DOI: 10.1111/cgf.12311
EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)
*Volume 33 (2014), Number 2*

# 3D Timeline: Reverse Engineering of a Part-based Provenance from Consecutive 3D Models

Jozef Doboš Niloy J. Mitra Anthony Steed

University College London

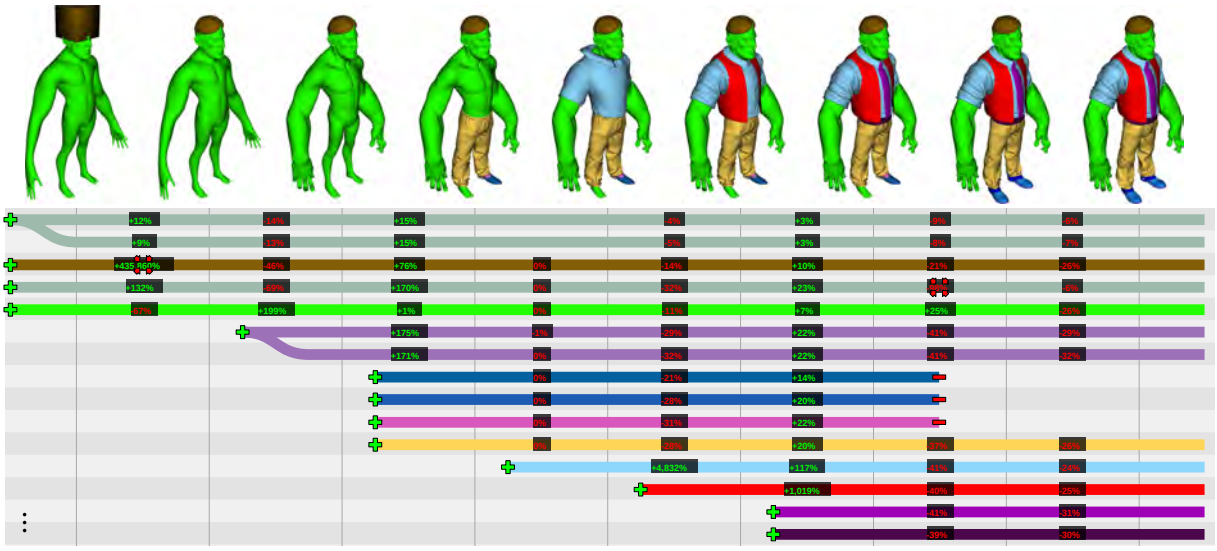

**Figure 1:** *Extracted and collapsed editing timeline (bottom) from 9 keyframes (top) of a genuine modeling sequence with over 7.6 million polygons in total. Legend to the detected operations is listed in §5 and the full timeline is in supplemental materials.*

**Abstract**

*We present a novel tool for reverse engineering of modeling histories from consecutive 3D files based on a timeline abstraction. Although a timeline interface is commonly used in 3D modeling packages for animations, it has not been used on geometry manipulation before. Unlike previous visualization methods that require instrumentation of editing software, our approach does not rely on pre-recorded editing instructions. Instead, each stand-alone 3D file is treated as a keyframe of a construction flow from which the editing provenance is reverse engineered. We evaluate this tool on six complex 3D sequences created in a variety of modeling tools by different professional artists and conclude that it provides useful means of visualizing and understanding the editing history. A comparative user study suggests the tool is well suited for this purpose.*

## 1. Introduction

As 3D geometry processing tools become more accessible, their use in applications ranging from games through to 3D printing is surging. Consequently, there is a growing need to inspect and organise large model collections. These might arise from archives of similar models that need to be categorised by type or shape. However, we identify a complementary problem; The organisation of models in a temporal domain of editing history. The problem arises since many tools do not save the editing history, and even if they do, only for a few recent steps. Such native histories can anyway be manually deleted and are also lost when exporting into interchange formats. Although most tools allow file names to be auto-increment and auto-saved, management of such files is

poorly supported. This leads to what we might call the *disk full of models problem*: There is a disk, or a version control repository, with various versions of models, but understanding the provenance of these potentially large datasets is hard due to little associated metadata. For example in Arup, a multinational engineering firm that is also one of Autodesk's largest customers, this problem is experienced frequently as projects span multiple specialized 3D software. Their practice is to periodically save snapshots of whole 3D files. Similar approach is also prevalent in creative industries, e.g., the former Black Rock Studios or Wham Bam Productions, where tools to explore model histories are sorely needed.

Therefore, we present a novel tool that takes sets of models and builds a visualization of a high-level provenance. The tool does this by reverse engineering a *plausible* history of parts (components) of a model and then by summarising important changes. These are displayed on a timeline that makes it easy to visually track the lifetime of each part and its relation to other parts. Using this tool the user would be able to answer important questions about the model history, such as the timing of a particular change, or the steps at which an error was introduced. Figure 1 shows an example.

Grossman et al. [GMF10] visualize the history of images in a timeline, but to do so they instrument an editor and video record the entire session. Their system has been extended to Autodesk's CAD software, too. Nevertheless, such a capture might not necessarily exist. This is certainly the case for the majority of legacy 3D models that one might want to inspect. Hence, our tool is agnostic to the editors that generated the models and takes as input complete files. It does not require any instrumentation, nor any editing sequence. In our demonstrations, we used models saved from Autodesk Maya, Pixologic ZBrush, Trimble Sketchup, Blender and Luxology Modo. The models that we tackle initially consist of millions of polygons. To make this tractable from an analysis point of view, and to achieve a concise visualization, our work is based on the observation that many models are composed of separate parts rather than single manifold surfaces. Models are thus often comprised of duplicated, symmetric or self-similar parts [MPWC13]. Hence, our focus is on reverse engineering the changes in parts and the aspects of duplication, provenance from common roots and instantiation.

Similarly to the very recent inverse image editing [HXM*13], we propose a set of simple geometric rules and methods to detect common edit operations and extract semantic provenance. We further provide a summarization by collapsing non-conflicting edits and removing redundant model snapshots. The analyzed and abstracted timeline can then be displayed in our custom viewer to quickly browse over the edit history and focus on key events. We tested the system on a range of models, often spanning 50-100 snapshots each with thousands of components, and recovered compact and informative edit summaries, see Fig. 8.

## 2. Related Work

Grabler et al. [GAL*09] generate visual tutorials of photo editing sessions based on author demonstrations in an instrumented version of GIMP. Chronicle by Grossman et al. [GMF10] supports document history exploration by linking the editing events and components of the UI into a video playback. The resulting video is indexed and hierarchically clustered. This, however, requires software instrumentation and large disk space for generated sequences. In the context of 3D models, Denning et al. [DKP11] obtain sequences using an instrumented Blender plug-in, which records all editing operations. These are clustered by analyzing the frequency of repeated operations using substituting regular expressions. In contrast, our method takes just a sequence of models as saved to disk. Even commercial products such as VisTrails [BCC*05] plug-in to Autodesk Maya preview recorded edits. We extract histories when instrumented editing software is unavailable.

**Shape analysis.** Over the years, researchers have investigated how to compute consistent alignments and correspondences within surface pairs [vKZHCO11], and on collections of 3D models [NBCW*11, HKG11]. Since in many contexts point-correspondence can be ambiguous and fuzzy, more abstracted part-based correspondence has been investigated. Golovinskiy and Funkhouser [GF09] first proposed consistent segmentation in the context of mesh pairs. They relied on rigid alignment and nearest neighbors to establish correspondences, and jointly segment all the input models into parts. Subsequently, several methods have been developed to address the problem of consistent segmentation and labeling by clustering points in an embedded space of local shape features [KHS10,HFL12,SvKK*11,WAvK*12] or using coupled modal analysis [KBB*13], either in supervised or unsupervised settings. Recently, Kim et al. [KLM*13] jointly optimize for correspondence, part segmentation, and part-level deformation to analyze model collections. Unlike these, our goal is to detect editing operations (mesh refinement, instancing), while the untouched parts of the meshes remain identical across the frames. The linear sequences result in models that are not suited for coanalysis, since geometric similarities are confined to neighboring frames.

**Mesh morphing.** In presence of point-level correspondence information across mesh pairs, early efforts in computer animation proposed multiresolution mesh interpolation frameworks [LDSS99, MKFC01]. Subsequently, algorithms have been developed to interpolate mesh collections by constructing and navigating underlying shape spaces [SZGP05]. Jain et al. [JTRS12] synthesize new 3D shapes by *interpolating* between a pair of input meshes. They segment the models into hierarchies of connected components via intersection detection. A single slider creates intermediate meshes by using the underlying contact and relation graphs. We too interpolate models via a slider, although, our challenge is to reverse engineer the edit trees from the input data.

**Version control.** Chen et al. [CWC11] propose a non-linear version control for image edits. They use a directed acyclic graph to represent sequences with graph nodes being edit operations and graph edges being the corresponding spatial, temporal, and semantic relationships. The recorded graphs are then visualized to display revision history. They also propose an easy-to-use interface to support common revision operations (e.g., review, replay, diff, addition, merging, etc.). In the context of 3D models, Doboš and Steed [DS12a, DS12b] used *two* and *three-way* differencing and merging on 3D assets and introduced a tool to merge changes in scene graph components while resolving conflicts by accepting or rejecting entire revisions. Correspondence was established assuming component-level unique name identifiers. More recently, in an interesting system, Denning and Pellacini [DP13] approximate edit distance as a cost of matching elements across two meshes. By treating faces and vertices as nodes of a graph with edges representing their adjacency relations, they convert the task of mesh differencing into a maximum common subgraph isomorphism problem. The method relies on spatial adjacency and does not handle modeling operations (e.g., instantiation, free-form shape manipulation, remeshing, etc.), which is the focus of our work.

## 3. System Overview

Given a set of consecutive 3D models, which we call *keyframes*, our goal is to reverse engineer a modeling tree that explains their relationships. We do this by decomposing the keyframes into parts, and for each part, track its *provenance*, i.e., how it is related to identical or similar parts in adjacent frames. The full modeling tree will then explain the life history of every identified model component. Being an inverse problem, the task is ambiguous with multiple intermediate edits potentially explaining the input model sequence. Hence we do not attempt to recover the actual history, nor do we attempt to exhaustively generate all possible permutations of the histories. Instead, we seek to infer a plausible flow of steps that fulfils our assumptions with regards to permitted editing operations, and provides a consistent explanation of the geometry found in the input models.

### 3.1. User Interface

From the user perspective, the detected operations together with their visualization and playback form the core output of our system, see Fig. 2. Even though modeling might be regarded as continuous in the temporal domain, we discretize the construction into individual events, see §5.1. In this work we focus on the following basic component-level operations: *addition* and *deletion*, changes in *polycount* and *size* of corresponding parts as well as detection of *transformations*, *duplications*, *instancing*, and *repeated copying*. As validated on various modeling workflows in §7, such operations are dominant in edit histories and understanding them provides valuable insight, albeit not necessarily capturing all of the steps an artist might have performed. To generate a timeline, the user first loads the model keyframes from which the system automatically extracts and estimates a component-level correspondence flow across the frames. Subsequently, in the key analysis phase, we imply edit operations and visualize them using a timeline metaphor (see supplementary video). The user can scrub through the timeline, as they might in a video editor, and an animation is created that demonstrates how the model evolves over time. Similarly to [GMF10], a timeline compression further simplifies the displayed information without violating extracted provenance (Fig. 1 bottom). Note that in rare cases when automatic correspondence fails, the user can manually override the assignment.

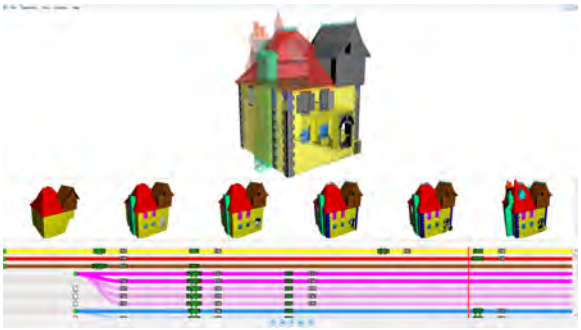

**Figure 2:** *Prototype GUI implemented in a cross-platform framework Qt. Morph window (top), keyframes with correspondence (middle) and estimated timeline (bottom).*

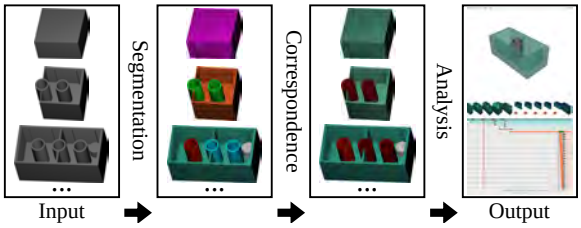


**Figure 3:** *Pipeline. A collection of consecutive models is loaded. In a pre-processing step the meshes are individually segmented and a part-based correspondence is estimated. This is analyzed and the implied editing history is visualized.*

### 3.2. Processing Pipeline

The input keyframes or modeling "snapshots" provide direct evidence from which our reverse engineering solution implies the missing editing steps that were not recorded, Fig. 3:

1. First, in pre-processing (§4), we perform an independent component analysis to extract model segments and establish the mutual correspondence across the frames.
2. Next, via semantic analysis (§5), we detect editing operations across the keyframes. The extracted operations are

then grouped, and possibly collapsed by merging non-conflicting edit operations across time.
3. Finally, the extracted provenance is visualized as a timeline (§6) with color-coded correspondence, marked individual events, and a playback option. The compressed timelines provide succinct edit summaries showing a quick overview, particularly for long sequences.

## 4. Pre-processing

Artists typically create, represent, and manipulate shapes as collections of *components* that are commonly supported by various primitive-based modeling tools. Hence, in contrast to coanalysis of model collections, we have a different problem (see §2). As demonstrated in [GF09], relying strictly on a pre-alignment of the meshes can yield poor results. Therefore, we do not make assumptions with regards to the initial global alignment of the models. Instead, we build upon the observation that early design stages are often characterized by *massing*, i.e., outlining of the most prominent volumes first before progressively adding detail later [ES11]. Hence, we focus on establishing a component-level correspondence across the frames, starting with the dominant largest components that then provide additional contextual information for the less prominent parts allowing factoring out of global rigid transforms. Furthermore, the neighbouring models in a construction sequence tend to be highly correlated with many sections being locally unmodified. We take advantage of this crucial characteristic and formulate a sequential correspondence estimation as explained in §4.1.

**Segmentation.** We use classical *hierarchical face clustering* [GWH01] to independently generate non-overlapping components as clusters. This algorithm builds a dual graph of a mesh surface such that nodes represent clusters initially seeded by individual faces while edges, ordered according to the cost of collapsing, their adjacency. At each iteration, the lowest cost edge is removed, its clusters are merged and the costs of their inherited edges are recalculated. Apart from clusters' planarity, the cost can express conformity to shapes such as spheres and cylinders (c.f., [AFS06]). The algorithm proceeds until the edges have been exhausted, effectively identifying disjoint manifold components.

### 4.1. Correspondence Flow Estimation

In pursuance of a correspondence flow

$$F = \{C_{t_i} \rightarrow C_{t_{i+1}} \rightarrow \ldots \rightarrow C_{t_{i+n}}\}, \tag{1}$$

i.e., an assignment of a component $C$ at time $t$ to a component $C'$ at time $t'$, it is simply not sufficient to find the most similar meshes since components can be deformed, refined, copied, etc. Rather, the exact same component needs to be identified and tracked across all keyframes so that its provenance can be reliably implied. Thus, the correspondence measure has to be robust to changes in shape and location, yet discriminate duplication.

**PCA-aligned bounding boxes.** First, we calculate a principal component analysis (PCA)-aligned bounding box so as to establish a rough descriptor for each component, c.f., [JTRS12]. PCA of vertices $v_i$ in a component $C$ weighted by the cumulative area $A_i$ of parent faces provides a transformation from the global to a local coordinate system. Let $C_\square[\bar{v}, w, h, d]$ be the bounding box of $C$ in this coordinate system with a centroid $\bar{v} = \sum_{i=1}^{n} A_i v_i / \sum_{i=1}^{n} A_i$, and $[w, h, d]$ its respective width, height and depth. This bounding box reflects the spread of vertices along principal axes, is rotation invariant and robust to local geometry modifications.

**Part-based hierarchy.** Building a part-based hierarchy is a common way of adding contextual support and relative localization to otherwise loose components [SSS*10, JTRS12]. Instead of building a complex tree of components, in the next step at each keyframe independently we start with a *forest* of one-level deep trees rooted at the largest components. Hence, a parent $C_P$ of a component $C$ is the one that contains its bounding box and has the largest volume. This provides localized systems where the largest components have an explicit global reference at the origin. Note that the parental component might not be the same part of a scene as components can be translated, duplicated, disconnected, etc.

**Correspondence estimation.** Based on the bounding boxes and the hierarchy, correspondence between components can be estimated. Let $E_S$ be a similarity error between two components $C$ and $C'$ defined as a $l^2$-norm of their bounding boxes irrespective of their centroids $\bar{v}, \bar{v}'$

$$E_S := \|C_\square[w, h, d] - C'_\square[w', h', d']\|. \tag{2}$$

We use this error measure to first group self-similar components in each keyframe independently, as shown in Fig. 4. Let $E_L$ be a localization error defined as an absolute difference of Euclidean distances of centroids of the component bounding boxes $C_\square, C'_\square$ to the centroids of the bounding boxes of their associated parental components $C_{P_\square}, C'_{P'_\square}$ as:

$$E_L := \left| \|\bar{v} - \bar{v}_P\| - \|\bar{v}' - \bar{v}'_{P'}\| \right|. \tag{3}$$

By combining definitions (2) and (3) for each pair of components in two frames we obtain an affinity matrix $S$ as a weighted sum of the similarity and localization errors:

$$S_{i,j} := \alpha E_S + (1 - \alpha) E_L. \tag{4}$$

**Correspondence propagation.** Finally, a greedy one-to-one assignment based on the affinity matrix $S$ estimates the initial correspondence from frame $t_i$ to $t_{i-1}$, see step (2) of Fig. 4. This assignment is then checked for consistency based on a majority vote within a group. Outliers at $t_{i-1}$ with correspondence assigned from a non-matching self-similarity group at $t_i$ are reassigned to the remaining components of the desired group at $t_i$. This enforces a consistent

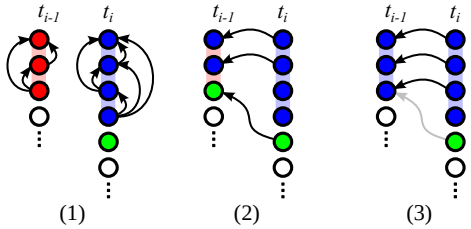


**Figure 4:** *Correspondence assignment from time $t_i$ to $t_{i-1}$. (1) Firstly, self-similarity groups are established independently within each timeframe. (2) Next, a one-to-one correspondence is assigned. (3) Finally, a consistency check ensures that all correspondences come from a single self-similarity group in $t_i$ based on a majority vote in $t_{i-1}$.*

flow between groups of components rather than the individuals, see step (3) of Fig. 4. We proceed through pairs of neighboring frames from the last to the first. If a correspondence cannot be reliably established in two neighbors, a frame can be skipped in order to try and find a match in the next frame repeating such an attempt until a suitable candidate is found or the beginning of the sequence is reached. Due to a membership in a self-similarity group, most components gain not only a one-to-one but also a one-to-many correspondences. This is illustrated in Fig. 8, where massive "funnels" represent extracted interconnected groups as duplication.

## 5. Semantic Analysis

As illustrated in (5), correspondence flows can be represented as a sparse binary $m \times n$ matrix $\Phi$, where $m$ is the number of flows and $n$ the number of keyframes.

$$\Phi_{m,n} = \begin{array}{c} \\ F_1 \\ F_2 \\ \vdots \\ F_m \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & \cdots & t_n \end{array} \\ \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m,1} & C_{m,2} & \cdots & C_{m,n} \end{pmatrix} \end{array} \tag{5}$$

Entries in $\Phi$ express a component $C_{i,j}$ being or not being present in a keyframe at time $t_j$. Hence, each row defines a single logical scene-part tracked over time, while columns are collections of components that belong to a particular keyframe. In such a representation the natural temporal ordering is from left to right, i.e., from the first to the last keyframe. By sorting the rows such that a presence in an earlier column $t$ and more overall entries across all columns are favored, we also gain a top to bottom temporal ordering.

### 5.1. Editing Operations

Once we have the correspondence flows in matrix $\Phi$, we can inspect the part-by-part changes between pairs of keyframes. Changes are classified into one or more of the following operations. We include visual icons for reference.

**Addition.** A component $C_{i,j}$ has been *added* between $t_{j-1}$ and $t_j$ iff it is the first in its self-similarity group and there is no associated corresponding component $C_{i,j-1}$.

**Deletion.** Conversely, a component $C_{i,j}$ has been *deleted* between $t_j$ and $t_{j+1}$ iff no correspondence at $t_{j+1}$ exists.

**Life-span.** The difference in time between a component being added and deleted represents its *life-span*.

**Duplication.** A component $C_{i,j}$ is a *duplicate* iff it is added but not a *template*, i.e., not the first in its group.

**Polycount increase.** A component $C_{i,j}$ has *increased in polycount* between $t_{j-1}$ and $t_j$ iff it has a larger number of polygons than $C_{i,j-1}$.

**Polycount decrease.** Conversely, a component $C_{i,j}$ has *decreased in polycount* between $t_{j-1}$ and $t_j$ iff it has a smaller number of polygons than $C_{i,j-1}$.

**Size increase.** A component $C_{i,j}$ has *increased in size* between $t_{j-1}$ and $t_j$ iff its bounding box volume is larger than that of $C_{i,j-1}$.

**Size decrease.** Conversely, a component $C_{i,j}$ has *decreased in size* between $t_{j-1}$ and $t_j$ iff its bounding box volume is smaller than that of $C_{i,j-1}$.

**Translation.** A component $C_{i,j}$ has been *translated* between $t_{j-1}$ and $t_j$ iff there is a difference $T$ in the global position of its bounding box centroid $\bar{v}_{i,j}$ to that of $C_{i,j-1}$ or to its template at $t_j$ if it is a duplicate.

**Repeated copy.** A component $C_{i,j}$ is a *repeated copy* between $t_{j-1}$ and $t_j$ iff it is a duplicate and its $T$ belongs to a list of at least 3 successive translations, see §5.2.

**Instancing.** A component $C_{i,j}$ is *instanced* iff it is a duplicate and its life-span operations match the template.

Our semantic labeling iterates through each row of $\Phi$ and for each column (keyframe) it detects these operations one-by-one via a lookup table. Certain operations such as changes in polycount and size can and often do occur simultaneously.

### 5.2. Repeated Copying Detection

Apart from instancing, another special case of duplication is repeated copying. A template which was duplicated and belongs to a self-similar group $G_i$ at time $t$ is a component with the largest lifetime. In a case of multiple components fulfilling this criterion, the choice is arbitrary. Repeated copying, however, differs from basic duplication in that the translation from the template to each copy is repetitive, i.e., can be expressed as an incremental succession of the same translation $T$ such that the most immediate copy is assigned $1 \times T$, the next $2 \times T$ and so on while $T$ is minimal. Essentially we are looking for a 1-parameter regular structure (c.f., [PMW*08]), where the component-based instances and repetitions are exact, and hence easier to discover. We focus on repeated copying that is equally spaced.

In order to unravel repeated copying, the detection algorithm proceeds as follows. Given a self-similarity group $G$, an arbitrary component $C_S \in G$ is selected as a seed. The aim

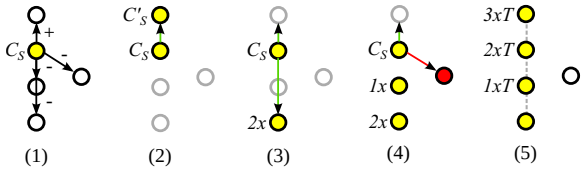


**Figure 5:** *Repeated copying detection. (1) Distances from an arbitrary seed $C_S$ are calculated. (2) The smallest vector defines the desired line direction. (3) Components with parallel vectors are selected and unparallel (4) rejected. (5) The furthest component becomes the new template.*

is to find at least 2 other components that would form a repeated sequence with $C_S$, their spatial ordering, the template component and the translation $T$ that governs the repetition. Firstly, distances from $C_S$ to all components are calculated, step (1) in Fig. 5. Let $\overrightarrow{AC_S}$ be a vector from any other arbitrary component $A \in G \wedge A \neq C_S$. A distance from $C_S$ to $B \in G$ is considered negative iff $\overrightarrow{AC_S} \cdot \overrightarrow{BC_S} < 0$. The components are then sorted according to the signed distance from the most negative to the most positive with $C_S$ being also included with a trivial non-negative zero distance to itself. Let $C'_S \neq C_S$ be the component with the smallest unsigned distance (in absolute terms) to $C_S$. Vector $\overrightarrow{C_S C'_S}$ then defines a line on which the repeated copying is expected to occur while its magnitude the desired repetition distance, step (2) of Fig. 5. Next, the components are checked one-by-one in the order of signed distances. Component $D$ is considered a member of a repeated copying subgroup iff $\|\overrightarrow{C_S D} \times \overrightarrow{C_S C'_S}\| = 0$, i.e., the vector from a seed component to $D$ is parallel with the desired line and $mod(\|\overrightarrow{C_S D}\|, \|\overrightarrow{C_S C'_S}\|) = 0$, step (3) of Fig. 5. If less than 3 components fulfil the condition, vector $\overrightarrow{C_S C''_S}$ with the next smallest magnitude is selected as the new seed vector and the process repeats with $C'_S$ removed from $G$. If, however, 3 or more components were found, these form the desired copying subgroup with the head of the list being the new template and the translation calculated as multiplies from it, step (5) of Fig. 5. The subgroup is removed from $G$ and the algorithm repeats until there is not enough components or all possible seed distances have been exhausted.

### 5.3. Timeline Compression

In the `Medieval` dataset (Fig. 2), for example, there are 510 individual components across the sequence, yet these form only 17 component groups altogether. Further, many editing operations are repeated across multiple components such as is the case of instancing while others are independent of each other. Our goal is to simplify the apparent complexity of the timeline matrix $\Phi$ while preserving the essence of the reverse engineered provenance. Therefore, we perform two analytically independent collapsing steps.

Instanced duplicates by definition have the same operations applied to them as do their templates. Hence, a *row-wise collapse* merges all instances into their parental components while remaining components are left unmodified. This significantly reduces the matrix height, in the case of the `Medieval` dataset from 189 to 28 rows. Operations in neighboring frames that do not affect the same correspondence flow are considered independent. It is thus possible to perform a *column-wise collapse* given the operations at $t_i$ do not collide with those at $t_{i-1}$ and vice versa, i.e., when operations do not occur in neighboring frames simultaneously.

## 6. Timeline Visualization

Representing events in a timeline visualization is a common way of abstracting complex temporal interactions into a meaningful easily understandable flow. Apart from linear dependencies, timelines can also display hierarchical information [SNF10] and even be used as a collaboration platform [BBB*10]. In computer graphics, timelines are mostly used for animation compositing such as is the case of many 3D authoring tools. We have chosen a hierarchical timeline as it matches the linear succession of the input data, yet enables us to display dependency relationships between components and their groups, see Fig. 2. Such a timeline encourages both manual exploration and automated playback. Our viewer, similarly to [DKP11], provides a main blending preview at the top and a sequence of thumbnail models ordered from left to right underneath. In addition, the reverse engineered provenance timeline is at the bottom. Models are initially colored based on their independent segmentation using a random color scheme, but once the correspondence estimation has been completed, the colors are consistent across the frames. It is possible to select a single component or their groups to highlight the corresponding parts in the remaining frames. The thumbnails can be navigated synchronously while exploring the main preview independently.

### 6.1. Timeline interface

The timeline is divided into equally spaced buckets that illustrate the elapsed time between neighboring frames, see vertical lines in Fig. 1. This is an approximation as there is no requirement for the input models to be developed in equal amounts of time. Alternating rows signify a correspondence flow each. Life-span of a component is visualized as a collection of cubic Bézier curves forming a path that can diverge from the assigned timeline row whenever a duplication has been detected. These are laid out from top to bottom in the order of a template followed by repeated copies and general duplicates. Their coloring is consistent with the correspondence assignment in the thumbnail 3D views. Instanced duplicates have decreased opacity for easy identification even when the timeline is not collapsed as shown in Fig. 2. Double clicking on a path highlights the assigned components in the thumbnail views. Vectorized icons of the

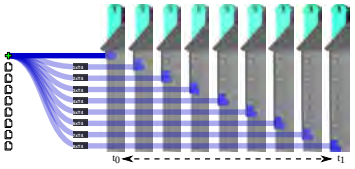

**Figure 6:** *Repeated copying blending (detected here on the* `Medieval` *dataset, see Fig. 2) is interpolated sequentially.*

detected editing operations, as listed in §5.1, are placed on top of the paths. From left to right the additions are laid out first, followed by relative changes in polycount and size with translations being the last between pairs of 3D models. Such an arrangement ensures that for a duplication the relative geometrical changes appear only once in the timeline while the transformations are placed directly on top of the matching paths. Even though there is no evidence that the events have happened in this particular order, it declutters the interface.

**Playback blending.** Just like in 2D animations, the time between two successive keyframes is linearly interpolated so that components at $t_j$ are morphed into their known state at $t_{j+1}$ via the detected editing operations. In addition, the opacity of a component $C_{i,j-1}$ changes from 100% to 0% while for $C_{i,j}$ it changes in reverse. Those components that do not change are shown in gray so that modifications stand out during playback. Repeated copies, however, change their opacity one-by-one for a pleasing visualization of the successive construction, see Fig. 6. Even though linearly interpolated, more emphasis can be put on certain operations by slowing down the playback speed for desired event classes.

## 7. Evaluation

We evaluated our tool on a variety of modeling sequences created by several artists in Autodesk Maya, Pixologic ZBrush, Trimble Sketchup, Blender and Luxology Modo authoring tools. Each sequence provides a distinct set of challenges including detailing, large number of polygons and even organic sculpting. Tab. 2 lists statistics for these sequences while Fig. 8 shows the implied timelines and some of the input models with assigned correspondence coloring. Even if the actual editing history existed, it would represent only one plausible evolution that achieves the same sequence, hence we do not attempt ground truth comparison. For Eq. (4) we used $\alpha = 0.3$ and for the repeated copying detection in §5.2, a zero tolerance threshold $\sigma = 0.0001$.

The timeline reveals deformations and other modifications that are hard to detect by visual inspection alone. For example, between the first two keyframes of the `Medieval` dataset (top of Fig. 8), there is a vertical stretch of the façade geometry with an upward translation of its roof structures. Even in a direct side-by-side comparison it is difficult to notice this, yet both the timeline and the blending preview reveal it precisely. At close inspection of the `Brick` and `Engine` datasets in Fig. 8, it can be also seen that the logos were created separately and put in place once completed.

An interesting revelation comes from the `Portico` dataset consisting of 158 distinct models, the largest sequence we have tested with our system. A common practice when creating massive 3D models is to temporarily disable or even remove certain parts of a scene in order to lower its memory footprint, hence unburden the editor. This hidden geometry is then reintroduced at a later stage when the full scene is required. As visible in the first 29 keyframes (represented by 7 green thumbnails at the beginning of the last sequence in Fig. 8), the modeler created a set of pillars with attempted detailing, which then disappear altogether. Later the columns reappear, this time, however, with a different height and shape. Unlike the common practice described above, our tool identifies this as a massive deletion followed by an addition as the reintroduced geometry significantly differs from that of the supposed original. We have reached out to the modeler who confirmed that it was indeed the intention to delete the first version of the columns so that the positioning and the height of the roof would govern their construction.

**MeshGit comparison.** MeshGit [DP13] focuses on vertex-level differencing and merging, hence a complementary reverse engineering subproblem. Being higher-level and component-based, our detection is faster and scales to bigger datasets. For example on `shuttle`, their largest example evaluated on comparable hardware, it takes 9.7 minutes vs. mere 14 seconds in 3D Timeline. On our datasets where there are strong changes in adjacency of vertices and faces, MeshGit matches large areas that do not correspond as repositioned, added or deleted, see `Engine` in Fig. 7. We can also handle modifications that translate components significantly, the main limitation discussed in their paper. Never-

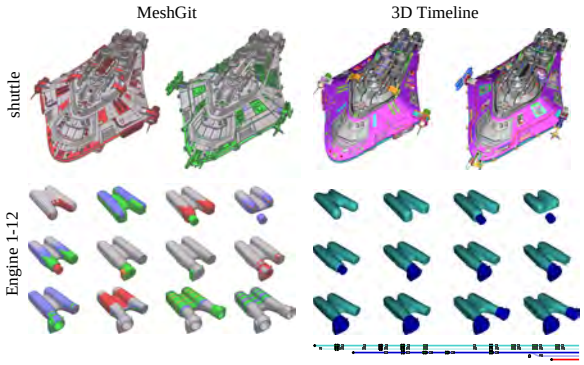


**Figure 7:** *MeshGit comparison with their* `shuttle` *model and the first 12 frames of our* `Engine` *dataset. MeshGit shows changes in adjacency (red/green), geometry (blue) and sequential changes (orange). Timeline shows unmodified component groups (gray) and modified (corresponding colors). More examples are in the supplemental materials.*

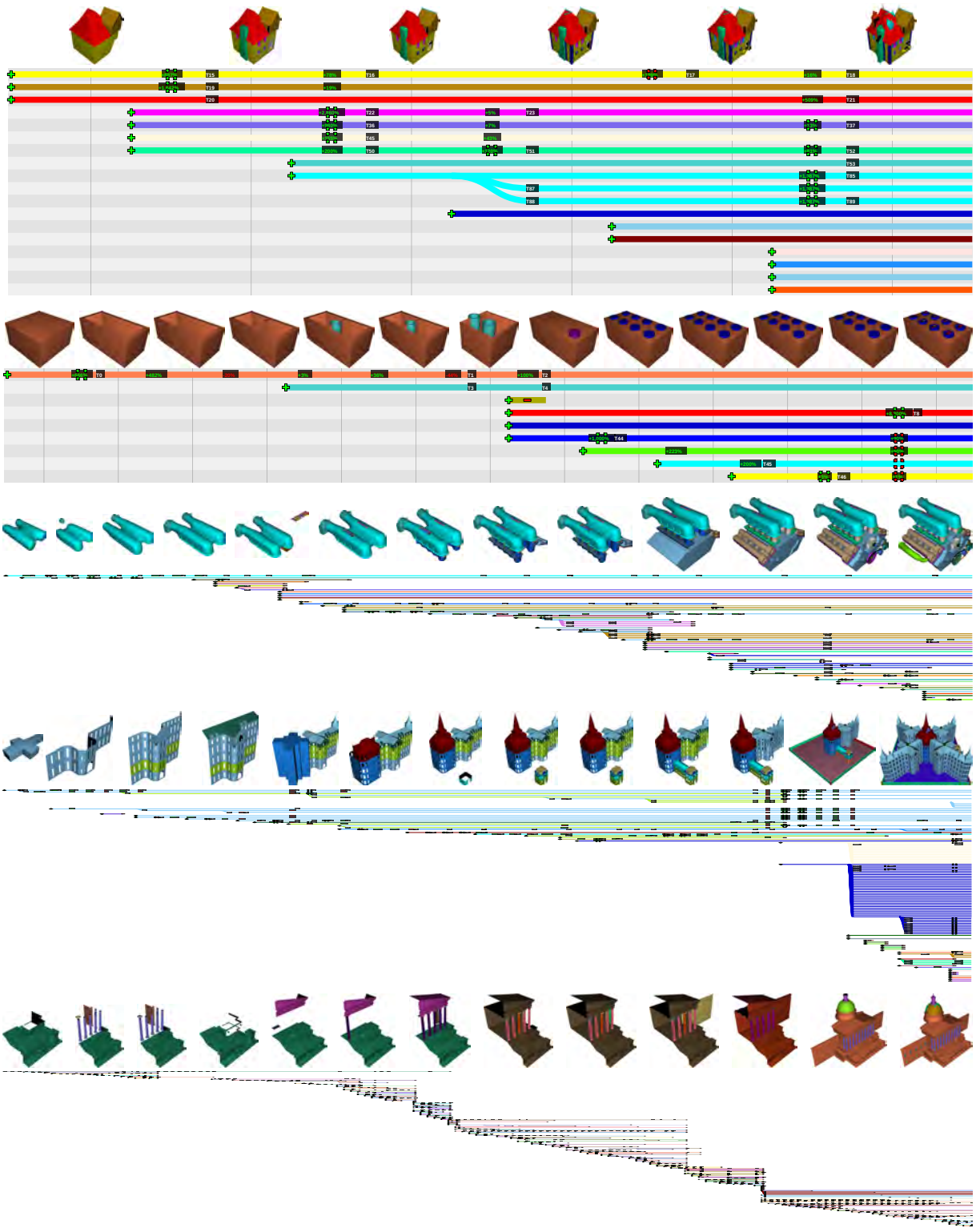

**Figure 8:** *Results: Collapsed timelines for* `Medieval`, `Brick`, `Engine`, `Cruciform` *and* `Portico` *datasets as presented in Tab. 2.* `Character` *sequence is shown in Fig. 1. Full timelines can be found in the supplemental materials. Please note that for brevity only a subset of the keyframes are shown for the* `Engine`, `Cruciform`, *and* `Portico` *datasets. See supplemental materials for full timelines or zoom in on PDF.*

theless, in most cases MeshGit provides valuable insight into fine-grained editing that our solution does not. Note that it is conceivable to use 3D Timeline for component-based analysis and MeshGit for vertex-level differences in conjunction.

**User study.** In order to evaluate the usability of our tool, we have conducted a preliminary user study with 8 postgraduate researchers in the field of computer graphics. Each participant indicated an intermediate or an expert level of experience in 3D modeling. To compare and contrast the timeline, a *multi-view* interface with only a basic side-by-side visualization and interlinked navigation was used. Participants were given a sample dataset before each trial to familiarize themselves with the interface. The task was to answer a short quiz with regards to the modeling provenance. After each session a system usability scale (SUS) [Bro96] questionnaire was handed out. The order of the datasets as well as of the interfaces was shuffled according to Latin square.

**Table 1:** *Pilot user study results based on 3 quiz questions in a multiview and a timeline interface, time to completion & the system usability scale (SUS) scores.* T = *true,* F = !T.

| | | Multiview | | | | | Timeline | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Experience** | **Q1** | **Q2** | **Q3** | **Time [m]** | **SUS** | **Q1** | **Q2** | **Q3** | **Time [m]** | **SUS** |
| **P1** | Itermediate | F | F | F | 5.9 | 70.0 | F | T | T | 3.3 | 67.5 |
| **P2** | Expert | T | T | T | 1.8 | 75.0 | T | T | T | 2.5 | 37.5 |
| **P3** | Intermediate | T | T | F | 3.1 | 75.0 | F | T | F | 2.0 | 72.5 |
| **P4** | Intermediate | F | F | T | 2.3 | 67.5 | F | T | F | 4.3 | 67.5 |
| **P5** | Expert | F | F | F | 3.6 | 44.5 | T | T | F | 1.5 | 77.5 |
| **P6** | Intermediate | F | F | T | 4.3 | 47.5 | T | F | F | 2.6 | 60.0 |
| **P7** | Intermediate | F | F | F | 2.4 | 37.5 | T | T | F | 1.9 | 85.0 |
| **P8** | Intermediate | F | F | T | 1.5 | 37.5 | F | T | F | 1.5 | 87.5 |
| **AVG** | | 25% | 25% | 50% | 3.1 | **56.8** | 50% | 88% | 25% | 2.4 | **69.4** |

The overall average success rate of the quiz in the timeline interface was 54% versus 33% in the multiview. As shown in Tab. 1, the timeline also scored better in terms of the system usability scale reaching grade B, i.e., an above the average user interface for the task at hand. According to this score, the users found it easy to familiarize themselves with the interface. In general, they also indicated that they would not need any technical assistance and would happily use it again. Note that the user study had only very small sequences with 6 consecutive frames to give the baseline viewer a fair chance. In our experience, on longer sequences (15+ frames), visual inspection becomes impossible.

### 7.1. Discussion

As shown in Tab. 2, the performance depends on the number of polygons, the number of components and their structural organization. On input, we do not make any assumptions about the temporal coherence of modeling effort neither rely on any scene graph organizational structures. If the frames are too close to each other, i.e., identical, our algorithm detects no changes and those are collapsed. If they are too far apart, i.e., too dissimilar, at some point the correspondence estimation will fail, hence components would be marked as deleted in one and added in the subsequent frame.

**Limitations.** The system in its current form cannot imply the relative ordering of operations between two frames. For example, it cannot be reliably decided whether an increase in the number of polygons has happened before or after a duplication. Therefore, future study into modeling behavior based on software instrumentation might provide evidence for patterns of events that occur more frequently than others. Similarly, the choice of a template component for duplication is arbitrary, although, it might be possible to devise additional sets of rules to break ties. Further, we do not detect any join operations such as multiple components becoming a single manifold surface. This is especially noticeable in the most challenging `Portico` dataset where most of the façade is modelled as a single continuous mesh. Our solution can successfully track major changes across most frames, however, in the 158 models it looses track in 4 instances visible as steps at the bottom of Fig. 8. Even though the correspondence estimation is greedy, hence not globally optimal, it is robust and together with our changes detection can uncover even non-rigid transformations such as bending. Unlike [PMW*08], however, the tight threshold of our repeated copying detection does not support approximate regularities.

## 8. Conclusions

We presented a tool for reverse engineering of a part-based provenance from linear sequences of 3D models. Our approach does not require recording of individual edits during construction, hence, it is applicable to all sorts of legacy datasets. The tool was successfully tested on 6 construction sequences spanning architectural modeling, CAD prototyping and even free form sculpting. Our pilot user study suggests this tool to be usable by untrained users who preferred it over a standard side-by-side visualization. This tool can be useful to artists revisiting modeling processes to learn from.

**Future work.** A simple addition in the future will be application of a heat map to visualize the rate of change on the morphed model itself. Further, we plan to add semantic differencing at the level of individual vertices and faces after the part-based correspondence has been established. An interesting avenue will be the exploration of automated intention preservation while modifying the timeline. We hope this type of UI will motivate software vendors in the future.

### Acknowledgements

We would like to thank Jules Bodenstein, Marcin Klicki, Johnathan Good and Ian Brown for modeling sequences, the participants of the user study, the reviewers for their feedback, and the authors of MeshGit for releasing their software. This research was sponsored by the UK EPSRC-funded EngD. Centre in Virtual Environments, Imaging and Visualisation (EP/G037159/1), the Rabin Ezra Trust, Arup, and the ERC Starting Grant SmartGeometry. A sabbatical at Electronic Arts informed parts of this work.

**Table 2:** *Statistics for the test sequences evaluated on a Thinkpad X230 with Intel Core i7-3520M CPU @2.90 GHz with 16 GB RAM on Windows 8. From left to right: the number of frames in a sequence, the cumulative number of polygons, the number of detected components, duration of the correspondence estimation, duration of the timeline analysis, sum of correspondence and analysis and the overall number of components processed per second (Components / (Correspondence + Analysis)* $\times 1000$*).*

| Dataset | Frames | Polycount | Components | Corr. [ms] | Analysis [ms] | Total [s] | Throughput [$C$/s] |
|---|---|---|---|---|---|---|---|
| Medieval | 6 | 16,005 | 510 | 51 | 40 | 0.09 | 5,604 |
| Brick | 16 | 16,703 | 141 | 47 | 25 | 0.07 | 1,958 |
| Engine | 55 | 3,414,103 | 2,460 | 1,435 | 512 | 1.95 | 1,264 |
| Cruciform | 74 | 924,123 | 23,695 | 74,712 | 1,140 | 75.85 | 312 |
| Portico | 158 | 2,442,104 | 3,622 | 1,908 | 784 | 2.70 | 1,346 |
| Character | 9 | 7,609,539 | 189 | 6,685 | 1,875 | 8.56 | 22 |

## References


[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput. 22*, 3 (Mar. 2006), 181–193.

[BBB*10] BOHØJ M., BORCHORST N. G., BOUVIN N. O., BØDKER S., ZANDER P.-O.: Timeline collaboration. In *Proc. SIGCHI* (2010), pp. 523–532.

[BCC*05] BAVOIL L., CALLAHAN S. P., CROSSNO P. J., FREIRE J., SCHEIDEGGER C. E., SILVA C. T., VO H. T.: Vistrails: Enabling interactive multiple-view visualizations. In *Visualization, 2005. VIS 05. IEEE* (2005), IEEE, pp. 135–142.

[Bro96] BROOKE J.: SUS: A quick and dirty usability scale. In *Usability evaluation in industry*, Jordan P. W., Weerdmeester B., Thomas A., Mclelland I. L., (Eds.). CRC Press, London, 1996.

[CWC11] CHEN H.-T., WEI L.-Y., CHANG C.-F.: Nonlinear revision control for images. In *ACM SIGGRAPH* (2011).

[DKP11] DENNING J. D., KERR W. B., PELLACINI F.: Meshflow: interactive visualization of mesh construction sequences. *ACM Trans. Graph. 30*, 4 (July 2011), 66:1–66:8.

[DP13] DENNING J. D., PELLACINI F.: Meshgit: diffing and merging meshes for polygonal modeling. *ACM Trans. Graph. 32*, 4 (July 2013), 35:1–35:10.

[DS12a] DOBOŠ J., STEED A.: 3d diff: an interactive approach to mesh differencing and conflict resolution. In *SIGGRAPH Asia 2012 Technical Briefs* (2012), pp. 20:1–20:4.

[DS12b] DOBOŠ J., STEED A.: 3d revision control framework. In *Proc. Web 3D* (2012), pp. 121–129.

[ES11] EISSEN K., STEUR R.: *Sketching: The Basics*. BIS Publishers B.V., 2011.

[GAL*09] GRABLER F., AGRAWALA M., LI W., DONTCHEVA M., IGARASHI T.: Generating photo manipulation tutorials by demonstration. *ACM SIGGRAPH 28*, 3 (2009), 66:1–66:9.

[GF09] GOLOVINSKIY A., FUNKHOUSER T.: Consistent segmentation of 3D models. *Proc. SMI 33*, 3 (2009), 262–269.

[GMF10] GROSSMAN T., MATEJKA J., FITZMAURICE G.: Chronicle: Capture, exploration, and playback of document workflow histories. In *Proc. UIST* (2010), ACM, pp. 143–152.

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P. S.: Hierarchical face clustering on polygonal surfaces. In *I3D* (2001).

[HFL12] HU R., FAN L., LIU L.: Co-segmentation of 3d shapes via subspace clustering. *SGP 31*, 5 (2012), 1703–1713.

[HKG11] HUANG Q., KOLTUN V., GUIBAS L.: Joint shape segmentation with linear programming. *ACM TOG 30*, 6 (2011).

[HXM*13] HU S.-M., XU K., MA L.-Q., LIU B., JIANG B.-Y., WANG J.: Inverse image editing: Recovering a semantic editing history from a before-and-after image pair. *TOG 32*, 6 (2013).

[JTRS12] JAIN A., THORMÄHLEN T., RITSCHEL T., SEIDEL H.-P.: Exploring shape variations by 3d-model decomposition and part-based recombination. *CGF 31*, 2pt3 (2012), 631–640.

[KBB*13] KOVNATSKY A., BRONSTEIN M. M., BRONSTEIN A. M., GLASHOFF K., KIMMEL R.: Coupled quasi-harmonic bases. In *CGF* (2013), vol. 32, pp. 439–448.

[KHS10] KALOGERAKIS E., HERTZMANN A., SINGH K.: Learning 3D mesh segmentation and labeling. In *ACM SIGGRAPH* (2010), pp. 102:1–102:12.

[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. *SIGGRAPH 32*, 4 (2013).

[LDSS99] LEE A. W. F., DOBKIN D., SWELDENS W., SCHRÖDER P.: Multiresolution mesh morphing. In *Proc. SIGGRAPH* (1999), pp. 343–350.

[MKFC01] MICHIKAWA T., KANAI T., FUJITA M., CHIYOKURA H.: Multiresolution interpolation meshes. In *Proc. Pacific Graphics* (2001).

[MPWC13] MITRA N. J., PAULY M., WAND M., CEYLAN D.: Symmetry in 3D Geometry: Extraction and Applications. *Computer Graphics Forum 32*, 6 (2013), 1–23.

[NBCW*11] NGUYEN A., BEN-CHEN M., WELNICKA K., YE Y., GUIBAS L.: An optimization approach to improving collections of shape maps. *SGP 30*, 5 (2011), 1481–1491.

[PMW*08] PAULY M., MITRA N. J., WALLNER J., POTTMANN H., GUIBAS L. J.: Discovering structural regularity in 3d geometry. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 43:1–43:11.

[SNF10] STAB C., NAZEMI K., FELLNER D. W.: Sematime - timeline visualization of time-dependent relations and semantics. In *Proc ISVC* (2010), pp. 514–523.

[SSS*10] SHAPIRA L., SHALOM S., SHAMIR A., COHEN-OR D., ZHANG H.: Contextual part analogies in 3d objects. *Int. J. Comput. Vision 89*, 2-3 (Sept. 2010), 309–326.

[SvKK*11] SIDI O., VAN KAICK O., KLEIMAN Y., ZHANG H., COHEN-OR D.: Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM SIGGRAPH Asia 30*, 6 (2011), 126:1–126:9.

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. *ACM SIGGRAPH 24*, 3 (2005), 488–495.

[vKZHCO11] VAN KAICK O., ZHANG H., HAMARNEH G., COHEN-OR D.: A survey on shape correspondence. *CGF 30*, 6 (2011).

[WAvK*12] WANG Y., ASAFI S., VAN KAICK O., ZHANG H., COHEN-OR D., CHEN B.: Active co-analysis of a set of shapes. *ACM SIGGRAPH Asia 31*, 6 (Nov. 2012), 165:1–165:10.