control the rotation, and also suggest to the user which of the two modalities (skeleton-based or trackball rotation) are best for a given context. Separately, we aim to deploy our joint skeleton-and-trackball rotation tool on touch displays (single or multiple inputs) and evaluate its effectiveness in supporting domain experts to perform 3D exploration for specific applications, such as the astronomical data exploration outlined in Section 5.4.

6 CO-SKELETONS: CONSISTENT CURVE SKELETONS FOR SHAPE FAMILIES

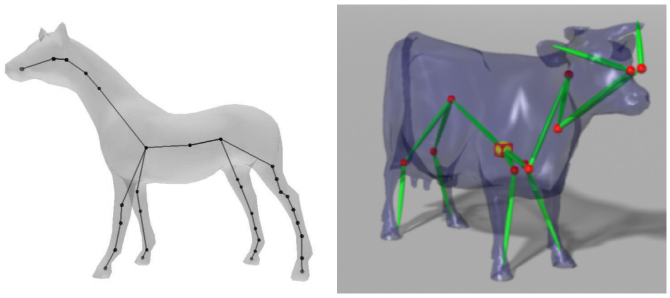
The work presented so far in this thesis addressed the tasks of exploration (Chapters 3 and 4) and examination (Chapter 5). These correspond to the highest level of aggregation — exploring an entire 3D collection — respectively the highest level of detail — exploring an individual shape. In this chapter, we focus on an in-between level, that of exploring a set of related 3D shapes, such as selected by the user in the visual overviews created in Chapters 3 and 4 or as the result of a content-based retrieval query. How can we *jointly* handle such a set of related shapes? Addressing this would enable multiple use-cases, such as aligning all these related shapes in a common reference frame, computing their variability or similarity, or extracting descriptors that jointly represent such a shape set.

We approach the above questions by proposing a skeletal descriptors that represents an entire collection of related 3D shapes. More specifically, we present a method that computes consistent curve skeletons for 3D shapes from a given family, leading to a family-level descriptor that we call co-skeleton. We argue that a group of related shapes can help us to deduce the semantic correlations between their curve skeletons. We also show the utility of our method by using co-skeletons for shape segmentation and shape blending on real-world data. Our method is based on the 3D curve skeletons computed by different independent state-of-the-art skeletonization methods and, as such, can be added to any 3D shape processing pipeline that admits the computation of such standard curve skeleton descriptors.

6.1 INTRODUCTION

Skeletons are thin and locally centered structures which describe the geometry, topology, and symmetry properties of shapes compactly and intuitively. This makes skeletons powerful tools for applications such as shape segmentation [119], manipulation [85, 178], and blending [2]. Existing skeletonization methods can be classified in methods that compute *surface* skeletons [3, 98] and methods that compute *curve* skeletons [5, 148]. Surface skeletons capture shape geometry better, but curve skeletons are much simpler (and faster) to compute, represent, and analyze, and are the dominant skeleton types currently used in applications [149].

Among the many existing curve skeletonization, important differences exist regarding the *quality* of the produced skeletons, which is



(a) Le *et al. 2003* [93] (b) Liu *et al. 2014* [85]

Figure 6.1: Curve skeletons extracted by Le *et al.* [93] and Liu *et al.* [85] for mesh rigging. As visible, the produced skeletons are not always locally centered and, at places, even exit the shape.

measured by criteria including thinness, centeredness, compactness, robustness to noise, homotopy equivalence to the input shape, and computational complexity [143]. Quality issues create problems when using skeletons in certain applications such as mesh rigging. Figure 6.1 shows the curve skeletons (CSs) extracted by two such methods. The extracted skeletons exhibit problems such as shrinkage of end branches with respect to the corresponding shape parts or even exiting the shape at places. Other skeletonization methods exhibit different problems with respect to the mentioned quality criteria. Given such problems, most of the skeleton-based mesh rigging approaches require that skeletons are manually specified for the input shape, a process which is time-consuming and error-prone.

Obtaining high quality curve skeletons — important for applications like rigging [7] and shape segmentation [119] — has been done so far by proposing increasingly improved skeletonization methods. Yet, new methods may introduce new problems, more user parameters, or have a more complex implementation [149]; and must be thoroughly tested on large shape collections [143, 144]. Another problem of all skeletonization methods is that they cannot guarantee that they preserve the *same* level-of-detail on similar parts of *any* input shape. Given an actual shape and parameter settings, details may be kept in the skeleton or simplified away. This creates *inconsistent* skeletons from the viewpoint of applications that use them further to manipulate shapes.

We propose a different approach: Inspired by recent approaches on co-analysis of 3D shape collections, rather than aiming to compute a high-quality skeleton from a single *shape*, we use a *collection* of shapes (of the same kind) to compute their skeletons. The intuition behind this is that a given skeletonization method will be able to extract good-quality skeletons from *most* parts of *most* shapes, and will not fail consistently on the same parts of all shapes. By combining information from

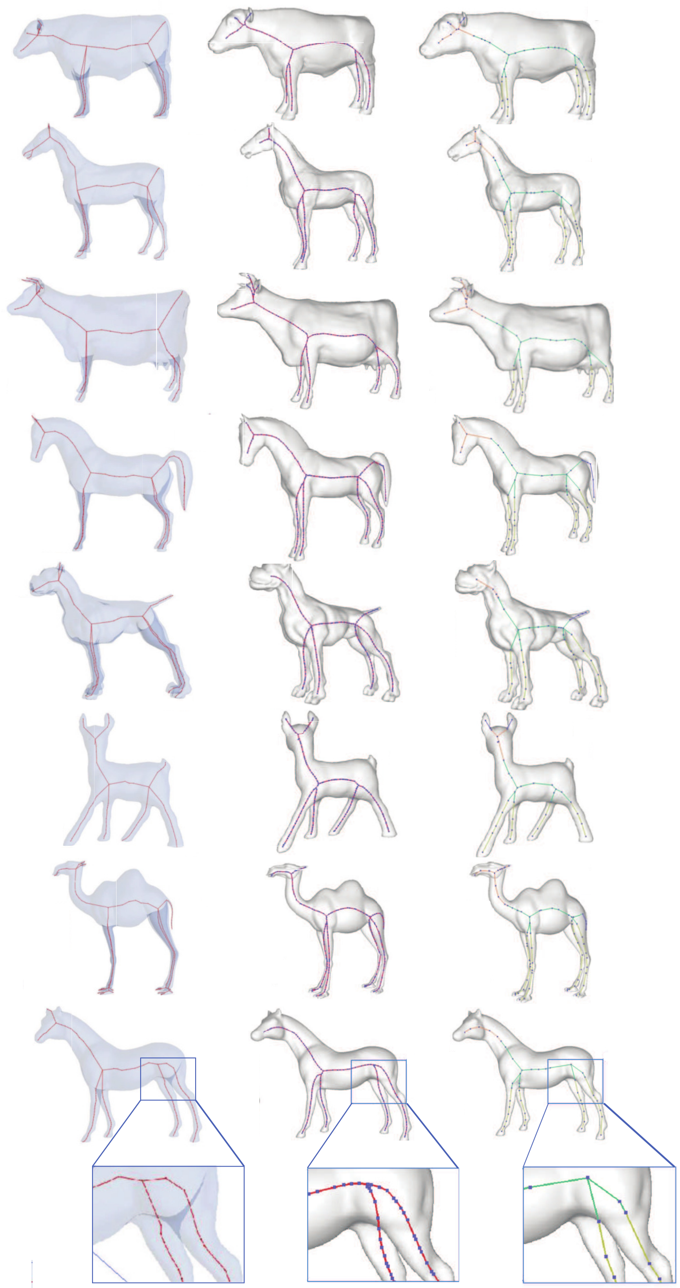


Figure 6.2: Left and middle columns: Skeletons extracted from the *fourleg* subset of the Princeton Shape Benchmark (PSB) [25] using the Mean Curvature Flow [148] and Mesh Contraction [5] methods, respectively. These skeletons have a variable sampling density and preserve (or not) similar details across different shapes. Right column: Co-skeletons extracted by our approach are more concise and consistent in preserving similar details across different shapes.

all extracted skeletons, we obtain *co-skeletons* which represent well all shapes in a given family with controlled and consistent sampling density and presence of significant details in all skeletons, even when large variations exist between individual shapes.

Our method works as follows: Given a 3D shape collection, we extract the curve skeletons from all shapes, using any existing good-quality CS method chosen by the user. Next, we use several descriptors to characterize these skeletons, and use them to cluster similar branches from different skeletons. Finally, we infer the semantic correlation among corresponding edges and use this information to *jointly* prune all skeletons to achieve *conciseness* (representing CSs with few sampling points and edges) and *consistency* (the same type of shape detail creates the same type of skeleton branch) over an entire shape family. We show our automatic co-skeleton extraction framework by applications of shape co-segmentation and shape blending.

The structure of this chapter is as follows. Section 6.2 reviews related work in curve skeletonization. Section 6.3 outlines our framework. Section 6.4 details our pruning algorithm for skeleton consistency. Section 6.5 presents results and applications. Section 6.6 discusses our proposal and outlines future work directions.

6.2 RELATED WORK

Skeleton extraction and shape co-analysis are two aspect about our work. In Section 2.4 we have introduced skeleton extraction, here we present the related work on shape co-analysis.

Shape Co-analysis: There has been recent increasing interest in the co-analysis of shape collections. The premise is that more information can be extracted by analyzing a collection than when analyzing individual shapes. An example hereof is *co-segmentation* — the simultaneous segmentation of all shapes in a set in a consistent manner. This has been shown to be of great utility for modeling and texturing [20, 74, 140]. Golovinskiy and Funkhouser [50] pioneered consistent co-segmentation by aligning all shapes and then clustering their primitives. Following this work, many co-segmentation approaches have been proposed, using unsupervised learning [61, 140, 174] or semi-supervised learning [169, 175]. Deep learning has shown excellent performance in this direction, with several methods proposed for shape segmentation [52, 168]. Yet, such methods heavily rely on large training datasets.

Besides co-segmentation, other approaches exist for the co-analysis of a shape-set. Laga *et al.* [82] presented an effective algorithm to obtain semantic correspondences between 3D shapes that finds part-wise correspondences. Kaick *et al.* [72] constructed a unified shape co-hierarchy from a shape set, providing a richer characterization of the shape-set be-

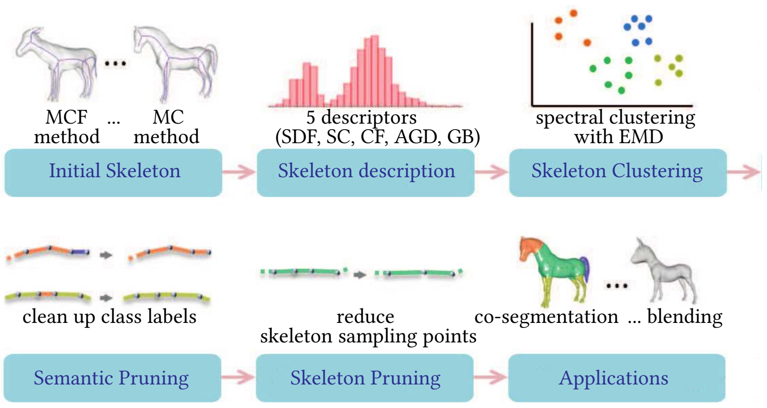


Figure 6.3: Our method has six steps. Curve skeletons are extracted from 3D input shapes using existing skeletonization methods. These skeletons are next reduced to five per-face descriptors. The descriptors of all skeletons from a shape family are next clustered. Finally, we prune (simplify) the clustered data to remove semantic noise (semantic pruning) and obtain skeletons with a small sample count (skeleton pruning). We use co-skeletons for shape co-segmentation and blending applications.

yond coarse template-based or part-level correspondence. Yumer and Kara [183] propose a co-abstraction method where shapes in a set are abstracted as much as possible while still preserving the unique geometric characteristics distinguishing them from each other. Xu *et al.* [177] synthesize new shapes by analyzing a given shape-set using genetic algorithms. Kim *et al.* [78] construct cuboid model templates of large shape sets. Fish *et al.* [48] learn the configurations of a shape-set as geometric distributions. Yumer and Kara [184] use co-constrained handles to deform shape-sets to find and respect the geometric and spatial constraints among different shape parts. Our work is inspired by these techniques: We compute family-consistent skeletons in terms of sampling density and semantic relevance. Hence, even if the underlying curve-skeletonization method that we use is imperfect, the problems that it creates on *individual* shapes are alleviated or removed by considering *all* shapes in the family.

6.3 PROPOSED METHOD

Figure 6.3 shows the pipeline of our method. Our input is a collection of N shapes $\mathcal{I} = \{I_1, \ldots, I_N\}$, $I_i \subset \mathbb{R}^3$, of one shape family. By a family, we mean a set of shapes that belong to the same semantic class, *e.g.*, four-legged animals or chairs. Shapes are represented as boundary meshes [149]. The goal of our method is to obtain family-consistent curve skeletons $\mathcal{S} = \{S_1, \ldots, S_N\}$, one for each shape in the input collection. For each shape I_i, we consider its curve skeleton S_i, modeled as a set of 3D sample points P_i connected by edges E_i, *i.e.*, $S_i = (P_i, E_i)$, $P_i \subset \mathbb{R}^3$, $E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,n_i}\} \subset P_i \times P_i$. For a collection $\mathcal{I}$, we first extract the initial skeletons S_i for each shape I_i individually, using existing state-of-the-art methods [5, 148]. We then extract features for all skeleton edges $e_{i,j}$ and cluster edges in a joint descriptor space to infer their semantic correlation. To obtain concise and family-consistent skeletons, we propose and apply two pruning algorithms, which lead to co-skeletons S_i suitable for shape co-segmentation and shape blending. We next describe each step of our pipeline in turn.

Initial Skeleton Extraction: We extract shape skeletons S_i using the mean curvature flow (MCF) method [148] or, alternatively, the mesh contraction (MC) method [5]. Any other curve skeletonization method can be used directly, if desired. For selecting alternatives to MCF and MC, one can study the survey of Sobiecki *et al.* [144] to pick the method of choice based on various desirable properties, such as speed, ease of use, type of input (mesh or voxel volume), or robustness. For brevity, we next show results based on the MCF method [148], which we found slightly easier to use than MC and producing smoother curve skeletons (see also Figure 6.2). However, using MC yields very similar co-skeletons to MCF, so the choice between the two is largely left to the user's preference. Regardless of the skeletonization method choice, we compute skeletons for each shape with approximately $|E_i| = 100$ skeleton edges each.

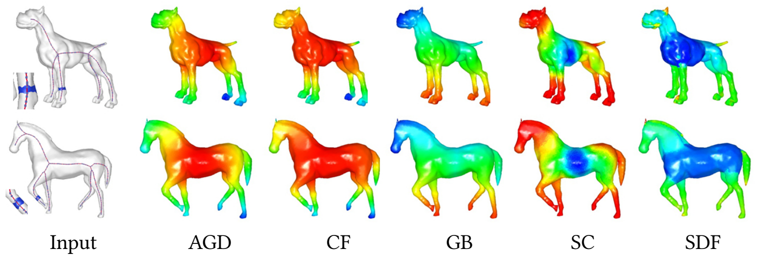


Figure 6.4: Feature extraction in our pipeline. Left column: Two examples of input shapes with their initial curve skeletons (red) and the shape faces (blue) associated with a selected skeleton edge. The other columns show our five feature descriptors computed on the two models (feature names are detailed in Section 6.3). These descriptors are aggregated, via their histogram distributions, to form the skeleton-edge descriptors.

Skeleton Description: No matter which skeletonization technique one uses, every skeleton edge can be mapped to a set of shape faces [149]. This mapping [57], known as the feature transform $FT : S \rightarrow \mathcal{P}(I)$, with $\mathcal{P}$ denoting the power set, maps $FT(e \in S)$ to the set of faces in I that correspond to a skeleton edge e. The FT complements topological shape information, captured by the curve-skeleton's branching structure, with geometric information that encodes which skeleton fragments capture which shape-surface details. Both information types are essential for semantic or functional prediction.

We use five shape descriptors to characterize surface details, similar to previous co-analysis approaches [140, 173, 174]: Shape Diameter Function (SDF) [132], Conformal Factor (CF) [11], Shape Contexts (SC) [9], Average Geodesic Distance (AGD) [58], and Geodesic distance to the Base of the shape (GB) [140]. These descriptors are defined on mesh faces. Taking the computation of SC as an example, given a mesh face, we compute the distribution of all other faces (weighted by their area) in logarithmic geodesic-distance bins and uniform-angle bins, where angles are measured relative to the normal of each face. Hence, each shape face is described by five scalar values that correspond to the five above mentioned descriptors.

Given a skeleton edge $e_{i,j} \in S_i$ with its associated faces $FT(e_{i,j}) \in I_i$ and their face descriptors, we use normalized histograms with a specified bin value to measure the feature distribution of $e_{i,j}$. Using histograms ensures that the number of faces $|FT(e_{i,j})|$ belonging to a skeleton edge $e_{i,j}$ is normalized over all skeleton edges. We next compute a so-called *descriptor space* over all shapes I_i in a family. Figure 6.4 shows two examples. The leftmost column shows the faces (blue) associated to a skeleton edge (red). The other columns show the five feature descriptors we compute, color-coded on a rainbow colormap. As explained, these descriptors are ultimately recorded on the skeleton edges via their feature histograms.

Skeleton Clustering: As already explained, there is no unanimously accepted formal definition of 3D curve skeletons, let alone of co-skeletons for a shape *family*. This implies that it is difficult to define consistency. Hence, we proceed by processing all skeleton edges in a shape family in a unified and global manner. As we do not have explicit semantic information, we resort to clustering, which is the approach of choice in many related co-analysis techniques. We therefore simplify (cluster) *all* edges of *all* skeletons of a shape family, together with their computed descriptor values, in the per-family descriptor space. To simplify notation, let e_a and e_b be two skeleton edges in the whole family. For each descriptor, let $p_a = FT(e_a)$ be the set of faces corresponding to an edge e_a. Let $h_{a,k}$ be the histogram over p_a of the k-th descriptor ($1 \leq k \leq 5$). We define the *dissimilarity* between two edges e_a and e_b with respect to the k-th descriptor as

$$d_k(e_a, e_b) = \mathrm{EMD}(h_{a,k}, h_{b,k}),$$

where $\mathrm{EMD}(h, \bar{h})$ is the Earth Mover's Distance [127] between histograms h and $\bar{h}$. EMD is a typical method for evaluating dissimilarity between two multidimensional distributions in a feature space. We next apply a Gaussian kernel to the distances d_k to build an affinity matrix $W_k = (w_{a,b,k})$ for each descriptor k, with entries

$$w_{a,b,k} = \exp\left(-\frac{d_k(e_a, e_b)}{2\sigma^2}\right), \tag{6.1}$$

where $w_{a,b,k}$ is the dissimilarity between e_a and e_b for the k-th descriptor. We set the number of bins to 50 for each histogram, and σ to the mean of all dissimilarities, respectively.

We now seek a way to combine the five affinity matrices W_k into a single matrix, to be next used to perform the co-skeleton computation. We note that our five descriptors characterize partially-related shape aspects. For instance, the AGD and CF descriptors take typically large values on a shape's center and low values on its extremities; see Figure 6.4. Hence, simply merging the five affinity matrices W_k into a single matrix would result in redundant information. To avoid this, we use affinity-aggregation spectral clustering [174] to jointly perform feature selection and clustering — that is, reduce the amount of redundant information and also decompose the resulting information into self-similar subsets. For this, we proceed as follows: Let $\alpha = [\alpha_1, \ldots, \alpha_5]$ be weights associated to the affinity matrices $W_1, \ldots, W_5$ that indicate how much each matrix contributes to describing similarity over the shape family. We formulate the affinity-aggregation spectral clustering as

$$\min_{\alpha, F} \sum_{k=1}^{5} \sum_{a,b} \alpha_k w_{a,b,k} \|f_a - f_b\|^2, \tag{6.2}$$

where $F = [f_1, \cdots, f_m]$ is the indicator vector in joint feature space having a total of m samples.

The minimization in Eqn. 6.2 involves two unknown vectors, α and F. To solve for them, we use a two-step minimization approach that alternatively fixes one unknown vector and varies the other. During optimization, two additional constraints must be satisfied: The first one comes from normalized spectral clustering, *i.e.*, the final diagonal matrix D must satisfy

$$1 = F'DF = F'(\alpha_1 D_1 + \cdots + \alpha_5 D_5)F = \sum_{k=1}^{5} \alpha_k s_k, \tag{6.3}$$

where

$$s_k = F' D_k F$$

and D_k is a diagonal matrix whose i-th diagonal element is the sum of the elements in the i-th row of W_k. Using this constraint, spectral clustering typically converges to a result [96]. The second constraint comes from the Cauchy-Schwartz inequality, which leads to constraining the sum of the weighted matrices in a normalized condition, *i.e.*,

$$\sum_{k=1}^{5} \sqrt{\alpha_k} = 1. \tag{6.4}$$

By applying the Lagrange multiplier method to the constraints in Eqns. 6.3 and 6.4, the problem of finding α can be reduced to a one-dimensional line-search problem, which is easy to solve. For more details, we refer the interested reader to [62].

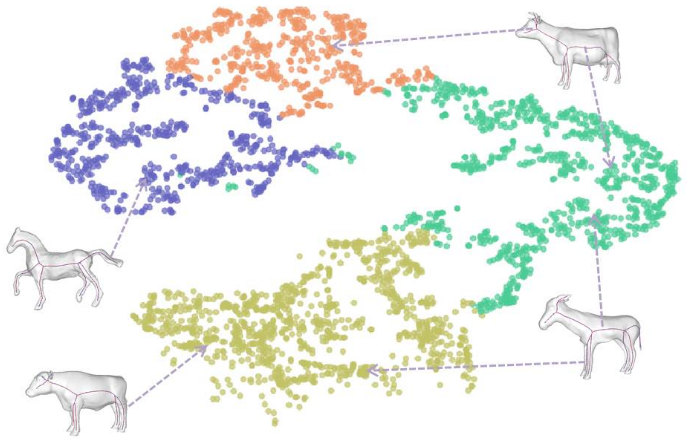


Figure 6.5: Skeleton edge clustering in joint feature space. Family-consistent semantic correlation can be deduced from the clusters.

After obtaining F, we run k-means in feature space to cluster the data into C classes, where C is assigned according to the number of parts of each shape in the input family $\mathcal{I}$, under human supervision. That is, for a given shape family, the user has to decide what is a suitable number of parts that typical shapes in that family have — or, putting it differently, by how many parts the user wants to model shapes in that family. Figure 6.5 shows our clustering result on the *fourleg* dataset, visualized using t-SNE [97] with different clusters colored differently. Points that are close in the embedding (2D) space have thus similar feature vectors. Skeleton edges that belong to the same cluster are assumed to be semantically similar. Note that this assumption is reasonable as many co-analysis algorithms operate under it (see Section 6.2). As the figure shows, four large clusters appear, which correspond to four parts of shapes in the *fourleg* dataset.

Semantic Pruning: We next use semantic pruning to remove so-called *semantic noise*, which occurs in our clustering due to three reasons:

(1) Shape meshes may contain noise, which propagates into the five descriptors. (2) The descriptors themselves contain a certain amount of fuzziness, as they only measure geometric properties rather than the 'true' semantic ones. (3) Our feature selection and clustering steps may introduce artifacts. To remedy these issues, we introduce a semantic pruning step. This step uses the connectivity of skeleton edges, based on the idea that two connected skeleton edges have a high probability of carrying the same semantic information. Importantly, semantic pruning only 'merges' the semantic information extracted from different skeletons in the same family; it does *not* actually remove skeleton nodes, a task which is done next by the skeleton pruning. We describe semantic pruning in detail in Section 6.4.1.

Skeleton Pruning: A final concern is to compute *compact* co-skeletons, *i.e.*, having a small number of points. This assists, speed-wise, all operations that next use co-skeletons. To obtain compact co-skeletons, we carry out skeleton pruning to reduce potential skeleton *over-sampling*. Since existing skeleton-pruning algorithms do not take into account semantic part information over a family of shapes [149], we propose a new pruning procedure that considers and respects such semantic properties. We describe skeleton pruning in detail in Section 6.4.2.

6.4 SKELETON PRUNING DETAILS

We next describe the semantic and skeleton pruning algorithms which aim to reduce semantic noise, respectively oversampling, in our produced co-skeletons.

6.4.1 *Semantic pruning*

As stated in Section 6.3, our semantic pruning exploits the observation that two connected skeleton edges usually hold the same semantic information. Hence, this connectivity information can add extra information to the initial clustering results.

Our semantic pruning algorithm exploits four properties of a skeleton edge to measure the confidence that two skeleton edges fall within the same semantic part. These are the length of the edge, the angles between the edge and the two edges connected to it, and the semantic information of the connected edges themselves.

Edge length: Given a skeleton edge e_i, we compute its normalized length l_i (with respect to the longest edge in I, so $l_i \in [0,1]$) and its angles $\beta_{i,j}$ to edges $N_i = \{e_j\}$ to which e_i is connected in the skeleton graph E. Let the cluster index of e_i, as computed by k-means (Section 6.3), be denoted by $c_i \in [1,\ldots,C]$. Given our clustering, c_i thus encodes semantic similarity of the edges. With the above, we define the *confidence score* K_i of e_i as

$$K_i = \lambda l_i + \sum_{e_j \in N_i} b_{i,j} l_j \gamma(i,j), \tag{6.5}$$

where $\gamma(i,j) = 1$ when $c_i = c_j$ and $\gamma(i,j) = -1$ otherwise. The hyperparameter λ (default: 1.5) defines the relative weight given to edge lengths *vs* edge angles in the confidence score.

Edge angles: The angles $\beta_{i,j}$ are first normalized into $[0,1]$ by computing

$$\hat{b}_{i,j} = (1 + \cos \beta_{i,j})/2 \tag{6.6}$$

and then mapped by a Gaussian kernel to yield the weights

$$b_{i,j} = \exp\left(-\frac{\hat{b}^2_{i,j}}{2}\right).$$

This way, the smaller the angle $\beta_{i,j}$, the smaller the final weight $b_{i,j}$. When $\beta_{i,j} = \pi$, we obtain the highest value of $b_{i,j} = 1$.

Equation 6.5 assigns a low confidence to an edge e_i when its connected edges e_j have different semantic definition, *i.e.*, come from different clusters than e_i. Conversely, a high confidence score tells that e_j has the same semantic information as e_i.

We next sort all skeleton edges e_i of a shape collection ascendingly on their scores K_i and process them in this order as follows. For each e_i, we refine its confidence score K_i to equal the one of its highest-confidence edge-neighbor, *i.e.*, $K_i = \max_{e_j \in N_i} K_j$. After processing an edge, we refresh its confidence score and that of its neighbours. We repeat the process until we have processed approximately 10% of all edges (see Figure 6.6). This value has been set empirically based on tests comprising many shape categories.

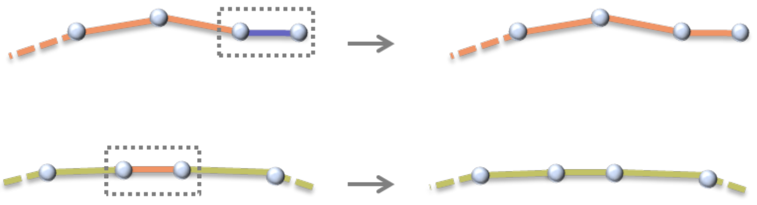


Figure 6.6: Semantic pruning for leaf skeleton edge (top) and a joint skeleton edge (bottom). Colors show class. The role of semantic pruning is to clean up class labels, not to remove edges; skeleton pruning does the latter.

6.4.2 *Skeleton pruning*

From the initial skeletons with their pruned semantic attributes, we now perform skeleton pruning to reduce over-sampling and to remove spurious branches. This way, we achieve *compact* co-skeletons that describe the respective shapes with a small number of sampling points and edges.

In contrast to semantic pruning, we now focus on the nodes (vertices) $\mathbf{x}_i \in P$ of the curve skeletons. We categorize all nodes into three groups: leaf, joint, and branch. A *leaf* node is incident with only one skeleton edge; a *joint* node with two edges; and a *branch* node with at least three edges. As most nodes are joint nodes in curve skeletons, we focus on pruning those only. Also, not pruning leaf or branch nodes ensures that the topology of the pruned skeletons stays identical to the initial ones. Node positions are not altered by pruning, so the pruned skeletons maintain their centeredness with respect to the original shapes. We

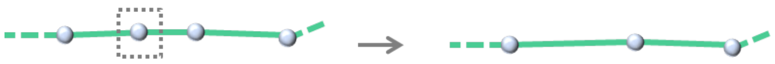


Figure 6.7: Illustration of our skeleton pruning process. A skeleton node is removed depending on its associated angle and the lengths of its incident edges.

exclude from pruning joint nodes whose incident edges carry different semantic information (cluster labels c_i). Pruning such nodes would be difficult, as we would need to somehow merge different cluster labels into newly created edges. Let e_i and e_j be the two incident edges for a node candidate for pruning, and let $\beta_{i,j}$ be the angle spanned by these edges. We use $\beta_{i,j}$ and the normalized edge lengths l_i and l_j, defined as in Section 6.4, to compute a node confidence score as

$$V_{i,j} = (l_i + l_j)\hat{b}_{i,j} \tag{6.7}$$

with $\hat{b}_{i,j}$ defined by Eqn. 6.6. Following Eqn. 6.7, nodes with large angles and with short incident edges have low confidence values. Conversely, nodes with small angles and long incident edges get high confidence values. This models the fact that we want to prune densely-sampled and relatively-straight skeleton branches. We sort all skeleton nodes ascendingly on their confidence values $V_{i,j}$ and prune (remove) nodes in this order, one at a time. After each node removal, we recompute the confidence scores $V_{i,j}$ of the node's two neighbors in the skeleton graph. We prune until approximately 75% of the joint nodes are pruned. Different thresholds yield a more, respectively less, aggressive skeleton simplification, as desired by the application at hand (see Figure 6.7).

6.5 RESULTS AND APPLICATIONS

We tested our method on a PC with an Intel 4GHz i7 processor and 8GB RAM. Our time complexity mainly depends on the descriptor computation. Clustering takes under 1 minute with 20 iterations for 2K skeleton edges. Semantic pruning and skeleton pruning computations are linear in skeleton size, taking one second for 2K skeleton edges. Overall, our end-to-end pipeline takes about 8 minutes for small datasets (20 shapes with around 2K skeleton edges/shape), and scales linearly for larger data.

We demonstrate the utility of our co-skeletons by comparing our results with the raw curve skeletons (extracted as explained in Section 6.5.1). We also show co-skeletons in action in two applications: shape segmentation and shape blending (Section 6.5.2).

6.5.1 *Co-skeleton results*

We compare our co-skeletons with the initial MCF and MC curve skeletons [5, 148]. As test data, we used the Princeton Shape Benchmark (PSB) dataset [25], which contains sets of shapes of multiple types, *e.g.*, animals, chairs, human models, furniture, and vehicles. Figure 6.2 shows that our co-skeletons are significantly more concise (have fewer nodes) than the original curve skeletons, while preserving overall desirable characteristics such as centeredness and topology. Moreover, each edge of our co-skeletons is annotated with semantic information (color-coded in Figure 6.2). For instance, all edges pertaining to the animals' heads, legs, or rump, have the same color. Such semantic information can next be used in a wide range of applications, such as shape matching, retrieval, or segmentation.

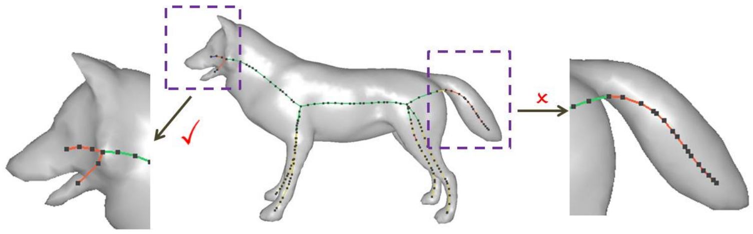
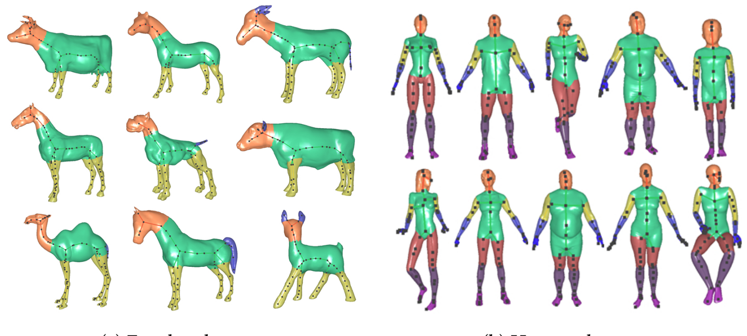


Figure 6.8: A correct case (left) and a wrong case (right) of semantic pruning in one example. Our approach may fail to deal with the case of successive semantic noise in skeleton edges.

Figure 6.8 shows an additional result of our pruning: In most cases, even in the presence of semantic noise, our approach succeeds. Yet, in cases with significant semantic noise, our approach may fail. This is due to the fact that our method uses a voting mechanism that takes into account the semantic information of connected skeleton edges. One potential remedy is the use of a better, more robust, initial skeletonization method, than [148] or [5]. Any (existing or future) skeletonization method that accepts 3D meshes as input, and produces a polyline representation of the curve skeleton, together with the feature transform of its points, is directly applicable.

6.5.2 *Co-skeleton applications*

We next aim to show the potential of co-skeletons by presenting two applications that may benefit from them: shape co-segmentation and shape blending.



(a) Fourleg dataset (b) Human dataset

Figure 6.9: Segmentation results based on our co-skeletons. Different colors depict different semantic parts. The pruned (co-)skeletons inherently encode co-segmentation results. Further results can be found in Figure 6.10.

Co-segmentation: Since skeleton edges are inherently linked to collections of faces of the input shape(s), we can use the semantic information our algorithm produces to segment shapes. Like state-of-the-art co-segmentation approaches [61, 140, 174], we also use the graph cut algorithm [15] to optimize the boundaries of different segments. Figure 6.9 shows several segmentation results based on our co-skeletons for the *fourleg* (Figure 6.9a) and *human* datasets (Figure 6.9b). As visible, the produced segmentations are consistent, in the sense that different shapes (from the same family) get segmented at approximately the same level of detail — four limbs, rump, and head, for the *fourleg* shapes, and rump, head, legs (thigh, calf, foot), and hands (forearm, arm), respectively. However, details such as ears or horns for the *fourleg* shapes, are sometimes not separately segmented, for the shapes in which they are very small. Here we showcase our method on further results. Figure 6.10 presents (co-)segmentation results based on our co-skeletons

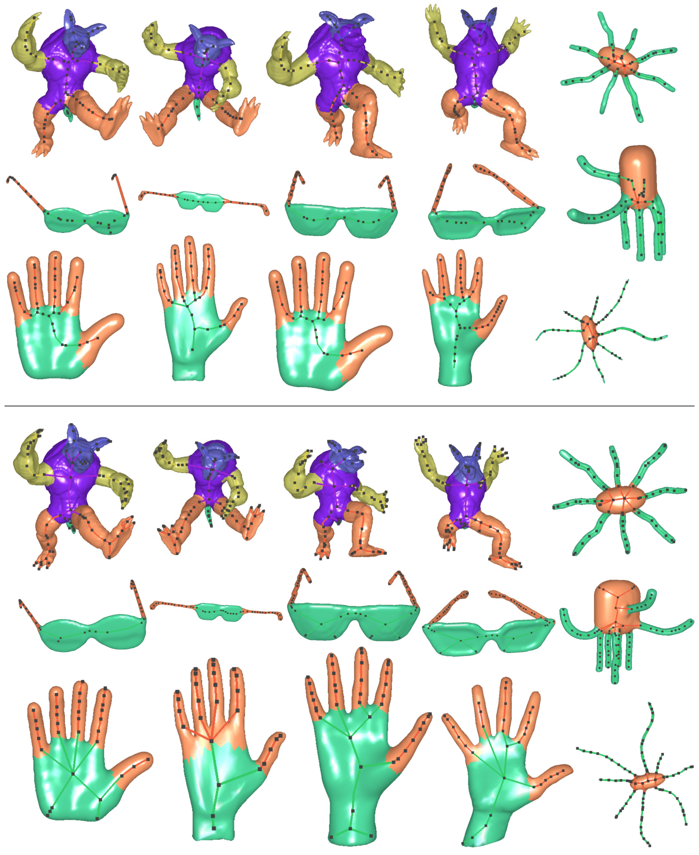


Figure 6.10: Co-skeletonization and co-segmentation results of our method based on Mean Curvature Flow [148] (top half) and Mesh Contraction [5] (bottom half).

building on the per-shape input skeletons computed using Mean Curvature Flow [148] and Mesh Contraction [5].

We next compare our co-segmentation results to five state-of-the-art methods [52, 61, 140, 168, 174]. Note that these are also co-segmentation methods which consider shape families rather than individual shapes. Similar to these methods, we measure the amount of area of a shape that is labeled correctly as

$$\mathrm{acc}(I) = \frac{\sum_i a_i \delta(c_i, t_i)}{\sum_i a_i}, \tag{6.8}$$

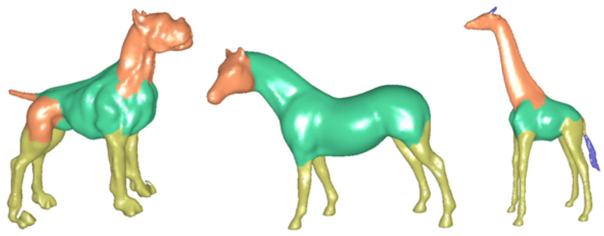
where a_i, c_i, and t_i are the area, label computed by our co-segmentation, and respectively ground-truth label of face i of a given shape I, and δ is Kronecker's delta.

Table 6: Comparison of the average accuracy (Eqn. 6.8) of our co-segmentation results *vs* existing techniques [52, 61, 140, 168, 174]. We separate unsupervised techniques [61, 140, 174] and supervised ones [52, 168] for fair comparison. [52, 168] use only 6 training shapes in their experiments.

Shape	**Average accuracy per category**					
category	Ours	[140]	[61]	[174]	[52]	[168]
Human	**83.2**	-	70.4	78.0	-	-
Glasses	95.8	-	98.3	92.4	96.78	97.15
Airplane	80.2	-	83.3	79.6	95.56	93.90
Ant	88.1	-	92.9	90.1	-	-
Chair	93.8	85.0	89.6	87.6	97.93	97.05
Octopus	92.4	-	97.5	96.8	98.61	98.67
Table	98.6	-	99.0	98.4	99.11	99.25
Teddy	89.8	-	97.1	94.9	98.00	98.04
Hand	83.4	-	91.9	90.3	-	-
Plier	80.9	-	86.0	83.4	95.01	95.71
Fish	80.3	-	85.6	82.4	96.22	95.63
Bird	76.9	-	71.5	72.0	87.51	89.03
Armadillo	67.6	-	87.3	78.5	-	-
Fourleg	**92.1**	77.3	88.7	87.7	-	-
Candelabra	82.5	84.8	93.9	97.2	-	-
Lamp	91.1	94.1	90.7	98.4	-	-

Table 6 lists the accuracy values averaged per shape family for the PSB benchmark, with unsupervised and supervised methods reported separately. It shows that our co-segmentation achieves comparable results for this benchmark in the unsupervised group. However, we gain better performance for some shape families, *e.g.*, human and fourlegs, due to the semantic pruning step. Our results are driven by skeletons, so they contain both *skeleton* and *segmentation* information, in contrast to other pure segmentation approaches. We also see that supervised learning methods perform overall better than unsupervised ones. Yet, as said, supervised methods require significant training data, which we do not need. See also Figure 6.11 for additional insights.

Finally, we compare our segmentation results with nine classical segmentation methods, which do not consider shape families (that is, which are not of the co-segmentation type). Figure 6.12 shows the results for the *hand* and *horse* shapes from the PSB benchmark. The rightmost two columns show our results obtained with co-skeletons constructed from Mesh Contraction (MC) [5], respectively Mean Curvature Flow (MCF) [148] base skeletons. We consider in the comparison both skeleton-based and surface-based segmentation methods, as follows. In


(a) Segmentation using the method of Sidi *et al.* [140]

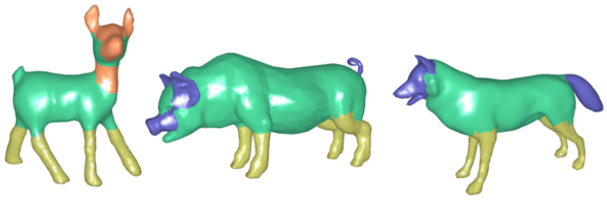

(b) Segmentation using the method of Wu *et al.* [174]

Figure 6.11: Segmentation results using existing methods can lead to wrong part prediction. Our method improves on these results (see Figure 6.9).

the first class, Reniers *et al.* [120] detect curve skeleton junctions and use these to trace geodesic cuts to segment the parts of a shape. The method was further improved in [121] to reduce oversegmentation. Tierny *et al.* [160] segment shapes by analyzing their Reeb graphs, which are related to curve skeletons. Lien *et al.* [92] formulate (and solve) shape segmentation and curve-skeleton computation as a joint optimization problem. Feng *et al.* [46] extend the geodesic-cut-based segmentation in [120, 121] to surface skeletons, which encode both shape geometry and topology, thus provide more information for the segmentation. Finally, Li *et al.* [91] use mesh decimation methods for both shape segmentation but also their curve-skeleton computation.

In the second class, we have methods that segment shapes purely based on the information encoded by their surface. Lee *et al.* [86, 87] segment surface meshes using snake cuts which are optimized based on local mesh features such as curvature and excentricity. Attene *et al.* [4] segment shapes by fitting primitives from a given set (library). Liu and Zhang [94] encode the shape's faces into a similarity matrix which they then decompose by spectral clustering.

Overall, we see that our segmentation results (Figure 6.12 rightmost two columns) compare very favorably with existing methods. In particular, our segment borders are smooth and wrap naturally around the shape, while this is not always the case for the other methods (see Figure 6.12 c, h, i). Also, our co-skeletons ensure that there is no oversegmentation present, a phenomenon that can be observed for some of the other methods (Figure 6.12 b, d, i). From these and other tested examples,

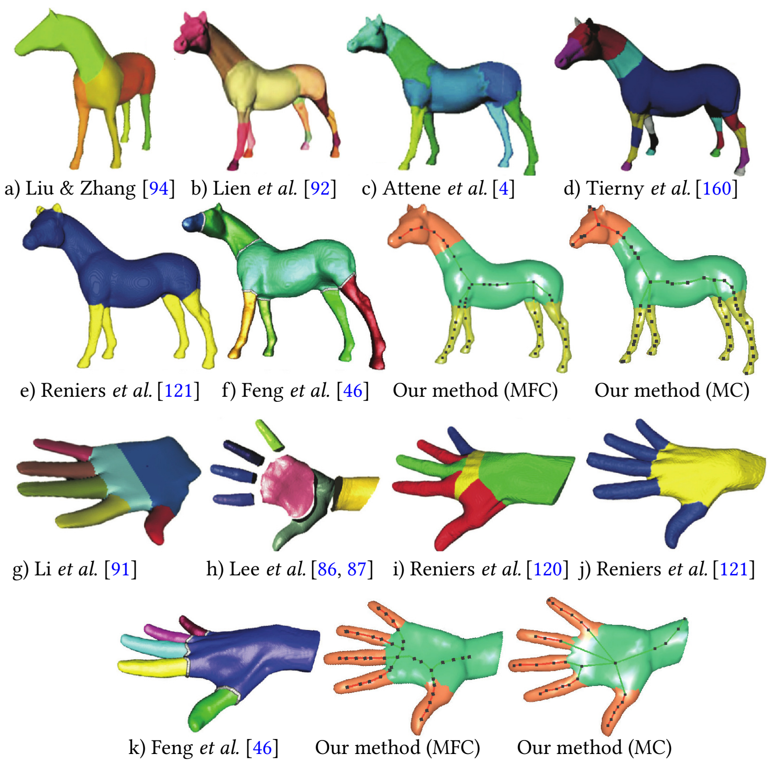


Figure 6.12: Comparison of our co-skeleton segmentation using MC and MCF skeletons with nine other segmentation methods.

we noticed that our segmentation method produces results most similar to the skeleton-cut method of Feng *et al.* (see Figure 6.12 f, k). This can be explained by the fact that both methods optimize for smooth cuts, though with different mechanisms (Feng *et al.* use geodesic tracing; we use graph cuts). Also, both methods use skeletons to drive the segmentation. However, while we use *curve* skeletons which, as explained in Section 6.2 are simple and fast to compute, in particular by the MC and MCF methods that we use here, Feng *et al.* use *surface* skeletons, which are considerably slower and more complex to compute and analyze.

Figure 6.12 shows an additional insight: We see that our segmentations obtained by skeletons computed with two quite different methods (MC and MCF) are very similar. This is due to the fact that we do not use the *raw* skeletons for segmentation, but the *co-skeletons* which, as explained, stabilize skeletons over an entire family by removing outlier details. Further, this suggests that our segmentation approach based on co-skeletons does not strongly depend on the underlying skeletonization method. Hence, one can obtain similar segmentation results by substituting MC or MCF with other, better (*e.g.*, faster and/or easier to use) skeletonization methods.

Shape Blending: Besides shape segmentation, other applications also benefit from co-skeletons, including shape blending. Raw skeletons extracted using even state-of-the-art algorithms are typically inadequate for shape blending, due to the lack of semantic information on skeleton edges and/or over-sampling. To use skeletons, manual post-processing for cleaning and/or annotation is typically needed. In contrast, our co-skeletons can be directly used for shape blending.

We show this by using our co-skeletons to perform shape blending by the technique of Alhashim *et al.* [2]. We first use skeleton edges to reconstruct the spatio-structural graph, and augment this graph by constructing morphing paths between semantically-correlated parts/edges of different shapes of a family. This allows us to keep track of evolving states of the shapes and maintain the topological constraints needed for blending. Next, we select from the obtained results those which show plausible blends and combinations (this selection is done by the user based on what one actually deems to be plausible for a given application context). Finally, we reconstruct shapes based on the structure graphs through the feature transform (FT) mapping from skeleton edges to the shape faces. This yields new blended shapes within the input family. Figure 6.13 shows several examples of blended shapes, demonstrating the effectiveness of our co-skeletons for family-based shape blending.

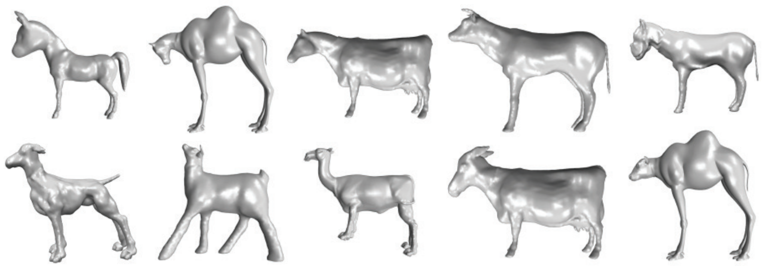


Figure 6.13: Evolution results on the *fourleg* dataset. Using our co-skeletons, we can easily generate new shapes by evolving different combinations across each family. See Figure 6.2 for the initial shapes in the family.

6.6 DISCUSSION AND CONCLUSION

We have presented a novel approach to extracting co-skeletons of a given set of related shapes. In contrast to per-shape skeletons, our co-skeletons have similar quality, measured in terms of simplification level, centeredness, and preservation of details across all considered shapes in a family. Our method has two main use cases. First, we reduce the dependence on the availability of a high-quality skeletonization method, which may not be easy to set up for any set of shapes. Secondly, we maximize the likelihood that the user obtains consistent skeletons over similar-type shapes with no additional parameter tweaking effort. We show the added value of co-skeletons on two applications: shape co-segmentation and shape blending.

While effective, easy to use, fast, and generic, our method has some limitations. First, we use multiple surface-based descriptors to infer the semantic relationships between skeleton edges by 'mapping' such edges to similar surface parts. Yet, the exact relation between specific types of skeleton fragments (*e.g.*, branch ends, junctions, or high-curvature zones) and specific surface details (*e.g.* edges, dents, tubular structures, or other detail types) is not yet fully clear. We aim to study this relationship in more detail. Secondly, we used two curve-skeletonization methods [5, 148] to extract initial skeletons. It is important to study how our co-skeleton proposal behaves when using other curve-skeletonization methods (for a candidate set, see [144, 149]), so as to increase the confidence that our co-skeleton quality does not (strongly) depend on the choice of the underlying skeletonization method. In the long run, we aim to remove this dependency on a specific skeletonization method by extracting co-skeletons *directly* from a shape set. Thirdly, we used here a set of 5 shape descriptors (Section 6.3) which are well-known for related tasks in shape analysis literature. Whether other descriptors would perform better for our task, is an open question.

Concerning the comparison with unsupervised shape segmentation methods, we should say that our co-segmentation benefits from additional information, present in the number of shape parts which is set by the user, which the aforementioned methods do not have. This shows, on the one hand, that adding such semantic information (which is available once we consider an entire shape *family*) benefits segmentation. On the other hand, this should not be seen as a limitation of unsupervised methods since these methods do not utilize such extra information.

Our co-skeleton validation is currently based on only two applications: shape co-segmentation and blending. While the initial results presented here are encouraging, it is of high added value to examine how co-skeletons work for other applications such as shape animation and morphing, and to evaluate our co-skeletons on other benchmark datasets, such as [84]. In this context, a specific application where co-skeletons could be used is precisely the creation of overviews of large 3D shape databases. As an alternative to the engineered features explored in Chapter 3, one could use features extracted from co-skeletons.

7 CONCLUSION

We conclude here our work on analysis and exploration of large 3D shape databases by revisiting the main research question listed in Chapter 1, and a discussion on the methods, limits, and further extension of our proposed answers.

Our main research question — how to help users in exploring, examining, and analyzing shapes and their families present in large 3D collections? — was addressed by considering separately its three parts (exploration, examination, and analysis). We discuss below our answers and findings to each of these parts.

7.1 SHAPE EXPLORATION

We first focused on this research question since, we argue, creating a good overview of a 3D shape database is the 'entry point' to motivate a user to further freely explore such data. As an underlying technique for this overview, we used the framework provided by dimensionality reduction, where we describe every 3D shape by a high dimensional feature vector and next project all vectors of all shapes in a collection to a two-dimensional scatterplot. Given that a good-quality projection will place similar 3D shapes close to each other in the 2D space, the user can next see and explore the structure of a whole 3D database by interactively exploring the resulting projection, suitably annotated with 3D shape thumbnails and supported by mechanisms such as zooming, panning, tooltips, and selection. We next proposed two instances of this framework based on different ways to extract the feature vectors and project them to 2D, as follows.

In Chapter 3, we extracted features by hand engineering, and projected them to 2D by t-SNE. The key advantage of this approach is that, using our visual analytics tools, the user can first and foremost see what is the effect of the features used by the projection. In other words, one can explain why certain shapes were found to be similar to others. Secondly, the user can interactively drive the construction of the projection by selecting different groups of features. There is, we believe, great potential for hand engineering feature sets. By using them, one can create customized views (projections) of the 3D shape database to emphasize, or on the contrary, to remove, the effect captured by certain features.

This strength of the method presented in Chapter 3 is, however, related to its main weakness: Hand engineered features are quite expensive to compute, especially for high-resolution 3D meshes. Also, features of different types can conflict with each other: some are good to separate some types of shapes but bad to separate other shape types. Moreover, while our feature pre-selection techniques discussed in Chapter 3 significantly reduce the search space for good feature-sets from the huge set of all possible combinations to more manageable (smaller) subsets, the space of possibilities is still very high, as it involves also tuning various hyperparameters of the respective feature extractors. Separately, the t-SNE projection used in Chapter 3, while simple to use, is quite slow and has a non-deterministic nature which makes it unsuitable in practice when one wants to create consecutive, stable, overviews of the same 3D database.

We address both above problems — feature extraction and feature projection — in Chapter 4 by using deep learning. For feature extraction, we leverage PointNet, an existing neural model for shape classification. For feature projection, we leverage NNproj, a recent neural model for generic dimensionality reduction. We present several architectures that combine the two models to achieve an end-to-end pipeline for constructing the desired 3D shape database overviews. Qualitative and quantitative studies show that our proposed pipeline achieves considerably faster execution times (even when considering both inference and training) than the hand engineered approach in Chapter 3. The resulting overviews, such as the one shown saliently on the cover of this thesis, are, we believe, of high visual quality, and give a compact but illustrative idea to users of what a 3D shape database contains. Our deep learning approach inherits also other desirable properties of such models, such as out-of-sample capability and being usable with minimal effort.

However, in symmetry with what was said above, the strength of the method presented in Chapter 4 is also related to its main weakness: While fast, automatic, and robust, this method is largely operating in a 'black box' manner. In contrast to the hand engineered method in Chapter 3, our deep learned models cannot be (easily) explained to the user, nor can the user (easily) steer a projection towards a different arrangement of the resulting 2D representations. While this is an expected limitation, that we share with other machine learning methods, it leaves the open question of how one could further unify the flexibility and explainability of the hand-engineered approach with the speed and simplicity of the deep-learned one.

7.2 SHAPE EXAMINATION

We take individual 3D shape examination as our second research focus as, we believe, this task follows immediately and naturally after the exploration of the entire 3D database overview. We focus here on the specification of 3D rotations which is, we argue, the key way in which users examine individual 3D objects in a viewer. We propose in Chapter 5 an addition to traditional trackball methods in the form of a new rotation mechanism that 'latches' automatically along 3D axes inferred from the projection (silhouette) of a 3D shape viewed from a given viewpoint. We propose a way to approximate the computation of such 3D axes from the computation of the corresponding 2D silhouette skeletons, the latter being very fast, robust, precise, and fully automatic. We achieve a good balance between the computation precision of the resulting 3D axes and the computation speed, both which are necessary for having an effective, respectively efficient, rotation tool.

We evaluated our new 3D rotation method using a controlled user study. The results show that, when taken in isolation, our new method cannot surpass the classical virtual trackball. However, when used alongside with virtual trackball, our new method brings in added value. On the conceptual side, our method is, to our knowledge, the first time when 2D image skeletons have been used for 3D interaction tasks. On the limitation side, there are still several aspects in which our method needs improvement: Certain 2D skeletal fragments do generate spurious, or at least, surprising, 3D rotation axes. Related to this, the 3D rotations produced by the 2D skeletons are not always what one would expect and may, as such, require further filtering, constraining, or tuning to become more natural to use. Also, identifying more specific manipulation contexts in terms of operations, applications, and/or shapes where the skeleton based rotation, in combination with virtual trackball and/or other interaction mechanisms, is an interesting direction for future work.

7.3 SHAPE ANALYSIS

Shape analysis is actually reflected in all the earlier chapters of this thesis. Indeed, we employ shape features and shape descriptors for shape exploration of an *entire* shape collection, respectively shape examination of a *single* shape. In Chapter 6, we take the middle road between these two extremes, and analyze a subset of *related* shapes from the perspective of extracting a joint 3D curve skeleton of the entire subset, which we call the co-skeleton. We show how co-skeletons can be extracted by leveraging existing state-of-the-art methods for extracting individual 3D curve skeletons from each shape after which the important parts common to, or representative for, the entire shape-set is generated. Our approach achieves more concise and family-consistent skeletons when compared to traditional per-shape methods, and as such, can be seen first as a more robust replacement of existing curve-skeletonization methods. Separately, our approach has the 'out-of-sample' property within a shape family, to use a term employed several times during this thesis to express generalization capabilities. This allows us to use co-skeletons for the robust treatment of shape-sets in applications such as consistent segmentation.

Yet, our co-skeleton extraction has some limitations. We use, for its computation, the shape's surface descriptors which are similar to the hand-engineered features discussed above in the context of Chapter 3. As such, co-skeletons are also affected by issues such as computational scalability and sensitivity of the respective descriptor extractor to several artifacts in the shape representations such as mesh quality or non-watertight meshes. Additionally, the relationship between surface patches and skeleton curves is not yet fully clear. More importantly, while we leverage, as explained, existing curve-skeletonization methods, it would be conceptually, and also arguably practically, more interesting to extract co-skeletons directly from the 3D shape descriptions.

7.4 FUTURE WORK

Many extensions of our work are possible based on the material presented in this thesis.

First and foremost, our proposed methods for shape exploration, examination, and analysis could be organically merged into a single 3D shape database system together with other traditional methods for shape analysis and processing. The workflow of such a system could start with creating overviews of an entire shape database (Chapters 3 and 4). The user would next zoom-and-pan this overview to find a subset of shapes of interest, which s/he could next examine in detail using the trackball-and-skeleton rotation (Chapter 5). Finally, if desired, various shape analysis methods could be used on such a subset to further process it, *e.g.*, segment the shapes jointly (Chapter 6). The above steps could be repeated several times and from several starting points, as needed.

Separately, it is important to underline other ways in which our three research targets can benefit from each other. For example, the information we get from co-skeletons can be used as features to help build visual overviews of 3D collections. We can use the cluster information and shape features extracted during the creation of overviews to automatically identify collections on which we want to run the co-skeleton analysis. Separately, co-skeletons can help us rotate or align an entire collection of related shapes rather than manipulate each such shape separately.

Another future work direction relates the possibilities for generalization. Since the methods presented in Chapters 3, 4, and 6 are orthogonal to each other, the techniques we use for feature extraction, feature aggregation, skeleton extraction, and data projection could be replaced by other suitable techniques that are better in one desirable aspect, such as speed, simplicity, or robustness, without affecting our proposed pipelines. Separately, since the deep learning approach in Chapter 4 is not constrained to 3D shapes, one could readily envisage its adaptation to handle collections of other media types, such as images, video, or audio data.

Finally, adding more visual explanations to our results can help our target users. All our approaches could benefit from this: Overviews could be enhanced by explaining better why certain shapes are found to be similar, thus placed close to each other, in analogy to related work in dimensionality reduction [100, 141]. Skeleton-based rotations could be enhanced by showing users more rotation possibilities, and ways to control these, before and during the rotation execution. Co-skeletons could be enhanced to show not just the common parts of a family, but the variance or variation therein as well. Last but not least, explaining the operation of the deep learning models used to create 3D shape database overviews is still an open and very actual research problem.

BIBLIOGRAPHY

[1] Aim@Shape. Aim@shape digital shape workbench 5.0, 2019. http://visionair.ge.imati.cnr.it.

[2] I. Alhashim, H. Li, K. Xu, J. Cao, R. Ma, and H. Zhang. Topology-varying 3D shape creation via structural blending. *ACM Trans. Graph.*, 33(4):158:1–158:10, 2014.

[3] N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proc. ACM SMA*, pages 249–266, 2001.

[4] M. Attene, B. Falcidieno, and M. Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *Visual Computer*, 22: 181–193, 2006.

[5] O. K. Au, C. Tai, H. Chu, D. Cohen-Or, and T. Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3):44:1–44:10, 2008.

[6] R. Bade, F. Ritter, and B. Preim. Usability comparison of mouse-based interaction techniques for predictable 3D rotation. In *Proc. Smart Graphics (SG)*, pages 138–150, 2005.

[7] I. Baran and J. Popovic. Automatic rigging and animation of 3D characters. *ACM Trans. Graph.*, 26(3):72, 2007.

[8] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Proc. NIPS*, pages 831–837, 2001.

[9] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE TPAMI*, 24(4):509–522, 2002.

[10] B. Belousov. Difference between two rotation matrices, 2016. http://www.boris-belousov.net/2016/12/01/quat-dist.

[11] M. Ben-Chen and C. Gotsman. Characterizing shape using conformal factors. In *Proc. 3DOR*, pages 1–8, 2008.

[12] L. P. Berg and J. M. Vance. Industry use of virtual reality in product design and manufacturing: a survey. *Virtual Reality*, 21:1–17, 2017.

[13] S. Bian, A. Zheng, E. Chaudhry, L. You, and J. J. Zhang. Automatic generation of dynamic skin deformation for animated characters. *Symmetry*, 10(4), 2018.

[14] H. Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, pages 362–381. MIT Press, 1967.

[15] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[16] M. Bruls, K. Huizing, and J. J. van Wijk. Squarified treemaps. In *Proc. Data Visualization*, pages 33–42, 2000.

[17] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranic. Feature-based similarity search in 3D object databases. *ACM Comput Surv*, 37(4):345–387, 2005.

[18] T. T. Cao, K. Tang, A. Mohamed, and T. S. Tan. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games*, pages 83–90, 2010.

[19] M. Chaouch and A. Verroust-Blondet. Alignment of 3D models. *Graphical Models*, 71(2):63–76, 2009.

[20] S. Chaudhuri, E. Kalogerakis, L. J. Guibas, and V. Koltun. Probabilistic reasoning for assembly-based 3D modeling. *ACM Trans Graph*, 30(4), 2011.

[21] L. Chen, G. Zeng, Q. Zhang, and X. Chen. Tree-lstm guided attention pooling of dcnn for semantic sentence modeling. In *International Conference on 5G for Future Wireless Networks*, pages 52–59. Springer, 2017.

[22] L. Chen, G. Zeng, Q. Zhang, X. Chen, and D. Wu. Question answering over knowledgebase with attention-based lstm networks and knowledge embeddings. In *2017 IEEE 16th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, pages 243–246. IEEE, 2017.

[23] M. Chen, S. Mountford, and A. Sellen. A study in interactive 3D rotation using 2D control devices. *Comput Graph Forum*, 22(4): 121–129, 1998.

[24] S. Chen, L. Zheng, Y. Zhang, Z. Sun, and K. Xu. Veram: View-enhanced recurrent attention model for 3d shape classification. *IEEE transactions on visualization and computer graphics*, 25(12): 3244–3257, 2018.

[25] X. Chen, A. Golovinskiy, and T. Funkhouser. A benchmark for 3D mesh segmentation. *ACM Trans Graph*, 28(3):1–12, 2009.

[26] X. Chen, G. Zeng, Q. Zhang, L. Chen, and Z. Wang. Classification of medical consultation text using mobile agent system based on naïve bayes classifier. In *International Conference on 5G for Future Wireless Networks*, pages 371–384. Springer, 2017.

[27] X. Chen, G. Zeng, Q. Zhang, L. Chen, and D. Wu. A fuzzy ontology for geography knowledge of china's college entrance examination. In *2017 IEEE 16th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, pages 237–242. IEEE, 2017.

[28] X. Chen, G. Zeng, J. Kosinka, and A. Telea. Visual exploration of 3d shape databases via feature selection. In *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP,*, pages 42–53. INSTICC, SciTePress, 2020. ISBN 978-989-758-402-2.

[29] X. Chen., G. Zeng, J. Kosinka, and A. Telea. Scalable visual exploration of 3d shape databases via feature synthesis and selection. In *Communications in Computer and Information Sciences (submitted)*. Springer, 2020.

[30] N. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE TVCG*, 13(3):597–615, 2007.

[31] L. Costa and R. Cesar. *Shape analysis and classification*. CRC Press, 2000.

[32] C. M. Cyr and B. B. Kimia. 3D object recognition using shape similiarity-based aspect graph. In *Proc. IEEE ICCV*, pages 254–261, 2001.

[33] K. Daniels, G. Grinstein, A. Russell, and M. Glidden. Properties of normalized radial visualizations. *Information Visualization*, 11 (4):273–300, 2012.

[34] L. Di Caro, V. Frias-Martinez, and E. Frias-Martinez. Analyzing the role of dimension arrangement for data visualization in radviz. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 125–132. Springer, 2010.

[35] J. Dubinski. When galaxies collide. *Astronomy Now*, 15(8):56–58, 2001.

[36] K. L. Duffin and W. A. Barrett. Spiders: A new user interface for rotation and visualization of N-dimensional point sets. In *Proc. IEEE Visualization*, pages 205–211, 1994.

[37] M. Eitz, R. Richter, T. Boubekeur, K. Hildebrand, and M. Alexa. Sketch-based shape retrieval. *ACM Trans Graph*, 31(4), 2012.

[38] M. Emory and G. Iaccarino. Visualizing turbulence anisotropy in the spatial domain with componentality contours. *Center for Turbulence Research Annual Research Briefs*, pages 123–138, 2014. URL https://web.stanford.edu/group/ctr/ResBriefs/2014/14_emory.pdf.

[39] J. D. *et al.* GRAVITAS: Portraits of a universe in motion, 2006. https://www.cita.utoronto.ca/~dubinski/galaxydynamics/gravitas.html.

[40] D. Engel, L. Hüttenberger, and B. Hamann. A survey of dimension reduction methods for high-dimensional data analysis and visualization. In *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering-Proceedings of IRTG 1131 Workshop 2011*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[41] O. Ersoy, C. Hurter, F. Paulovich, G. Cantareiro, and A. Telea. Skeleton-based edge bundling for graph visualization. *IEEE TVCG*, 17(2):2364 – 2373, 2011.

[42] M. Espadoto, R. Martins, A. Kerren, N. Hirata, and A. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE TVCG*, 27(3):2153–2173, 2019.

[43] M. Espadoto, N. Hirata, and A. Telea. Deep learning multidimensional projections. *J Inf Vis*, 9(3):247–269, 2020.

[44] A. Falcão, J. Stolfi, and R. Lotufo. The image foresting transform: Theory, algorithms, and applications. *IEEE TPAMI*, 26(1):19–29, 2004.

[45] A. Falcao, C. Feng, J. Kustra, and A. Telea. Multiscale 2D medial axes and 3D surface skeletons by the image foresting transform. In P. Saha, G. Borgefors, and G. S. di Baja, editors, *Skeletonization – Theory, Methods, and Applications*. Elsevier, 2017. Ch. 2.

[46] C. Feng, A. C. Jalba, and A. C. Telea. Part-based segmentation by skeleton cut space analysis. In J. A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot, editors, *Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 607–618, Cham, 2015. Springer International Publishing. ISBN 978-3-319-18720-4.

[47] C. Feng, A. C. Jalba, and A. C. Telea. Improved part-based segmentation of voxel shapes by skeleton cut spaces. *Mathematical Morphology –Theory and Applications*, 1(1), 2016.

[48] N. Fish, M. Averkiou, O. van Kaick, O. Sorkine-Hornung, D. Cohen-Or, and N. J. Mitra. Meta-representation of shape families. *ACM Trans. Graph.*, 33(4):34:1–34:11, 2014.

[49] E. Frokjaer, M. Hertzum, and K. Hornbaek. Measuring usability: Are effectiveness, efficiency, and satisfaction really correlated? In *Proc. CHI*, pages 345–352, 2000.

[50] A. Golovinskiy and T. A. Funkhouser. Consistent segmentation of 3D models. *Computers & Graphics*, 33(3):262–269, 2009.

[51] J. Guo, Y. Wang, P. Du, and L. Yu. A novel multi-touch approach for 3D object free manipulation. In *Proc. AniNex*, pages 159–172. Springer, 2017.

[52] K. Guo, D. Zou, and X. Chen. 3D mesh labeling via deep convolutional neural networks. *ACM Trans. Graph.*, 35(1):3:1–3:12, 2015.

[53] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3D point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[54] M. Hancock, S. Carpendale, and A. Cockburn. Shallow-depth 3D interaction: design and evaluation of one-, two- and three-touch techniques. In *Proc. ACM CHI*, pages 1147–1156, 2007.

[55] M. Hancock, T. ten Cate, S. Carpendale, and T. Isenberg. Supporting sandtray therapy on an interactive tabletop. In *Proc. ACM CHI*, pages 2133–2142, 2010.

[56] K. Henriksen, J. Sporring, and K. Hornbaek. Virtual trackballs revisited. *IEEE TVCG*, 10(2):206–216, 2004.

[57] W. H. Hesselink and J. B. T. M. Roerdink. Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. *IEEE TPAMI*, 30(12):2204–2217, 2008.

[58] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proc. ACM SIGGRAPH*, pages 203–212, 2001.

[59] K. Hinckley, J. Tullio, R. Pausch, D. Proffitt, and N. Kassell. Usability analysis of 3D rotation techniques. In *Proc. UIST*, pages 1–10, 1997.

[60] P. Hoffman, G. Grinstein, K. Marx, I. Grosse, and E. Stanley. Dna visual and analytic data mining. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*, pages 437–441. IEEE, 1997.

[61] R. Hu, L. Fan, and L. Liu. Co-segmentation of 3D shapes via subspace clustering. *Comput. Graph. Forum*, 31(5):1703–1713, 2012.

[62] H. C. Huang, Y. Y. Chuang, and C. S. Chen. Affinity aggregation for spectral clustering. In *Computer Vision and Pattern Recognition (CVPR)*, pages 773–780. IEEE, 2012.

[63] J. Hughes, A. van Dam, M. McGuire, D. Sklar, J. Foley, S. Feiner, and K. Akeley. *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 3 edition, 2013.

[64] J. Hultquist. A virtual trackball. In *Graphics Gems*, volume 1, pages 462–463, 1990.

[65] S. Ingram, T. Munzner, and M. Olano. Glimmer: Multilevel mds on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):249–261, 2008.

[66] ITI DB. The informatics & telematics institute database, 2019. `http://3d-search.iti.gr/3DSearch/index.html`.

[67] B. Jackson, T. Lau, D. Schroeder, K. Toussaint, and D. Keefe. A lightweight tangible 3D interface for interactive visualization of thin fiber structures. *IEEE TVCG*, 19(12):2802–2809, 2013.

[68] I. Jacob and J. Oliver. Evaluation of techniques for specifying 3D rotations with a 2D input device. In *Proc. HCI*, pages 63–76, 1995.

[69] A. Jalba, J. Kustra, and A. Telea. Surface and curve skeletonization of large 3D models on the GPU. *IEEE TPAMI*, 35(6):1495–1508, 2013.

[70] A. Jalba, J. Kustra, and A. Telea. Computing surface and curve skeletons from large meshes on the GPU. *IEEE TPAMI*, 35(6):783–799, 2013.

[71] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011.

[72] O. van Kaick, K. Xu, H. Zhang, Y. Wang, S. Sun, A. Shamir, and D. Cohen-Or. Co-hierarchical analysis of shape structures. *ACM Trans. Graph.*, 32(4):69:1–69:10, 2013.

[73] E. Kain. The 20 best-selling video games of 2019. *Forbes*, January 2019.

[74] E. Kalogerakis, A. Hertzmann, and K. Singh. Learning 3D mesh segmentation and labeling. *ACM Trans. Graph.*, 29(4):102:1–102:12, 2010.

[75] E. Kandogan. Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions. In *Proceedings of the IEEE Information Visualization Symposium*, volume 650, page 22. Citeseer, 2000.

[76] E. Kandogan. Visualizing multi-dimensional clusters, trends, and outliers using star coordinates. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 107–116, 2001.

[77] D. Kaye and I. Ivrissimtzis. Mesh alignment using grid based PCA. In *Proc. CGTA*, pages 174–181, 2015.

[78] V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. A. Funkhouser. Learning part-based templates from large collections of 3D shapes. *ACM Trans. Graph.*, 32(4):70:1–70:12, 2013.

[79] C. Kirbas and F. Quek. Vessel extraction techniques and algorithms: a survey. In *Proc. Third IEEE Symposium on Bioinformatics and Bioengineering*, 2003.

[80] J. Kustra, A. Jalba, and A. Telea. Probabilistic view-based curve skeleton computation on the GPU. In *Proc. VISAPP*. SciTePress, 2013.

[81] J. Kustra, A. Jalba, and A. Telea. Robust segmentation of multiple intersecting manifolds from unoriented noisy point clouds. *Comp Graph Forum*, 33(4):73–87, 2014.

[82] H. Laga, M. Mortara, and M. Spagnuolo. Geometry and context for semantic correspondences and functionality recognition in man-made 3D shapes. *ACM Trans. Graph.*, 32(5):150:1–150:16, 2013.

[83] H. Laga, Y. Guo, H. Tabia, R. B. Fisher, and M. Bennamoun. *3D Shape Analysis: Fundamentals, Theory, and Applications*. Wiley, 2019.

[84] G. Lavoué, J. P. Vandeborre, H. Benhabiles, M. Daoudi, K. Huebner, M. Mortara, and M. Spagnuolo. SHREC'12 track: 3D mesh segmentation. In *Proc. 3DOR*, 2012.

[85] B. H. Le and Z. Deng. Robust and accurate skeletal rigging from mesh sequences. *ACM Trans. Graph.*, 33(4):84:1–84:10, 2014.

[86] Y. Lee, S. Lee, A. Shamir, and D. Cohen-Or. Intelligent mesh scissoring using 3D snakes. In *Proc. IEEE Pacific Graphics*, pages 279–287, 2004.

[87] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H. P. Seidel. Mesh scissoring with minima rule and part salience. *CAGD*, 22:444–465, 2005.

[88] D. J. Lehmann and H. Theisel. Orthographic star coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 19(12): 2615–2624, 2013.

[89] D. J. Lehmann and H. Theisel. Optimal sets of projections of high-dimensional data. *IEEE transactions on visualization and computer graphics*, 22(1):609–618, 2015.

[90] B. Li, Y. Lu, C. Lib, A. Godil, T. Schreck, M. Aono, M. Burtscher, Q. Chen, N. K. C. amd Bin Fang, H. Fu, T. Furuya, H. Li, J. Liu, H. Johan, R. Kosaka, H. Koyanagi, R. Ohbuchi, and C. Zou. A comparison of 3D shape retrieval methods based on a large-scale benchmark supporting multimodal queries. *CVIU*, 131:1–27, 2015.

[91] X. Li, T. W. Woon, T. S. Tan, and Z. Huang. Decomposing polygon meshes for interactive applications. In *Proc. ACM SI3D*, pages 35–42, 2001.

[92] J. Lien, J. Keyser, and N. Amato. Simultaneous shape decomposition and skeletonization. In *Proc. ACM SPM*, pages 219–228, 2006.

[93] P. Liu, F. Wu, W. Ma, R. Liang, and M. Ouhyoung. Automatic animation skeleton construction using repulsive force field. In *Proc. IEEE Pacific Graphics*, pages 409–413, 2003.

[94] R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Proc. IEEE Pacific Graphics*, pages 298–305, 2004.

[95] M. Livesu, F. Guggeri, and R. Scateni. Reconstructing the curve-skeletons of 3D shapes using the visual hull. *IEEE TVCG*, 18(11): 1891–1901, 2012.

[96] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[97] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[98] A. Manzanera, T. M. Bernard, F. J. Prêteux, and B. Longuet. Medial faces from a concise 3D thinning algorithm. In *Proc. ICCV*, pages 337–343, 1999.

[99] R. Martins, D. Coimbra, R. Minghim, and A. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014.

[100] R. M. Martins, R. Minghim, A. C. Telea, et al. Explaining neighborhood preservation for multidimensional projections. In *CGVC*, pages 7–14, 2015.

[101] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[102] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv:1802.03426*, 2018.

[103] A. Meijster, J. Roerdink, and W. Hesselink. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 331–340. Springer, 2002.

[104] NASA. Nasa 3D resources, 2019. https://nasa3d.arc.nasa.gov.

[105] T. S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879, 2006.

[106] L. Nonato and M. Aupetit. Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG*, 2018. DOI:10.1109/TVCG.2018.2846735.

[107] L. Nováková and O. Štepánková. Multidimensional clusters in radviz. In *Proceedings of the 6th WSEAS International Conference on Simulation, Modelling and Optimization*, pages 470–475, 2006.

[108] R. L. Ogniewicz and O. Kübler. Hierarchic voronoi skeletons. *Pattern recognition*, 28(3):343–359, 1995.

[109] T. Partala. Controlling a single 3D object: Viewpoint metaphors, speed, and subjective satisfaction. In *Proc. INTERACT*, pages 536–543, 1999.

[110] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[111] F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG*, 14(3):564–575, 2008.

[112] F. V. Paulovich, C. T. Silva, and L. G. Nonato. Two-phase mapping for projecting massive data sets. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1281–1290, 2010.

[113] G. Peyre and L. Cohen. Geodesic computations for fast and accurate surface remeshing and parameterization. *Progress in Nonlinear Differential Equations and Their Applications*, 63:151–171, 2005.

[114] N. Pezzotti, B. P. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable t-SNE for progressive visual analytics. *IEEE TVCG*, 23(7):1739–1752, 2017.

[115] M. Potmesil. Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, 1987.

[116] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[117] P. E. Rauber, R. R. O. da Silva, S. Feringa, M. E. Celebi, A. X. Falcão, and A. C. Telea. Interactive image feature selection aided by dimensionality reduction. In *Proc. EuroVA*, pages 19–23, 2015.

[118] D. Reniers, J. J. van Wijk, and A. Telea. Computing multiscale skeletons of genus 0 objects using a global importance measure. *IEEE TVCG*, 14(2):355–368, 2008.

[119] D. Reniers and A. Telea. Skeleton-based hierarchical shape segmentation. In *Proc. SMI*, pages 179–188, 2007.

[120] D. Reniers and A. Telea. Hierarchical part-type segmentation using voxel-based curve skeletons. *Visual Computer*, 24:383–395, 2008.

[121] D. Reniers and A. Telea. Part-type segmentation of articulated voxel shapes using the junction rule. *CGF*, 27(3):1845–1852, 2008.

[122] A. A. G. Requicha and J. R. Rossignac. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, 12(5):31–44, 1992. DOI 10.1109/38.156011.

[123] R. S. V. Rodrigues, J. F. M. Morgado, and A. J. P. Gomes. Part-based mesh segmentation: A survey. *Comp Graph Forum*, 37(6):235–274, 2018.

[124] A. Rosenfeld and J. Pfaltz. Distance functions in digital pictures. *Patt Recogn*, 1:33–61, 1968.

[125] J. Rossignac. Interactive exploration of distributed 3D databases over the internet. In *Proc. IEEE CGI*, 1998.

[126] R. Rostami, F. S. Bashiri, B. Rostami, and Z. Yu. A survey on data?driven 3D shape descriptors. *Computer Graphics Forum*, 38 (1):356–393, 2019.

[127] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Intl J Computer Vision*, 40(2):99–121, 2000.

[128] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, 2000.

[129] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 3212–3217, 2009.

[130] ShapeNet. ShapeNet online repository. https://www.shapenet.org, 2019.

[131] L. Shapira, A. Shamir, and D. Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249–262, 2008.

[132] L. Shapira, S. Shalom, A. Shamir, D. Cohen-Or, and H. Zhang. Contextual part analogies in 3D objects. *Intl J on Comp Vision*, 89(1-2):309–326, 2010.

[133] Y. T. Shen, D. Y. Chen, X. P. Tian, and M. Ouhyoung. 3d model search engine based on lightfield descriptors. In *Eurographics 2003 - Posters*. Eurographics Association, 2003. DOI 10.2312/egp.20031031.

[134] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The Princeton shape benchmark. In *Proc. SMI*, pages 167–178, 2004. http://shape.cs.princeton.edu/benchmark.

[135] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. IEEE Symp. on Visual Languages*, pages 336–343, 1996.

[136] K. Shoemake. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Proc. Graphics Interface*, 1992.

[137] SHREC Committee. SHREC: Shape retrieval challenge, 2021. https://www.shrec.net.

[138] E. Shtrom, G. Leifman, and A. Tal. Saliency detection in large point sets. In *Proc. IEEE ICCV*, pages 3591–3598, 2013.

[139] K. Siddiqi and S. Pizer. *Medial Representations: Mathematics, Algorithms and Applications.* Springer, 2008.

[140] O. Sidi, O. van Kaick, Y. Kleiman, H. Zhang, and D. Cohen-Or. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM Trans Graph*, 30(6):126:1–126:9, 2011.

[141] R. R. O. D. Silva, P. E. Rauber, R. M. Martins, R. Minghim, and A. C. Telea. Attribute-based visual explanation of multidimensional projections. In *Eurovis Workshop on Visual Analytics*, 2015.

[142] R. R. da Silva, P. E. Rauber, and A. C. Telea. Beyond the third dimension: Visualizing high-dimensional data with projections. *Computing in Science & Engineering*, 18(5):98–107, 2016.

[143] A. Sobiecki, H. Yasan, A. Jalba, and A. Telea. Qualitative comparison of contraction-based curve skeletonization methods. In *Proc. ISMM.* Springer, 2013.

[144] A. Sobiecki, A. Jalba, and A. Telea. Comparison of curve and surface skeletonization methods for voxel shapes. *Patt Rec Lett*, 47:147–156, 2014.

[145] C. O. S. Sorzano, J. Vargas, and A. P. Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014.

[146] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[147] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proc. IEEE ICCV*, pages 945–953, 2015.

[148] A. Tagliasacchi, I. Alhashim, M. Olson, and H. Zhang. Mean curvature skeletons. *Comp. Graph. Forum*, 31(5):1735–1744, 2012.

[149] A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, and A. Telea. 3D skeletons: A state-of-the-art report. *Comp Graph Forum*, 35(2):573–597, 2016.

[150] J. Tangelder and R. Veltkamp. A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications*, 39(3):441–471, 2008.

[151] F. Tasse, J. Kosinka, and N. Dodgson. Cluster-based point set saliency. In *Proc. IEEE ICCV*, pages 163–171, 2015.

[152] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 902–907, 1995.

[153] A. Telea. Feature preserving smoothing of shapes using saliency skeletons. In *Proc. VMLS*, pages 136–148. Springer, 2011.

[154] A. Telea. Source code for salience skeleton computation, 2014. https://webspace.science.uu.nl/$\sim$telea001/Shapes/Salience.

[155] A. Telea. *Data Visualization: Principles and Practice*. CRC Press, 2 edition, 2014.

[156] A. Telea. Real-time 2D skeletonization using CUDA, 2019. http://www.staff.science.uu.nl/$\sim$telea001/Shapes/CUDASkel.

[157] A. Telea and A. Jalba. Voxel-based assessment of printability of 3D shapes. In *Proc. ISMM*. Springer, 2011.

[158] A. Telea and J. J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym*, pages 251–259. Springer, 2002.

[159] The Authors. Source code and videos of interactive skeleton-based axis rotation, 2019. http://www.staff.science.uu.nl/$\sim$telea001/Shapes/CUDASkelInteract.

[160] J. Tierny, J. Vandeborre, and M. Daoudi. Topology driven 3D mesh hierarchical segmentation. In *Proc. SMI*, pages 215–220, 2007.

[161] S. P. Tiwari, F. Tama, and O. Miyashita. Searching for 3D structural models from a library of biological shapes using a few 2D experimental images. *BMC Bioinformatics*, 19(320), 2018.

[162] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

[163] J. Tukey and P. Tukey. Computer graphics and exploratory data analysis: An introduction. In *The Collected Works of John W. Tukey: Graphics: 1965-1985*, 1988.

[164] I. TurboSquid. Turbosquid shape repository, 2019. https://www.turbosquid.com.

[165] R. Veltkamp and M. Tanase. Content-based image and video retrieval. In *A Survey of Content-Based Image Retrieval Systems (Ch. 5)*, pages 47–101. Kluwer, 2002.

[166] V. Verma and J. Snoeyink. Reducing the memory required to find a geodesic shortest path on a large mesh. In *Proc. ACM GIS*, pages 227–235, 2009.

[167] P. S. Wang, Y. Liu, Y. X. Guo, C. Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[168] P. Wang, Y. Gan, P. Shui, F. Yu, Y. Zhang, S. Chen, and Z. Sun. 3d shape segmentation via shape fully convolutional networks. *Comput. Graph.*, 70:128–139, 2018.

[169] Y. Wang, S. Asafi, O. van Kaick, H. Zhang, D. Cohen-Or, and B. Chen. Active co-analysis of a set of shapes. *ACM ToG*, 31 (6):165:1–10, 2012.

[170] M. Wattenberg. How to use t-SNE effectively, 2016. https://distill.pub/2016/misread-tsne.

[171] L. Wilkinson, A. Anand, and R. Grossman. High-dimensional visual analytics: Interactive exploration guided by pairwise views of point distributions. *IEEE TVCG*, 12(6):1363–1372, 2006.

[172] H. Y. Wu. *3D Visual Shape Processing based on Discrete Differential Geometry*. Lambert Academic Publishing, 2016.

[173] Z. Wu, M. Zeng, F. Qin, Y. Wang, and J. Kosinka. Active 3-d shape cosegmentation with graph convolutional networks. *IEEE Computer Graphics and Applications*, 39(2):77–88, 2019.

[174] Z. Wu, Y. Wang, R. Shou, B. Chen, and X. Liu. Unsupervised co-segmentation of 3D shapes via affinity aggregation spectral clustering. *Computers & Graphics*, 37(6):628–637, 2013.

[175] Z. Wu, R. Shou, Y. Wang, and X. Liu. Interactive shape co-segmentation via label propagation. *Computers & Graphics*, 38: 248–254, 2014.

[176] Z. Wu, X. Chen, L. Yu, A. Telea, and J. Kosinka. Co-skeletons: Consistent curve skeletons for shape families. *Computers & Graphics*, 90:62–72, 2020.

[177] K. Xu, H. Zhang, D. Cohen-Or, and B. Chen. Fit and diverse: set evolution for inspiring 3D shape galleries. *ACM ToG*, 31(4):57:1–57:10, 2012.

[178] H. Yan, S. Hu, R. R. Martin, and Y. Yang. Shape deformation using a skeleton to drive simplex transformations. *IEEE TVCG*, 14(3): 693–706, 2008.

[179] M. Yavartanoo, E. Y. Kim, and K. M. Lee. Spnet: Deep 3D object classification and retrieval using stereographic projection. In *Asian Conference on Computer Vision*, pages 691–706. Springer, 2018.

[180] H. You, Y. Feng, R. Ji, and Y. Gao. Pvnet: A joint convolutional network of point cloud and multi-view for 3d shape recognition. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1310–1318, 2018.

[181] L. Yu and T. Isenberg. Exploring one- and two-touch interaction for 3D scientific visualization spaces. *Posters of Interactive Tabletops and Surfaces*, Nov. 2009.

[182] L. Yu, P. Svetachov, P. Isenberg, M. H. Everts, and T. Isenberg. FI3D: Direct-touch interaction for the exploration of 3D scientific visualization spaces. *IEEE TVCG*, 16(6):1613–1622, 2010.

[183] M. E. Yümer and L. B. Kara. Co-abstraction of shape collections. *ACM Trans. Graph.*, 31(6):166:1–166:11, 2012.

[184] M. E. Yümer and L. B. Kara. Co-constrained handles for deformation in shape collections. *ACM Trans. Graph.*, 33(6):187:1–187:11, 2014.

[185] X. Zhai, X. Chen, L. Yu, and A. Telea. Scalable visual exploration of 3d shape databases via feature synthesis and selection. In *Communications in Computer and Information Sciences (submitted)*. Springer, 2020.

[186] X. Zhai, X. Chen, L. Yu, and A. Telea. Interactive axis-based 3d rotation specification using image skeletons. In *VISIGRAPP (1: GRAPP)*, pages 169–178, 2020.

[187] Y. J. Zhao, D. Shuralyov, and W. Stuerzlinger. Comparison of multiple 3D rotation methods. In *Proc. IEEE VECIMS*, pages 19–23, 2011.

[188] N. N. Zhou and Y. L. Deng. Virtual reality: A state-of-the-art survey. *International Journal of Automation and Computing*, 6:319–325, 2009.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my mentor Prof. dr. A.C. Telea, who led me to the amazing field of information visualization. His vision, sincerity, and creativity have deeply inspired me. I would like to express my special appreciation to my mentor Prof. dr. J. Kosinka, who helped me build a solid foundation on 3D graphics and helped me a lot in my research. Thanks to my mentor Prof. dr. G. Zeng. He encouraged me to apply for a scholarship for studying abroad. Thanks to the members/alumni of the SVCG research group, Dr. Zizhao Wu, Dr. Lingyun Yu, Gerben Hettinga, and Xiaorui Zhai. They helped me a lot and gave me advice during these years.

Finally, I must show my gratitude to my wife Zhuolin Wang. I want to thank you for always being there for me. I love you. Will you spend the rest of your life with me?

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LYX:

http://code.google.com/p/classicthesis/

Final Version as of April 15, 2021 (`classicthesis`).