EUROGRAPHICS 2018
K. Hildebrandt and C. Theobalt
(Guest Editors)

Volume 37 (2018), Number 2
STAR – State of The Art Report

A Survey on Data-driven Dictionary-based Methods for 3D Modeling

Thibault Lescoat[1] Maks Ovsjanikov[2] Pooran Memari[2,3] Jean-Marc Thiery[1] Tamy Boubekeur[1]

[1] LTCI, Télécom Paristech, Université Paris Saclay
[2] LIX, École Polytechnique, Université Paris Saclay
[3] CNRS

Abstract

Dictionaries are very useful objects for data analysis, as they enable a compact representation of large sets of objects through the combination of atoms. Dictionary-based techniques have also particularly benefited from the recent advances in machine learning, which has allowed for data-driven algorithms to take advantage of the redundancy in the input dataset and discover relations between objects without human supervision or hard-coded rules. Despite the success of dictionary-based techniques on a wide range of tasks in geometric modeling and geometry processing, the literature is missing a principled state-of-the-art of the current knowledge in this field. To fill this gap, we provide in this survey an overview of data-driven dictionary-based methods in geometric modeling. We structure our discussion by application domain: surface reconstruction, compression, and synthesis. Contrary to previous surveys, we place special emphasis on dictionary-based methods suitable for 3D data synthesis, with applications in geometric modeling and design. Our ultimate goal is to enlight the fact that these techniques can be used to combine the data-driven paradigm with design intent to synthesize new plausible objects with minimal human intervention. This is the main motivation to restrict the scope of the present survey to techniques handling point clouds and meshes, making use of dictionaries whose definition depends on the input data, and enabling shape reconstruction or synthesis through the combination of atoms.

CCS Concepts

•***Computing methodologies*** → *Shape modeling; Mesh models; Mesh geometry models; Point-based models; Shape analysis;*

1. Introduction

The recent availability of large datasets of 3D objects has inspired a new generation of algorithms that leverage the knowledge derived from the whole dataset in order to address fundamental problems, in fields such as correspondence [HWG14], segmentation [XXLX14, XSX*14], recognition [AME*14], surface reconstruction [XZZ*14, SFCH12], synthesis [KCKK12], modeling [YK14], or exploration [OLGM11]. These methods are able to learn the underlying computational models allowing representing and processing individual shapes or families of shapes, without relying on hard-coded or user-provided rules. However, the size and complexity of the input datasets affect both the memory and computational costs of algorithms, making the dataset representation and storage a particularly sensitive aspect.

In geometry processing and geometric modeling, data-driven methods have long been restricted to example-based methods, which, given an exemplar and a target object in a dataset, transfer information from the former to the latter. While achieving interesting results, these methods typically do not take advantage of the entire object collection. With the advent of large repositories of 3D shapes, novel methods have been proposed with the primary aim of extracting shapes or sub-shapes from the input and combine or blend them in order to form new models, giving the opportunity to help artists in designing new objects [CK10, FKS*04], to enrich an existing set [KCKK12] or to support geometry processing tasks such as shape reconstruction of scanned objects [PMG*05]. In particular, co-analysis of shape families allows for high-level shape understanding by finding correlations between multiple objects, which facilitates generating similar shapes using the learned generative grammar, as demonstrated by Talton et al. [TYK*12]. We refer the interested reader to the survey of Xu et al. [XKHK15] for further information on data-driven methods in geometry processing.

A key issue for many data-driven techniques is that the size of the data can be computationally prohibitive. For example, ShapeNet [CFG*15] contains more than 60,000 models and requires more than 100 gigabytes of storage. In some scenarios, it is important to use a concise representation of the dataset, and dictionaries make that possible by exploiting the fact that large datasets are made of numerous similar shapes that can be reduced to a limited

number of "atoms". The exact definition of an "atom" highly depends on the type of the dictionary considered. For example, it is possible to segment shapes and use the resulting segments as the atoms of a dictionary. In linear algebra and analysis, combining atoms (or words) of a dictionary is in general not obtained through simple concatenation or placement, but instead by using a (possibly linear) weighting; simple and famous examples are the Fourier series, or the canonical basis $(\vec{x},\vec{y},\vec{z})$ of $\mathbb{R}^3$ for 3D shapes. In signal processing, atoms in dictionaries were often restricted to orthogonal bases at first, for ease of use and signal decomposition, but this orthogonality proved too limiting for expressiveness and overcomplete dictionaries – meaning that some atoms can be expressed as combinations of other atoms – became increasingly popular. We refer to Rubinstein et al. [RBE10] for a survey retracing the history of dictionaries in signal processing. Dictionaries can also be very useful to reduce computations, as one can focus on processing a few atoms instead of operating on the entire input dataset.

While dictionaries are inherently sparse, they are not the only sparse representation one can use in shape processing (other possibilities include implicit surfaces or low-rank matrices, for example). Covering all sparse representations is out of the scope of this article, and we refer to Xu et al. [XWZ*15] for a survey on sparsity in geometric modeling.

Motivation. Many existing methods build and use dictionaries for shape modeling and analysis. Although there are surveys on data-driven techniques [XKHK15], there has been little treatment in shape analysis specifically dedicated to dictionary-based methods. One of the strengths of dictionary-based approaches is that they allow both fitting input data to the atoms in the dictionary (e.g., for shape reconstruction) and synthesizing novel shape instances for shape modeling by recombining atoms. Moreover, dictionary-based methods are versatile in the sense that the atoms and the data do not have to be of the same type (triangle meshes vs. point clouds vs. parametric planar patches). As an example, this property is used in the surface reconstruction method of Xiong et al. [XZZ*14], where the atoms are 3D points while the dictionary is coding an entire mesh. These techniques are also often used as sub-parts of other methods, to analyze shape collections for instance. We therefore claim that it is useful and important to consider dictionaries as objects of study in their own right. To this end we present an overview of recent techniques for building and using dictionary-based representations in computer graphics and in particular, geometry processing and modeling.

Scope. We restrict the scope of this survey to techniques handling point clouds and meshes, making use of dictionaries whose definition depends on the input data (data driven dictionaries), and explicitly enabling shape reconstruction or synthesis from the dictionary through the combination of atoms in a new point cloud or new mesh – for the latter either by composing parts, interpolating linearly or stitching atoms together. An important aspect of dictionary-based analysis and modeling is to be able to encode second-order constraints, which characterize how the atoms fit together, in addition to what the atoms actually are. We discuss ways of building and analyzing such second-order relations, via statistical models such as Bayesian networks or inferred grammars for example. Our goal is to describe the versatility and efficiency of dictionary-based techniques in shape modeling and analysis, but also their latent and unexploited potential, and to point out some promising future directions of research.

Structure. First, we will define the scope more precisely and will describe the various concepts used throughout this article in Section 2; this section also includes a comparative table of the presented methods. We will then present the relevant articles grouped by their application domain, starting with surface reconstruction (Section 3), followed by compression (Section 4), and then shape synthesis and modeling (Section 5). We conclude with a discussion of the presented methods in Section 6.

2. Preliminaries

2.1. Data-drivenness

Data-driven methods are motivated by the fact that the study of a whole set often brings more understanding than the study of the objects taken independently. The aim is to learn a representation from the dataset enabling complex and precise processing on large sets of shapes. Before the advent of data-driven techniques, such a goal was achieved with knowledge-driven methods, in which patterns are extracted using complex hard-coded rules that cope poorly with the large structural variability of big datasets. Knowledge-driven methods (such as the surface approximation of Xu et al. [XWY*16], using a dictionary of user-defined polynomials) are out of the scope of this survey. We refer to Xu et al. [XKHK15] for a survey on data-driven methods in modeling geometry in general.

2.2. Dictionaries

Definition. The term "dictionary" is very broad and sometimes not precisely defined. We refer to a dictionary as a set of elements, called atoms, equipped with a combination operator between elements in order to create new elements. The set of elements is usually overcomplete, meaning that there are more atoms than necessary in order to generate a given combination. Formally, we define a dictionary as:

$$dictionary = \{\{atoms...\}, combination\}$$
$$combination: \quad coefficients \rightarrow object$$

The choice of atoms, coefficients and combination operator depends on the application. For example, in geometric modeling atoms can be meshes in perfect correspondence, with a linear interpolation as combination operator, or entire shapes, with the union as combination operator in order to compose entire scenes, whereas, in some applications of language processing, atoms can be words and the combination be a concatenation operator. Hence we try to characterize what types of atoms and combinations are considered at each stage of this survey. Let us note that this abstract notion of dictionary is similar to the definition of *structure* in the survey of Mitra et al. [MWZ*13] on structure-aware shape processing. Our focus is complementary: their *structure* reflects the relations between distinct parts of shapes while we consider

techniques that build and use *dictionaries* derived from shape collections, and that do not necessarily use (distinct) parts.

From dictionaries to shapes. In most of the approaches covered in this survey, atoms will be shapes or shape parts. However, for a given type of atoms, the combination method can vary greatly. A conceptually simple combination method is a linear interpolation of atoms, as it is often done in *3D morphing*, where the vertex positions of different poses are interpolated following the (potentially animated) coefficients given by the user, which describe the amount of influence of each pose in the output. Note that such a combination require perfect correspondence between meshes. Some methods use more advanced interpolations involving the minimization of a stretch energy which takes into account some constraints that a simple linear interpolation method would ignore, like muscle deformations at the elbow when going from a straight arm to a folded arm [GCLX16]. Another combination operator example is concatenation: here, the atoms of the dictionary are thought as different components of a shape that are put together when constructing a new element. The coefficients are not just real number in this case, but more complex structure encoding which parts are present, as done by Kalogerakis et al. [KCKK12] for example. It is also possible to blend different meshes using dictionaries, and in this case the coefficients will be spatial functions that control which region of which mesh to stitch into the final one.

In this survey, the effectiveness of dictionary-based modeling relies on the ability to reconstruct shapes from a dictionary and a set of coefficients, which can have various forms, from simple real numbers to user sketches. This excludes methods such as bag-of-words methods (or bag-of-features) from data-driven dictionary-based approaches, notably used in object retrieval from databases (as in Shape Google [BBGO11], or the shape retrieval technique of Lavoué [Lav11]). To the best of our knowledge, generative applications using bag-of-words, such as PhotoSketcher [ERH*11], are only present in the image processing field. Similarly, while one could build a dictionary from the result of segmentation or symmetry detection techniques, such as the partial symmetry detection of Mitra et al. [MGP06], most of these methods do not focus on shape modeling and hence are not included in this survey. We refer the reader to the survey on symmetry of Mitra et al. [MPWC13] for more information.

2.3. Dictionary learning

Obtaining a dictionary can be done in several ways: it can be provided explicitly, in which case it is not data-driven (and hence methods using such dictionaries will not be discussed here); it can be the input dataset itself, or it can be learned from the dataset. Atoms as well as the combination operator and coefficients can sometimes be learned. Dictionary-learning is a part of machine-learning and a large body of algorithms and techniques have been developed for this matter. Before describing the classification of the survey, we will introduce some of these algorithms, focusing on the most fundamental ones shared by many techniques presented in this survey.

PCA. The simplest and most commonly-used algorithm is Principal Component Analysis: given a set of vectors as input (all of the same dimension), it outputs an orthogonal basis: the basis vectors (also called components) are the directions of (progressively) least-variance of the data. This algorithm computes both the atoms – the basis vectors – and the coefficients, and is broadly used for dimensionality reduction. Atoms are combined using a linear combination, i.e. by multiplying the basis matrix with a vector of coefficients.

A limitation of PCA is that the coefficients that it produces can involve many basis vectors. In practice one is often interested in learning a *sparse* representation, that allows to express the data using as few as possible coefficients. In general, finding a dictionary with a sparse representation is a NP-hard problem [Vav09, Til15], so algorithms aim at finding a good approximation in an acceptable time. Below we review some commonly used approaches.

K-SVD. Introduced by Aharon, Elad and Bruckstein [AEB06], K-SVD is a generalization of the k-means algorithm. It represents a dictionary of K atoms $\{\boldsymbol{d}_j\}$ in $\mathbb{R}^n$ by a $n \times K$ matrix D. With $\boldsymbol{y} \in \mathbb{R}^n$ a signal, its decomposition is represented as a coefficient vector $\boldsymbol{c} \in \mathbb{R}^K$, such that $\boldsymbol{y} = D\boldsymbol{c}$ (or as close as possible). This is achieved by optimizing:

$$\min_{D,C} \|Y - DC\|_F^2 \qquad \text{subject to} \quad \forall i\,, \|\boldsymbol{c}_i\|_0 \leq T_0$$

where Y is the matrix of signals $\{\boldsymbol{y}_i\}$, C the matrix of coefficients (so we have $\boldsymbol{y}_i = D\boldsymbol{c}_i$), F is the Frobenius norm, T_0 is a constant and $\|.\|_0$ is the number of non-zero coefficients. The constraint enforces sparsity of the decomposition onto the output dictionary. This optimization is performed in two main steps: first, D is fixed and the best (approximate) coefficients C are found, using an algorithm like Orthogonal Matching Pursuit (described below); second, C is fixed, and D is updated, one atom at a time in a greedy fashion, the other atoms being fixed. This is done by minimizing for every k:

$$\|Y - DC\|_F^2 = \left\| \left(Y - \sum_{j \neq k} \boldsymbol{d}_j \boldsymbol{c}'_j \right) - \boldsymbol{d}_k \boldsymbol{c}'_k \right\|_F^2 = \|E_k - \boldsymbol{d}_k \boldsymbol{c}'_k\|_F^2$$

where $\boldsymbol{c}'_j$ is the j-th row of C. This step is commonly solved using Singular Value Decomposition – hence the name of the algorithm – and the solution to this new minimization is the updated column vector. All of this is done iteratively until sufficient convergence, i.e., $\|Y - DC\| \leq \varepsilon$. Note however, that the global optimum is not guaranteed to be found.

Orthogonal Matching Pursuit (OMP) is an extension of Matching Pursuit, which was introduced by Pati et al. [PRK93]. From an input dictionary and coefficients, it aims at producing better coefficients to favor sparsity, keeping the dictionary constant. The optimization is cast as decreasing iteratively the residual by greedily considering new atoms onto which the signal is decomposed. Given a dictionary $D \in \mathbb{R}^{n \times m}$ a vector $\boldsymbol{f} \in \mathbb{R}^n$, and a greediness factor $\lambda \in [0, 1]$, the initialization is as follows:

$D_0 = \varnothing$ already processed atoms
$\boldsymbol{R}_0 = \boldsymbol{f}$ residual of $\boldsymbol{f}$ still to be approximated

At iteration k (starting at 0), the following steps are performed:

1. find l so that $\boldsymbol{d}_l \in D \setminus D_k$ and $|\langle \boldsymbol{R}_k, \boldsymbol{d}_l \rangle| \geq \lambda \sup_{\boldsymbol{d} \in D \setminus D_k} |\langle \boldsymbol{R}_k, \boldsymbol{d} \rangle|$
2. if this product is lower than a given threshold, stop,
3. permute columns $k+1$ and l of the dictionary D,
4. compute the set of coefficients $\{b_j^k\}_{j=1}^k$ such that

$$\boldsymbol{d}_{k+1} = \sum_{j=1}^{k} b_j^k \boldsymbol{d}_j + \boldsymbol{\gamma}_k \qquad \text{with } \langle \boldsymbol{d}_j, \boldsymbol{\gamma}_k \rangle = 0,\ \forall j \in [1..k]$$

5. with $\alpha_k = \langle \boldsymbol{R}_k, \boldsymbol{d}_{k+1} \rangle \|\boldsymbol{\gamma}_k\|^{-2}$, compute $\{c_j^{k+1}\}_{j=1}^{k+1}$:

$$\forall j \in [1..k],\ c_j^{k+1} = c_j^k - \alpha_k b_j^k \quad ; \quad c_{k+1}^{k+1} = \alpha_k$$

6. update the model:

$$\begin{aligned} \boldsymbol{f}_{k+1} &= \sum_{j=1}^{k+1} c_j^{k+1} \boldsymbol{d}_j \\ \boldsymbol{R}_{k+1} &= \boldsymbol{f} - \boldsymbol{f}_{k+1} \\ D_{k+1} &= D_k \bigcup \{\boldsymbol{d}_{k+1}\} \end{aligned}$$

The last coefficients found are the result of the algorithm, decomposing $\boldsymbol{f}$ on D, with the guarantee that no more than m iterations will be needed.

Alternating direction method of multipliers (ADMM) is an optimization algorithm that is widely used in machine learning and was proposed by Glowinski and Marrocco [GM75] and Gabay and Mercier [GM76]. It regained interest more recently after the related publication of Boyd et al. [BPC*11], and is also used to learn dictionaries. Given 2 functions $f, g : \mathbb{R}^n \to \mathbb{R}$, we iteratively solve the following problem:

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) + g(\boldsymbol{y}) \\ &\text{subject to} \quad A\boldsymbol{x} + B\boldsymbol{y} = \boldsymbol{c} \end{aligned}$$

The augmented Lagrangian is defined as

$$L_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = f(\boldsymbol{x}) + g(\boldsymbol{y}) + \boldsymbol{z}^\top (A\boldsymbol{x} + B\boldsymbol{y} - \boldsymbol{c}) + \frac{\rho}{2} \|A\boldsymbol{x} + B\boldsymbol{y} - \boldsymbol{c}\|_2^2$$

and each iteration is composed of three steps:

$$\begin{aligned} &\text{Minimize } \boldsymbol{x}: && \boldsymbol{x}_{k+1} = \operatorname*{argmin}_{\boldsymbol{x}} L_\rho(\boldsymbol{x}, \boldsymbol{y}_k, \boldsymbol{z}_k) \\ &\text{Minimize } \boldsymbol{y}: && \boldsymbol{y}_{k+1} = \operatorname*{argmin}_{\boldsymbol{y}} L_\rho(\boldsymbol{x}_{k+1}, \boldsymbol{y}, \boldsymbol{z}_k) \\ &\text{Dual update:} && \boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \rho(A\boldsymbol{x}_{k+1} + B\boldsymbol{y}_{k+1} - \boldsymbol{c}) \end{aligned}$$

This method is an extension of the Method of Multipliers, which considers an additional term $\frac{\rho}{2} \|A\boldsymbol{x} + B\boldsymbol{y} - \boldsymbol{c}\|_2^2$ in order to regularize the dual update. ADMM is guaranteed to converge for fixed ρ, assuming that f and g are closed proper convex functions, though it is not guaranteed to find the global optimum. This algorithm is widely used in machine learning [BPC*11], and it has been used successfully to learn dictionaries in some methods presented in this survey. In this context, the coefficients and the dictionary are often optimized alternatively, sparse coding being used to learn the coefficients and ADMM being used to learn the dictionary.

While these algorithms are commonly used in the literature presented in this survey, they are not specific to 3d modeling, and are widely used in other such as signal and image processing, with image denoising as a common example. Authors such as Mairal [Mai10] and Elad [Ela10] have thoroughly detailed dictionary-learning and sparse coding, from the point of view of image processing. We will focus on 3d modeling in the next section.

2.4. Classification

We group techniques by the application domains in which dictionaries have been exploited the most: surface reconstruction, compression, synthesis and modeling. However, even within the same application domain, related methods can feature a great variability in the way dictionaries are used. We summarize all the methods presented in this survey in Table 1. The columns represent which elements compose a dictionary, what is learned and how it is learned, what type of shape is obtained by combining atoms, and the method application domain.

Atom type. Most of the methods use shapes or shape parts as atoms. In this column of the table, *shape* means that the representation of the shape is irrelevant, because it will only be placed in a scene – usually in part-based synthesis techniques [CKGK11, TYK*12]. A certain number of techniques need *meshes* as atoms as they blend or stitch them, and the algorithm they use is specialized for this representation. Less common, *mesh vertices* are used for surface reconstruction, and allow considering the reconstructed mesh as the dictionary itself.

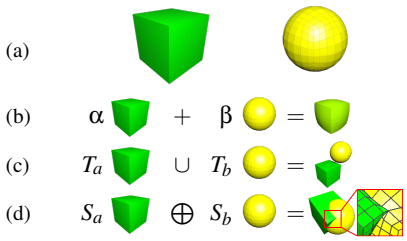

Figure 1: *Various combination operators. (a) example atoms used in the following: a cube and a sphere with the same connectivity. (b) interpolating atoms by interpolating vertex attributes, here position and color (coefficients are not always scalars). (c) compositing (placing) shapes together, using transformation matrices. (d) stitching objects together, resulting in a watertight output mesh. S_a and S_b depict the regions of interest for each atom.*

Combination operator. The second important ingredient in the definition of a dictionary is the combination operator, which here can be either *interpolation*, *composition*, or *stitching*. An overview of the differences between these classes of combination operators is shown in Figure 1. *Interpolation* refers to linear interpolation, with scalar coefficients that can be defined per atom [BV99] or can be spatially-varying [NVW*13]. This class of combination operators may impose constraints on the representation of the atoms (e.g., atoms may be meshes constrained to contain the same number of vertices and connectivity). *Composition* of a set of atoms refers to placing atoms, potentially with rotation and scaling, into the final object, which can be an artistic creation [CKGK11] or even a fully reconstructed scene [LDGN15]. It usually only needs

Method	Section	Dictionary		Learning				Output	Application
		Atom Type	Combination	$\mathcal{A}$	$\mathcal{C}$	$\mathcal{O}$	Algorithm		
[XZZ*14]	3.1	Mesh vertices	Interpolation	●	●	○	Sparse coding + ADMM	Mesh	Surface reconstruction
[PMG*05]	3.2	Mesh	Stitching	○	●	○	Region growing segmentation	Mesh	Surface reconstruction
[SFCH12]	3.2	Shape	Composition	○	●	○	Match silhouette contours in 2D and 3D	Scene	Surface reconstruction
[SKAG15]	3.2	Shape	Composition	○	●	○	Random Decision Forest [Bre01]	Point cloud	Surface reconstruction
[SMNS*13]	3.3	Shape	Composition	○	●	○	Voting scheme	Scene	Surface reconstruction
[LDGN15]	3.3	Shape	Composition	○	●	○	Match corner descriptors	Scene	Surface reconstruction
[vdHRD*15]	3.3	Shape	Composition	○	●	○	Minimize silhouettes differences	Scene	Surface reconstruction
[NXS12]	3.3	Shape	Composition	○	●	○	Random Decision Forest [Bre01]	Scene	Surface reconstruction
[KMYG12]	3.3	Shape	Composition	○	●	○	RANSAC Primitive Fitting [FB81]	Scene	Surface reconstruction
[SXZ*12]	3.3	Shape	Composition	○	●	○	Graph cuts [BVZ01]	Scene	Surface reconstruction
[AM00]	4.1	Mesh	Interpolation	●	●	○	PCA	Mesh	Compression
[KG04]	4.1	Mesh	Interpolation	●	●	○	PCA	Mesh	Compression
[LCS13]	4.1	Mesh	Interpolation	●	●	○	PCA	Mesh	Compression
[DCV14]	4.2	Image patch	Interpolation	●	●	○	K-SVD	Point cloud	Compression
[YLÖ*16]	4.2	Image patch	Interpolation	●	●	○	K-SVD	Point cloud	Compression
[GDGP16]	4.3	Image patch	Interpolation	●	●	○	K-SVD / Custom	Mesh	Compression / Synthesis
[BV99]	5.1.1	Mesh	Interpolation	●	●	○	PCA	Mesh	Synthesis / Morphing
[SQRH*16]	5.1.1	Mesh	Interpolation	●	●	○	PCA	Mesh	Synthesis / Morphing
[NVW*13]	5.1.2	Mesh	Interpolation	●	●	○	Sparse coding + ADMM	Mesh	Synthesis / Morphing
[HYZ*14]	5.1.2	Mesh	Interpolation	●	●	○	Sparse coding + ADMM	Mesh	Synthesis / Morphing
[WLZH16]	5.1.2	Mesh	Interpolation	●	●	○	Sparse coding + ADMM	Mesh	Synthesis / Morphing
[SZGP05]	5.1.3	Mesh	Interpolation	○	●	●	Minimize deformation gradients	Mesh	Synthesis / Morphing
[FB11]	5.1.3	Mesh	Interpolation	○	●	●	Minimize discrete shell energy	Mesh	Synthesis / Morphing
[Wam16]	5.1.3	Mesh	Interpolation	○	●	●	Minimize local deformations	Mesh	Synthesis / Morphing
[GCLX16]	5.1.4	Mesh	Interpolation	○	○	●	Dijkstra + smooth interpolation path	Mesh	Synthesis / Morphing
[KJS07]	5.2.1	Mesh	Stitching	●	○	○	Segmentation	Mesh	Synthesis / Modeling
[XXM*13]	5.2.1	Mesh	Stitching	●	○	●	Match suggestive contours	Mesh	Synthesis / Modeling
[AKZM14]	5.2.1	Shape	Composition	○	○	●	[KLM*13] + Multi Dimensional Scaling	Scene	Synthesis / Modeling
[CKGK11]	5.2.2	Shape	Composition	○	○	●	JointBoost [TMF07] on Bayesian network	Scene	Synthesis / Amplification
[KCKK12]	5.2.2	Mesh	Stitching	○	○	●	Greedy search + Cheeseman-Stutz score [CS96]	Mesh	Synthesis / Amplification
[TYK*12]	5.2.3	Shape	Composition	○	○	●	Markov Chain Monte Carlo	Scene	Synthesis / Amplification
[XZCOC12]	5.2.4	Shape	Composition	○	●	○	Genetic algorithm	Scene	Synthesis / Exploration

Table 1: *Summary of presented methods. Legend: $\mathcal{A}$ = atoms, $\mathcal{C}$ = coefficients, $\mathcal{O}$ = combination operator / ● learned, ○ not learned. "Not learned" means raw input for atoms, and independent from data for combinations and coefficients. A scene is a set of disjoint shapes, or more precisely $\{(S_i, T_i) \,|\, S_i = shape\, i, T_i = transformation\, i\}$.*

4×4 transformation matrices, with the possibility of placing the same atom at several locations; in general, this operator can work with any shape representation. Note that it is possible to have meshes assembled by composition, although in that case, they will simply be composed of several disconnected components. Finally, *stitching* meshes together refers to taking atoms (or their parts) and gluing the triangles in order to form a watertight mesh, to output a unique mesh [PMG*05] or to support organic shapes [KJS07] for instance. For example, *composition* and *stitching* are typically used by part-assembly methods.

Learning. This section of the table describes the elements of the dictionary that are learned: atoms, coefficients, and combination method, when applicable. The learning step is often done by minimizing an energy on the input dataset. Sometimes atoms and coefficients are not sufficient for the combination operator to output a shape, and some information may be added – such as the result of another energy minimization depending on the input data for example; in this case we say that the combination operator is *learned*. Finally this section of the table is completed with a short summary of the learning algorithms that are used, the most frequent ones being explained in the preliminaries (Section 2).

All the methods mentioned in this survey appear in Table 1 in the order in which they are introduced in the text, and are discussed in the Sections 3, 4, and 5 corresponding to the specific application that they address in geometry processing and modeling.

3. Surface reconstruction

The first application domain that we consider is surface reconstruction from unorganized point clouds. Surface reconstruction algorithms aim at generating piecewise smooth 2-manifolds – often in the form of triangle meshes – from point clouds potentially corrupted by noise, outliers and missing parts. In this context, dictionaries are often sets of high-quality shapes or point-clouds, whose knowledge helps removing noise, detecting outliers and filling holes in the input.

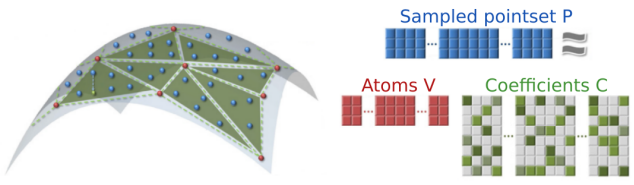

Figure 2: *Overview of the dictionary of Xiong et al. [XZZ*14]. Left: points in blue are from the point-cloud, points in red are the reconstructed vertices. Right: problem mathematical formulation.*

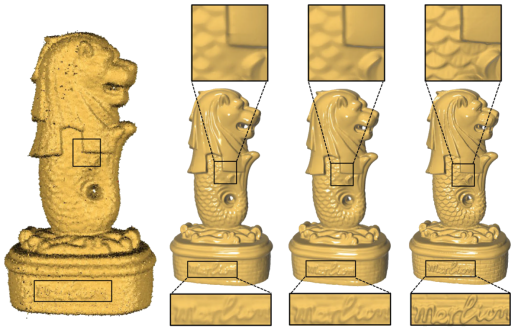

Figure 3: *Comparative results for surface reconstruction. From left to right: input point cloud, reconstruction using APSS, using RIMLS, and using the method of Xiong et al. [XZZ*14].*

3.1. Mesh as a dictionary

Xiong et al. [XZZ*14] propose a dictionary-based method to reconstruct a surface given as input a noisy point cloud, with the input points $P \in \mathbb{R}^{3\times n}$ being considered as individual signals. These points are expressed as linear combinations of the vertex positions $V \in \mathbb{R}^{3\times m}$ of the output mesh, which acts as the learned dictionary (see Figure 2). Specifically,

$$P = VC + Z$$

where Z is an offset matrix whose deviation to 0 is penalized. This decomposition is obtained as the optimizer of

$$\min_{V,C} \sum_i \|\boldsymbol{p}_i - V\boldsymbol{c}_i\|_2^q + \mathcal{E}_{\text{Reg}}(V,C)$$
$$\text{s.t. } \|\boldsymbol{c}_i\|_0 \leq 3, \|\boldsymbol{c}_i\|_2 = 1, \boldsymbol{c}_i \geq 0$$

$\mathcal{E}_{\text{Reg}}(V,C)$ being a *regularization* energy favoring equilateral triangles and enforcing input normals if given in the input point cloud. Additionally, the output mesh is constrained to be manifold.

The learning algorithm updates C and D iteratively, using ADMM for the update for the atoms, and optimizing mesh connectivity (while preserving manifoldness) during the process.

Analysis. The method of Xiong et al. [XZZ*14] is quite atypical in the sense that the dictionary *is* the result of the method, instead of just being an intermediate tool, so that the entire problem is transformed into a dictionary learning one, allowing to leverage advances in this domain. By using a $l_{2,q}$ norm ($q \leq 1$), the method can gain robustness to outliers and can outperform standard moving least squares (MLS) techniques such as algebraic point set surfaces (APSS) [GG07] or robust implicit moving least squares (RIMLS) [ÖGG09] in challenging settings (Figure 3), while being similar performance-wise, even if the non-convex optimization model does not guarantee global optimality. This method was extended beyond point clouds in the image meshing algorithm of Xie et al. [XfT16], in which images are considered as 5D point clouds with components $(x, y, red, green, blue)$.

3.2. Shape completion

A different approach was proposed by Pauly et al. [PMG*05], which uses *a database* of manifold surfaces as a dictionary, and shows how to complete scanned point clouds in order to fill holes, but also remove noise and detect outliers. The algorithm takes as input a point cloud and a "knowledge database", which is a dictionary of manifold meshes. The input shape is first queried in the database in a supervised manner, the user providing input in the form of tags and the query making use of point-based descriptors defined on the input point cloud, which are enriched with a confidence score to account for input defects robustly. All models that are found are non-rigidly registered to the point cloud and deformed. They are then segmented following the distortion and geometric errors of the deformation; these segments are stitched together to construct the output mesh fitting the input cloud. As more and more objects are scanned and processed, the final reconstructed surfaces further enrich the dictionary. Therefore, previous scans help in the processing of the following ones. Figure 4 illustrates the reconstruction of a scan that is missing large parts.

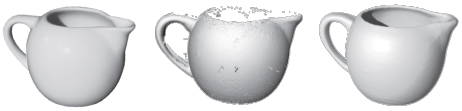

Figure 4: *Reconstruction using the method of Pauly et al. [PMG*05]. This technique can cope with large missing parts in scans, such as the other side of the creamer (upper part of the scan).*

Contrary to the method of Pauly et al. [PMG*05], which uses segments extracted from the shapes in its dictionary, methods introduced by Shen et al. [SFCH12] and Sung et al. [SKAG15] directly take a database of segmented and labeled shapes as input. However, they differ in their registration process and in the type of content they output, a scene for the first and a consolidated point cloud for the second.

RGBD-based. The approach presented by Shen et al. [SFCH12] takes as input RGBD data, processes both the point cloud and the RGB captured image, and outputs an assembly of parts present in its dictionary (see Figure 6 for the overview). The first step consists in registering shapes in the dictionary to the input cloud. Individual parts are then compared to the input and selected parts are assembled to create the output shape. This two-level matching is less computationally expensive than the direct fitting of segmented parts to the input point cloud since it allows narrowing the search window of individual parts to those that are located around the target local geometry, and it provides robustness as it enforces global coherence in the output, in the sense that parts are placed where they are expected to be present considering the input dataset. For example, the seat of a chair will only be searched and placed near the center of the scan.

The matching of individual parts is made by comparing the shape and the part both in 3D (geometry) and in 2D (comparing the 2D distance fields obtained from the silhouettes of the input image and the part, provided its current alignment). After the matching step, aligned parts are given a matching score, and ordered per category. A subset of top ranked parts is then selected per category,

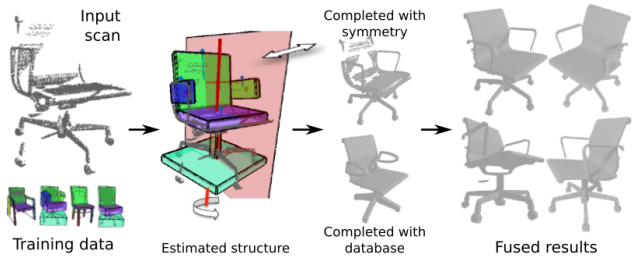

Figure 5: *Overview of the method of Sung et al. [SKAG15]. From an input scan and training data (left), the scan structure is inferred (middle left); possible reconstruction from symmetry and parts (middle right) are used to complete the final point cloud (right).*

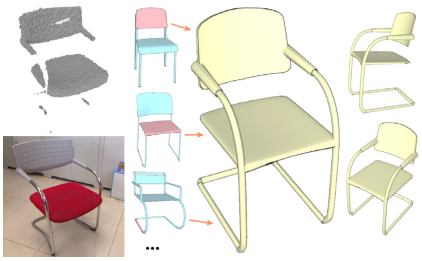

Figure 6: *Structure recovery of Shen et al. [SFCH12]. Given an input point-cloud and its associated RGB image from the scanner (left), and a dictionary of parts and objects (middle, parts in light red), a similar object is reconstructed (right). While structurally the same, the original object and the reconstruction exhibit differences (e.g., the position where the seat begins, the shape of the back, etc).*

based on criteria such as geometric fidelity, proximity, and part overlap. At this step, symmetry is accounted for by duplicating related parts, if the symmetry associated with the input object is successfully matched against the input point cloud. Finally, parts are joined by minimizing contact point distances, in order to form a well-connected and visually-pleasing output model.

Point-based. In contrast to the method of Shen et al. [SFCH12], the approach of Sung et al. [SKAG15] does not use RGB images, does not assume that shapes can be rigidly aligned with the point cloud, and outputs a consolidated point set only. The authors estimate the structure of the shape from the dictionary, which is composed of a set of parts and a set of symmetries, by minimizing an energy that takes into account low-level components such as point segmentation (favoring smooth and accurate segments) and part assignation to points (favoring points close to the surface of their part), and high-level components such as pairwise relationships and symmetries. Finally, in the completion step, new points are added: first, points are duplicated following the symmetries of the parts they belong to, and second, points are generated from the voxelized parts in the database (see Figure 5 for a visual overview of this workflow).

Analysis. These methods produce various types of output: the modeling by example of Pauly et al. [PMG*05] outputs a watertight mesh, the structure recovery of Shen et al. [SFCH12] outputs an assembled mesh (i.e., a combination of parts) and the approach of Sung et al. [SKAG15] outputs a point cloud. They all provide good results in presence of moderately-incomplete data and / or noise, and the last two techniques [SFCH12, SKAG15] make extensive use of symmetries either detected in the point cloud or present in the dictionary to help recovering shapes with a satisfactory global structure even when large parts are missing in the input point cloud, although the method of Sung et al. [SKAG15] reaches better reconstruction quality in that case. However, the first method [PMG*05] requires user input and is therefore prone to human-error in the choice of tags describing the scanned object, while the second one [SFCH12] does not reconstruct the scanned object but outputs an object with similar structure, since it is restricted to the assembly of parts that are present in its dictionary. The atom placement model also does not make possible shape variations not explained by spatial layout, such as facades with irregular structures. Since the third method [SKAG15] only outputs a point cloud, an additional surface reconstruction algorithm [GG07, ÖGG09, KH13, XZZ*14] has to be used to recover a surface manifold from it. Finally, a drawback of all these methods is their heavy reliance on the completeness of the dictionary, for target shapes to be expressible with the atoms.

3.3. Scene reconstruction

In the same spirit, it is possible to reconstruct entire 3D scenes by placing shapes extracted from a dictionary. In this context, a given shape can appear several times at several locations of the scene. While the combination is simple (union), the extraction of the coefficients (which are transforms) can be rather complex. SLAM++ [SMNS*13] and the method of Li et al. [LDGN15] are particularly efficient, and allow the user to reconstruct an entire room in real-time using a simple hand-held sensor. All captured frames are processed (not just the last one), meaning that the accuracy of the reconstruction augments progressively.

The SLAM++ method [SMNS*13] tracks the object's position and camera pose simultaneously. Objects are recognized and their placement determined at the same time, using the method of Drost et al. [DUNI10], which consists in finding similar point-pairs in the depth image and in the dictionary. First, the point-pairs are generated for each object in the database by randomly sampling points and computing all possible pairings, and these pairs are then put in a dedicated search structure (one per shape). After filtering the input image, point-pairs are generated the same way and these pairs are queried in the structure via a voting scheme, which gives the position of matched shapes. Note that this method is GPU-friendly and therefore allows for a fast scene reconstruction.

The problem with SLAM++ is that these point-pair descriptors are highly discriminative and require almost exact matches, which implies that in practice it works well only for extremely rich dictionaries, and that scanned models should be cleaned before use.

To address this issue, Li et al. [LDGN15] instead match key-point descriptors, computed similarly on each shape of the dataset and on the input image, the latter being converted to a signed distance function (SDF) [NZIS13, NDF14]. They are obtained by detecting corners and enriching them with descriptors

of their geometric neighborhood (such as the local distance field), so the dictionary atoms of this approach are shapes with their set of key-point descriptors. Large databases are managed by clustering the objects on their descriptors. The object matching the scan (i.e. the one with the closest descriptors) is then added to the scene. Since all necessary comparisons are easily parallelized on the GPU (as with SLAM++ [SMNS*13]), this technique provides real-time results. Furthermore, the main advantage of this method compared to the previous one is that it is highly robust to missing data, notably because the SDF representation aggregated over several frames is accurate, yields robust normals and is resistant to noise.

Van den Hengel et al. [vdHRD*15] reconstruct the structure of real-world Lego assemblages from a set of silhouette images and a dictionary of Lego parts (see Figure 7). By expressing silhouettes as vectors, with one vector for all the input silhouettes and one vector per part (which is the concatenation of this part rendered with the projections of all input images), they solve the structure recovery problem with an energy minimization, the first term favoring combinations of silhouettes close to the input images, and the second favoring sparse coefficients vectors. For plausible results, they take in account overlapping silhouettes, and they aim for physical plausibility by avoiding atom intersections and floating parts. By using silhouettes, this method is robust to noise. However, it is computationally expensive due to the complexity of the system to solve, and can reconstruct shapes with holes when viewed from angles not corresponding to an input image.

Indoor scene reconstruction. Among previous data-driven dictionary-based scene reconstruction methods, some are specialized to indoor scenes, as shown by Nan et al. [NXS12], Shao et al. [SXZ*12], and Kim et al. [KMYG12]. For all of these techniques, the atoms of the dictionary are shapes given as input, and they are composed together as the user scans the room.

The main idea of the method of Nan et al. [NXS12] is to alternate between segmentation and classification in a feedback loop. Before scanning a scene, shapes (atoms) are preprocessed : they are used to learn a Randomized Decision Forest, which is a robust classifier in presence of incomplete data. In the runtime phase, the input point cloud is oversegmented, and the adjacency of these segments is computed. Each step in the loop works on a candidate patch (aggregation of segments), first trying to extend it with the neighbor segment with the highest classification likelihood. To improve the segments, templates associated with the class of the patch is fitted to the patch point-cloud, in order to detect outliers and exclude them.

The approach of Kim et al. [KMYG12] requires first scanning all potential objects individually, before they can be automatically segmented and approximated by simple primitives using RANSAC. During runtime, the user scans an indoor room, and the scan is separated into connected components that are matched against the database of objects. First, dominant planes corresponding to walls, floor, etc, are extracted using RANSAC; then objects are matched iteratively, and once a transformation is found for an object, its corresponding points are removed from the point-cloud. The loop continues until no objects can be matched, and the matching in itself is performed by minimizing the distance between features measured on segments and models to match. A drawback of the technique of Kim et al. [KMYG12] is that it requires a significant amount of training data as well as careful parameter tuning, compared to previous methods [LDGN15, NXS12, KMYG12].

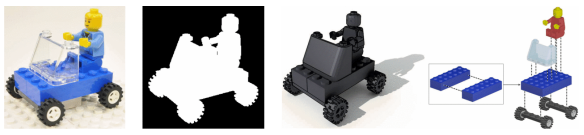

Figure 7: *Extracting Lego structures [vdHRD*15]. From left to right: an input image is transformed into a silhouette image, used to retrieve the object 3D structure. Finally, a possible application is the automatic generation of assembly instructions.*

Shao et al. [SXZ*12] address the segmentation problem with a supervised system: given RGBD images of indoor scenes, each image is segmented on a set of predefined semantic labels (10 are used in the original article: sofa, table, monitor, wall, chair, floor, bed, cabinet, ceiling and background). These segmentations are achieved through a minimization of a Conditional Random Field energy with two terms, the first one measuring the likelihood of a specific label for a specific pixel, and the second one measuring the labeling consistency between two pixels. The actual minimization is done using graph-cuts [BVZ01], and these segmentations can be refined by the user if needed. The segmented point clouds are then matched with objects in the database for the final composition.

Analysis. The use of a dictionary for the reconstruction of scenes allows faster execution times, as the algorithm just has to classify parts of the scene and find the appropriate placement. Moreover, compared to the space of possible point clouds or voxels, dictionaries allow removing ambiguous and impossible shapes from the search. These methods work best with man-made objects due to the highly repetitive structure, and can reconstruct high quality scenes since the quality of the composition only depends on the quality of the atoms. This enables algorithms such as SLAM++ [SMNS*13] to run in real-time. The two first approaches [SMNS*13, LDGN15] are more general than the other presented in this subsection, as they do not include domain-specific knowledge such as basic room organization [NXS12, KMYG12, SXZ*12] or grid structures [vdHRD*15] – note that the latter can be applied to general shapes but the quality of the reconstruction is not on par with the reconstruction of shapes with a grid structure. Nonetheless, akin to the shape completion methods shown in the previous section, the drawback of these methods resides in having to fill the dictionary at first, requiring manual modeling or scanning beforehand, which becomes cumbersome for large number of shapes; this is attenuated for the primitive fitting method of van den Hengel [vdHRD*15], as it only need a few atoms. Additionally, with these scene reconstruction methods [SMNS*13, LDGN15, NXS12, KMYG12, SXZ*12], not reconstructing an object can be due to either the lack of similar shapes in the dictionary, or the lack of scans, and the distinction between these two cases is difficult.

4. Compression

A natural application of dictionaries is data compression, where the input data can be represented as the sum of a signal expressed in a dictionary and an approximation residual (see Section 2.3); the smaller the residual, the better the approximation, and since the size of the dictionary is limited (for the compression), a better approximation means a better dictionary.

4.1. Compressing animated meshes

Most mesh compression algorithms do not use dictionaries of shapes, but prefer relying instead on some carefully crafted coding format and apply scalar or vector quantization, as is shown in the survey of Maglo et al. [MLDH15]. There are several different approaches when it comes to mesh compression, and depending on the technique, the focus is put more on compressing the connectivity or on the geometry. It is worth mentioning that for static meshes, the valence coder of Alliez and Desbrun [AD01] was proven optimal with regards to connectivity-driven compression.

Indeed, this is mainly in the domain of mesh sequences compression that dictionaries are used, starting with the method of Alexa and Muller [AM00] which uses a PCA to learn a dictionary of frames. Usually, animation is done by linearly interpolating between a set of poses. While this is intuitive for an artist, it is not efficient in terms of memory storage (note that this interpolation is a use of a dictionary in itself). To find better poses, i.e., atoms enabling sparser coefficients, Alexa et al. [AM00] perform a PCA on the f vectors in $\mathbb{R}^{3v}$ representing all the frames, v being the number of vertices. The authors show that only a few principal directions are required to obtain satisfactory reconstructions while allowing them to compress animations up to 97%.

This PCA-based technique was first extended by Karni and Gotsman [KG04], in which linear prediction coding was used on the coefficients to improve compression. The aim of Linear Prediction Coding is to express element i in a series as a linear combination of its k preceding elements. Sattler et al. [SSK05] propose instead to perform Clustered Principal Analysis on the trajectories over a few frames, thus taking advantage of the fact that large regions of the mesh undergo similar motion in standard animations. Luo et al. [LCS13] choose to do a temporal clustering, i.e., cluster similar poses, which is beneficial for long animations (i.e., when the number of frames is greater than the number of vertices). Finally, Váša and Skala presented a series of articles [VS07, VS09, VS10, VS11], in which they combine PCA on the trajectory space, with accurate coefficients and basis prediction and a set of optimization strategies on the traversal order. In their last article on this problem [VS11], they present compression rates ranging from 0.5 to 5 bits per frame per vertex – to compare with the 96 bits required to store one vertex in one uncompressed frame.

4.2. Compressing point clouds

Digne et al. have shown in [DCV14] how to compress 3D point-clouds using a dictionary; this method is also used by Yoon et al. [YLÖ*16]. Starting from a point cloud which is assumed to be dense enough to unambiguously represent a surface, and a radius R given by the user, the first step is to select a set of seeds which will be used to cover the input point cloud with patches. R is assumed to be greater than the noise magnitude and big enough so that each R-neighborhood centered at the seeds contains several points to allow for patch fitting and together cover the whole point set. This sampling is done in a greedy manner by marking a point as a seed if it is not already covered by the R-neighborhood of an existing seed. Isolated points are marked as outliers during this process.

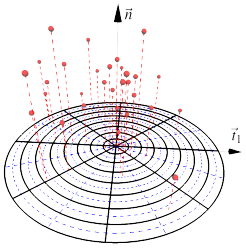

Figure 8: *From Digne et al. [DCV14], local frame of a seed, described by a heightmap over a radial grid.*

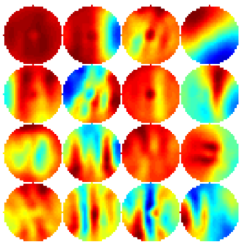

Figure 9: *Dictionary learned with the method of Digne et al. [DCV14], atoms are shown by order of importance.*

For each seed, a local frame is aligned onto the patch normal (see Figure 8), which is either derived from the point cloud normals or from a local PCA. With this frame in hand, each point in the neighborhood is expressed in cylindrical coordinates (r, θ, z) and quantized into a $k \times k$ circular patch image, which acts as its descriptor. Finally, K-SVD is run on the set of patch images, and the output is a dictionary whose atoms are patches and coefficients scalars (Figure 9).

To reconstruct the point cloud from the learned dictionary and coefficients, patches are first repositioned using the stored frames and a point is created for each radial pixel in the patch image. To handle oversampling at the patch boundaries, points falling at the intersection between several seeds' neighborhoods are merged as the average of the corresponding points in each neighbor seed patch.

This method is able to handle large point clouds with higher quality than competing methods [KV05, SMK08, HMHB08] (which are not dictionary-based in the sense of this survey). For a fixed compression rate between 0 and 2 bits per point, the Peak Signal to Noise Ratio (PSNR) of the approach of Digne et al. [DCV14] is greater than these methods by roughly 9dB, 8dB and 2dB respectively. A minor drawback is that the point set boundaries are generally not preserved after decompression, as those are not encoded in the atoms and points are added in the whole neighborhood of the seeds regardless.

4.3. Terrain compression

In a similar spirit, Guérin et al. [GDGP16] propose to compress terrains using the self similarity of such data, as terrains are often stored as heightmaps and thus trivially assimilable to (random-accessible) point clouds. The input terrain is defined by

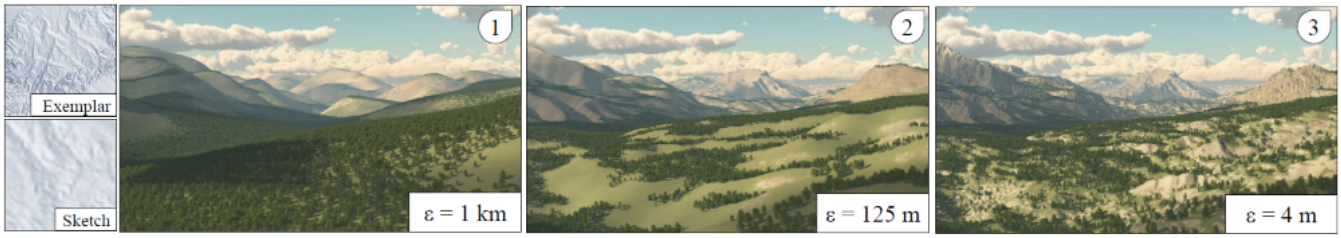

Figure 10: *Sparse terrain amplification as an application of Sparse Terrains [GDGP16]. This example use the elevation map of Australia, first with a resolution of 1 km (left), 125 m (middle) and 4 m (right), generated by successive amplifications. Texturing and vegetation were applied as a postprocess to outline the variations of the landscape created by the amplifications.*

an elevation function $h : \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^2$. This elevation map is decomposed as a set of patches, which are called *primitives*:

$$h(\boldsymbol{p}) = \sum_{i=0}^{n-1} \alpha_i(\boldsymbol{p}) h_i(\boldsymbol{p}) \Big/ \sum_{i=0}^{n-1} \alpha_i(\boldsymbol{p}) \;\; ; \;\; \alpha_i(\boldsymbol{p}) = \left(1 - \frac{\|\boldsymbol{p} - \boldsymbol{x}_i\|^2}{R^2}\right)_+^3$$

where α_i is a spatial weight function localizing the influence of primitive h_i. The whole domain must be covered by these primitives, potentially with overlaps. These primitives are decomposed into dictionary height map atoms $\{d_j\}_{j=0}^{N-1}$ as

$$h_i(\boldsymbol{p}) = z_i + \sum_{j=0}^{N-1} c_i^j d_j(\boldsymbol{p} - \boldsymbol{x}_i)$$

where z_i is a primitive-representative elevation, $\boldsymbol{x}_i$ is its center and the $\{c_i^j\}$ are the coefficients allowing expressing a primitive h_i in terms of atoms $\{d_j\}$. The atoms can be analytic (i.e., defined at every continuous location, for example represented as variations of Perlin noise) or sampled (e.g., represented as elevation grids). In either case, these atoms are geometrically embedded in $\mathbb{R}^K$, and since there are N atoms and n coefficients, the dictionary can be represented by a matrix $D \in \mathbb{R}^{K \times N}$ stacking all atoms, and coefficients as $C \in \mathbb{R}^{N \times n}$. For the application of terrain compression, the dictionary atoms $\{d_j\}$ as well as the coefficients $\{c_i^j\}$ are learned using K-SVD, and base altitudes are $z_i = h(\boldsymbol{x}_i)$.

Terrain modeling and amplification. While the Sparse Terrains method [GDGP16] achieves high compression ratios, its generic and hierarchical representation allows for additional interesting applications. In particular, terrain modeling can be achieved through the change of the atoms while preserving the coefficients, and resolution amplification, which boils down to replacing low-resolution atoms with higher-resolution counterparts.

In these applications, it is important to obtain a naturally-looking set of atoms, and the learning of the dictionary is slightly changed to account for this constraint. Specifically, their learning algorithm is a variant of K-SVD, in which they force dictionary atoms $\{d_j\}$ to match input primitives $\{h_i\}$ that are either extracted from a realistic terrain database or extracted from a high-resolution model directly as patches, while accounting for the sparsity s of the solution. Their optimization can be written as:

$$\min_{D,C} \|H - DC\|_F^2 \quad \text{s.t.} \begin{cases} \forall i \in [1..n], \; \|\boldsymbol{c}_i\|_0 \leq s \\ \forall j \in [1..N], \; \exists\, l \in [1..n], \; \boldsymbol{d}_j = \boldsymbol{h}_i \end{cases}$$

In the case of terrain amplification, the system is given a high-resolution exemplar, which is decomposed on a set of derived atoms $\{d_j\}$, and a low-resolution terrain (e.g., resulting from a rough sketch). The atoms $\{d_j\}$ are down-sampled and low-resolution counterparts $\{\bar{d}_j\}$ are created. The low-resolution terrain is decomposed into $\{\bar{d}_j\}$, and an amplification can be trivially obtained (see Figure 10). Note that this process can be performed in a multiresolution fashion to increase robustness.

Analysis. Dictionaries have been used extensively and successfully for compression, and in particular for lossless compression (for example, by the LZMA algorithm used in the popular archiver *7-zip* [Pav16]). However, in the field of surface and point cloud compression, existing dictionary-based algorithms are for the most part lossy. Interestingly, general surface compression is not done using dictionaries, unless the connectivity is not important and the set of vertices can be interpreted as a point cloud, enabling self-similarity [DCV14]. Heightmap terrain are such an example, as the connectivity is following a fixed pattern, which does not need to be stored in the compressed data. Similarly, when compressing a mesh animation [AM00, KG04, LCS13], the connectivity is given in the base mesh, the other poses being treated as point clouds. Compressing the animation trajectories [SSK05, VS11] does not involve the connectivity, as it is only present once for all the meshes. Finally, Guérin et al. [GDGP16] show that dictionaries can be used for both compression and synthesis, here by changing the atoms to increase or decrease the level of detail on the terrain, with limitation that synthesized terrain will follow the logic learned in the dictionary, and not geological laws. All these techniques show that most shapes or shape animations have a lot of self-similarities that can be compressed using dictionaries.

5. Synthesis and Modeling

We have seen how dictionaries can be useful for synthesis. One can use them as a high-level control in order to create shape variations by changing atoms or editing their weights. In particular, the use of such a workflow allows reducing drastically the time needed to design interesting shapes and *explore* the resulting shape space. We will emphasize this aspect when describing related techniques in this section, divided into two categories: *mesh morphing* using linear interpolation of vertex positions, and *part assembly* allowing the creation of new shapes from existing parts.

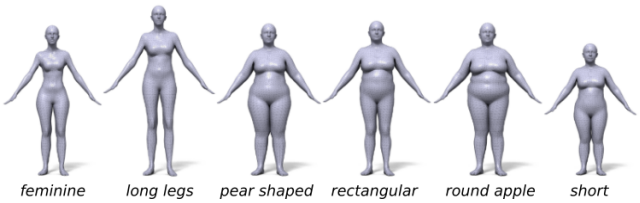

Figure 11: *Results of Body Talk [SQRH*16], which learns a model linking 3D shapes and linguistic descriptions using crowd-rated 3D models. Each shape shown here is the most likely one, given the descriptive word below them.*

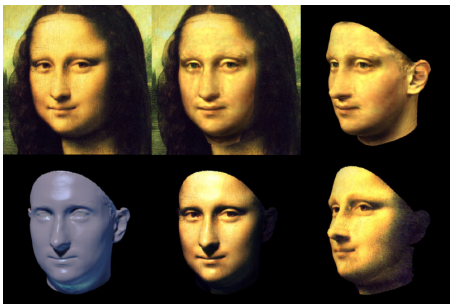

Figure 12: *The method of Blanz and Vetter [BV99] allows editing a painting – here the Mona Lisa (top left) – by extracting the face (top center and right) and then modifying the illumination (bottom left and center) or the orientation (among others).*

5.1. Mesh morphing

Unlike reconstruction or compression, where the goal is to encode a single object or scene, shape morphing consists in finding an interpolation between multiple shapes. In its its simplest form it consists in linearly interpolating vertex positions, if the meshes share their connectivity (e.g., a mesh animation). This, however, can lead to unrealistic shapes with significant distortion, which has also motivated the use of dictionaries in this setting. The simplest approaches often use PCA-based dictionaries to interpolate the coefficients, but have also used widely varying artistic controls, ranging from word attributes (as in Body Talk [SQRH*16]) to handles for inverse-kinematics (the method of Wampler [Wam16] for example) or even full example shapes as guides [GCLX16].

5.1.1. Words-to-shape models

A good use of dictionaries with a simple morphing as combination operator is described by Blanz and Vetter [BV99], in which a dictionary for human faces is developed. The input is a set of scanned faces, in the form of meshes put in correspondence with each other, and whose vertices are equipped with two attributes: their position S and their color T (encoded in a texture). Constructing a new face boils down to interpolating these attributes with coefficients $(a_i)_{i=1}^m$ for the positions and $(b_i)_{i=1}^m$ for the colors. In this work, both sets of coefficients must sum up to 1 and are therefore barycentric weights. The shape dictionary is computed using a PCA on the delta shapes $\Delta S_i = S_i - \overline{S}$, $\overline{S}$ denoting the shape average; a similar technique is used to obtain the texture dictionary. In Body Talk [SQRH*16], input shapes (entire human bodies, all in the same pose) are expressed over a derived dictionary, and is very similar to the previous method in spirit, with the notable difference that no texture dictionary is output. Note that the latter chose to separate female and male bodies, and also keep the first 8 vectors of the PCA only, thus emphasizing the low-dimensional nature of the human shape dataset that they process.

These dictionaries, learned via PCA on the input dataset, are not easy to manipulate for a human, and a set of user-friendly attributes are developed to allow for high-level control in each of these two works. Both methods ask users to rate a model according to N_A predefined attributes such as "masculine", "feminine", "smiling", "skinny", "muscular", etc. For each attribute j, the input of the user is a factor μ_{ji} for shape S_i describing the rating of shape i over the attribute j. For both methods, these user-provided ratings are the key to map natural language tags to the dictionary atoms (see Figure 11). However, both methods differ in their model reconstruction methodology.

The approach of Blanz and Vetter [BV99] suggests matching a change $\Delta\boldsymbol{\mu}_j$ of attribute j (corresponding to, e.g., a desired increase of "smiling", etc...) to a change ΔS^j of an input shape computed as $\Delta S^j = \sum_{i=1}^m \mu_{ji}\Delta S_i$. An input shape S with texture T is therefore updated as

$$S \leftarrow S + \sum_{j=1}^{N_A} \Delta\boldsymbol{\mu}_j \sum_{i=1}^{m} \mu_{ji}\Delta S_i \quad ; \quad T \leftarrow T + \sum_{j=1}^{N_A} \Delta\boldsymbol{\mu}_j \sum_{i=1}^{m} \mu_{ji}\Delta T_i$$

The user is therefore given the ability to edit an input shape using, e.g., a simple set of sliders $\{j\}$, which are mapped to changes of attributes $\{\Delta\boldsymbol{\mu}_j\}_j$. Although this interface is quite straightforward, it has proven useful and powerful for high-level editing. Additionally, Blanz and Vetter [BV99] show how to edit an input photograph of a face (it can even be a painting, as demonstrated on the Mona Lisa painting in the original article) using their framework. They start by matching a 3D morphed face to the input image, and render the model atop the image (see Figure 12). This matching is made possible by the use of a dictionary and of a simple lighting model (Phong). The problem then becomes a search for the optimal coefficients minimizing the distance between the source image and a rendered image, which is achieved using a gradient-descent. By later editing the morphed face, applications such as relighting, shape enriching and expression or pose editing are offered to the user.

In contrast, in the work of Streuber et al. [SQRH*16], the coefficients $\boldsymbol{\mu}_j$ are averaged from the crowd's ratings. This results in a vector of ratings of the form $\boldsymbol{d}_i = (\mu_{1i}, \mu_{2i}, \ldots, \mu_{Wi})^\top$ for shape i, with W attributes (in their experiments $W = 30$). By assembling those vectors $\boldsymbol{d}_i$ in a matrix D (each row defined as $(1, \boldsymbol{d}_i^\top)$), the input data Y is decomposed into the dictionary atoms as

$$Y = DC + \varepsilon \tag{1}$$

where each row of Y is the matrix of all bodies (one per row) expressed in its low 8-dimensional PCA space. Assuming therefore an affine linear relationship between ratings and body geometry in the PCA space, the regression coefficients C are found in

the least-squares sense. This gives effectively the words-to-shape model, and given a new rating $\boldsymbol{\mu}$ (which is a row vector), the corresponding shape geometry (in the PCA space) can be obtained by $\boldsymbol{y}(\boldsymbol{\mu}) = \boldsymbol{\mu} C$. Figure 11 shows a set of shapes associated with a few descriptive words. Note that a shape-to-words model can be derived by simply inverting the model described by Equation 1.

Analysis. While these methods are very similar, the technique of Blanz and Vetter [BV99] does not model correlations between attributes, a problem tackled in Body Talk [SQRH*16], allowing accurate reconstruction of shapes from words alone. Still, Body Talk does not account for the non-linear relationship between user ratings and shape coefficients for some attributes such as "skinny". While both methods are simple, considering they provide a powerful modeling system suitable for experts as well as novice users, the use of the first N components of the PCA limits the space of possible shapes, excluding uncommon variations. Note that Streuber et al. [SQRH*16] provide an interactive web tool that allows to visualize a human shape by playing with the attributes.

5.1.2. Localized decomposition for shape deformation

In contrast to the methods presented in the previous section [BV99, SQRH*16] for which the PCA results in components whose support is global, methods such as SPLOCS [NVW*13] and the work of Huang et al. [HYZ*14] optimize for the components' sparsity by penalizing the size of their support, which results in better artistic control. All of these methods use meshes with exact one-to-one correspondences, and most examples are frames of a mesh animation. SPLOCS [NVW*13] starts by encoding all meshes as differences with a base mesh (e.g., first mesh or average of meshes), and for M meshes with N vertices, the whole input data can be stored as a $X \in \mathbb{R}^{M\times 3N}$ matrix. The aim is to find the dictionary $D \in \mathbb{R}^{K\times 3N}$ and the coefficients $C \in \mathbb{R}^{M\times K}$ such that:

$$X = CD$$

Directly performing a PCA to obtain D and C would yield global components, which has unwanted practical consequences. For instance, on the facial animation in Figure 13, moving the bottom of the lips would also deform the eyebrows. SPLOCS [NVW*13] models the sparsity of the decomposition as

$$\mathcal{E}_{sparsity}(D) = \sum_{k=1}^{K}\sum_{i=1}^{N} \lambda_{ki}\|\boldsymbol{d}_k^i\|_2$$

where $\boldsymbol{d}_k^i$ is the vertex i displacement in component k, and weights λ_{ki} of the displacement $\boldsymbol{d}_k^i$ depend on the geodesic distance to the component center (the center being the location with the largest displacement). Fitting the input data as closely as possible while enforcing sparsity amounts to minimizing the following energy:

$$\mathcal{E} = \|X - CD\|^2 + \mathcal{E}_{sparsity}(D)$$

which is a non-convex problem addressed with ADMM.

Analysis. While the components extracted using SPLOCS [NVW*13] enable better artistic control than, e.g., a simple PCA, as illustrated in Figure 13, it is not suitable for input mesh sequences featuring large rotations for example, and hence has been improved [HYZ*14, WLZH16]. Huang et al. [HYZ*14] use

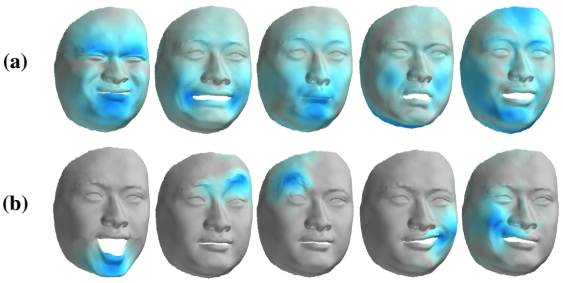

Figure 13: *Comparison of decompositions on the same set of captured faces, between PCA (a) and SPLOCS [NVW*13] (b). The intensity of the blue shows the magnitude of vertex displacements.*

deformation gradients to represent a shape with respect to a rest shape, in order to handle large rotations. Still, this encoding is too limited for sequences featuring large global rotations (e.g., a horse running in circle) as the deformation gradients are not intrinsic geometry quantities and also are not local regarding rotation deformation. These issues are addressed by Wang et al. [WLZH16], who also localize decomposition, but use *edge lengths* and *dihedral angles* to represent a given shape (this representation encodes the first and the second fundamental forms of a discrete shape, and has been used successfully in a body of work dealing with *shape interpolation* [WDAH10]). Reconstruction from such a representation is possible with a linear solve, as shown in this method [WLZH16]. All these methods have a runtime on the order of minutes, with the last being the fastest.

5.1.3. Example-based mesh inverse kinematics

While the methods described in the previous section obtain results that outperform non-data-driven approaches, they rely on a simple combination operator that is not *learned* from the input examples. Using the dataset to alter the way atoms are combined, although more complex than a simple linear interpolation, enables rich and example-inspired morphing [SZGP05, GCLX16, Wam16]. These methods take as input a dataset of meshes with shared connectivity. The latter equips an object with a set of handles, which are points associated with a subset of the vertices, in order to deform the shape via inverse kinematics (IK).

Sumner et al. [SZGP05] were the first to introduce an example-based mesh IK method using *blending* of feature vectors at its core. Specifically, given m poses of a shape with n vertices, a feature vector $\boldsymbol{d}_i$ $(1 \leq i \leq m)$ is associated with pose $\boldsymbol{V}_i \in \mathbb{R}^{3n}$. The *example manifold* $\mathcal{M}$ is then described as some form of combination of these feature vectors defined by coefficients $\boldsymbol{c} = \{c_i \in \mathbb{R}\}$ denoting the amount of influence of pose i in the output, this example manifold being simply given by the vector space spanned by $D = \{\boldsymbol{d}_i\}$ in the simple, linear case.

The key idea of Sumner et al. [SZGP05] is to find, given a set of constrained handles, the shape geometry that is closest to the example manifold $\mathcal{M}$, which amounts to optimizing for the coefficients $\boldsymbol{c}$ while the dictionary atoms $\{\boldsymbol{d}_i\}$ are kept untouched.

In the seminal article of Sumner et al. [SZGP05], the feature

vectors are the deformation gradients of the triangles, which can be computed using the gradient matrix G as $\boldsymbol{d}_i = G\boldsymbol{V}_i$. By splitting the vertex positions into two separate vectors $\boldsymbol{V}_H$ and $\boldsymbol{V}_F$, denoting the handle positions and the free vertices respectively, the vertex positions resulting from the handle positions are therefore obtained as the result of the following minimization problem:

$$(\boldsymbol{V}_F^*, \boldsymbol{c}^*) = \underset{(\boldsymbol{V}_F, \boldsymbol{c})}{\operatorname{argmin}} \|G\boldsymbol{V}_F + G\boldsymbol{V}_H - \boldsymbol{blend}(\boldsymbol{c}, D)\|$$

which in the case of a linear blending defining $\mathcal{M}$ (i.e., $\boldsymbol{blend}(\boldsymbol{c}, D) = \{c_i \boldsymbol{d}_i\}$) requires a single linear solve.

As simple linear blending of deformation gradients is too limited for a natural deformation behavior, non-linear blending is required in practice, and Sumner et al. [SZGP05] adjusts $\boldsymbol{blend}(\boldsymbol{c}, D)$ by expressing the deformation gradient of triangle j as

$$\boldsymbol{blend}(\boldsymbol{c}, D)_j = \exp(\sum_i c_i \log(R_{ij})) \sum_i c_i S_{ij} \quad ; \quad R_{ij} S_{ij} = T_{ij}$$

with T_{ij} being the polar decomposition of the deformation gradient of triangle j in pose i. This decomposition is commonly used for gradient-based shape morphing [XZWB05] and allows obtaining much better results than simple linear blending. However, interpolating the gradients (even in a sophisticated way) results in artifacts mentioned in Section 5.1.2, as shortest path interpolation or blending of local rotations is not adapted in 3D where shape interpolation typically requires complex interpolation paths of the rotations, which cannot be deduced from a local analysis only.

This problem is addressed by Fröhlich and Botsch [FB11], who use edge lengths and dihedral angles as features, resulting in full rotation invariance. As previously written, these quantities are natural features to interpolate for shape morphing [WDAH10]. They are also natural variables for computing and minimizing discrete shell energies [GHDS03], which are efficient geometric energies penalizing stretching and bending in a richer way than rigid energies can achieve. Note that the Riemannian geometry of the *space of shells* has been studied by Heeren et al. [HRS*14], allowing them to extend concepts such as geodesics and parallel transport to the space of deformations governed by these energies.

More recently, the approach of Wampler [Wam16] shows another problem, independent of the actual feature representation of the example manifold, and which relates instead to the under-constrained nature of this space. More precisely, if several linear combinations c provide the same positions for the handles – such as illustrated in Figure 14 where the handles of the green pose can be obtained as a linear combination of the handle positions of the blue and red poses – the result from the latter optimization is under-constrained, and some poses may eventually be ignored as a result (see Figure 14d, for an illustration of such behavior).

To enforce pose awareness, one of the contributions of Wampler [Wam16] is to add an *energy interpolation term*, interpolating energies from individual pose:

$$\mathcal{E}_{\mathcal{I}}(\boldsymbol{c}, \boldsymbol{p}_i, \boldsymbol{q}) = \sum_i c_i E_d(\boldsymbol{q}, \boldsymbol{p}_i)$$

where $\boldsymbol{p}_i$ denotes the i^{th} pose, $\boldsymbol{q}$ denotes the output interpolated geometry, and $E_d(\boldsymbol{q}, \boldsymbol{p}_i)$ denotes an elastic energy describing the amount of "stretch" for deforming $\boldsymbol{p}_i$ into $\boldsymbol{q}$. This kind of energy interpolation method is highly popular for shape interpolation [CPSS10, VTSSH15], when the weights $\boldsymbol{c}$ are provided by the user and the technique outputs barycentric weighting of the poses $\{\boldsymbol{p}_i\}$. As optimizing for $\mathcal{E}_{\mathcal{I}}$ alone favors the nearest pose in the shape space, so local configurations matching locally the input poses are preferred. This can be observed in Figure 14b, where the green pose is no longer ignored in the output.

Analysis. To summarize, the key motivation behind the use of dictionaries for shape manipulation is that some deformation behaviors (localization of articulations, muscle bulge, rigidity of limbs vs elasticity of jaws and fat) cannot be easily captured by geometric elastic energies only, whatever their degree of complexity, but are captured trivially when learning on poses that are representative of the degrees of freedom desired by the user. In our opinion, the recent technique of Wampler [Wam16] constitutes a breakthrough for mesh IK and is successfully demonstrated on challenging inputs such as highly stretched and exaggerated cartoon animations, but we suspect that there is room for improvement, especially in the mathematical formulation of the final energy, as non-smooth transitions near input poses in the output can be observed in Figure 14b. Further investigation is required to tell if these effects are negligible or not in practice. Among interesting future work directions mentioned by the author, we note with a particular interest the learning of animations instead of simple poses or the possible inclusion of kinetics in order to include some notion of time in the output animation.

5.1.4. Example-guided shape interpolation paths

The simpler method of Gao et al. [GCLX16] uses the knowledge of a shape dataset in order to find *good* interpolation paths between a source and a target shape. Note that the source and target shapes are considered to be inside the dataset (the dataset can eventually be completed with those two shapes, as no heavy preprocessing is required by the technique). Objects in the dataset are represented with respect to a base model using patch-based Linear Rotation Invariant (LRI) coordinates. The LRI representation of meshes was introduced by Lipman et al. [LSLCO05], and is composed of a first order differential representation of the directed edges, which are

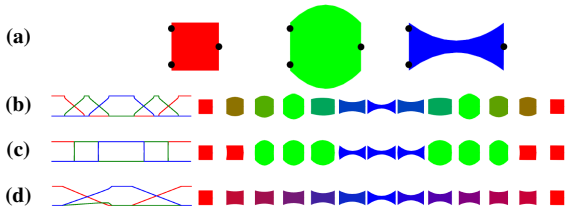

Figure 14: *Comparative results of Wampler [Wam16]. (a) The pose dataset, where the red cube should puff outwards when stretched (green) and then thin inward if further stretched (blue). The 3 handles of this dataset are the black dots. The rightmost handle is dragged to stretch then unstretch in the following subfigures, with blending weights over time shown at the left. (b) Method of Wampler [Wam16]. (c) Energy interpolation of Chao et al. [CPSS10]. (d) Linear shape space.*

expressed with respect to local frames attached to the vertices, and of a second order differential representation expressing change of local frames across an outgoing directed edge in the vertex' local frame. Such a representation is invariant to local rotations of the geometry, necessitates two linear solves to recover the geometric embedding from a set of LRI coordinates, and have been used successfully for mesh interpolation.

With the input source and target meshes in hand, an initial interpolation path is found using Dijkstra's algorithm over the k-nearest neighbors graph in the shape space (each shape in the input data set being a point with LRI coordinates as geometric embedding). This initial path is not smooth however, and it is further refined to obtain a pleasing morphing model. Given P_k the interpolated path after k iterations, $\boldsymbol{x}_i^k$ its i^{th} sample, and $\{\boldsymbol{n}_{i,j}^k\}_{j=1}^m$ the m nearest models of $\boldsymbol{x}_i^k$ in the shape space ($m=6$ in their examples), the cost to deviate from a smooth interpolating path is defined by Gao et al. [GCLX16] as

$$\mathcal{E}_k = \sum_i \sum_j w_{i,j}^k \|\boldsymbol{x}_i^k - \boldsymbol{n}_{i,j}^k\|^2 + \gamma \sum_i \|\boldsymbol{x}_{i+1}^k - \boldsymbol{x}_i^k\|^2 + \lambda \|L_k P_k\|^2 + \delta \|P_k - P_{k-1}\|^2$$

where $w_{i,j}^k = exp(-\|\boldsymbol{x}_i^{k-1} - \boldsymbol{n}_{i,j}^k\|/\sigma)$, γ, λ, and δ are weights controlling which terms to prioritize, and L_k is a tridiagonal Laplacian matrix, whose entries are

$$\begin{cases} L_k(i,i) & = 1 \\ L_k(i,i-1) & = -\frac{\|\boldsymbol{x}_i^{k-1} - \boldsymbol{x}_{i-1}^{k-1}\|}{\|\boldsymbol{x}_i^{k-1} - \boldsymbol{x}_{i-1}^{k-1}\| + \|\boldsymbol{x}_{i+1}^{k-1} - \boldsymbol{x}_i^{k-1}\|} \\ L_k(i,i+1) & = -\frac{\|\boldsymbol{x}_{i+1}^{k-1} - \boldsymbol{x}_i^{k-1}\|}{\|\boldsymbol{x}_i^{k-1} - \boldsymbol{x}_{i-1}^{k-1}\| + \|\boldsymbol{x}_{i+1}^{k-1} - \boldsymbol{x}_i^{k-1}\|}. \end{cases}$$

The first term of $\mathcal{E}_k$ makes the path adhere more to shapes of the dataset and can be seen as a smooth projection operator over the shape space manifold, in the spirit of MLS schemes [ABCO*01]. The second term favors shorter paths, the third term favors smoother paths and the fourth term prevents large updates between successive iterations for numerical stability. This energy functional is quadratic and can be minimized efficiently by solving a linear system. Lastly, after each iteration, the path is resampled with an even spacing between samples.

To recover the shape on the path at an arbitrary position t, one can interpolate the LRI representations of the closest samples, and reconstruct the corresponding shape. Results of this method are shown in Figure 15.

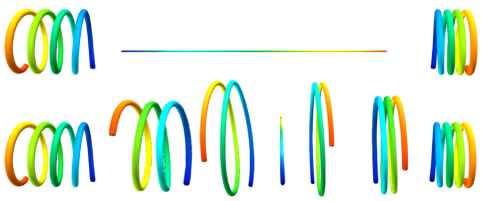

Figure 15: *Data-driven morphing as defined by Gao et al. [GCLX16]. Top: input shapes, with the source, intermediate and target model from left to right. Bottom: the morphing result.*

Analysis. Overall, the two methods of Wampler [Wam16] and Gao et al. [GCLX16] are able to learn the morphing pattern from a dataset, and interpolate between atoms of the dictionary using a high-level interface which hides the underlying complexity from the user in favor of an intuitive artistic control. It is worth mentioning that the technique of Gao et al. [GCLX16] does not create new shapes by just interpolating the atoms, but is also able to extrapolate new shapes from the dictionary. This indirect use of a dictionary is extremely interesting and novel, and could inspire further research in areas where dictionaries have not been used successfully so far because of a lack of flexibility. Performance-wise, both methods are suitable for fast editing; although the pre-computation phase of Wampler [Wam16] is significantly slower than the one of Gao et al. [GCLX16], its runtime phase is faster and allows real-time morphing, compared to interactive rates for the other.

5.2. Part assembly

All the methods previously described in this section take as input meshes with one-to-one vertex correspondence and use blending (or morphing) as combination operator. Another possibility is to consider meshes made of distinct parts, and use composition to combine these atoms together.

This trend was motivated by frameworks presented in the early 2000s such as Modeling by Example [FKS*04], which made the observation that modeling from scratch requires expert knowledge whereas reusing existing models could inspire and help beginners in the design of complex shapes. In this framework, a user could *search* for existing models, *select* part of them, and *combine* these parts to create a new model.

Recent approaches [KJS07, XXM*13, KCKK12, TYK*12] consider a predefined (co-)segmentation of shapes in the input dataset – thus effectively using dictionaries of shape parts derived from the input at their core, in order to amplify the input dataset by synthesizing meshes or accelerating the design process for users.

5.2.1. Design accelerator

These part-dictionaries enable easy workflow for non-expert users and allow them to quickly create interesting models, with methods such as Shuffler [KJS07] or Sketch-to-Design [XXM*13].

Figure 16: *The Shuffler [KJS07] system. The edited model is in the center, and is the result of the stitching of several parts, obtained by selecting them in the source models shown at the side. The segmentation for the user selection of parts is colored.*

For both methods, the workflow is to iteratively select a part of the shape to edit, choose between the candidates from a search by

Figure 17: *Overview of the workflow induced by Sketch-to-Design [XXM*13]. The user begins with an arbitrary shape (in gray) and can sketch (in red) the parts he wants to put in the final model. Parts found in the database are in blue, and symmetries are taken into account (e.g. the arms of the chair). It is possible to change the view for a better sketching experience.*

similarity in the database, merge the chosen part with the currently edited model, and repeat until the user achieves the desired goal.

The *Shuffler* system [KJS07] addresses the problem of modeling shapes through swapping of interchangeable components. It is worth noting that the authors of this article introduced a co-segmentation preprocessing step that was later reused in numerous part-based composition articles. In fact, this segmentation phase is often external now, and the input dataset is directly segmented and labeled.

At modeling time, the interface suggests to the user a currently edited model as well as other models that serve as source of components to be taken from (see Figure 16). Similar to SnapPaste [SBSCO06], the edited model and the candidate part are stitched together: boundaries are extended to overlap, then are snapped using soft ICP before generating the connectivity.

This modeling interface is relatively powerful, but starts being impractical when the dataset grows and it becomes unfeasible for the user to search through the entire database. By reversing the way parts are chosen (artists sketch what they want instead of being presented all possibilities), the Sketch-to-design method [XXM*13] solves this problem. In this approach, queries are sketches drawn on top of the currently-edited model and automatically matched to parts in the dictionary to be assembled into the final object (see Figure 17).

Based on the sketch-based retrieval method by Eitz et al. [ERB*12], the matching between user sketches and dictionary parts is built upon contour descriptors, which requires the dataset to be first preprocessed. This preprocessing phase consists in rendering the suggestive contours of each part for 169 different camera positions. In this phase, spatial relations between parts and symmetries are also learned (per part and per model), hence enriching the dictionary with relations between atoms. In the runtime phase, the user can change an object by sketching parts (for example sketching a new back for a chair), and the algorithm will return a ranked list of matched parts. The score of a part depends not only on its similarity with the user sketch, but also on its consistency with the rest of the model being edited.

As searching and computing scores on the whole set each time the user draws a sketch would be very expensive performance-wise, the score evaluation is limited to a subset of the dictionary

To achieve that, Xie et al. [XXM*13] use an auxiliary dictionary, whose atoms (forming a "visual vocabulary") are line features of contours. Given a sketch query, its representation on this auxiliary dictionary is computed and used to confine the search. Finally, match results are placed with respect to how parts of the same category are placed in the object, then snapped and deformed for their contact points (recorded during the preprocessing) to match, using the method of Müller et al. [MHTG05].

As shown in the two previous methods, improving the productivity of the user often requires efficiently searching huge datasets, a problem more visible now because of the availability of large object databases. Given a dataset, early methods only searched for the closest matching object (or ranked list of closest objects) based on the user query, without combining objects. For example, the method of Ovsjanikov et al. [OLGM11], in which the user modifies a template shape (a set of boxes) which is then transformed into shape descriptors used for the actual search, or the fuzzy exploration of Kim et al. [KLM*12], in which a user query is a part of an object. In short, these methods explore a dataset that is given as input.

On the contrary, the ShapeSynth system [AKZM14] allows the user to explore a dataset represented by a dictionary of parts, combined using part assembly. This means that the full dataset is the set of all possible combinations of the atoms in the dictionary. Here parts are stored as labeled segments from a set of compatibly labeled and segmented shapes. After a preprocessing phase on the input dataset, the user is presented with a 2D point cloud, where each point corresponds to an object of the dataset. By clicking in the blank space between the points, the system will generate a new object by combining parts from different shapes of the dataset.

In the preprocessing phase, a box template consisting in boxes encompassing the possible parts of the objects, is extracted [KLM*13]. Then, a descriptor vector of dimension $6t$ – where t is the number of boxes in the general template – is attached to each object; these descriptors represent the position and size of each box in the template. These descriptors are then projected in $\mathbb{R}^2$ by using Multi-Dimensional Scaling (MDS). Note that for performance reasons this is not done for all objects; instead a set of *landmark models* is selected to be projected using MDS, while the other models are defined by their barycentric coordinates with regards to the *landmark models*. Finally, points are clustered using mean shift clustering, and for each mode, the corresponding object can be re-projected; this process is repeated in a supervised manner, with the user clicking on the mode to reprocess. The resulting hierarchy is used to organize the data and help the synthesis.

In the runtime phase of ShapeSynth [AKZM14], when the user clicks between 2D points in the exploration view, the system

displays the corresponding template (descriptor in $\mathbb{R}^{6t}$, shown as a set of boxes), which can be locked when the user is satisfied with it for the boxes to be filled by parts that are to be least deformed when fit to their corresponding box. Thus the shown object might not be present in the initial dataset, but instead be the result of a combination of parts derived from the dataset. An overview of the user interface is shown in Figure 18.

Analysis. As seen in Sections 3.2 and 3.3, the absence of atoms in the dictionary directly translates into reduced quality of the results (including lack of artistic control). However, as illustrated for the last three methods [KJS07, XXM*13, AKZM14], difficulties may also arise when the dictionary contains lots of redundant atoms. While this might not be detrimental for purely automatic methods (apart from performance reasons), it may hinder the design process for users by providing them with too many similar choices. This motivates the development of techniques allowing for the intuitive exploration of large dictionaries, in order to present only the relevant variabilities to the user. Shuffler [KJS07] and Sketch-to-design [XXM*13] are more closely related than ShapeSynth [AKZM14], in the sense that the main difference lies in the retrieval criterion (geometric similarity vs sketches), whereas ShapeSynth also adds a more high-level view of the space of possible shapes. This exploration problem is less present in methods that automatically create models following the overall style of an input dataset, as will be shown in the next section.

5.2.2. Graph-based statistical set representation

A different class of methods uses graph-based representations that facilitate encoding the shape structure and the possible inter-dependence between different elements. In this context, prominent methods include the approaches of Chaudhuri et al. [CKGK11] and Kalogerakis et al. [KCKK12] that both take as input compatibly-segmented shapes into labeled parts (e.g., for human shapes: arm, leg, torso, ...), and operate in 2 major phases: a preprocessing phase during which a probabilistic model of the input data is derived and a runtime phase during which synthesis and modeling of new shapes is performed. In the context of this survey, the dictionary is the set of all parts, and the combination method is a composition under the constraint that the result fits

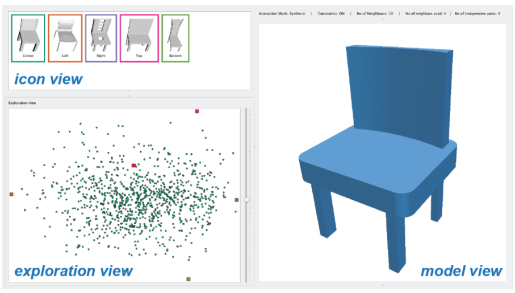

Figure 18: *Interface of the ShapeSynth system [AKZM14]. The* icon view *shows the representatives for each style, obtained via the hierarchical clustering; the* exploration view *presents the embedding of the shapes, and the* model view *displays the synthesized model.*

the derived probabilistic model. The nature of these probabilistic models differs a lot between the two frameworks, which we explain in detail in the next paragraphs.

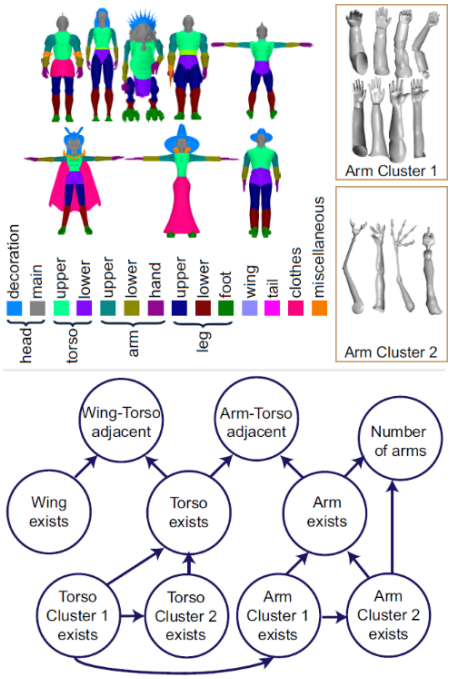

Figure 19: *From a dataset of segmented and labeled shapes (top left), the method of Chaudhuri et al. [CKGK11] starts by clustering segments inside label categories by geometric style (top right). Then a probabilistic model is learned (representative subset in bottom), encoding the dependencies between labels, geometric styles, part adjacencies, symmetries, and cardinality of each category.*

Learning Bayesian networks. Chaudhuri et al. [CKGK11] start by clustering the segments within each label category, based on descriptors such as the Shape Diameter Function [GSCO07], curvature, or Shape Context [BM00], and the different clusters give rise to the different so-called geometric styles of the components available in the label category.

The probabilistic model used by Kalogerakis et al. [CKGK11] is a Bayesian network [Pea88, KF09], which is a graph whose nodes are the observed variables (existence of label l in a shape, cardinality of label l in the shape, existence of adjacency of labels l and l', style $S_{s,l}$ describing whether a component of label l exists with geometric style s, and symmetry $R_{l,l'}$ describing whether a component of label l has a symmetric counterpart of label l') and whose directed edges represent the conditional dependency between them. Unconnected nodes represent conditionally independent variables. Each node also stores the probability of its variable taking a specific value given the values of its parents, and a shape can be viewed as an outcome of a joint probability distribution of all these variables. An overview of the method is shown in Figure 19.

The model is then learned by maximizing the Bayesian Information Criterion [Sch78] score on the observed variables:

$$\log P(O \mid G) \approx \log P(O \mid G, \theta) - \frac{1}{2} k_G \log(n)$$

where O is the set of observed variables (described before, such as the existence of a label, its cardinality, etc...), G the model to evaluate, θ the parameters of the model, k_G the number of free parameters for model G and n the number of shapes in the dataset. The model structure is found by exploring possible structures by adding and/or deleting edges (note that nodes are determined by the observed variables introduced previously).

At runtime, the user creates a shape progressively by selecting components and joining them together. Each time the user adds or removes a component, the model is used to sort components by likelihood, i.e. probability of it to be found in the the input dataset knowing the current partial shape, permitting the system to make suggestions for the user and enabling fast modeling. It is also possible to use this model to automatically generate new shapes, as done by Kalogerakis et al. [KCKK12] for results comparison.

This model is flat by nature, with no implicit nor explicit hierarchy, and this hinders its ability to generate plausible shapes automatically, as it only keeps track of direct dependencies between variables and not indirect ones, thus not tracking latent causes of structural variability. This observation is the main motivation for the technique introduced by Kalogerakis et al. [KCKK12], which we now detail.

The model of Kalogerakis et al. [KCKK12] is represented as a tree-like graph, whose nodes are random variables, and where latent unobserved variables (shape style, component style) are parent nodes of observed variables (number of components of a given category, continuous geometric descriptor and discrete geometric descriptor), as shown in Figure 20. It is possible to have lateral edges between observed variables, representing conditional dependencies. The core of this method is to learn latent sources of structural differences and relate these to the probabilistic relationships between components. This gives the model a higher-level understanding of the models, to better adapt compatible segments based on the shape and component styles.

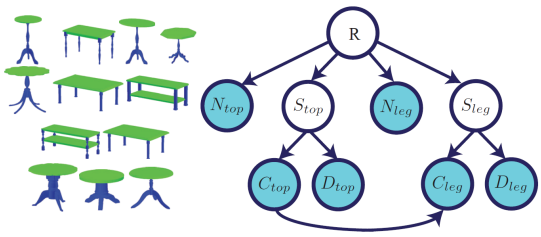

Figure 20: *The probabilistic model (right) of Kalogerakis et al. [KCKK12] learned on a small dataset of tables (left). R is the shape style, and for a category l, N_l is the number of components, S_l is the style of the components, $\boldsymbol{C}_l$ is the continuous geometric descriptor and $\boldsymbol{D}_l$ the discrete geometric descriptor. Here there are only 2 categories: $l \in \{top, leg\}$. Note that observed variables are in blue while inferred latent variables are in white.*

This model is learned by first computing the observable variables for each component, resulting in a feature-vector $\boldsymbol{O}$ (see Figure 20). The output probabilistic model G is the one with the highest probability given $\boldsymbol{O}$, which, as given by the Bayes rule, equals

$$P(G|\boldsymbol{O}) = \frac{P(\boldsymbol{O}|G)P(G)}{P(\boldsymbol{O})} \tag{2}$$

Since $P(\boldsymbol{O})$ does not change for different probabilistic models and $P(G)$ is assumed uniform over all possible structures, finding $G = \text{argmax}(P(G|\boldsymbol{O}))$ boils down to maximizing the marginal likelihood $P(\boldsymbol{O}|G)$. This is done by first determining the hierarchy (trying to add potential shape and component styles until the probability stops increasing), then finding lateral edges (adding or removing edges until the probability stops increasing).

By exploring the set of possible shapes of the probabilistic model, a collection of shapes can also be derived automatically to enrich the input dataset. For example, Kalogerakis et al. [KCKK12] used this strategy to synthesize 1267 new planes inspired by a training dataset composed of 100 airplanes only. Note that it is not specific to man-made objects, as demonstrated by the synthesis of animal shapes. It is possible to constrain the generation process, such as constraining possible shape or component styles, limiting the available component categories and/or explicitly specifying the set of acceptable components, in order to give the user a fine-grained control over the results of the generation.

Analysis. For both approaches [CKGK11, KCKK12] the dictionary atoms are parts and the combination method is a simple composition, however their use of second-order relations on the atoms is instrumental as it provides a simple, high-level representation of shapes that allows them to analyze the underlying structure of large sets of shapes, which is key to deriving a statistical model representing these datasets. The fact that the models are graphs helps to efficiently represent the joint probability distribution modeling the input dataset and also allow to extrapolate plausible new shapes. Results from both methods were compared in a user-study, confirming that the technique of Kalogerakis et al. [KCKK12] yields more plausible shapes. Both methods can be used in supervised mode, which make them closer to Shuffler [KJS07] than ShapeSynth [AKZM14] in terms of user experience, as they lack a high-level overview of the possible shapes space. However, compared to the grammar-based representation that we will review next, these methods remains limited regarding folding power i.e., "loops" (self-repeating motifs) are hard to reproduce.

5.2.3. Part grammars

Akin to graph-based representations, part grammars allow the user to create large variations of shapes. One of the earliest frameworks of this kind is the L-systems [PL90] (introduced in 1968 by Lindenmayer), which aimed originally at describing plants in an algorithmic manner. More recently, the data-driven method of Talton et al. [TYK*12] allows retrieving from the input dictionary of parts a grammar able to generate objects exhibiting a structure similar to the input training set of shapes (decomposed as labeled trees relating parts to each other). Figure 21 gives a visual overview of the whole method.

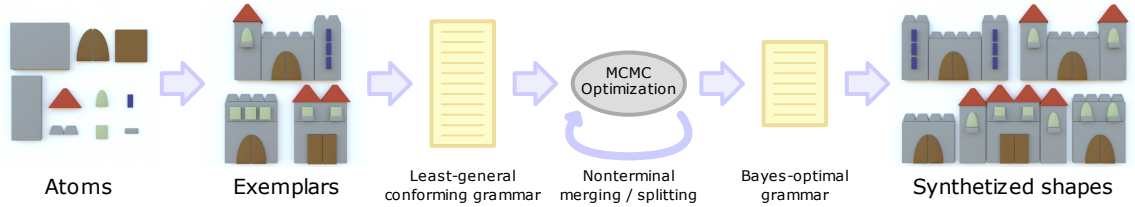

Figure 21: *Overview of the method of Talton et al. [TYK*12]. Starting with a dictionary of labeled parts and a set of exemplars created using these parts, the algorithm first find a grammar that can only produce the exemplars, before generalizing it using a Markov Chain Monte Carlo optimization. The resulting grammar is able to produce objects similar to the input exemplars.*

The grammars they consider are rigorously defined by a tuple $G =< V, T, \omega, R, \theta >$, where V is the set of non-terminal symbols, T is the set of terminal symbols, $\omega \in (V \cup T)^+$ is the axiom, $R \subset (V \rightarrow (V \cup T)^+)$ is the set of production rules, and $\theta : R \rightarrow [0,1]$ is the probability of the rules. The generation is done by iteratively replacing the current symbols by compatible productions, starting from the axiom. This algorithm stops when there are no non-terminals in current symbols (which imply that no production rule is compatible). In the approach of Talton et al. [TYK*12], production rules include geometric placement of introduced elements with respect to the replaced element. An example of such a grammar is visible in Figure 22.

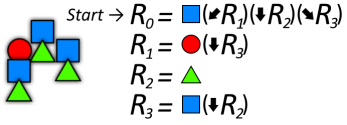

Figure 22: *An example of a grammar for shape generation. Starting with R_0 as initial word, successive replacements are made following the grammar rules (right), until no more non-terminals remain, resulting in the final shape (left)*

The first step of the pipeline proposed by Talton et al. [TYK*12] consists in generating a least-general conforming grammar, by processing all example trees iteratively: leaf nodes are represented by terminal symbols, and intermediate nodes are represented by non-terminal symbols; a production rule is added for these nodes, transforming the node symbol into its children symbols.

In a second step, this grammar is simplified using Bayesian inference in a way that is similar to what is done by Kalogerakis et al. [KCKK12], which results in a more general grammar (see Equation 2). Here O denotes the set of example designs. However, $P(G)$ is not assumed to be uniform, and simpler grammars are given a higher probability than complex ones (a grammar complexity being estimated by the length of its description). This maximization is done using the Markov Chain Monte Carlo algorithm.

Once the grammar is computed, it is used to generate large datasets, which exhibit similarities with the training set (see Figure 23). Note that this approach is not limited to 3D objects, and the authors also applied their algorithm to web page generation by learning the Document Object Model (DOM, the tree defining the document structure) of 30 web pages.

Analysis. What motivated the use of a concise dictionary of parts by Talton et al. [TYK*12] is that working on a few atoms allows keeping the complexity of the search over the possible assemblies reasonable, and results in a set of shapes exhibiting large variations while preserving the overall coherence of the set. As with previous methods [CKGK11, KCKK12], this requires having a model describing second-order constraints on the atoms (such as a Bayesian network or a grammar). This is interesting for performance reasons (by eliminating irrelevant possibilities) and also for perceptual reasons (coherent shapes with regards to the style of the examples). However, interactive control methods over these generation tools could be improved, so that desired shapes could be favored when sampling the possible realizations of the model. For instance, providing explicit means for the user to model second order constraints interactively could greatly improve the style control during modeling.

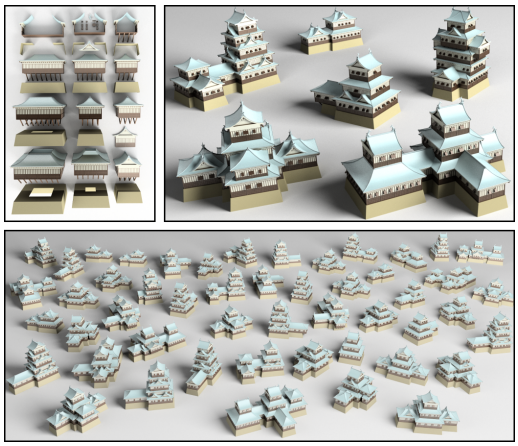

Figure 23: *Results obtained using the grammar learning of Talton et al. [TYK*12]. The inputs are the atoms of the dictionary (top left), and example compositions of these atoms (top right). After learning a grammar on these examples, it is possible to generate objects similar to examples (bottom).*

5.2.4. Set evolution

An interesting step further into the assistance in designing shapes and collections of shapes is the "Fit and Diverse" system [XZCOC12]. This technique introduces a genetic algorithm that allows *evolving* entire datasets, guided by user preferences. It relies on a crossover operator, allowing combining shapes that were pre-segmented into compatible parts stored in a part-dictionary.

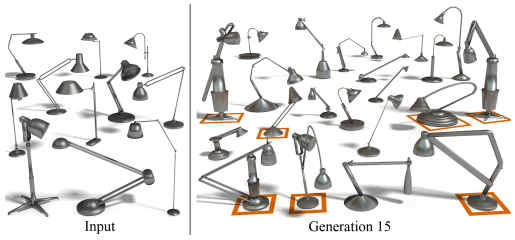

Figure 24: *Based on an initial set of object (left), the "Fit and Diverse" system [XZCOC12] uses a genetic algorithm to generate new objects similar to the input but also incorporating significant variations, with the aim of inspiring users.*

Contrary to the techniques previously described in this section, Xu et al. [XZCOC12] *voluntarily* compromise on the fitness of the evolved dataset, in order to preserve *set diversity* and offer shapes that are often surprising and inspiring to the user while preserving their overall geometric structure, thus encouraging *user creativity*. A rapid comparison of the results of this technique presented in Figure 24 with the results of Talton et al. [TYK*12] presented in Figure 23 illustrates this difference through more artistic applications.

Synthesis analysis Overall, the task of synthesizing new shapes with the help of dictionaries can be done in widely different ways, which could be distinguished between *continuous* and *discrete* methods. *Continuous* ones work with meshes with identical vertex set (including connectivity), which usually are the atoms of the dictionary. The combinations range from simple linear interpolation (with powerful artistic controls) [BV99, SQRH*16] to localized blending (controls can be coefficients [NVW*13, HYZ*14, WLZH16] or even inverse kinematics [SZGP05, FB11, Wam16]). It is even possible to have coherent extrapolation with regards to the input set of shapes, following an interpolation path specified by a few shapes [GCLX16]. *Discrete* methods instead work with dictionaries of parts, which are composed together (sometimes even stitched as in the Shuffler system [KJS07]). The key point then is to learn the second-order constraints between atoms; models used to represent these constraints differ widely, from custom descriptors (as in Sketch-to-Design [XXM*13]) to Bayesian networks [CKGK11] or even grammars [TYK*12]. This enable these techniques to synthesize shapes coherent with the style of the input examples, while allowing the generation algorithm to prune irrelevant realizations early for better performance.

6. Discussion

In this section, we give a summary of the presented approaches, focusing on the pervasiveness and applicability of dictionary-based approaches in different domains of geometric modeling. We also discuss the main concerns that could affect the use of dictionary-based methods, such as scalability to data size, generalization and the challenges induced by more sophisticated dictionary learning techniques. Finally, we describe the relation of dictionary-based approaches covered in this survey to methods based on deep learning and discuss the main open problems and possible future directions.

Application domain. In the context of shape recognition and retrieval, although some techniques use the classical bag-of-words representation, e.g., Shape Google [BBGO11], they typically do not allow recovering the geometry from the dictionary representation. When representing a set of shapes, the main purpose of a dictionary is to express its elements using a small base set that is usually over-complete, with the ability to go from coefficients to shapes, the latter property being crucial to representing the set of shapes in itself. Still, even without the reconstruction property, dictionaries can be used for classification, for example by doing PCA on a set of descriptors derived from the input objects. Dictionaries are not a silver bullet adapted to every possible use, and are sometimes less efficient than concurrent approaches making better use of the specificities of the problem; for example, while segmentation can be thought of as finding a dictionary of segments to cover a mesh, it is typically not performed this way in practice and recent more successful techniques mainly approach the problem using neural networks that are trained on the whole input dataset [XXLX14, GZC15].

In shape reconstruction, several methods consider different dictionaries, even when they have the same input: from a point cloud, the approach of Xiong et al. [XZZ*14] expresses the points with a dictionary, while other methods match shapes from an input collection with the point cloud in order, e.g., to complete scans [PMG*05] or to retrieve the object's structure [SFCH12].

In shape compression, dictionaries are often used for compactly representing point clouds, as shown by Digne et al. [DCV14] for instance. This is inspired by what is done in the image processing field, where approaches such as non-local means [BCM05] exploit the idea of using similar pixels for denoising, regardless of their position in the image. In general, mesh compression methods are not dictionary-based [MLDH15], except for mesh sequences compression [AM00, VS11].

In shape synthesis and geometric modeling, there are two major types of approaches: the first one, *continuous* in its nature, uses entire meshes as atoms and performs interpolation between them, which can either be linear as in Body Talk [SQRH*16] or more advanced as demonstrated by Gao et al. [GCLX16]. The second class of approaches, which are *discrete* in their nature, uses a dictionary of parts, obtained by segmenting and labeling the input dataset, and then combines them, using statistical models [KCKK12] or grammars [TYK*12] for example. Both kinds of approaches are often split into a preprocessing (or learning) phase and a runtime phase in which the learned model is used to synthesize new shapes. For the moment, methods in the first class require input meshes to be in perfect correspondence, i.e., to have the same number and ordering of vertices and the same connectivity. This requirement can be quite cumbersome as it is not verified by most datasets, with the exception of poses in an animation of the same initial mesh [SSK05, LCS13], and meshes registered to a template such as body capture [BV99, SQRH*16]. We name these methods *continuous* since interpolated shapes do not have subparts that can appear or disappear in a discrete fashion. Conversely, algorithms within

the second category lack the continuous interpolation properties, and instead do not have specific requirements, apart from having segmented and potentially consistently labeled shapes. They are often also more expressive, since for example using a grammar [TYK*12] allows for complex shapes that would not be easy to construct using simple interpolation.

Scalability. This is one of the main advantages presented by dictionary-based methods. For most presented methods, the runtime phase can approach real-time performance. For example, morphing bodies following a set of descriptors as shown in Body Talk [SQRH*16] can be rendered in 3D in a web browser without framerate drops. One reason for this is the drastic reduction of the size of the problem, i.e. doing the computationally demanding work only on a set of atoms instead of the whole dataset, i.e. on a significantly smaller set; this ability is perfectly demonstrated in the application of terrain amplification of Guérin et al. [GDGP16]. For specific man-made objects, it is possible to have small dictionaries for large datasets due to the similarity between objects, which improves the performance compared to doing the processing independently on each object. Using a dictionary also means drastically reducing the space of possible objects by removing ambiguous or impossible shapes. Not only this enable faster algorithms due to the limited set, but it also enable methods to do a second-order analysis on the atoms [vdHRD*15, KCKK12, TYK*12] for a better understanding of the whole set, which can be used to further reduce computations [SFCH12] or generate plausible shapes [KCKK12].

Generalization difficulty. One limitation of dictionary-based methods is that they work best when staying inside the given learned model, and generalizing learned structures and relations to a broader set of shapes is typically not reliable. For a given dictionary, it could be difficult to construct a specific object which is not similar to the ones in the initial set. However, having a set of atoms can still be useful: for example, Sketch-to-Design [XXM*13] uses multiple dictionaries, one of object parts and one of object descriptors allowing faster search in the first dictionary. In other words, dictionaries are good for summarizing a dataset, while making it more sparse, but their extrapolation power is limited.

More advanced dictionary learning. PCA and K-SVD are not the only dictionary learning algorithms. The dictionary representation itself can be different, as with translation-invariant dictionaries [ESH07], multiscale dictionaries [MSE08] or sparse dictionaries [RZE10] (where atoms are assumed to have a sparse representation over a second dictionary, so $D_1 = D_2 C_{atoms}$). Furthermore, most methods covered above use two passes, the first in which the dictionary is learned and the second pass where it is used. This is called offline learning, and assumes the data is entirely available and fits in memory. There are situations where such assumptions do not hold, for example when the data is received as a stream, in which case one can use online dictionary learning [MBPS10]. Dictionary learning is actively developed in signal and image processing, for denoising, deblurring, compression, separation, inpainting or computer vision [Ela10, Mai10], which may inspire geometric counterparts. Similarly, in addition to classical methods such as The Method of Optimal Direction (MOD) [EAH99], and the K-SVD algorithm [AEB06], sparse decomposition methods [EA06] are widely applied to image denoising or enhancement. They rely on the assumption that each patch of an image can be decomposed as a sum of a small number of atoms from a dictionary.

Such a framework is also well suited to texture modeling [Pey09] where new texture patches can be created from a learned dictionary, simply by imposing a sparsity constraint on the use of the dictionary atoms. In a more recent work [TGP15], a variational approach for texture synthesis is proposed which uses a sparse, patch-based dictionary and allows the reconstruction of geometric textures efficiently, thanks to constraints on the spectrum of images.

Relations to deep learning. Dictionary learning and deep learning are two flavors of machine learning, which appear as complementary when it comes to *representation* learning. Both methodologies can be used to automatically discover and exploit characteristic features of a specific class of raw input objects, to give rise to higher level parametric models of the class. Doing so, they make possible to generate new instances of the class by interpolating or even extrapolating from the learned subspace. Deep neural network architectures are numerous, with in particular *convolutional neural networks* (CNNs) which have been successfully used for large scale visual recognition [KSH12, SZ14]. They are typically built by stacking 1D or 2D layers of neurons, with a spatial resolution that progressively diminishes and local convolutions to connect neurons from one layer to the next [LBBH98]. At training phase, input samples (e.g., images) are propagated through the network, a loss function is computed on the output w.r.t. ground truth data and the resulting error is propagated backward in the network, optimizing for the network weights using variants of the gradient descent method. Several deep learning architectures based on CNNs have been used for 3D geometric modeling. In particular, *Autoencoders* (AEs) [Ben09] have recently become popular to reveal a latent space characterizing a set of objects. The are composed of two sub-networks: (i) the *encoder* which takes the form of a CNN mapping the input data to a layer of neurons of reduced size, producing a sparse code at the center of the AE and (ii) the *decoder* which symmetrically amplifies sparse codes to progressively higher resolution layers. The center layer ensuring the transition between encoding and decoding is usually referred as the *space of codes*. Learning with such architectures is performed by minimizing the deviation between input data and encoded-then-decoded data. New data may then be synthesized by sampling the space of codes and decoding. The properties of the resulting data interpolation/extrapolation are still intensively studied in many visual computing scenarios.

Thus, AEs and dictionary learning share similar purposes [OF96, RMA*17] but AEs have typically more parameters (sizes of the hidden layers) which are often hard to tune, resulting in a tedious try-and-test process on a per-problem (or per-dataset) basis. Although they use simpler operators (e.g., convolutions, maxout, dropout) which maps well to parallel processors such as GPUs, the space of code may be quite unintuitive and hard to navigate which, compared to shape atoms, is a weakness when it comes to 3D modeling. Instead of finding a sparse representation of the

input signal, *variational autoencoders* (VAE) [KW13] learn the probability distribution of the input, which eases the generation of new plausible data. *Generative Adversarial Networks* (GAN) [GPAM*14] are composed of two networks, a generator aiming to create plausible data and a discriminator aiming to distinguish between real and generated data; they are more precise than VAEs but also more finicky to train. These two methods can also be made complementary, by sharing the VAE decoder with the GAN generator [LSLW16]. Furthermore, these methods make possible to address data classification and synthesis at the same time. One promising research direction aims at combining deep learning and dictionary learning to exploit their complementary, using for instance multiple layers dictionaries [TMSV16] or dictionary learning to train hidden layers [SSM16].

Most applications of neural networks to geometric modeling are naturally derived from image scenarios, which naturally fit GPU computing. One key challenge in deep learning for 3D shapes is therefore to find a way to vectorize 3D data in order to set them as input/output to neural networks. *ShapeNets* [WSK*15] chooses to cast 3D shapes as voxel grids and uses a convolutional deep belief network [LGRN09] to either recognize a shape, complete a shape or indicate the most disambiguating camera placement for recognition. Wu et al. [WZX*16] and Jiang et al. [JM17] instead used a GAN (or a VAE-GAN as a variant) to perform better shape recognition and generation. With GRASS [LXC*17], Li et al. are able to handle structure (parts modeled as a set of boxes), with recursive VAE-GANs; the geometry of the parts is synthesized as a voxel grid obtained with another neural network. Unfortunately, the curse of dimension can already be observed in 3D, with dense voxel grids representations being too heavy and inefficient to train/infer from when it comes to higher resolution objects. As one can notice that most of the relevant information about a 3D shape is encapsulated by its surface boundary, vectorizing the surface only to feed deep learning architectures appears as a natural alternative. In particular, Sinha et al. [SUHR17] use geometry images [GGH02] to better scale, although their method is limited to genus-0 surfaces. Following *Sketch-to-Design* [XXM*13] (previously discussed), Lun et al. [LGK*17] completely reconstruct a shape from sketches, without parts dictionary, by training AEs on a set of predefined views. The vectorized shape model (network inputs and output) takes the form of depth and normal (2D) maps from which one can recover a point cloud. Some methods directly use point clouds for 3D point cloud reconstruction from a single image [FSG16], classification [QSMG17] and exploration and synthesis [NW17]. The two formers define their custom learning architecture while the latter uses an AE. For an overview of the numerous recent approaches to 3D deep learning, we refer the reader to the recent work conducted by Wang et al. [WLG*17].

Open problems. The first challenge related to meshes that we identify is to be able to store meshes of different connectivities and different number of vertices in a dictionary. At the moment, existing methods [SQRH*16, Wam16, SSK05] require perfect vertex correspondence in order for the interpolation to remain simple, which limits their usability. Such a problem is fundamentally related to abstracting the inner representation, to have a dictionary of *shapes*, which can accommodate objects regardless of their underlying representation e.g., parametric or implicit. One possible direction is to reformulate the notion of shape atoms through basic geometric operators only (e.g., intersection test, distance estimation, etc.), hiding completely the way surfaces are stored.

The limited extrapolation power of dictionaries, which we mentioned earlier, makes necessary the rapid updates of the dictionary atoms, given some user inputs, which would require on-line learning strategies. Complementary to editing efficiently the atoms during the optimization of the dictionary, we suspect that spatially-varying combination operators would be needed to be able to take into account user inputs, specifically in regions of the constructed space where input shapes are under-represented. Such flexibility would allow for larger extrapolation while providing user-friendly modeling interfaces. In order to investigate regions, which might require user inputs, efficient visualization of the dictionary and its relation to the input dataset might be useful.

Another largely unexplored direction lies in combining dictionary-based techniques with more advanced learning methods, including those based on deep learning, which has recently shown a remarkable success in shape analysis [BKR*16]. One advantage of dictionary-based approaches is that they result in atoms, which can often be interpreted and explicitly re-combined into new shapes, unlike deep learning-based methods, which are more difficult to interpret (deep learning interpretability is a current research subject [BZK*17]). Some very recent work in point cloud processing [QSMG17] has proposed constructing and learning so-called symmetry functions that capture structural shape properties. Such functions, which are learned using a neural network architecture can, at an abstract level, be considered as atoms and it would be interesting to explore this connection more fully.

Furthermore, most methods use dictionaries that are simple to understand (atoms are whole meshes or mesh segments) but not specifically designed for the processing algorithms. One approach in this direction is the mesh reconstruction of Xiong et al. [XZZ*14], where the final mesh *is* the dictionary. Following the idea of having the best representation for a specific algorithm, and changing representations between different phases of the pipeline, one can think of having dictionaries where the processing on the whole shape can be obtained by modifying the atoms and then reconstructing, therefore using dictionary to model shape alterations instead of shapes alone. This type of approach is used in image processing, e.g., for image denoising [EA06, BSF13], but has received limited attention in geometric modeling.

Last, beyond pure geometric representation, dictionaries seem particularly adapted to code and factorize shape-dependent objects, such as light fields or visibility complexes. Such objects are typically both highly complex and redundant, while many of their application scenarios can handle a certain degree of approximation as long as their synthesis is fast enough. Dictionaries may play a major role in making such objects practical and dynamic in graphics applications, following the emerging trend which consists in substituting pure simulation by inference from learned models.

Acknowledgments

All figures are courtesy of the authors of the article cited in their caption. We also thank Chris Platz for designing, building, and texturing the 3d models used in Figure 23. Parts of this work were supported by the ERC Starting Grant EXPROTEA (StG-2017-758800), chaire Jean Marjoulet from Ecole Polytechnique, a Google Focused Research Award, the French National Research Agency (ANR) under grant ANR 16-LCV2-0009-01 ALLEGORI and BPI France, under grant PAPAYA.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proceedings of the Conference on Visualization '01* (Washington, DC, USA, 2001), VIS '01, pp. 21–28. 14

[AD01] ALLIEZ P., DESBRUN M.: Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, pp. 195–202. 9

[AEB06] AHARON M., ELAD M., BRUCKSTEIN A.: K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing 54*, 11 (Nov 2006), 4311–4322. 3, 20

[AKZM14] AVERKIOU M., KIM V., ZHENG Y., MITRA N. J.: Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. *Computer Graphics Forum (Special issue of Eurographics 2014)* (2014). 5, 15, 16, 17

[AM00] ALEXA M., MÜLLER W.: Representing animations by principal components. *Computer Graphics Forum 19*, 3 (2000), 411–418. 5, 9, 10, 19

[AME*14] AUBRY M., MATURANA D., EFROS A., RUSSELL B., SIVIC J.: Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), pp. 3762–3769. 1

[BBGO11] BRONSTEIN A. M., BRONSTEIN M. M., GUIBAS L. J., OVSJANIKOV M.: Shape Google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics 30* (2011), 1:1–1:20. 3, 19

[BCM05] BUADES A., COLL B., MOREL J.-M.: A non-local algorithm for image denoising. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02* (Washington, DC, USA, 2005), CVPR '05, pp. 60–65. 19

[Ben09] BENGIO Y.: Learning deep architectures for ai. *Found. Trends Mach. Learn. 2*, 1 (2009), 1–127. 20

[BKR*16] BRONSTEIN M., KALOGERAKIS E., RODOLA E., MASCI J., BOSCAINI D.: Deep learning for shape analysis. In *Proceedings of Eurographics Tutorials* (2016), EG '16, pp. 2:1–2:1. 21

[BM00] BELONGIE S., MALIK J.: Matching with shape contexts. In *2000 Proceedings Workshop on Content-based Access of Image and Video Libraries* (2000), pp. 20–26. 16

[BPC*11] BOYD S., PARIKH N., CHU E., PELEATO B., ECKSTEIN J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning 3*, 1 (2011), 1–122. 4

[Bre01] BREIMAN L.: Random forests. *Machine Learning 45*, 1 (2001), 5–32. 5

[BSF13] BECKOUCHE S., STARCK J.-L., FADILI J. M.: Astronomical Image Denoising Using Dictionary Learning. *Astronomy and Astrophysics - A&A 556*, A132 (2013), 14 pp. 21

[BV99] BLANZ V., VETTER T.: A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1999), SIGGRAPH '99, pp. 187–194. 4, 5, 11, 12, 19

[BVZ01] BOYKOV Y., VEKSLER O., ZABIH R.: Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 11 (2001), 1222–1239. 5, 8

[BZK*17] BAU D., ZHOU B., KHOSLA A., OLIVA A., TORRALBA A.: Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition* (2017). 21

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., XIAO J., YI L., YU F.: *ShapeNet: An Information-Rich 3D Model Repository*. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 1

[CK10] CHAUDHURI S., KOLTUN V.: Data-driven suggestions for creativity support in 3d modeling. *ACM Transactions on Graphics (Proc. Siggraph Asia) 29*, 6 (Dec. 2010), 183:1–183:10. 1

[CKGK11] CHAUDHURI S., KALOGERAKIS E., GUIBAS L., KOLTUN V.: Probabilistic reasoning for assembly-based 3D modeling. *ACM Transactions on Graphics (Proc. Siggraph) 30*, 4 (2011), 35:1–35:10. 4, 5, 16, 17, 18, 19

[CPSS10] CHAO I., PINKALL U., SANAN P., SCHRÖDER P.: A simple geometric model for elastic deformations. *ACM Transactions on Graphics (Proc. Siggraph) 29*, 4 (2010), 38:1–38:6. 13

[CS96] CHEESEMAN P., STUTZ J.: Advances in knowledge discovery and data mining. Menlo Park, CA, USA, 1996, ch. Bayesian Classification (AutoClass): Theory and Results, pp. 153–180. 5

[DCV14] DIGNE J., CHAINE R., VALETTE S.: Self-similarity for accurate compression of point sampled surfaces. *Computer Graphics Forum 33*, 2 (2014), 155–164. 5, 9, 10, 19

[DUNI10] DROST B., ULRICH M., NAVAB N., ILIC S.: Model globally, match locally: Efficient and robust 3d object recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (June 2010), pp. 998–1005. 7

[EA06] ELAD M., AHARON M.: Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing 15*, 12 (Dec 2006), 3736–3745. 20, 21

[EAH99] ENGAN K., AASE S. O., HUSOY J. H.: Method of optimal directions for frame design. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)* (1999), vol. 5, pp. 2443–2446 vol.5. 20

[Ela10] ELAD M.: *Sparse and redundant representations, from theory to applications in signal and image processing*. New York, NY, USA, 2010. 4, 20

[ERB*12] EITZ M., RICHTER R., BOUBEKEUR T., HILDEBRAND K., ALEXA M.: Sketch-based shape retrieval. *ACM Transactions on Graphics (Proc. Siggraph) 31*, 4 (2012), 31:1–31:10. 15

[ERH*11] EITZ M., RICHTER R., HILDEBRAND K., BOUBEKEUR T., ALEXA M.: Photosketcher: interactive sketch-based image synthesis. *IEEE Computer Graphics and Applications* (2011), 56–66. 3

[ESH07] ENGAN K., SKRETTING K., HUSØY J. H.: Family of iterative ls-based dictionary learning algorithms, ils-dla, for sparse signal representation. *Digital Signal Processing 17*, 1 (2007), 32 – 49. 20

[FB81] FISCHLER M. A., BOLLES R. C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM 24*, 6 (1981), 381–395. 5

[FB11] FRÖHLICH S., BOTSCH M.: Example-driven deformations based on discrete shells. *Computer Graphics Forum 30*, 8 (2011), 2246–2257. 5, 13, 19

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. In *ACM Transactions on Graphics (Proc. Siggraph)* (New York, NY, USA, 2004), SIGGRAPH '04, pp. 652–663. 1, 14

[FSG16] FAN H., SU H., GUIBAS L. J.: A point set generation network for 3d object reconstruction from a single image. *CoRR abs/1612.00603* (2016). 21

[GCLX16] GAO L., CHEN S.-Y., LAI Y.-K., XIA S.: Data-driven shape interpolation and morphing editing. *Computer Graphics Forum* (2016). 3, 5, 11, 12, 13, 14, 19

[GDGP16] GUÉRIN E., DIGNE J., GALIN E., PEYTAVIE A.: Sparse representation of terrains for procedural modeling. *Computer Graphics Forum (Proc Eurographics) 35*, 2 (2016), 177–187. 5, 9, 10, 20

[GG07] GUENNEBAUD G., GROSS M.: Algebraic point set surfaces. In *ACM Transactions on Graphics (Proc. Siggraph)* (New York, NY, USA, 2007), SIGGRAPH '07. 6, 7

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. *ACM Trans. Graph. 21*, 3 (July 2002), 355–361. 21

[GHDS03] GRINSPUN E., HIRANI A. N., DESBRUN M., SCHRÖDER P.: Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 62–67. 13

[GM75] GLOWINSKI R., MARROCO A.: Sur l'approximation, par éléments finis d'ordre un, et la résolution, par penalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires. *Revue Française d'Automatique, Informatique, Recherche Opérationnelle. Série Rouge 9*, 2 (1975), 41–76. 4

[GM76] GABAY D., MERCIER B.: A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications 2*, 1 (1976), 17 – 40. 4

[GPAM*14] GOODFELLOW I. J., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., COURVILLE A., BENGIO Y.: Generative adversarial networks. *ArXiv e-prints* (June 2014). 21

[GSCO07] GAL R., SHAMIR A., COHEN-OR D.: Pose-oblivious shape signature. *IEEE Transactions on Visualization & Computer Graphics Issue No. 02 - March/April (2007 vol. 13)* (March 2007), 261–271. 16

[GZC15] GUO K., ZOU D., CHEN X.: 3d mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics 35*, 1 (2015), 3:1–3:12. 19

[HMHB08] HUBO E., MERTENS T., HABER T., BEKAERT P.: Self-similarity based compression of point set surfaces with application to ray tracing. *Computers & Graphics 32*, 2 (2008), 221 – 234. 9

[HRS*14] HEEREN B., RUMPF M., SCHRÖDER P., WARDETZKY M., WIRTH B.: Exploring the geometry of the space of shells. *Comput. Graph. Forum 33*, 5 (Aug. 2014), 247–256. 13

[HWG14] HUANG Q., WANG F., GUIBAS L.: Functional map networks for analyzing and exploring large shape collections. *ACM Transactions on Graphics (Proc. Siggraph) 33*, 4 (2014), 36:1–36:11. 1

[HYZ*14] HUANG Z., YAO J., ZHONG Z., LIU Y., GUO X.: Sparse localized decomposition of deformation gradients. *Computer Graphics Forum 33*, 7 (Oct. 2014), 239–248. 5, 12, 19

[JM17] JIANG C., MARCUS P.: Hierarchical detail enhancing mesh-based shape generation with 3d generative adversarial network. *CoRR abs/1709.07581* (2017). 21

[KCKK12] KALOGERAKIS E., CHAUDHURI S., KOLLER D., KOLTUN V.: A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics 31*, 4 (July 2012), 55:1–55:11. 1, 3, 5, 14, 16, 17, 18, 19, 20

[KF09] KOLLER D., FRIEDMAN N.: *Probabilistic Graphical Models : Principles and Techniques*. 2009. 16

[KG04] KARNI Z., GOTSMAN C.: Compression of soft-body animation sequences. *Computers & Graphics 28*, 1 (2004), 25 – 34. 5, 9, 10

[KH13] KAZHDAN M., HOPPE H.: Screened poisson surface reconstruction. *ACM Trans. Graph. 32*, 3 (July 2013), 29:1–29:13. 7

[KJS07] KREAVOY V., JULIUS D., SHEFFER A.: Model composition from interchangeable components. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (Washington, DC, USA, 2007), PG '07, pp. 129–138. 5, 14, 15, 16, 17, 19

[KLM*12] KIM V. G., LI W., MITRA N. J., DIVERDI S., FUNKHOUSER T.: Exploring Collections of 3D Models using Fuzzy Correspondences. *ACM Transactions on Graphics (Proc. Siggraph) 31*, 4 (2012). 15

[KLM*13] KIM V. G., LI W., MITRA N. J., CHAUDHURI S., DIVERDI S., FUNKHOUSER T.: Learning part-based templates from large collections of 3D shapes. *ACM Transactions on Graphics (Proc. Siggraph) 32*, 4 (jul 2013). 5, 15

[KMYG12] KIM Y. M., MITRA N. J., YAN D.-M., GUIBAS L.: Acquiring 3d indoor environments with variability and repetition. *ACM Transactions on Graphics (Proc. Siggraph Asia) 31*, 6 (2012), 138:1–138:11. 5, 8

[KSH12] KRIZHEVSKY A., SUTSKEVER I., HINTON G. E.: Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 2012, pp. 1097–1105. 20

[KV05] KALAIAH A., VARSHNEY A.: Statistical geometry representation for efficient transmission and rendering. *ACM Transactions on Graphics 24*, 2 (Apr. 2005), 348–373. 9

[KW13] KINGMA D. P., WELLING M.: Auto-encoding variational bayes. *ArXiv e-prints* (Dec. 2013). 20

[Lav11] LAVOUÉ G.: Bag of words and local spectral descriptor for 3d partial shape retrieval. In *Proceedings of the 4th Eurographics Conference on 3D Object Retrieval* (Aire-la-Ville, Switzerland, Switzerland, 2011), 3DOR '11, pp. 41–48. 3

[LBBH98] LECUN Y., BOTTOU L., BENGIO Y., HAFFNER P.: Gradient-based learning applied to document recognition. 2278 – 2324. 20

[LCS13] LUO G., CORDIER F., SEO H.: Compression of 3d mesh sequences by temporal segmentation. *Computer Animation and Virtual Worlds 24*, 3-4 (2013), 365–375. 5, 9, 10, 19

[LDGN15] LI Y., DAI A., GUIBAS L., NIESSNER M.: Database-assisted object retrieval for real-time 3d reconstruction. *Computer Graphics Forum 34*, 2 (2015), 435–446. 4, 5, 7, 8

[LGK*17] LUN Z., GADELHA M., KALOGERAKIS E., MAJI S., WANG R.: 3d shape reconstruction from sketches via multi-view convolutional networks. *CoRR abs/1707.06375* (2017). 21

[LGRN09] LEE H., GROSSE R., RANGANATH R., NG A. Y.: Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning* (2009), ICML '09, pp. 609–616. 21

[LSLCO05] LIPMAN Y., SORKINE O., LEVIN D., COHEN-OR D.: Linear rotation-invariant coordinates for meshes. In *ACM Transactions on Graphics (Proc. Siggraph)* (2005), pp. 479–487. 13

[LSLW16] LARSEN A. B. L., SØNDERBY S. K., LAROCHELLE H., WINTHER O.: Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of the 33nd International Conference on Machine Learning, ICML* (2016), pp. 1558–1566. 21

[LXC*17] LI J., XU K., CHAUDHURI S., YUMER E., ZHANG H., GUIBAS L.: Grass: Generative recursive autoencoders for shape structures. *ACM Trans. Graph. 36*, 4 (July 2017), 52:1–52:14. 21

[Mai10] MAIRAL J.: *Sparse coding for machine learning, image processing and computer vision*. Theses, École normale supérieure de Cachan - ENS Cachan, Nov. 2010. 4, 20

[MBPS10] MAIRAL J., BACH F., PONCE J., SAPIRO G.: Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research 11* (Mar. 2010), 19–60. 20

[MGP06] MITRA N. J., GUIBAS L. J., PAULY M.: Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. 25*, 3 (July 2006), 560–568. 3

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. In *ACM Transactions on Graphics (Proc. Siggraph)* (New York, NY, USA, 2005), SIGGRAPH '05, pp. 471–478. 15

[MLDH15] MAGLO A., LAVOUÉ G., DUPONT F., HUDELOT C.: 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Computing Surveys 47*, 3 (2015), 44:1–44:41. 9, 19

[MPWC13] MITRA N. J., PAULY M., WAND M., CEYLAN D.: Symmetry in 3d geometry: Extraction and applications. *Comput. Graph. Forum 32*, 6 (Sept. 2013), 1–23. 3

[MSE08] MAIRAL J., SAPIRO G., ELAD M.: Learning multiscale sparse representations for image and video restoration. *Multiscale Modeling & Simulation 7*, 1 (2008), 214–241. 20

[MWZ*13] MITRA N., WAND M., ZHANG H. R., COHEN-OR D., KIM V., HUANG Q.-X.: Structure-aware shape processing. In *SIGGRAPH Asia 2013 Courses* (New York, NY, USA, 2013), SA '13, ACM, pp. 1:1–1:20. 2

[NDF14] NIESSNER M., DAI A., FISHER M.: Combining inertial navigation and icp for real-time 3d surface reconstruction. 13–16. 7

[NVW*13] NEUMANN T., VARANASI K., WENGER S., WACKER M., MAGNOR M., THEOBALT C.: Sparse localized deformation components. *ACM Transactions on Graphics (Proc. Siggraph Asia) 32*, 6 (2013), 179:1–179:10. 4, 5, 12, 19

[NW17] NASH C., WILLIAMS C. K. I.: The shape variational autoencoder: A deep generative model of part-segmented 3d objects. *Computer Graphics Forum 36*, 5 (2017), 1–12. 21

[NXS12] NAN L., XIE K., SHARF A.: A search-classify approach for cluttered indoor scene understanding. *ACM Transactions on Graphics (Proc. Siggraph Asia) 31*, 6 (2012), 137:1–137:10. 5, 8

[NZIS13] NIESSNER M., ZOLLHÖFER M., IZADI S., STAMMINGER M.: Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (Proc. Siggraph Asia) 32*, 6 (Nov. 2013), 169:1–169:11. 7

[OF96] OLSHAUSEN B. A., FIELD D. J.: Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature 381* (07 1996), 607–9. 20

[ÖGG09] ÖZTIRELI A. C., GUENNEBAUD G., GROSS M.: Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression. *Computer Graphics Forum* (2009). 6, 7

[OLGM11] OVSJANIKOV M., LI W., GUIBAS L., MITRA N. J.: Exploration of continuous variability in collections of 3d shapes. *ACM Transactions on Graphics (Proc. Siggraph) 30*, 4 (2011), 33:1–33:10. 1, 15

[Pav16] PAVLOV I.: Lzma sdk version 16.04. http://www.7-zip.org/sdk.html, apr 2016. 10

[Pea88] PEARL J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA, 1988. 16

[Pey09] PEYRÉ G.: Sparse Modeling of Textures. *Journal of Mathematical Imaging and Vision 34*, 1 (May 2009), 17–31. 20

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. New York, NY, USA, 1990. 17

[PMG*05] PAULY M., MITRA N. J., GIESEN J., GROSS M., GUIBAS L. J.: Example-based 3d scan completion. In *Proceedings of the Third Eurographics Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), SGP '05. 1, 5, 6, 7, 19

[PRK93] PATI Y. C., REZAIIFAR R., KRISHNAPRASAD P. S.: Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers* (Nov 1993), pp. 40–44 vol.1. 3

[QSMG17] QI C. R., SU H., MO K., GUIBAS L. J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). 21

[RBE10] RUBINSTEIN R., BRUCKSTEIN A. M., ELAD M.: Dictionaries for sparse representation modeling. *Proceedings of the IEEE 98*, 6 (June 2010), 1045–1057. 2

[RMA*17] RANGAMANI A., MUKHERJEE A., ARORA A., GANAPATHY T., BASU A., CHIN S. P., TRAN T. D.: Critical points of an autoencoder can provably recover sparsely used overcomplete dictionaries. *CoRR abs/1708.03735* (2017). 20

[RZE10] RUBINSTEIN R., ZIBULEVSKY M., ELAD M.: Double sparsity: Learning sparse dictionaries for sparse signal approximation. *IEEE Transactions on Signal Processing 58*, 3 (March 2010), 1553–1564. 20

[SBSCO06] SHARF A., BLUMENKRANTS M., SHAMIR A., COHEN-OR D.: Snappaste: an interactive technique for easy mesh composition. *The Visual Computer 22*, 9 (Sep 2006), 835–844. 15

[Sch78] SCHWARZ G.: Estimating the dimension of a model. *The Annals of Statistics 6*, 2 (03 1978), 461–464. 17

[SFCH12] SHEN C.-H., FU H., CHEN K., HU S.-M.: Structure recovery by part assembly. *ACM Transactions on Graphics (Proc. Siggraph Asia) 31*, 6 (2012), 180:1–180:11. 1, 5, 6, 7, 19, 20

[SKAG15] SUNG M., KIM V. G., ANGST R., GUIBAS L.: Data-driven structural priors for shape completion. *ACM Transactions on Graphics (Proc. Siggraph Asia) 34*, 6 (Oct. 2015). 5, 6, 7

[SMK08] SCHNABEL R., MÖSER S., KLEIN R.: Fast vector quantization for efficient rendering of compressed point-clouds. *Computers and Graphics 32*, 2 (Apr. 2008), 246–259. 9

[SMNS*13] SALAS-MORENO R. F., NEWCOMBE R. A., STRASDAT H., KELLY P. H. J., DAVISON A. J.: Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition* (Washington, DC, USA, 2013), CVPR '13, pp. 1352–1359. 5, 7, 8

[SQRH*16] STREUBER S., QUIROS-RAMIREZ M. A., HILL M. Q., HAHN C. A., ZUFFI S., O'TOOLE A., BLACK M. J.: Body talk: Crowdshaping realistic 3d avatars with words. *ACM Transactions on Graphics (Proc. Siggraph) 35*, 4 (2016), 54:1–54:14. 5, 11, 12, 19, 20, 21

[SSK05] SATTLER M., SARLETTE R., KLEIN R.: Simple and efficient compression of animation sequences. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2005), SCA '05, pp. 209–217. 9, 10, 19, 21

[SSM16] SINGHAL V., SINGH S., MAJUMDAR A.: How to train your deep neural network with dictionary learning. *CoRR abs/1612.07454* (2016). 21

[SUHR17] SINHA A., UNMESH A., HUANG Q., RAMANI K.: Surfnet: Generating 3d shape surfaces using deep residual networks. *CoRR abs/1703.04079* (2017). 21

[SXZ*12] SHAO T., XU W., ZHOU K., WANG J., LI D., GUO B.: An interactive approach to semantic modeling of indoor scenes with an rgbd camera. *ACM Transactions on Graphics (Proc. Siggraph Asia) 31*, 6 (2012), 136:1–136:11. 5, 8

[SZ14] SIMONYAN K., ZISSERMAN A.: Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014). 20

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. *ACM transactions on graphics 24*, 3 (2005), 488–495. 5, 12, 13, 19

[TGP15] TARTAVEL G., GOUSSEAU Y., PEYRÉ G.: Variational Texture Synthesis with Sparsity and Spectrum Constraints. *Journal of Mathematical Imaging and Vision 52*, 1 (2015), 124–144. 20

[Til15] TILLMANN A. M.: On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Processing Letters 22*, 1 (Jan 2015), 45–49. 3

[TMF07] TORRALBA A., MURPHY K. P., FREEMAN W. T.: Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 5 (2007), 854–869. 5

[TMSV16] TARIYAL S., MAJUMDAR A., SINGH R., VATSA M.: Deep dictionary learning. *IEEE Access 4* (2016), 10096–10109. 21

[TYK*12] TALTON J., YANG L., KUMAR R., LIM M., GOODMAN N., MĚCH R.: Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2012), UIST '12, pp. 63–74. 1, 4, 5, 14, 17, 18, 19, 20

[Vav09] VAVASIS S. A.: On the complexity of nonnegative matrix factorization. *SIAM J. on Optimization 20*, 3 (2009), 1364–1377. 3

[vdHRD*15] VAN DEN HENGEL A., RUSSELL C., DICK A., BASTIAN J., POOLEY D., FLEMING L., AGAPITO L.: Part-based modelling of compound scenes from images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 878–886. 5, 8, 20

[VS07] VÁŠA L., SKALA V.: Coddyac: Connectivity driven dynamic mesh compression. In *2007 3DTV Conference* (May 2007), pp. 1–4. 9

[VS09] VÁŠA L., SKALA V.: Cobra: Compression of the basis for pca represented animations. *Computer Graphics Forum 28*, 6 (2009), 1529–1540. 9

[VS10] VÁŠA L., SKALA V.: Geometry-driven local neighbourhood based predictors for dynamic mesh compression. *Computer Graphics Forum 29*, 6 (2010), 1921–1933. 9

[VS11] VÁŠA L., SKALA V.: A perception correlated comparison method for dynamic meshes. *IEEE Transactions on Visualization and Computer Graphics 17*, 2 (Feb 2011), 220–230. 9, 10, 19

[VTSSH15] VON-TYCOWICZ C., SCHULZ C., SEIDEL H.-P., HILDEBRANDT K.: Real-time nonlinear shape interpolation. *ACM Transactions on Graphics 34*, 3 (2015), 34:1–34:10. 13

[Wam16] WAMPLER K.: Fast and reliable example-based mesh ik for stylized deformations. *ACM Transactions on Graphics (Proc. Siggraph Asia) 35*, 6 (Nov. 2016), 235:1–235:12. 5, 11, 12, 13, 14, 19, 21

[WDAH10] WINKLER T., DRIESEBERG J., ALEXA M., HORMANN K.: Multi-scale geometry interpolation. *Computer Graphics Forum 29*, 2 (2010), 309–318. 12, 13

[WLG*17] WANG P.-S., LIU Y., GUO Y.-X., SUN C.-Y., TONG X.: O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (SIGGRAPH) 36*, 4 (2017). 21

[WLZH16] WANG Y., LI G., ZENG Z., HE H.: Articulated-motion-aware sparse localized decomposition. *Computer Graphics Forum* (2016), n/a–n/a. 5, 12, 19

[WSK*15] WU Z., SONG S., KHOSLA A., YU F., ZHANG L., TANG X., XIAO J.: 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 1912–1920. 21

[WZX*16] WU J., ZHANG C., XUE T., FREEMAN W. T., TENENBAUM J. B.: Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *CoRR abs/1610.07584* (2016). 21

[XfT16] XIE H., FENG TONG R.: Image meshing via hierarchical optimization. *Frontiers of Information Technology & Electronic Engineering 17*, 1 (2016), 32–40. 6

[XKHK15] XU K., KIM V. G., HUANG Q., KALOGERAKIS E.: Data-driven shape analysis and processing. *CoRR abs/1502.06686* (2015). 1, 2

[XSX*14] XU W., SHI Z., XU M., ZHOU K., WANG J., ZHOU B., WANG J., YUAN Z.: Transductive 3d shape segmentation using sparse reconstruction. *Computer Graphics Forum 33*, 5 (2014), 107–115. 1

[XWY*16] XU L., WANG R., YANG Z., DENG J., CHEN F., LIU L.: Surface approximation via sparse representation and parameterization optimization. *Computer Aided Design 78*, C (2016), 179–187. 2

[XWZ*15] XU L., WANG R., ZHANG J., YANG Z., DENG J., CHEN F., LIU L.: Survey on sparsity in geometric modeling and processing. *Graphical Models 82* (2015), 160 – 180. 2

[XXLX14] XIE Z., XU K., LIU L., XIONG Y.: 3d shape segmentation and labeling via extreme learning machine. *Computer Graphics Forum 33*, 5 (2014), 85–95. 1, 19

[XXM*13] XIE X., XU K., MITRA N. J., COHEN-OR D., GONG W., SU Q., CHEN B.: Sketch-to-design: Context-based part assembly. *Computer Graphics Forum* (2013). 5, 14, 15, 16, 19, 20, 21

[XZCOC12] XU K., ZHANG H., COHEN-OR D., CHEN B.: Fit and diverse: Set evolution for inspiring 3d shape galleries. *ACM Transactions on Graphics (Proc. Siggraph) 31*, 4 (2012), 57:1–57:10. 5, 18, 19

[XZWB05] XU D., ZHANG H., WANG Q., BAO H.: Poisson shape interpolation. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling* (New York, NY, USA, 2005), SPM '05, pp. 267–274. 13

[XZZ*14] XIONG S., ZHANG J., ZHENG J., CAI J., LIU L.: Robust surface reconstruction via dictionary learning. *ACM Transactions on Graphics (Proc. Siggraph Asia) 33* (2014). 1, 2, 5, 6, 7, 19, 21

[YK14] YUMER M. E., KARA L. B.: Co-constrained handles for deformation in shape collections. *ACM Transactions on Graphics (Proc. Siggraph Asia) 33*, 6 (2014), 187:1–187:11. 1

[YLÖ*16] YOON Y.-J., LELIDIS A., ÖZTIRELI A. C., HWANG J.-M., GROSS M., CHOI S.-M.: Geometry representations with unsupervised feature learning. vol. 00, pp. 137–142. 5, 9