Metamorphers: Storytelling Templates For Illustrative Animated Transitions in Molecular Visualization

Johannes Sorger[a], Peter Mindek[a], Peter Rautek[b], Eduard Gröller[a,c], Graham Johnson[d], Ivan Viola[a]

[a]*TU Wien, Vienna, Austria*
[b]*King Abdullah University of Science and Technology, Thuwal, Saudi Arabia*
[c]*VRVis Research Center, Vienna, Austria*
[d]*Allen Institute for Cell Science, Seattle, Washington, United States*

Abstract

In molecular biology, illustrative animations are used to convey complex biological phenomena to broad audiences. However, such animations have to be manually authored in 3D modeling software, a time consuming task that has to be repeated from scratch for every new data set, and requires a high level of expertise in illustration, animation, and biology. We therefore propose *metamorphers*: a set of operations for defining animation states as well as the transitions to them in the form of re-usable storytelling templates. The re-usability is two-fold. Firstly, due to their modular nature, metamorphers can be re-used in different combinations to create a wide range of animations. Secondly, due to their abstract nature, metamorphers can be re-used to re-create an intended animation for a wide range of compatible data sets. Metamorphers thereby mask the low-level complexity of explicit animation specifications by exploiting the inherent properties of the molecular data, such as the position, size, and hierarchy level of a semantic data subset. We demonstrate the re-usability of our technique based on the authoring and application of two animation use-cases to three molecular data sets.

Keywords: Molecular visualization, animation, animated transitions, illustrative visualization, storytelling

1. Introduction

In recent years, we have seen a rapid increase in the communication of topics from molecular biology, such as through the work of Drew Berry [2], Graham Johnson [1], Gaël McGill [3], or Janett Iwasa [4]. These topics involve complex processes, such as the copying of DNA, the transport of oxygen in the blood stream, or the comparison of molecular structures within a cell. Illustrators use animations to communicate these processes in an explicit and intuitive manner. An example sequence from such an animation is shown in Figure 1 where an organic cell structure is smoothly transformed into a more diagrammatic representation. The scientific animator [1] used this transformation to promote quantitative aspects of the data over its structural appearance. To achieve their goals, animators make use of well established visual abstraction techniques from scientific illustrations. *Exploded views*, for instance, are suitable for revealing how hierarchical structures, for example, the compartments of a microorganism, are layered (Fig. 2, left). Such animations are typically handcrafted with 3D modeling and animation tools, such as Maya. These tools offer great flexibility in the variety of achievable results. However, the low-level approach that they apply, requires a high level of expertise in illustration, animation, and biology to create an animation that conveys a complex biological phenomenon. A manually key-framed animation cannot be transferred to other data sets.

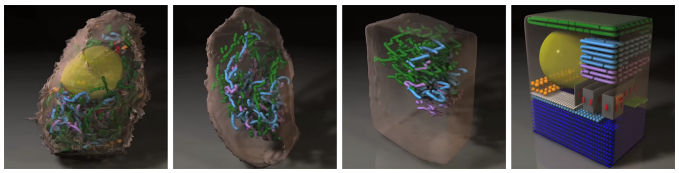


Figure 1: Transition of an organic representation of a cell to a more structured one that conveys the quantities of the cell's components [1].

Another problem of illustrative animation authoring on a low level arises in the context of molecular biology. In the visualization of molecular biology, we are dealing with very complex models, often containing tens of thousands of objects that represent individual cells, molecules, or atoms. Techniques routinely applied in hand-crafted scientific illustrations and animations are difficult to apply to such dense and large-scale data sets.

Besides manual low-level approaches, there are high-level approaches [5, 6] that semi-automatically generate a certain visual transformation for a supplied data set. These approaches relieve the user from manually creating key frames for the thousands of objects in complex molecular scenes. Further, they can be applied to arbitrary data sets that share a similar data structure. This advantage comes at the price of the flexibility that low-level techniques offer in terms of result variability.

Our goal is therefore to develop a technique for authoring animations that unites the advantages of low-level and high-level approaches. We support *flexibility* and *re-usability*, while providing the means to mask the complexity of low-level approaches from inexperienced users. To achieve these goals, we propose modular animation operators, i.e., *metamorphers*, that act as

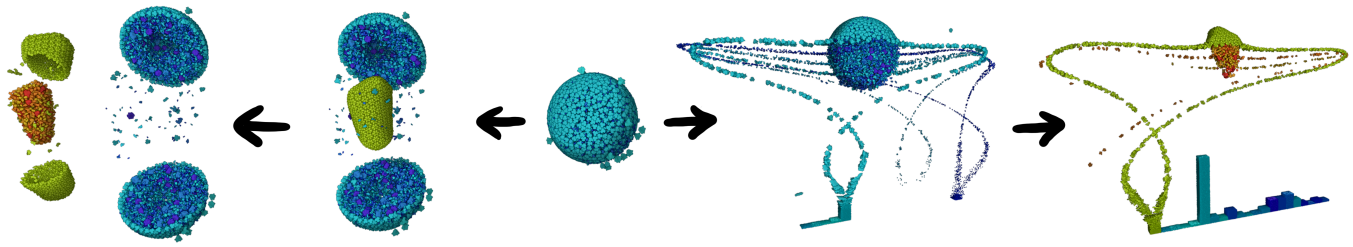


Figure 2: Metamorphers provide a flexible and re-usable interface for the creation of animated transitions of mesoscale data. Here, two different combinations of metamorphers are applied to create an exploded view (left) of an HIV model, and a bar chart (right) that represents protein quantities.

combinable storytelling templates. Metamorphers belong to six different classes of operations (data restructuring, layout, morphing, trajectory, timing, and camera control) that collectively describe the intermediate or target states of an animation, as well as the transitions between these animation states. A single metamorpher modifies the properties of the molecular data in regard to one of these classes. *Re-usability* ensures that the metamorpher produces an intended result for any compatible input data. *Flexibility* ensures that arbitrary metamorphers can be combined in a modular way to create a wide range of results. Both of these properties allow the user the combination of multiple metamorphers into higher-level operations. Such high-level metamorphers are again re-usable and flexibly combineable. A high-level metamorpher serves as a hierarchic container that masks the complexity of the contained lower-level metamorphers from the user.

In the remainder of this paper, we describe the technical details that are required to understand and reproduce metamorphers for the application to the domain of molecular biology. We illustrate our technique by presenting exemplary metamorphers in the scope of a proof of concept implementation. Our proof of concept implementation features a visual scripting interface in the form of a hierarchical node editor that enables the parametrization and combination of metamorphers as well as the creation of re-usable high-level metamorphers. Based on our implementation, we demonstrate the flexibility and re-usability of metamorphers by creating two different animations and re-applying them to three different data sets from molecular biology. Finally, we reflect on our technique as well as give some informal feedback that we gathered from domain experts in illustration, animation, and biology.

2. Related Work

Animation and transitions between data representations serve as powerful tools for the dissemination of complex relations in space, time, and abstract dimensions. They are frequently used in visual storytelling as well as for the depiction of relations between different data representations. Kosara and Mackinlay [7] emphasize the importance of storytelling in visualization not only for presentation purposes but also in decision making and process analysis. They suggest that opening up a visualization for interaction at the storys end, provides a convenient starting point for exploration and goes beyond a simple slideshow. Our implementation creates animations of the molecular data that can be paused and explored in real-time at any point during the transition. Wohlfart and Hauser [8] present a guided interactive volume-visualization approach in terms of camera path, transfer-function parameters, and annotations. Grimm et al. [9] introduce V-Objects as an abstraction of volume data. Their approach enables interactive specification of key-frames for animation paths, transfer-function fades, light movement, clipping planes, change of data sources, and enabling and disabling of objects. These approaches do not consider the re-usability of the animations for different data sets as they are based on the explicit authoring the key-frames. Hyun et al. [10] propose a more implicit approach in the context of storytelling, by using grammars to synthesize the animation of characters. They generate animation sequences for virtual players by employing a language that captures the behavioral structure of human movements in a basketball game. The grammar is used to check the validity of a sequence of movements and to suggest new plausible candidates for the next move. Karp and Feiner [11] present a scripted system that generates animations from communication goals. The system plans an animation by breaking it down into sequences, scenes, and shots down to the level of individual frames. Seligman and Feiner [12] present a system that generates technical illustrations using illustration rules. Annotations, highlights, and rendering styles are chosen to convey the intended information. Mühler et al. [13] present a scripting language for the generation of illustrations and animations in the context of medical intervention planning. Mühler and Preim [14] introduce *keystates* that describe a visualization on an abstract level, including information about visibility, rendering style, etc. of each structure. Keystates are used to automatically generate animations and are re-usable for data sets that contain the same structures. Iserhardt-Bauer et al. [15] developed a system that generates video sequences for intra-cranial aneurysms. Visualization parameters and the camera path are automatically generated from a standardized predefined protocol.

While these approaches convey purely spatial phenomena, other approaches reveal the abstract relationships within the 3D spatial data. Hurter et al. [16] use animation to interpolate between different (abstract) projections of the original data dimensions in a 3D volume. Basch [17] uses animated transitions of voxels to abstract volume representations, such as histograms of voxel intensity values. With metamorphers we propose a flexible technique that supports the depiction of spatial, as well

Figure 3: Classes of metamorphers according to the six stages of a pipeline for animated transitions. The data restructuring, layout, and morphing stages define an animation state. Trajectory, timing, and camera control define how an animation state transitions to the next one, and thereby structure the presentation of the transition.

as abstract relationships by transforming the spatial data into an abstract frame of reference.

3. Metamorphers

We describe the complete transformation of an object between two animation states by six stages of a pipeline for animated transitions (see Fig. 3). The first three stages of the pipeline (data restructuring, layout, and morphing) define, *what* the original model or data set should represent in an intermediate or target state of the animated transition. In the simplest case, an animation is created by linearly interpolating between the animation states (key frames) of the individual molecules in a scene at the same time. However, the simultaneous interpolation of tens of thousands of molecules, is hard to read due to clutter and occlusion. We refer to such an interpolation as an *unstructured interpolation*. By *spatially and temporally structuring* the transition, i.e., with trajectories, timings of the individual molecule transitions, and by controlling, which parts of the scene the camera presents to the viewer, a more pleasant animation can be created. The last three stages of the pipeline (trajectory, timing, and camera control) are thus responsible for defining, *how* a state should transform into the next one, and how the transition should be presented to the viewer.

Initially, we give a definition of how metamorphers can support re-usability and flexibility, we provide a description of the molecular data that they operate on, and we specifiy the interface between metamorphers. Then we describe the different classes of metamorphers that are necessary to create all aspects of an animated transition-

3.1. Re-usability & Flexibility

In order to support *re-usability*, a metamorpher has to adhere to two rules to achieve intended results for arbitrary data sets:

1) a metamorpher has to define an animation state not in an absolute way but *relative* to the previous animation state. For instance, a new position Pos_{new} for an object A should not be defined as $Pos_{new}(A) = X$, but instead as $Pos_{new}(A) = Pos_{old}(A) + Y$.

2) A metamorpher has to define an animation state not explicitly but *implicitly*, i.e., based on inherent properties of the scene and the data set. In the above example, X should not be an explicit value, e.g., $X = 500$, but should be given or derived from inherent properties $X = width\ of\ object\ A's\ bounding\ box$. Exceptions to these rules occur if absolute and explicit values have the same semantic implication for all data sets, such as moving an object to the center of the scene.

In order to support *flexibility*, a metamorpher has to provide a modular interface that enables arbitrary combinations with other metamorphers. Such a modular interface is achieved by using the same data structure for the input as well as the output of each metamorpher. A metamorpher takes a data (sub)set as an input, modifies specific properties, such as the positions over time, and passes the modified data (sub)set as output to the next metamorpher.

Both of these qualities support to *abstract* the complexity of authoring an animation. Multiple metamorphers can combine their lower-level operations into a single high-level operation with complex functionality. A high-level metamorpher serves as a hierarchic container that masks the combined lower-level metamorphers from the user. For instance, the steps involved in the creation of an exploded view can be summarized into a high-level metamorpher. A non-expert user can now apply the exploded view to an arbitrary data set without having to know about the individual operations that are involved. Flexibility assures that this high-level metamorpher can still be combined with other metamorphers. These characteristics make metamorphers re-usable storytelling templates that can be flexibly combined to convey diverse messages. At the same time, the low-level complexity of the operations involved in authoring illustrative animation sequences is masked.

3.2. Molecular Data: Inherent Properties

Our pipeline operates on biological data that is composed of individual proteins. These proteins collectively form the compartments and inner structures of the microorganisms that they depict. Proteins are therefore the building blocks of these data sets. The data set of the HIV virion that is depicted in Figure 2, for instance, contains approximately 20.000 molecules distributed across 42 protein types, with an atom count of 60 millions.

The molecular data is organized in a hierarchical structure. The smallest entity is an *atom a* of a certain *type*, i.e., chemical element. It is represented by a sphere that has a *position* in its local coordinate system, and a *scale* that describes its radius. A *molecule m* represents a certain protein *type* and has a *position*, *orientation*, and *scale* in the global coordinate system.

A molecule is represented by a list of atoms $m = a_1, ..., a_n$ that yields an *atom count* and a *volume*. It is contained within the spatial extents of a *bounding box BB*. A *scene S* is comprised of a list of molecules $S = m_1, ..., m_j$ that can be partitioned into *hierarchical subsets M*, describing semantic structures of the microorganism. The leaf nodes are the individual molecules. Each node in the hierarchy thus contains a list of subsets or a list of molecules that yields a *molecule count* and *volume*, and is contained in a *bounding box*.

To support animation, the hierarchic data structure is extended by *control points* and *time curves* for subsets and molecules. A control point stores a node's properties, such as the position, and bounding box, for an animation state. Each control point corresponds to a key frame of the animation. Time curves determine the speed at which a molecule transitions between each pair of control points. Metamorphers manipulate these properties and automatically create new control points and time curves that are interpolated to create an animation.

3.3. Metamorpher Modular Interface

The modular nature of the metamorpher input/output interface supports the flexible combination of metamorphers. This enables the user to create diverse animation sequences. The interface takes as input the data hierarchy and a pointer to the parent node of the subsets that should be processed. The metamorpher then accesses and modifies the respective properties of the node's children according to its functionality. The interface passes on the modified node as a single output node. Multiple outputs have to be used if the individual subsets of a hierarchy level should be forwarded to different metamorphers (as shown in Fig. 5). In this case, the pointers to the individual subsets are passed on to separate outputs. Since the separate outputs are explicitly linked to different metamorphers, the number of outputs has to be known, i.e., it cannot depend on the data set. A spatial data-restructuring metamorpher, for instance, creates always two subsets if it splits the space into two parts. If the number of subsets that a data restructuring metamorpher creates is unknown, the subsets have to be forwarded as the children of a single node.

3.4. Metamorpher Classes

In the following we will elaborate on the six metamorpher classes and give examples for concrete implementations for each of them. We will use the exploded view and the transition to a bar chart from Figure 2 as guiding examples.

3.4.1. Data-restructuring Metamorphers

Data-restructuring metamorphers partition the data structure to define, on which parts of the data, and in which granularity metamorphers from subsequent pipeline stages will operate, i.e., *they create semantic subsets* from the list of molecules. The data hierarchy in a target representation can differ from the one in the initial representation, and therefore has to be adapted for subsequent operations. A data set can be partitioned *semantically*, according to inherent abstract data properties, or *spatially*, according to inherent geometric properties. The partitioning

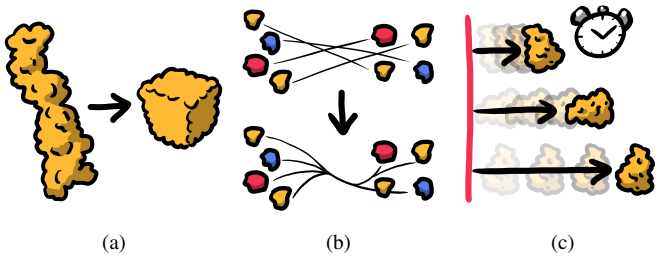


Figure 4: Schematic representations of the effects of different metamorphers from the a) morphing, b) trajectory bundling, c) timing stage.

defines a scene hierarchy for each animation state. The original topology is always preserved in the initial animation state.

For the exploded view in Figure 2 (left), for instance, the data first has to be partitioned semantically into the three subsets that represent the layered cell compartments. *Split(comp, S)* where $S = m_1, ...m_j$, partitions the input data into k cellular compartments so that $S = M_1, ..., M_k$. The compartments are in our case inherently given in the hierarchy of the data format. If no such hierarchy is present, segmentation algorithms are needed to determine semantic clusters.

In the exploded view in Fig. 2 (left), the compartment subsets are *spatially* partitioned into a lower and an upper half. *Split(spatial(plane), S)*, where $S = m_1, ..., m_j$ assigns all molecules to subsets M_{upper} and M_{lower} depending on which side of a plane they are located. To achieve a re-usable spatial partitioning, inherent geometric information, such as the bounding box, can be exploited, e.g, by placing a cutting plane at the center of the input subset's bounding box.

By providing operations that work on semantic data properties, the user is not required to have an extensive knowledge of the data composition.

3.4.2. Layout Metamorphers

Layout metamorphers *spatially rearrange* subsets of molecules that were defined in the first pipeline stage, in order to create a desired target representation. Layout metamorphers define a target state with a sequence of transformations of the positions, rotations, and scales of the molecules in a given subset.

A $rotate(hinge(axis, degree), M)$ metamorpher needs an implicit axis, such as the axis of a scene's coordinate system or the edge of a bounding box, to support re-usable rotations. In the exploded view (Fig. 2, left), for instance, we rotate the halves of a compartment around a common hinge where their bounding boxes align. The rotation angle can be explicit, since it yields the same results for arbitrary data sets. An angle of, e.g., 90 degrees between two planar surfaces provides a view into the interior, independent of the data.

Similarly, for translations that have the same impact on arbitrary input data, the translation direction and distance of a *translate(direction, distance, M)* metamorpher are chosen so that they have the same semantics. Such implicit layout operations can mask the complexity of low-level geometric transformations from the user.

3.4.3. Morphing Metamorphers

Metamorphers from the morphing stage change the visual appearance of individual molecules (Fig. 4a). In some scenarios, the representation of individual objects in a scene needs to be adapted in order to better convey an intended message. Molecules in mesoscale biological models can be displayed as space-filling models, stick models, or one of many other representations commonly used in molecular graphics [18, 19]. In the bar chart in Figure 2 (right), for instance, *morph(slab, S)* transforms proteins into square-shaped slabs that are then stacked upon each other by a layout metamorpher. A morphing metamorpher *morph(slab—stick—cube—sphere—scale)* re-assigns the atoms of a given protein to random positions within the extents of the intended shape. The random positions approximate the defined shape. Due to the random positioning within the specified geometric borders, the resulting shape is the same for arbitrary input molecules. These positions represent new control points for atom positions in the molecule's local coordinate system.

Morphing metamorphers mask the low-level complexity of molecular data structures that users would need to access and modify in order to change their representation.

3.4.4. Trajectory Metamorphers

The trajectory stage is the first of the three pipeline stages responsible for structuring the transition between two animation states (Fig. 3). Due to the size of the data (tens of thousands of molecules), the structure of the transition plays an especially important role, e.g., for visual clutter reduction, occlusion handling, and the guidance of the viewer's attention.

Trajectory metamorphers *spatially structure* the transition by defining the trajectories along which the individual molecules of a given subset will move. A common technique for spatially structuring the transition of particles is trajectory bundling. *Bundle(p, M)* reroutes the trajectories of the particles in M through a common point p, i.e., a *bundling point*, instead of directly interpolating between their initial and target position (see Fig. 4b). The position of a re-usable bundling point is derived from the given data semantics. In the transition to a bar chart (Fig. 2, right), we define bundling points with respect to the bounding boxes of the initial and final animation states.

Trajectory metamorphers hide from the user the complexity of manually specifying explicit control points for specific subsets with hundreds or thousands of objects.

3.4.5. Timing Metamorphers

Timing metamorphers are responsible for the *temporal structuring* of the transition between two animation states. Temporal structuring is achieved by manipulating the start time, duration, and speed of individual transitions of molecule subsets. These properties can be adjusted with *time curves* that are associated with control-point pairs between molecules m and subsets M.

Temporally structuring the transition of thousands of densely packed animated objects is essential to reduce the cognitive load on the viewer. A *staged* animation can reduce occlusion and clutter [20]. It determines the order and speed in which objects move, as well as the number of objects that move at the same time. Other temporal distortions that create more pleasant animations, such as slow-in/slow-out pacing [21], are handled in this stage as well. Timing metamorphers can also be used to convey information about the chronology of the illustrated events.

A *stage(distance(p), S)* metamorpher, for instance, delays the animation of molecules in respect to their distance to a certain point p. This distance-based delay temporally structures the transition so that molecules appear to be peeled away from their original position, layer by layer. A *stage(type, S)* metamorpher orders the time curves of the molecules in an input subset by their protein type so that all molecules of the same type are animated in tandem. A delay d defines how long each molecule waits before starting its transition (Fig. 4c). The delay can be defined explicitly ($d = X$) or implicitly (for instance, $d = 1/j * Y$, where j represents the total number of molecules in a subset, and Y a scaling factor).

By relying on a given semantic or geometric properties, timing metamorphers mask the complexity of sophisticated animation timing from the user. The resulting temporal structuring would be very difficult to achieve by manual authoring, e.g., on the level of individual protein instances.

3.4.6. Camera-Control Metamorphers

Camera-control metamorphers govern the camera position and viewing direction at each time step during an animation sequence. Camera steering is an essential component of a non-interactive explanatory visualization. It effectively determines, *which parts* of a dynamic scene are presented to the viewers when guiding them through a sequence of events, as well as *how* these parts are presented, i.e., from which viewing angle and distance. A wide range of sophisticated semi- and fully-automatic camera-control techniques [22] have been developed in the field of computer graphics. Re-usable camera-control metamorphers that automatically react to dynamic changes in an animation, can rely on inherent data properties, such as molecule types and subsets, as well as their animation states (position, rotation, scale) at each time step.

A simple *camera(follow(BB), S)* metamorpher checks the extents of the input subset's bounding box and adjusts the camera zoom and look-at vector to keep the subset inside the view frustum during the animation. In more complex cases, a *camera(follow(semantic), S)* metamorpher could derive semantic importances from salient features, such as objects that are currently in motion.

Camera-control metamorphers that react automatically to the dynamic changes in an animation, relieve the author from manually specifying camera paths in complex dynamic scenes.

4. Implementation

Our implementation is an extension of the cellVIEW framework [23], an open-source tool for the real-time visualization of large mesoscale molecular models that is realized within Unity3D. The molecular models are loaded from files supplied in the cellPACK format [24]. We extend the cellVIEW data

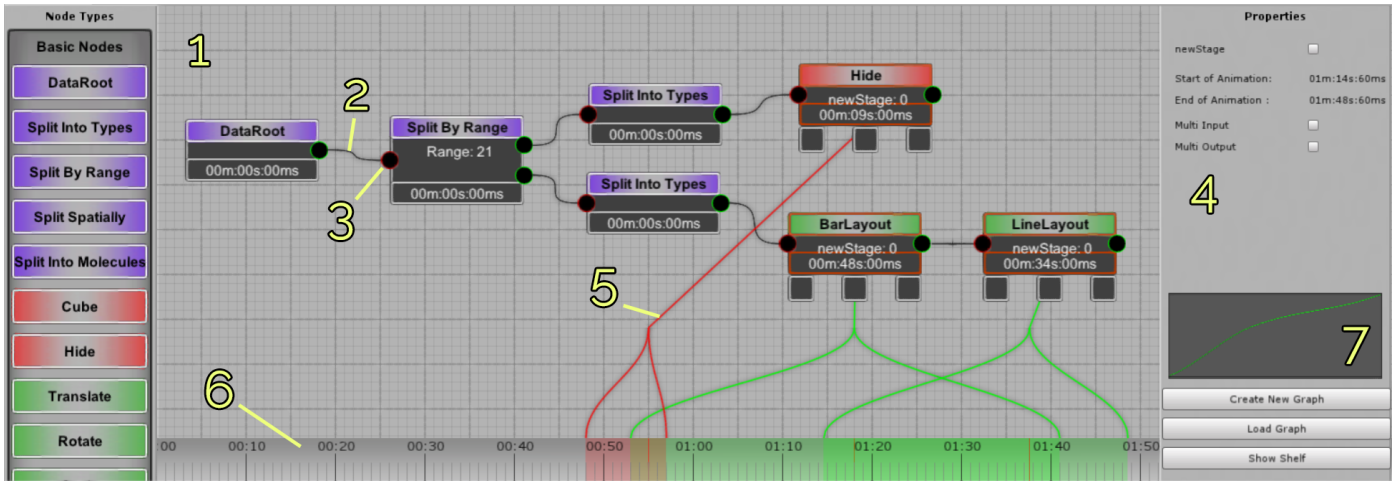


Figure 5: The node editor acts as a visual scripting interface to the metamorpher API function calls. 1: the canvas for arranging metamorpher nodes, 2: a link between input & output of two metamorphers, 3: input handle of a metamorpher node, 4: a node's property window, 5: a time anchor, 6: the time line, 7: a node's time curve.

structure of atom and molecule types, positions, and orientations, with the means to create and store hierarchical subsets of molecule instances, as well as control points and time curves (as described in Sec. 3.2).

The animation pipeline is realized as an application programming interface (API) in the C# programming language. All metamorphers are executed as parametrized function calls that operate on arbitrary subsets of the data hierarchy. On the C# level, new metamorphers can be defined by creating new functions that implement the interface that we defined in Section 3.3.

4.1. Interface

To allow users a more intuitive parametrization and combination of metamorphers, we implemented a node editor (Fig. 5) that acts as a visual scripting interface to our API. Each node in the editor represents a metamorpher. The inputs and outputs of nodes can be connected (2) to link two metamorphers. Metamorphers can be dragged from the node shelf on the left onto the canvas (1). Nodes are parametrized via the property window (4) that is accessed by clicking on a node. While the order of linked metamorphers and the applied timing metamorphers create an implicit timing for the animation, the editor allows the user manual fine tuning. The start time and duration of a metamorpher is set via its anchor (5) to the time line (6) at the bottom of the editor. The time curve (7) of a metamorpher can be accessed in a node's property window. The horizontal axis represents the time, and the vertical axis represents the progress of the respective operation. The time curve thus controls the start, end, speed, and acceleration of an operation. Data-restructuring metamorphers do not require timings.

Each node in the editor is associated with an *evaluate()* function that is called when the chain of connected metamorpher nodes is traversed upon execution. The *evaluate()* function calls the respective metamorpher with the parameters defined by its node's properties. It passes the metamorpher output as an input to the metamorpher of the following connected node. For an additional level of control, experienced users can access and edit the C# code of *evaluate()* functions and the associated metamorphers by double clicking on a node. Upon saving, Unity3D will automatically re-compile and accommodate the changes.

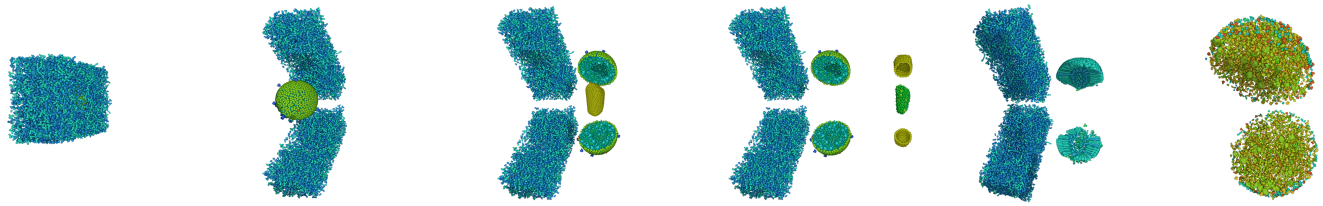
4.2. High-Level Metamorphers

High-level metamorphers hide the functionality of multiple metamorphers contained in a hierarchical group node. The node editor enables the user to create and store such group nodes on the fly. These high-level metamorphers are flexible and reusable, as well. The contents of a high-level node still can be accessed and modified by users who have the required expertise and domain knowledge.

4.3. Producing the Animation

An animation is defined by a chain of linked metamorphers. The key frames of an animation are produced by interpolating the animation states that are stored in the control points of individual molecules in consideration of the specified time curves. To create the key frames for each molecule, the data hierarchy is traversed. The transformations of higher-level nodes in the hierarchy are propagated to the transformations on lower levels, down to the individual leaf nodes, i.e., the molecules. For displaying the animation, the key frames are interpolated, and the animation is played back in real-time within cellVIEW. The last state of a metamorpher setup can be inspected at any time during the authoring process by pressing the *evaluate* button.

Our current implementation calculates the key frames on the CPU. Linear interpolations between control points can be achieved in real-time, i.e., the key frames do not have to be calculated before the animation is played back. If control points require cubic interpolations, key frames have to be calculated prior to playback. The key frame calculations of the videos in the supplemental material took between 10 and 20 seconds on current consumer hardware.



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

Figure 6: Exploded views of three datasets: (a) - (d) a sequence of four animation states of the mature HIV dataset, (e) final animation state of the immature HIV dataset, and of the (f) mycoplasma dataset.

5. Results

In this section, we showcase the authoring of the two animations illustrated in Figure 2, i.e., the transition to an exploded view and the transition to a bar chart. We thereby refer to the metamorphers that we introduced in Section 3.4. We demonstrate the re-usability of our technique by applying each metamorpher setup to three molecular data sets, i.e., to computational models of the immature HIV virion, the mature HIV virion, and the mycoplasma bacterium. The resulting animations can be viewed in the supplementary video.

The first two data sets describe different development stages (immature and mature) of the structural model of the human immunodeficiency virus (HIV), which is built-up from more than 20.000 macromolecules. The HIV is surrounded by blood serum, and therefore not visible in the initial state of the data set (Fig. 6a). The mature virion features better defined hierarchical compartments and also differs in protein types and their spatial distribution, in comparison to the immature virion. Both data sets have an onion-like hierarchical structure. The mycoplasma bacterium, while spherical as well, has a less developed hierarchy. Most of its proteins are distributed loosely within its interior.

5.1. Exploded View

The three data sets that we present are all densely packed. Their outer shells obstruct the view to inner structures. In this example, we want to reveal the inner structures as well as the hierarchy in which they are arranged. We therefore create an exploded view that opens the individual compartments and places them side by side.

We start by creating the data structure that we need for the final transition state. First, we create molecule subsets that represent the individual compartments. We therefore feed our initial data S into a *split(comp, S)* metamorpher that yields i subsets M_i for S, where i is the number of compartments. Second, we want to split all compartments horizontally at their center. We achieve this by applying a *split(plane(BB_{center}), S)* metamorpher that uses a plane at the bounding-box center BB_{center} of each subset M_i. This will add an additional hierarchy layer to the data structure by splitting each M_i into an $M_{i_{upper}}$ and $M_{i_{lower}}$.

Next, we define the layout for the data structure that we created. To explode (or open up) the individual compartments, we rotate the lower and upper compartment halves by 45 and -45 degrees respectively. As rotation hinge we use the upper (back) and lower (back) edge of their bounding boxes. We pass the upper compartment halves to a *rotate(hinge(BB_{lower}, 45), $M_{i_{upper}}$)* metamorpher and the lower ones to a *rotate(hinge(BB_{upper}, -45), $M_{i_{lower}}$)* metamorpher.

To layout the exploded compartments side by side, we use a *translate(x-axis, BB_{width}, S)* metamorpher. The children of S, i.e., the compartments, will thereby be placed along the x-axis with respect to their bounding box width. We do not change the appearance of individual molecules, so no morphing metamorpher is needed.

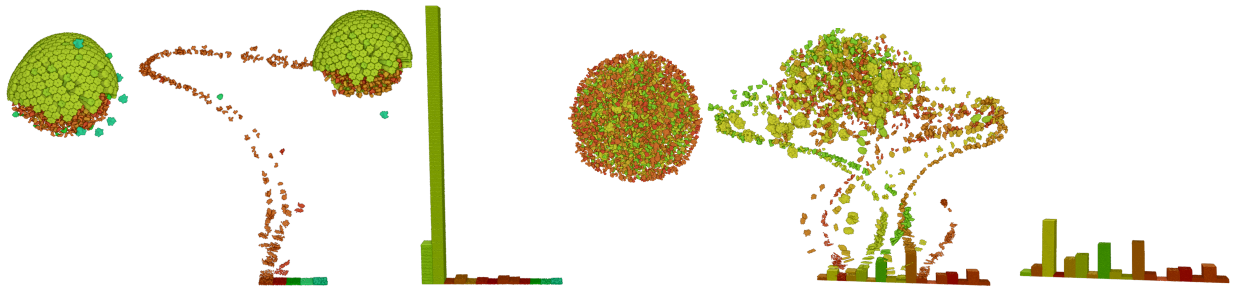
After defining the target representation, we design the transition to it. We want the individual compartments to be revealed one by one. We simply apply a *stage(subset, S)* metamorpher that individually delays the transition of each compartment, i.e., the child nodes of S. Since we move individual compartments as a whole, no trajectory metamorpher is required. We use a *camera(maintain, S)* metamorpher to guarantee that the data stays inside the view frustum during the transition.

We group the final chain of metamorphers into a high-level *explode(S)* metamorpher that can now be applied to different data sets in order to create an exploded view of their hierarchical compartments. The resulting transitions can be seen in Figures 6a-d for the mature HIV. Figures 6e and 6f show the results of the same metamorpher sequence on the immature HIV and the mycoplasma data set. Even though the number of compartments and the associated molecule types differ in each data set, our chain of metamorphers creates an appropriate exploded view for each of them. Since the mycoplasma data set does not have a hierarchical structure, the application of an exploded view might be of lower value. Still, even here the result is semantically correct in regard to the desired target visualization.

5.2. Bar Chart

In the bar chart example, we convey knowledge about the quantities of proteins in a microorganism. We choose to represent these quantities in a bar chart. Each bar in the chart represents the volume that the respective molecule type occupies in a given structure.

Since we want to show the quantities per molecule type, we partition the data with a *split(type, S)* metamorpher into i subsets M_i, where i represents the number of molecule types. To create the individual bars, we apply a *translate(y-axis, length, S)* metamorpher that will stack the molecules of a specific type



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

Figure 7: The bar chart metamorpher setup applied to the immature HIV (a-c), and the mycoplasma data set (d-f).

along the y-axis in intervals of the defined length. The length is defined by a user-specified maximum bar height, normalized by the maximal number of molecules among all molecule types. Next, to give the stacked molecules the appearance of a bar, we use a *morph(slabs, S)* metamorpher in order to change the original molecule shapes into square slabs of equal size. To align the bars side by side along the x-axis, we apply a *translate(x-axis,* BB_{width}*, S)* metamorpher. The bounding-box width corresponds to the width of the slabs in each stacked bar.

In order to increase the readability of the animation, we structure the transition spatially and temporally. For the spatial structure of the transition, we re-route the trajectories of the molecules so that they do not move though the virus on the way to their target positions. Instead, we want the molecules to move to trajectory-bundling points to the right and left of the initial structure. We therefore partition the data further by applying a *split(plane(*$BB_{center.yz}$*, S)* metamorpher that splits each typed subset M_i at the yz-plane of the bounding-box center into $M_{i_{left}}$ and $M_{i_{right}}$ halves. The newly created subsets are now used by a *bundle(*BB_l*,* $M_{i_{left}}$*)* and *bundle(*BB_r*,* $M_{i_{right}}$*)* metamorpher to create bundling points on the left and right side of the virus structure's initial bounding-box.

We also want to create the impression of molecules falling into their respective bar from the top. We achieve this with a *bundle(*BB_{top}*, S)* metamorpher that creates a bundling point for each subset of S, at the top of each histogram bar. The trajectory-bundling metamorphers are inserted before (*BB* of the initial structure) and after the layouting (*BB*s of the bars), depending on which structural information they need to relate to.

In order to *temporally structure* the transition, we firstly stage the animation per molecule type (*stage(type, S)*). Secondly, we apply a distance-based delay per molecule (*stage(distance(*BB_{center}*), S)*) to peel away the molecules layer by layer depending on their distance to the initial bounding-box center. Finally, we use a *camera(maintain, S)* metamorpher to guarantee that the data stays inside the view frustum during the transition.

The results of applying this sequence of metamorphers to two different data sets can be seen in Figures 7 a-c for the immature HIV, and Figures 7 d-f for the mycoplasma data set. The result for the mature HIV data set is displayed in Figure 2 (right). The intended result, i.e., a transition to a bar chart of protein quantities, has been created successfully for all three data sets. The principal requirement for this is the existence of typed entities (molecules) in the data.

6. Discussion

6.1. Expert Feedback

In order to judge the feasibility of our technique, we gathered informal feedback on metamorphers from three experienced biological illustrators. They commented that their work flow to create complex illustrations would greatly benefit from metamorphers. They consider the produced results as *"incredibly valuable"* to communicate knowledge. The current visual-scripting interface received some criticism for being not very intuitive. One expert suggested that the interface could benefit from *"icons (or animated icons) that show what each effect is roughly intended to perform"*.

According to one of the experts, setting up an illustration using his accustomed work flow *"would still take me a minimum of one hour to create a prototype, and then several hours or a couple of days to refine it"* – despite having 17 years of experience with 3D modeling software. With metamorphers, he claims, he can *"iterate on it in real-time in a matter of minutes"*. In addition, most of the illustrations that result from the accustomed work flow are not reusable or portable to other data without significant customization. The re-usability of animation setups was therefore regarded as highly valuable.

6.2. Applicability

While our technique is in principle applicable to different data types, it is especially valuable for large-scale molecular data. On the one side, the authoring of illustrative animations for large-scale molecular structures is a cumbersome task, as reported by domain experts. On the other side, large-scale molecular data bridges the disciplines of molecular visualization and cellular visualization, as microorganisms are depicted on molecular/atomic resolution. In terms of molecular visualization, the visual transformation of individual elements in a data set is common, as many different well established visualization techniques exist. In the visualization of complex biological structures, illustrative abstractions of the organic structures and compartments are commonly applied to convey specific information about an

organism. Metamorphers are especially suitable for this type of data, as abstractions of both molecular and biological structures are supported by our approach, i.e., in the morphing and layout stages respectively.

Our technique supports in theory arbitrary animations of typed data with point cloud characteristics. It is especially well suited for the purpose of creating short illustrative animations, most notably, transitions between different representation forms. Re-usability is mainly a benefit in application scenarios where the same illustrative technique is frequently re-applied to different data sets. An example is to highlight a certain characteristic in a series of related data sets for the purpose of comparison. Stories with a more complex plot typically do not need to be re-usable, in the sense that the contained animations and transformations are applied to different data sets. For complex stories, the overhead of creating metamorphers would most likely outweigh the benefit if re-usability is not a concern.

The re-usability of our technique remains intact as long as the conditions in Section 3.1 are met. An explicit description of an animation state might require less effort than programmatically creating multiple metamorphers from scratch. However, once a metamorpher is defined, it can instantly be applied to other appropriate data as well. The flexible interface further broadens the scope of achievable results in combination with already defined metamorphers. The potential results are only limited by transformations that cannot be defined in relation to implicit semantic or geometric properties of the data.

6.3. Future Work

There are several directions that we consider as possible extensions to the concept of metamorphers. In terms of general applicability, our method could be adapted to be used with point clouds, or volumes in general, as the molecular data is represented by a set of points in 3D space. The topologies of objects that are formed by spatial proximity of individual elements in point clouds and volumes are not changed, e.g., when spatially splitting a data set along a plane. However, for the application of our approach to polygonal data, an implementation would need to support changes of the mesh topology.

Another interesting direction for further research is the transition between multiple scales at different orders of magnitude, such as the transition from the scale of a blood cell to the scale of the human body as presented by Hsu et al. [25].

Segel and Heer formulate a design space for storytelling in visualization [26]. They classify a narrative visualization by genre, visual narrative, and narrative structure. Our technique gives the animation author great flexibility in terms of visual structuring, highlighting, transition guidance, and ordering. Since in this work we focus on the definition of animation states and transitions, we currently do not explicitly support interaction and messaging. These two aspects are priorities for our follow-up work, in order to further improve the storytelling capabilities. Since the environment, in which we implemented our method [23], renders the animated transitions in real-time, interaction is possible but not yet explicitly supported.

To offer a compensation for the fleetingness of animations, the metamorpher output could be sparsely sampled to produce a series of static images. This could be achieved, for instance, with object-based key-frame extraction techniques, as described in previous work [27, 28].

The set of metamorphers that we presented in this paper serves as a proof of concept to illustrate our technique and to create the examples in Section 5. It is by no means feature complete. An interesting possibility would be to create a public data base where users can share the metamorphers that they defined and modify existing ones.

7. Conclusion

Our method allows users to create re-usable animated transitions for molecular data sets. The re-usability has the advantage that target representations and transitions to them do not have to be manually re-modeled and key-framed. Instead, users can create animated transitions by combining pre-made templates, which we refer to as metamorphers. The same chain of metamorphers can be used to generate animated transitions for multiple data sets, while individual metamorphers can be re-used in different combinations. The set of introduced metamorphers already demonstrates the flexibility of our technique, but the presented list of operations is by no means complete. To extend the list of achievable animations, our interface provides access to the source code of individual metamorphers. It thereby allows users with programming expertise to modify existing metamorphers, or to create new ones. The grouping to high-level metamorphers hides complex low-level animation concepts. In this way, metamorphers can be exchanged between users from different domains and of different levels of expertise.

Acknowledgments

This project has been funded by the Vienna Science and Technology Fund (WWTF) through project VRG11-010 and was supported by the EC Marie Curie Career Integration Grant through project PCIG13-GA-2013-618680, as well as by the OeAD Scientific & Technological Agreement SK 14/2016 through the ManyViews project. The authors would like to extend their gratitude to Bara Kozlikova, Manuela Waldner, Martin Ilk, and Wiktor Manczarski for their valuable input.

References

[1] Johnson, G.T., Noske, A., Marsh, B.. Rapid visual inventory and comparison of complex 3d structures. accessed: 3/5/2017. URL `https://www.youtube.com/watch?v=Dl1ufW3cj4g`.

[2] Berry, D.. Molecular animations. accessed: 3/5/2017. URL `http://www.molecularmovies.com/movies/viewanimatorstudio/drew%20berry/`.

[3] Jenkinson, J., McGill, G.. Using 3D animation in biology education: Examining the effects of visual complexity in the representation of dynamic molecular events. Journal of Biocommunication 2013;39(2):E42–E49.

[4] Iwasa, J.H.. Animating the model figure. Trends Cell Biol 2010;20(12):699–704.

[5] Li, W., Ritter, L., Agrawala, M., Curless, B., Salesin, D.. Interactive cutaway illustrations of complex 3d models. ACM Trans Graph 2007;26(3).

[6] Díaz, J., Monclús, E., Navazo, I., Vázquez, P.. Adaptive cross-sections of anatomical models. Computer Graphics Forum 2012;31(7):2155–2164.

[7] Kosara, R., Mackinlay, J.. Storytelling: The next step for visualization. IEEE Computer 2013;46:44–50.

[8] Wohlfart, M., Hauser, H.. Story telling for presentation in volume visualization. In: Proceedings of the 9th Joint Eurographics / IEEE VGTC Conference on Visualization. EUROVIS'07; Eurographics Association; 2007, p. 91–98.

[9] Grimm, S., Bruckner, S., Kanitsar, A., Gröller, M.E.. Flexible direct multi-volume rendering in interactive scenes. In: Vision, Modeling, and Visualization. 2004, p. 386–379.

[10] Hyun, K., Lee, K., Lee, J.. Motion grammars for character animation. Computer Graphics Forum 2016;35(2):103–113.

[11] Karp, P., Feiner, S.. Automated presentation planning of animation using task decomposition with heuristic reasoning. In: Proceedings of Graphics Interface '93. GI '93; Toronto, Ontario, Canada: Canadian Human-Computer Communications Society; 1993, p. 118–127.

[12] Seligmann, D.D., Feiner, S.. Automated generation of intent-based 3d illustrations. SIGGRAPH Comput Graph 1991;25(4):123–132.

[13] Mühler, K., Bade, R., Preim, B.. Adaptive script based animations for intervention planning. In: Proceedings of the 9th International Conference on Medical Image Computing and Computer-Assisted Intervention - Volume Part I. MICCAI'06; Berlin, Heidelberg: Springer-Verlag; 2006, p. 478–485.

[14] Mühler, K., Preim, B.. Reusable Visualizations and Animations for Surgery Planning. Computer Graphics Forum 2010;29(3):1103–1112.

[15] Iserhardt-Bauer, S., Hastreiter, P., Tomandl, B., Köstner, N., Schempershofe, M., Nissen, U., et al. Standardized analysis of intracranial aneurysms using digital video sequences. Medical Image Computing and Computer-Assisted InterventionMICCAI 2002 2002;:411–418.

[16] Hurter, C., Taylor, R., Carpendale, S., Telea, A.. Color tunneling: Interactive exploration and selection in volumetric datasets. In: Proceedings of IEEE Pacific Visualization Symposium (PacificVis 2014). 2014, p. 225–232.

[17] Basch, C.. Animated transitions across multiple dimensions for volumetric data. Master's thesis; Institute of Computer Graphics and Algorithms, Vienna University of Technology; 2011.

[18] Richardson, J.S.. The anatomy and taxonomy of protein structure. Advances in protein chemistry 1981;34:167–339.

[19] Goodsell, D.S.. Atomistic vs. continuous representations in molecular biology. In: Visual Representations and Interpretations. Springer; 1999, p. 146–155.

[20] Heer, J., Robertson, G.. Animated transitions in statistical data graphics. IEEE Transactions on Visualization and Computer Graphics 2007;13(6):1240–1247.

[21] Dragicevic, P., Bezerianos, A., Javed, W., Elmqvist, N., Fekete, J.D.. Temporal distortion for animated transitions. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM; 2011, p. 2009–2018.

[22] Christie, M., Olivier, P., Normand, J.M.. Camera control in computer graphics. In: Computer Graphics Forum; vol. 27. Wiley Online Library; 2008, p. 2197–2218.

[23] Le Muzic, M., Autin, L., Parulek, J., Viola, I.. cellVIEW: a tool for illustrative and multi-scale rendering of large biomolecular datasets. In: Eurographics Workshop on Visual Computing for Biology and Medicine. 2015, p. 61–70.

[24] Johnson, G.T., Autin, L., Al-Alusi, M., Goodsell, D.S., Sanner, M.F., Olson, A.J.. cellpack: a virtual mesoscope to model and visualize structural systems biology. Nature methods 2015;12(1):85–91.

[25] Hsu, W.H., Ma, K.L., Correa, C.. A rendering framework for multiscale views of 3d models. ACM Trans Graph 2011;30(6):131:1–131:10.

[26] Segel, E., Heer, J.. Narrative visualization: Telling stories with data. IEEE Transactions on Visualization and Computer Graphics 2010;16(6):1139–1148.

[27] Lee, T.Y., Lin, C.H., Wang, Y.S., Chen, T.G.. Animation key-frame extraction and simplification using deformation analysis. IEEE Transactions on Circuits and Systems for Video Technology 2008;18(4):478–486.

[28] Huang, K.S., Chang, C.F., Hsu, Y.Y., Yang, S.N.. Key probe: a technique for animation keyframe extraction. The Visual Computer 2005;21(8):532–541.