FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Hybrid Frames

Animated Narrative Sequences of Transitions in Molecular Visualization

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Stappen Stefan
Matrikelnummer 1329020

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assoc. Prof. Dipl.-Ing. Dr. techn. Viola Ivan
Mitwirkung: Sorger Johannes, MSc

Wien, 3. Juli 2017

Stappen Stefan	Viola Ivan

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Hybrid Frames

Animated Narrative Sequences of Transitions in Molecular Visualization

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Stappen Stefan
Registration Number 1329020

to the Faculty of Informatics

at the TU Wien

Advisor: Assoc. Prof. Dipl.-Ing. Dr. techn. Viola Ivan
Assistance: Sorger Johannes, MSc

Vienna, 3rd July, 2017

Stappen Stefan	Viola Ivan

Erklärung zur Verfassung der Arbeit

Stappen Stefan

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Juli 2017

Stappen Stefan

Danksagung

Ich bedanke mich bei meinem Betreuer für das konstruktive und motivierende Feedback während der Arbeit. Außerdem bedanke ich mich für das Korrekturlesen einer unvoreingenommenen Person bei Michael Vögler.

Acknowledgements

I say thanks to my adviser Johannes Sorger, who always gave me constructive and motivating feedback during this thesis. I also thank Michael Vögler for proofreading as a non-biased person.

Kurzfassung

Bei der Visualisierung von Bio-Molekular Daten werden Animationen verwendet um Zustandsübergänge zwischen verschiedenen Repräsentationen darzustellen. Zum Beispiel können Animationen dabei helfen, den Übergang von einer physikalisch korrekten Repräsentation eines HIV Moleküls zu einer abstrakten Visualisierung wie eine Bar Chart, zu verstehen. Mit dieser Technik kann man die Bewegung und Transformation der einzelnen Strukturen während des Übergangs verfolgen. Dadurch werden diese Strukturen in den verschiedenen Repräsentationen in eine direkte Verbindung gebracht. Der Nachteil von einer Animation ist, dass die repräsentierte Information nur kurz erscheint und dann wieder durch die nächste ersetzt wird. Dies erschwert den Vergleich der Zustandsänderung über die Zeit und verhindert die Analyse von Details. Um diesen Nachteil zu reduzieren, beabsichtigen wir eine Hybride Visualisierung, bestehend aus einer Sequenz von Standbildern und einer Animation, einzusetzen. Mit den Standbildern wird ein Vergleich der Zustandsänderungen ermöglicht und gleichzeitig eine Übersicht über den Übergang gegeben. Da aber bei Standbildern der Bezug der Strukturen in den verschiedenen Repräsentationen verloren geht, wird zusätzlich eine Animation eingesetzt. Es ist möglich, direkt durch das Klicken auf eines der Standbilder, die Animation von diesem Zeitpunkt aus zu starten. Die Animation wird automatisch beim nächsten Standbild der Serie angehalten. Weil die Aussagekraft der Sequenz von Standbildern stark davon beeinflusst wird, welche Bilder ausgewählt werden, verwenden wir eine Object-Based Key Frame Extraction namens *Key Probe*, entwickelt von Huang et al. [HCHY05]. Um *Key Probe* auf Molekular Daten, welche aus Tausenden von Proteinen bestehen können, anzuwenden, waren Anpassungen notwendig. Deswegen haben wir einige Optimierungen für *Key Probe* entwickelt. Wir zeigen, dass die angepasste *Key Probe* Technik angemessene Ergebnisse in einer vertretbaren Berechnungszeit liefert. Die so extrahierten Key Frames werden dann zusammen mit einer animierten, 3D Visualisierung der Molekular Daten angezeigt. Damit wird nicht nur ein schneller Überblick über die Zustandsübergänge ermöglicht, sondern auch eine detaillierte Analyse von spezifischen Subsequenzen.

Abstract

In bio-molecular visualizations animations are used to convey transitions between different representation states. For example, to understand the transition from a physical correct representation of a HIV molecule to an abstract visualization like a bar chart, an animation can be used. This method allows to trace the structures during the transition and therefore brings them into a direct relation. The disadvantage of an animation is that the presented content is fleeting. Therefore, the change over time cannot be compared or investigated in detail. To mitigate this disadvantage, we propose a hybrid visualization composed of an animation and a series of still images. The series of still images allows an overview over the transition and comparison of states, but the relation of the representations may get lost. For this reason, the animation is used. By clicking on a single frame inside the sequence the animation is played starting from the selected frame to the next frame in the sequence. The expressiveness of the narrative sequence of images is determined by the selected images. Therefore, we adapted *Key Probe*, an object based key frame extraction method developed by Huang et al. [HCHY05], to be operable on molecular data. Molecular data consists of thousands of protein instances and is therefore challenging for object based key frame extraction methods. We introduced several optimization techniques to *Key Probe*. We show that the adapted *Key Probe* returns reasonable result within a reasonable computation time. The so extracted key frames are then displayed in a sequence of images together with a 3D animated view of the molecular data. This allows, additionally to the fast overview, an in-depth exploration of a specific sub-sequence of the animation.

Contents

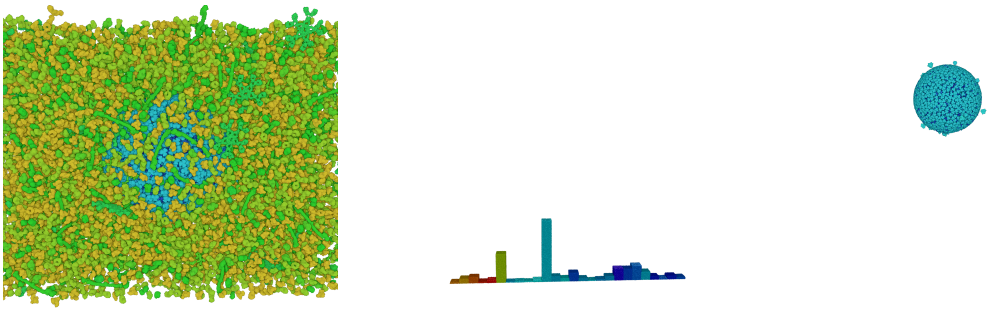
CHAPTER 1

Introduction

Animation is a convenient tool to support the transfer of information that changes over time. As such, it can display the exact order of events inside a process. In addition to the order, the time intervals of events are also exactly represented. While the order may be easily visualized with still images, timings will need an additional visual encoding, like numbers, and the fine grained transitions between animation states may not be efficiently pictured in still images at all.

For instance, let us consider the visualization of molecular data (Figure 1.1 a). Such data can be represented in different states that represent different life cycle states, statistical information (Figure 1.1 b) or visually abstracted representations (Figure 1.1 c). The relations between the different visual states of the same data are not always obvious. Animated transitions can therefore be used to convey the relation in an intuitive way. However, an animation is not always possible or desired.

Like Tversky et al. [TMB02] described, animation has failed to prove significant advantages over graphics in the research literature before their work was published. Tversky et al. also found a root cause for this failure: the human perception and cognition are at their limits when it comes to comprehend the fleeting information states of an animation. The user must be able to understand the structure and content of the presented information. This is especially hard for animations, because the presented information is short-lived and seems not to have an explicit structure, like this is the case, for example, in tables. The inspection of details and the comparison of events in animations is complicated too and requires interaction methods, such as pausing the animation and scrolling forwards and backwards in time. As stated by Tversky et al. [TMB02] graphics, we call them still images, are used by humans to outsource information and so help off-loading their mind. Graphics also support groups to come to a common understanding and let individuals convey and understand information. Still images might achieve these goals for static information. As long as a time changing aspect may be discretized enough it may be depicted with a sequence of still images, as well. However, this strategy is not infinitely



(a) A physically correct visualization of molecular data.

(b) A bar chart visualization of the same molecular data.

(c) An abstracted form of the molecular data, omitting the cytoplasm.

Figure 1.1: Examples of molecular data representation

scalable. If the number of still images needed, which are representable at an acceptable size, exceeds the capacities of a display or the field of view, then this technique of discretization has reached its limits. Here, animation could help to overcome these issues by displaying the whole changing information set in a single transition over time. In the end, both techniques have their individual advantages and disadvantages. In this work, we attempt to join the strengths of both representation forms in order to alleviate their downsides.

Therefore, we developed a visualization technique to lead a user through a transition between representation states. On the one side, a user should be guided through the transition with key frames that promote the comprehension. These key frames of the animation hold crucial information of the transition. They allow an overview of the main events in the transition and an inspection of the details and comparison of these events. On the other side, if users want to know about the detailed transition between two states/key frames, the animation information between those frames is at their disposal by clicking onto the respective key frame thumbnail. We call this technique *Hybrid Frame Vis*, see Figure 1.2.

Our approach is based on a framework for visualizing biological data at a resolution of atoms in real time developed by Le Muzic et al. [LMAPV15]. The framework is capable of fluently visualizing the transition between two states of molecular data. This is achieved by an animation, which intuitively describes this transition. For this framework, a hybrid visualization should be developed that shows a sequence of frames from the starting molecular state to the final molecular state. The goal of the hybrid visualization is to offer the user a technique that combines the advantages of still images and animation.

To achieve this, key frames of the animation must be extracted from the molecule transition and displayed together in a capable visualization. This visualization must efficiently combine the animation with still images. It must signal the user the affordance

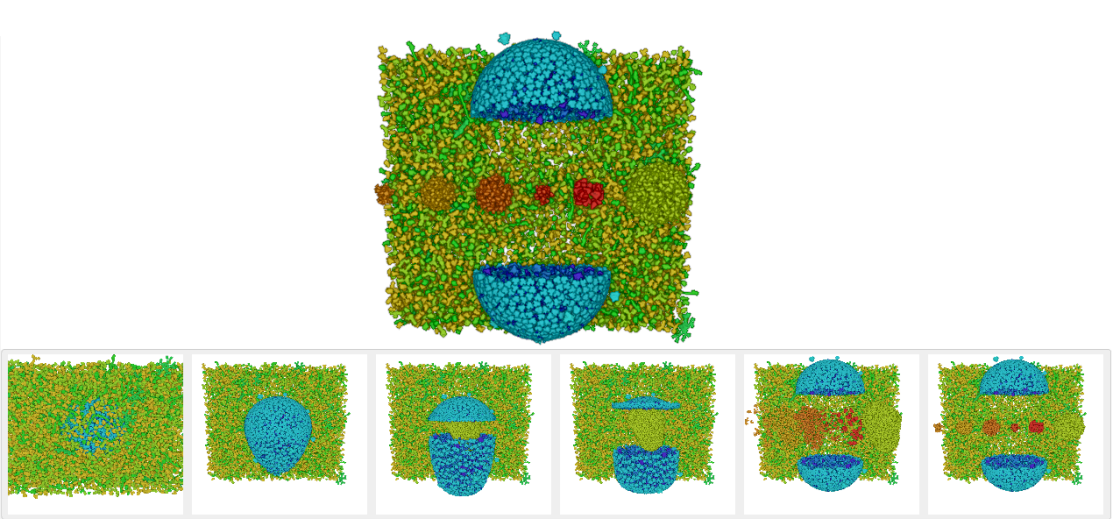


Figure 1.2: The hybrid frame visualization developed by us.

of exploring the animation in a fast way, while preserving the capability of an in-depth analysis. To fulfill these requirements, we propose a hybrid fame visualization. The hybrid frame visualization will contain an area to display the whole animation and an area for the key frames. While the animation is already offered by the framework and just needs integration with the frame visualization, key frame extraction is a critical point of the visualization. Key frames are a still image representation of a molecule transition. They must embody the essence of the transition. In other words, it should be possible to view the frames and have a grasp of the transition. We consider key frame extraction methods to fulfill these needs. The next challenge therefore is to find a suitable object based or image based key frame extraction method. While object based approaches directly work on 3D spatial objects, image based approaches need image data that is segmented or directly analyzed. The extraction method needs to be capable for operating on large molecular data sets containing thousands of protein instances with properties, such as shape, type or color. Besides the large spatial resolutions of these data sets, their temporal resolution from 30 to 60 fps must also be taken into account, when choosing an appropriate key frame extraction method.

The four major parts of this work therefore consist of:

- an extensive literature study of object based key frame extraction algorithms
- the implementation and adaption of the *Key Probe* keyframe extraction technique, developed by Huang et al. [HCHY05]
- the implementation of the hybrid frame approach within the *cellVIEW* environment
- an evaluation of the implemented keyframe extraction approach

In the following, Chapter 2, we review the related work concerning spatio-temporal visualizations and key frame extraction methods. In this review, the existing methods for key frame extraction and the existing hybrid frame visualizations will be analyzed and compared. Then the selection of the key frame extraction method will be justified and the selected method will be discussed in more detail. In the next chapter, Chapter 3, the features and functionality of the *Hybrid Frame Vis* will be specified. This contains a description of the user interactions and an explanation of *Key Probe* from a theoretical point of view. In Chapter 4 the solution will be presented. The used data structures, implementation details of the hybrid frame visualization and the implementation of the key frame extraction method are described thoroughly. All the implemented optimization techniques of the key frame extraction method with their impact to the extracted key frames and their performance, will also be discussed in detail. Furthermore, considered optimizations of the key frame extraction technique that are not applicable or introduce severe problems are discussed too. To evaluate our approach, we will compare the selected and adapted key frame extraction method to uniform sampled frames and discuss the results of the solution in Chapter 5.

CHAPTER 2

Related Work

In this chapter, the current body of literature related to our approach of the visualization of time-dependent 3D molecular data will be discussed. The related work can be split up into two parts: one concerning spatio-temporal data visualizations and the other one, concerning key frame extraction techniques needed for our visualization.

2.1 Spatio-Temporal Data Visualization

Visualization of spatio-temporal data is a challenging field for scientific and information visualization. As mentioned in the introduction, animation is capable of visualizing this kind of data, but has its limitations. A survey giving a systematic view on time-dependent data visualization was presented by Aigner et al. [AMM+07]. Andrienko et al. [AAG03] give an overview of exploratory spatio-temporal visualization based on a geo-spatio-temporal data example. Compieta et al. [CDMB+07] investigate large scale spatio-temporal data sets for decision support. They utilize data mining techniques combined with visualization tools to filter out relevant information and display it. Their approach works for example on google earth data or on association rules, which are the result of mining algorithms. In contrast to our work, their focus lies on visual analytics of large scale spatio-temporal data.

Liao et al. [LHM14] create animations from data that is generated from tracking interactions of scientists with a volume renderer that is well known by them. Their tool tracks the user inputs, calibrations of the renderer, viewpoint changes and the time spent in a specific configuration and automatically calculates an animation based on this information. This animation should support the understanding of the spatial structure of the volume data and its changes over time. As their approach is about authoring an animation it was not considered for this work.

What differentiates our approach from other visualization approaches for spatio-temporal data is the nature of our data. The molecular data sets can be compared to large point clouds of millions of individual points, i.e., the atoms. Persistent visualization of the events of such dynamic point cloud data has not yet been explicitly addressed in literature. In their survey of point cloud data, Richter and Döller [RD14] discuss time varying point clouds for computation of steady and moving objects as well as the reduction of temporal redundancy in the data, but no visualizations are presented. For this reason, we considered analogies to time-dependent data of other formats, like volume data. Point clouds correspond to volume data that is organized on a grid. Each point is given explicitly and the neighborhood is defined by distance measures. Time varying volume data visualization capable of supporting scientists in exploring and understanding the dynamic aspects of physical and chemical processes is a well-known challenge in visualization research as described in the review of Ma [Ma03]. Widanagamaachchi et al. [WCPB12] proposes a method using dynamic tracking graphs of extracted features from volume data to interactively explore large-scale time varying data. The tracking graphs are presented to the users with a linked 3D view of the volume data for the current selected time-step. Inside the tracking graph, events of features like births, splits, merges and deaths can be easily grasped and features can be filtered. Features can be selected in the tracking graph and are then visualized with a different color in the 3D view. Therefore, the exploration of time and space is split into two views. A very similar approach is presented by Reinders et al. [RPS01]. They use a feature tracking graph combined with a 3D view as well. The 3D view shows only glyphs of the features and is thus displayed on a more abstract level. They also added a prediction of paths. The focus lies not on feature filtering but on path finding of features through time. These kinds of methods require feature extraction and are not scalable with thousands of features. Feature extraction of molecular data animations, where each atom is allowed to move differently, is not applicable as too many features for such visualizations would be generated. On the other hand, if feature extraction would be set too coarse, too much information would get lost for adequately conveying the animation. Lu and Shen [LS08] have developed a method to visualize volume data with a large number of time steps, which is applicable for dynamic molecular data. They visualize the data as an interactive storyboard of key frames connected by paths. The positions on the path encode the similarities of consecutive frames. Therefore, the time-line does not increase from left to right but can be intertwined in the 2D space of the storyboard. To not lose the direction of time, it is color coded on the paths with a rainbow or grey color scale. The scale is depicted beneath the storyboard.

Another analogy to time-dependent point clouds can be found in flow data. While point clouds just encode positions and meta data of points, flow data has a spatial as well as temporal component per data point. To visualize this type of data, often particle based systems are used where one or more particles are set into the flow data and moved along the flow directions. These visualizations and other types of flow visualizations might be applicable for animated point clouds. Doleisch et al. [Dol07] visualize simulated flows with linked views. There are two kinds of views: filtering and 3D views. Filtering

views can be used to filter the data in the linked 3D views. 3D views visualize the flow at different time steps and so enable the selective exploration of flow over time. Guo et al. [GWY+11] use three linked views to visualize traffic data at a road intersection collected with laser scanners and other devices. A spatial view shows the traffic with trajectory information at a given timestamp selected with two hierarchical time-sliders. A themeriver with glyphs and scatterplots visualizes the directions in the time domain and parallel coordinates show multiple properties like speed, distance and angles of vehicles and pedestrians. Bruckner and Möller [BM10] developed a tool to visually explore the parameter space of physical simulations of phenomena, such as fire or smoke, for content creation in computer animation. They randomly sample the parameter space, compute simulation sequences and evaluate spatio-temporal similarities. These simulation sequences are segmented into visually distinct phases. The so computed parameters, clusters and sequences are then visualized in multiple linked views. A 3D animation view shows the currently selected sequence. In the sequence view, the user can browse through the sequences of the simulation, which are visualized as film-strips. The cluster view gives an overview of the visual variations across the sequences and allows the user to search for specific visual properties. Waser et al. [WKS+14] supports disaster management with a visual system to make and justify decisions. This system takes into account logistics, resource limits, building times of walls etc., and presents these factors in grouped tracks of events on a time line. In this time line, scenarios and time-steps can be selected, which are then visualized in a 3D animated view. In this view, building damages can be observed, as well as construction details of barriers and other factors, like the logistics for building the barriers. Logistics are visually described by truck paths, their position, their target and the progress of building a barrier. Konev et al. [KWS+14] have similar targets but use different visualizations. With flood simulation actions plans are automatically computed. These action plans are visualized to convey details and justify decisions. Therefore, a storyboard like approach is used. Storyboards are used to visualize an action plan or visually justify decisions. The visualizations are composed of story blocks. A story block represents a barrier placement. With a toggle button, the perspective can be switched between a close-up perspective of details for the barrier placement and an overview perspective for surroundings. A story block also works as a thumbnail to a main view where the user can navigate through a 3D rendering and get details, like materials needed, of a barrier. Such storyboards or sequence visualizations could be useful to convey spatio-temporal data, especially in combination with a 3D view showing the animation of frames in the sequence. This allows a fast overview and a detailed exploration of the animation.

An alternative visualization method for spatio temporal data is proposed by Meyer et al. [MGA+08], who use 4D multitemporal visualizations. In a query driven visualization approach, frames are selected, which are then composited and color coded to convey temporal and spatial data in a 4D image. A hybrid visualization of time varying data was developed by Rufiange and McGuffin [RM13]. They visualize dynamic networks changing over time with three types of tiles in a frame sequence. Diff tiles show the differences of the network over a certain time interval. Animation tiles show the evolution of the

graph over a time interval by pulling the clicked cursor from left to right. Small multiple tiles show the graph state at an individual time slice. These slices can be interactively manipulated. For example, it is possible to combine three small multiple tiles to an animated tile or split them up again. All time slices of the data are visualized in a sequence order from left to right. To search in a large data set of human motions, Bernard et al. [BWK+13] integrated hierarchical views, a motion explorer and a filtering view. The hierarchical view, a dendrogram, shows motion sets clustered by their type. The filtering view enables searching inside the motion data sets. The motion explorer displays motions as linked sequences of frames and allows semantic zooming with different levels of detail showing more or fewer key frames and allows filtering for specific motions.

The visualization of dynamic molecular data with the goal of conveying transitions has not yet been explicitly addressed in the current body of literature. However, there are already techniques utilizing a combination of still images and animations, but with different goals, working on different data sets. Bruckner and Möller [BM10] have the goal to find parameters for visually appealing simulations and not to convey the motion of these simulations. The goal of Konev et al. [KWS+14] is very similar to ours. They want to support the user in conveying the action plans and decisions made. In contrast to our work, they know the key events in their simulated data, namely barrier placements. As such, their data is completely different and their approach cannot be directly applied to dynamic molecular data.

2.2 Key Frame Extraction

For the already mentioned hybrid visualization, frames are needed that describe the transition in a comprehensible way and do not leave out the important details that allow to reconstruct the transition in mind. In the first prototype of the hybrid visualization, frames with a constant time interval were selected for representing the sequence of events in the animation. However, this approach left out important frames that are indispensable to comprehend the transition. Another problem with this method is, that in phases such as the end of the transition, where not much change is occurring in the animation, redundant information is represented, as the extracted frames are too similar. A regular sampling of the animation therefore does not convey all important parts of a transition and also results in redundant frames. For these reasons, we considered a method to extract key frames that are based on the amount of change that is occurring in the animation in order to capture all important details and to avoid redundancy. During a literature review about key frame extraction techniques, many methods were found, but only few fit the specific needs of this application. First of all, only papers were considered that give enough technical details on how a method works so that it could be implemented. Afterwards, the papers were validated in terms of whether they could be used to extract key frames of 3D spatio-temporal data. The vast majority of literature for key frame extraction is focused on image based 2D video, such as the work of Zhuang et al. [ZRHM98]. These methods do not take advantage of the 3D spatial information of individual entities in the point cloud transition. 3D object based key

frame extraction approaches have the luxury of utilizing data that is already segmented into individual objects of which the change can be tracked over time. However, 2D image based approaches have to work on the level of individual pixels or have to segment the objects in a frame and track such objects across a sequence of frames, which due to occlusion or image quality is not always possible.

These findings led us to investigate object based key frame extraction methods. These papers were already much more fitting, because they considered 3D information and animation. Most of these papers that are concerned with animation have their focus on skeleton animation and human motion recognition, like the work of Baak et al. [BMS08] and Yamasaki et al. [YA06]. These methods could in principle be adapted to animations of point clouds but are not designed for them and therefore would not deliver the key frames needed to comprehend the transition. For this reason, also the method based on decimation of unimportant frames by importance, of Li et al. [LOT05], was not used. Lee et al. [LLWC08] proposed another method for key frame extraction with the goal of a compact representation for motion and geometry features of an animation. While this would almost fulfill all criteria to extract key frames of the molecular transition, it is based on deformation analysis of the meshes, which is not applicable for molecular point clouds. Two methods were found, which fit all needs and therefore could be adapted for 3D molecular transitions: *Key Probe* by Huang et al. [HCHY05] and a method utilizing self-similarity and volume-shape histograms by Huang et al. [HHS08]. The decision was made in favor of the Key Probe approach, because on the one side, a feature vector may contain not only geometric information but also other information, like the type of molecules. This influences the key frame extraction and takes advantage of the additional information. On the other side, the whole animation can be reconstructed by blending the key frames with a weight. This promotes the expressiveness of the selected key frames. Another advantage of this method is the hierarchical representation of key frames, which could be utilized to make an in-depth search through the selected key frames.

CHAPTER 3

Hybrid Frame Visualization

In order to create a visualization that joins the advantages of still images and animated sequences, we propose our Hybrid Frame Visualization. The visualization combines both, the high detail temporal visualization of the transition via an animation and the overview of the whole transition with a sequence of selected frames, as depicted in Figure 1.2.

In the following the features and the functionality from a user's point of view will be discussed. Then the theoretical background of the key frame extraction method *Key Probe* will be explained.

3.1 Features and Functionality

The Hybrid Frame Visualization presented in this thesis combines a sequence of frames with an animated 3D viewer. The sequence of frames is displayed at the bottom of the 3D viewer that plays back the animation in real-time and allows simple interaction. The complete UI of the Hybrid Frame Visualization is depicted in Figure 3.1. It shows the 3D view of *CellView*, developed by Le Muzic et al. [LMAPV15], with the frame sequence. On the right side, a panel for configuring the parameters of the key frame computation and layer rendering is depicted. This panel is shown in detail in Figure 3.2. The parameters configurable with this panel include the key frame count, which defines the numbers of key frames to be computed. After configuring the parameters in the panel, the computation of the key frames can be started with a button, marked with 1 in Figure 3.2. When the computation of the key frames has finished or when clicking on the "Update Layer" button, marked with 2 in Figure 3.2, the key frames are shown in the bottom panel. The user can click on a frame in this panel and the 3D viewer starts rendering the animation from the clicked frame to the next key frame in the sequence. If the number of key frames selected by the user exceeds the screen space, a scroll bar is shown to navigate through the sequence.

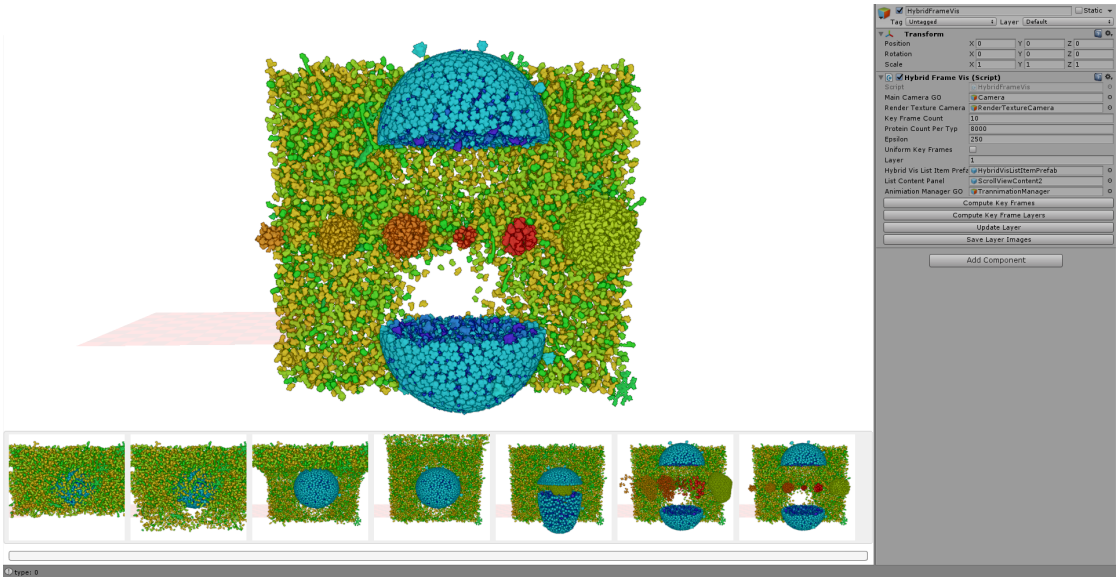


Figure 3.1: The complete Hybrid Frame Visualization including its graphical user interface.

Additionally, we implemented a hierarchical visualization, mentioned by Huang et al. for *Key Probe* [HCHY05]. This is presented by hierarchical layers that can be viewed in Figure 3.3. The layers describe different abstraction levels of the key frames. One layer is computed from the key frames of the underlying layer inside a recursive algorithm. This allows a fast and high-level overview of the animation and a drill-down workflow to explore more details. The layer of key frames can be selected with the right panel with an integer input form, depicted as 3 in Figure 3.2. Another button, marked with 4 in Figure 3.2, computes all layers higher than the initial layer and requires the computation of the initial key frames first. With a third button, marked with 2 in Figure 3.2, the selected layer is rendered to the sequence of frames in the UI.

The main goal of the sequence of images is, to give the user a fast overview of the spatio-temporal data and allow the user to grasp the key states and navigate through them. This sequence has therefore the requirement to describe the states of the animation. To fulfill this requirement, we used an object-based key frame extraction method: *Key Probe*, developed by Huang et al. [HCHY05].

3.2 Theoretical Background of *Key Probe*

Key Probe is a constraint based key frame extraction technique developed by Huang et al. [HCHY05]. It operates on skeleton-based motions or animated meshes. Animations are transformed to a key frame-based representation. To achieve this, the problem is defined as a constrained matrix factorization problem, which is solved by an algorithm

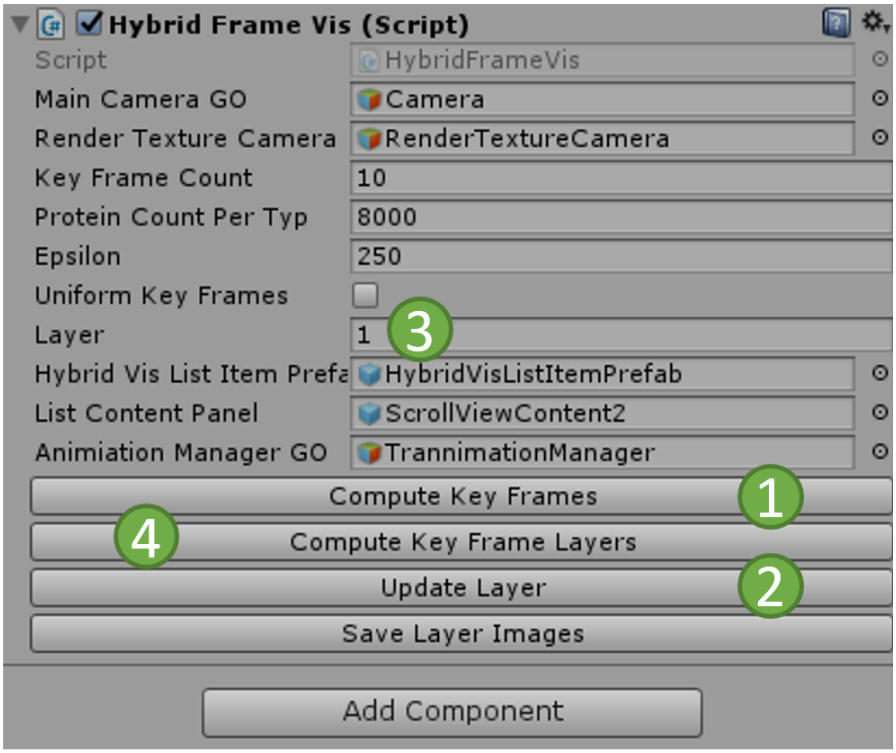


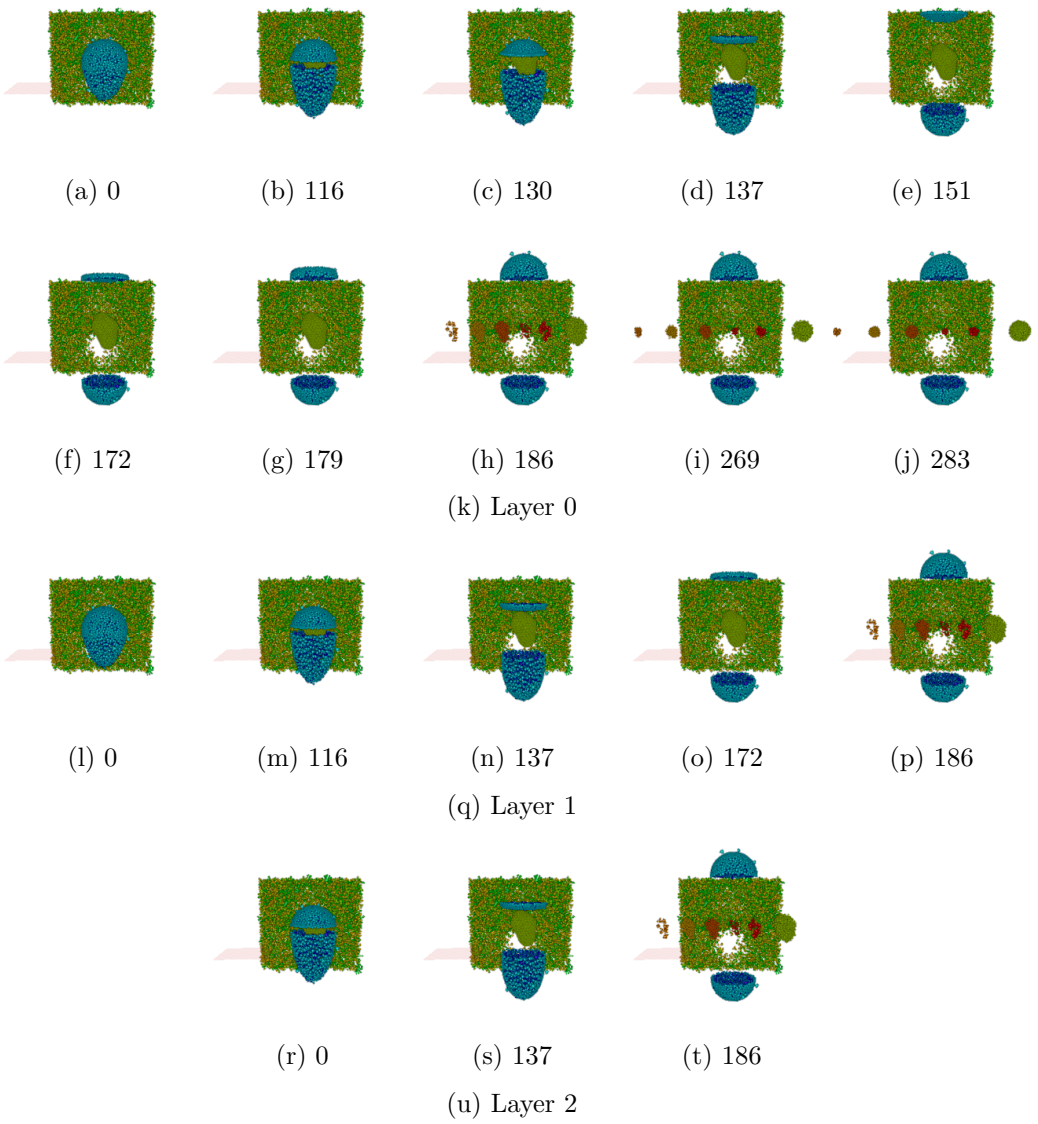
Figure 3.2: The right panel of the Hybrid Frame Visualization for configuring the parameters of the key frame computation. The important points are annotated: 1. The button for key frame computation. 2. the button to render the computed key frames. 3. the selection of the layer for the hierarchical visualization. 4. the button to compute hierarchical layers.

based on the least-squares optimization technique. This allows two applications: In the first application, animations could be compressed into key frames and reconstructed by blending these key frames with a weight matrix. The second application allows animations to be browsed by key frames to give an overview. The method was empirically proven to be faster than methods using principal component analysis or independent component analysis. Experiments have also shown the high quality and compression rates achieved with their algorithms.

The constrained matrix factorization problem is defined by the following equation. The animation, encoded as a matrix A is composed by two matrix factors W and H:

$$A \approx WH \tag{3.1}$$

In this equation W defines a weight matrix and H the key matrix. A defines the whole animation with n frames, which are represented by the rows. One row encodes one key frame with v points. In a 3D space, a point has three components and therefore the matrix



(a) 0 (b) 116 (c) 130 (d) 137 (e) 151

(f) 172 (g) 179 (h) 186 (i) 269 (j) 283

(k) Layer 0

(l) 0 (m) 116 (n) 137 (o) 172 (p) 186

(q) Layer 1

(r) 0 (s) 137 (t) 186

(u) Layer 2

Figure 3.3: Extracted key frames for the hierarchical visualization. The layers can be found underneath each group. The frames are annotated with their frame numbers.

has $m = 3v$ columns. If not only positions, but for example deformation at points or rotational aspects, are available, they can be encoded in the row of a frame as well. This allows to utilize much more information visible to the viewer of the animation than just changes in positions. The number of key frames, which should represent the animation, is defined by k. Therefore, the weight matrix W has the dimensions $n \times k$ and the key matrix H has the dimensions $k \times m$. The multiplication of these matrices approximates the animation matrix A. To achieve a compression and enable browsing the number

of key frames k should be lower than the number of all frames n. To approximately reconstruct the lth frame the following equation is used:

$$v_l = w_{l,1}h_1 + w_{l,2}h_2 + ... + w_{l,k}h_k \tag{3.2}$$

Where v_l is the lth frame, $w_{l,i}$ is the weight of the lth frame and the ith key frame and h_i is the ith key frame. Compared to the overall number of frames, usually few key frames are used, e.g., 10 key frames/300 frames. To achieve a good approximation of the original animation with these low number of key frames, they must encode the core information within the animation. Therefore, the key frames represent the animation faithfully. The reconstruction is basically a linear combination of key frames and weights, which allows an approximate reconstruction of every frame of the original animation. For this reason, key frames must not essentially be frames of the animation, as a reconstruction would be mathematically possible with other base vectors too. In *Key Probe*, each key frame belongs to the animation, which constrains the selection of key frames. The constraint of not allowing such derived key frames enables a reduced factorization:

$$A_{nkey} \approx W^{q\times k}H^{k\times m} \tag{3.3}$$

As the key frames are part of the animation, only the non-key frames must be reconstructed. The number of the non-key frames is computed with $q = n - k$. The complete animation can then be computed with the union of the key frames and the reconstructed non-key frames. To preserve the temporal sequence of the frames besides the weight matrix $W^{q\times k}$ and the key matrix $H^{k\times m}$, the original indices of the key frames must be saved.

To compute the key matrix and the weights an iterative algorithm is proposed:

$$u \leftarrow arg \min_{v\in A_{nkey}} ||A_{nkey} - A_{nkey}H^{\dagger}H||^2 \tag{3.4}$$

$$H^{i+1} \leftarrow H^i \oplus A^u_{nkey} \quad \text{and} \quad A_{nkey} \leftarrow A_{nkey} \ominus A^u_{nkey} \tag{3.5}$$

$$W^{i+1} \leftarrow A_{nkey}H^{\dagger i+1} \tag{3.6}$$

In Equation 3.4 the index u of the frame $v \in A_{nkey}$ that reduces the reconstruction error of the remaining non-key frames best, is selected with the method of least squares. Afterwards, in Equation 3.5 the selected key frame is inserted in the key matrix H and removed from the non-key matrix A_{nkey}. Finally, in Equation 3.6 the weights matrix is computed by multiplying the non-key matrix A_{nkey} and the pseudoinverse of the key matrix H. As $A_{nkey} \approx WH$, the weight matrix is approximated with the pseudoinverse of H. The pseudoinverse is needed as the inverse is undefined for non-square matrices. It

is also used to compute an intermediate W in Equation 3.4: $A_{nkey}H^{\dagger}$. The iteration is stopped when a user specified number of key frames is reached, which can be counted by i. The iteration can also be stopped when a user specified tolerance regarding the reconstruction error is reached.

To achieve a hierarchical representation if there are too many key frames for browsing or too much redundancies, a recursive approach is presented by Huang et al. [HCHY05]. To compute a hierarchical layer, the key frames from the previous layer are interpreted as the whole animation and the process is reapplied. In other words, key frames from key frames are selected.

CHAPTER 4

Technical Realization

In this chapter, we explain the details of the technical realization of our Hybrid Frame Visualization described in Chapter 3. First of all, we will discuss the data structures. This will include a description of the molecular data structures and the animation data structures, the data structures we used for *Key Probe* data and the data structures we used for the implementation of the Hybrid Frame Visualization. Large-scale spatio-temporal data sets, like our animated molecular data, have a large memory footprint and *Key Probe* was not intended for this type of data and is therefore computationally expensive. To overcome these bottlenecks, adaptions of *Key Probe* were needed. We have enabled the operability of this method on large-scale spatio-temporal data sets with various optimizations techniques. Our implementation of *Key Probe*, including the adaptions, will be described in detail in the section following the data structures. This section is followed by an analysis of our considered optimizations techniques including implementation details. Then, we describe how the Hybrid Frame Visualization was implemented. Afterwards, the implementation environment will be covered briefly. Finally, we explain the general applicability of the implemented Hybrid Frame Visualization.

4.1 Data Structures

4.1.1 Molecular Data Structures

The molecular data is represented by a list of proteins containing the positions, rotations and scale factors. Each protein again consists of a list of atoms, containing positions, rotations and scale factors. This data representation is also called a point cloud. The molecular data sets were created by *Cellpack*, developed by Johnson et al. [JAAA+15]. The proteins are taken from the protein database. For more details, see the *CellView* publication by Le Muzic et al. [LMAPV15].

4.1.2 Animation Data Structures

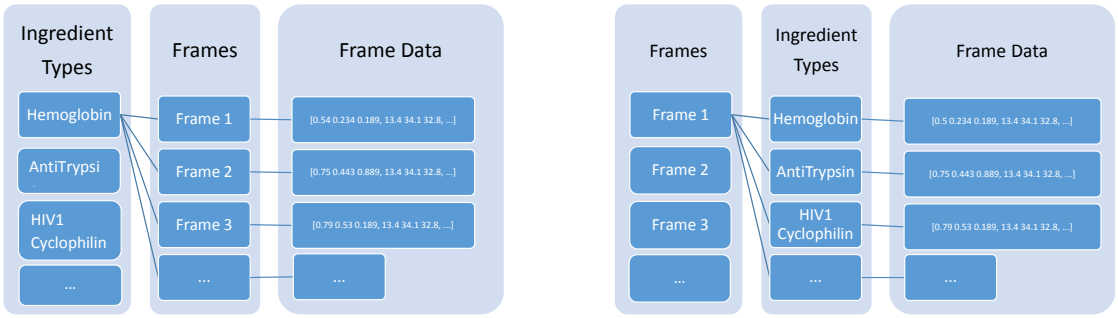
The animations are stored as a list of frame objects. Each frame contains again a list of positions, a list of rotations, a list of scale factors and a list of group IDs for each protein. The positions describe the 3D position in world space coordinates. The rotations and scale factors encode the visual features of a point. The group id is a reference to the animation group. The index in each of these lists is a reference to a list containing the index to so called ingredients. An ingredient represents a type of a structure in a molecule. The ingredient index may be used to query data like the ingredient name, the count of atoms or the color of the ingredient. With another list, a mapping between an ingredient index and the hierarchy of molecular data is achieved. Inside this hierarchy the names of the ingredients are hierarchically stored as strings. A layer of hierarchy is separated by a point inside the string. These are the relevant data structures of the cell animation that are given as input.

4.1.3 Key Probe Data Structures

Key Probe, presented by Huang et al. [HCHY05], operates on matrices. There are a weight matrix, a key matrix and a non-key matrix. The weight matrix defines just the weights for reconstruction and the cells of it have no specific meaning. The key matrix and the non-key matrix contain the frames of the animation in their rows. A frame in its simplest form contains only positions of protein instances. In a 3D space, a position has three values. The three values of the positions are subsequently inserted into one vector. This vector then represents the frame. These vectors are then placed in the rows of the matrix. It is possible to encode more data inside these matrices to emphasize specific properties of a point. These properties may contain local rotations, deformations, colors and so forth. Besides the mathematical data structures, the matrices, no other data structures were specified by Huang et al. [HCHY05].

To take into account ingredient data, groups of such matrices are possible. We consider two organizations of these groups, depicted in Figure 4.1. First, a list containing lists for each ingredient type, see Figure 4.1 (a). These lists contain all frames of one type. One frame is represented by a double array. This structure has the advantages that all frames of one type are contained in a list and are not needed to be reconfigured, if the *Key Probe* method is executed only on one ingredient type. The disadvantage is, that in a preprocess step, the frames need to be reorganized. Another disadvantage can be found in the deletion and insertion process of key frames. When a key frame is selected it must be deleted from the non-key matrix and inserted into the key matrix. With the grouped data structures, this would be equal to collect the double arrays, representing the frames, from each ingredient type list in the non-key list, delete them and then added them to the respective ingredient type key lists. Second, an alternative structure would be to store a list of key frames, see Figure 4.1 (b). Each frame, inside this list, is a list of ingredient groups. This ingredient groups contain a double array that represents the frame data of this group. This structure allows direct deletion and insertion of whole frames with all their groups from the non-key matrix and the key matrix, as it is needed during a

key frame selection. For the computation of a *Key Probe* iteration all frames of one group must be collected in a list and transformed into a matrix. This adds substantial data reorganization steps to each iteration. Another data structure needed, is the index lookup dictionary. It is a sorted dictionary where the keys are integers and the values are integers too. It saves the index of a key frame in the current iteration as key and as value the global index of the key frame in the whole animation. The interesting fact about this data structure is that it may contain a key twice, which is achieved with a special comparator. This is important, because in an iteration of *Key Probe* the same index as a previous index may occur, as the non-key matrix is reduced subsequently. However, the index inside the whole animation must be different as it is unique.



(a) A list of frames inside a list of ingredient types.

(b) A list of ingredient types inside a list of frames.

Figure 4.1: Different data structures for the implementation of *Key Probe*.

4.1.4 Hybrid Frame Visualization Data Structures

The data structure for the Hybrid Frame Visualization is very simple. The user can select the number of key frames to compute. For this reason, a dynamic growing list of hybrid frame objects is used. If the user selects more key frames to compute than before, the list grows. If the user selects less key frames to compute than before, the not used list items are set inactive and are not rendered. A hybrid frame object contains a texture, which functions as a thumbnail. It also contains the index of the start and the end frame, and a state that tells if it is active or not.

4.2 Implementation of *Key Probe*

One of the four core tasks of this work, was to adapt the *Key Probe* technique for extracting key frames of molecular data. Molecular data is presented by point clouds of atoms. One molecular data set could have potentially thousands, millions or even several billions of atoms. Each point has additional data attached as described in Chapter 4.1. *Key Probe* operates on data structures like animated meshes or motion captures. A theoretical overview of *Key Probe* can be found in Chapter 3.2. A straight forward port of *Key Probe* to molecular data would use the positions of the points in the point cloud like positions

of the mesh or the motion captures. As animated meshes and skeleton-based motions have only few vertices compared to point clouds of cell data, major optimizations had to be introduced. For example, an intermediate step in the least square selection of a key frame would require a dense matrix of the size $m \times m \quad \text{with} \quad m = 3v$, where v describes the number of points. The matrices in a 32-bit process are limited to 16384×16384, which would take 2GiB of memory for the double data type. Double is used to avoid too strong precision errors. This limitation is introduced by C# as arrays, the way matrices are saved, are limited to 2GiB of memory as stated by Rüegg [Rü]. Even if a 64-bit environment would be used and *gcAllowVeryLargeObjects* would be enabled in *.net* the max size of the matrix would be limited 46340×46340, which would take 16GiB of memory. This is not applicable for data sets that for example have about 20000 points. These data sets would need a 60000×60000 matrix, which exceeds the limitations by far. For this reason, we adapted the key frame extraction to work only on groups of molecules. The basic algorithm for the key frame selection for one group is depicted in Formula 4.1.

Algorithm 4.1: Basic Key Frame Selection with *Key Probe*

Input: Matrix with all frames $\mathbf{A_{nkey}} = (a_{nm})$, empty matrix $\mathbf{A_{key}} = (a_{km})$, integer **keyFrameCount** $= k$

Output: Matrix with key frames $\mathbf{A_{key}} = (a_{km})$, integer list of key indices **keyIndices**

```
1 for i ← 0 to keyFrameCount-1 do
2 |   int keyIndex = ComputeKeyIndex(A_nkey, A_key);
3 |   double[] keyFrame = A_nkey.getRow(keyIndex);
4 |   A_nkey = A_nkey.removeRow(keyFrame);
5 |   A_key = A_key.addRow(keyFrame);
6 |   keyIndices.add(keyIndex);
7 end
8 return keyIndices;
```

It selects k key frames of the non-key matrix A_{nkey} that is initialized with the entire animation. The selection is done iteratively. In each iteration, a key frame is selected by the Formula 4.2. The selected key frame is then fetched from the non-key matrix and removed from it. Then it is added to the key matrix and the index is added to a key indices list. This index is a local index. It is only valid inside the current iteration. To compute the global index inside the whole animation a very specific data structure is needed. This is presented by a sorted dictionary, in which the global indices are saved as values. The key to the global index is the local index of the iteration the key frame was found. The sorted dictionary was configured with a special sorter that allows duplicate keys. This is needed because the same key index can occur multiple times in different iterations. For example, the 5th frame is selected in the first iteration. This key frame is then removed from the non-key list. The global index is 5. In a later iteration, the 5th frame is again selected from the non-key list. This cannot be the same frame as selected before. The global index is different, but the local index is the same. Now all iterations of

Key Probe that were computed before and have had a lower global index then the current selected index would have globally, need to be taken into account. This is achieved by iterating the sorted dictionary when a new key frame index is added, as long as there are keys found that are smaller or equal than the current found local index. For each local index that is lower or equal than the current index, the current index is increased by one. The so computed index may now already be contained as a value in the sorted dictionary. That means that a previous iteration was not handled by the lower local index iteration and the key frame index must be increased by one. As long as the new key index is contained in the list it must be increased by one. The so found global index is added as value with the local index of this iteration as key to the sorted dictionary.

As an example, see Figure 4.2, consider a sequence of the following frames: 0 1 2 3 4 5 ... Now the local index 0 is selected. As the sorted dictionary is empty, there could not be any local indices before that were lower or equal to zero. Therefore, the index is directly added with its global index (0,0). The remaining sequence of frames then is 1 2 3 4 5 6 ... Now the local index 1 is selected. It can be assumed that the global index is 2. The algorithm now iterates over the keys in the sorted dictionary. There is only one: 0. 0 is lower than 1. Therefore, the index is increased by one. The new-found index is 2. No other keys of the sorted list are lower or equal to 1. Therefore, 2 is the global index and is added to the sorted dictionary (1,2). The remaining sequence of frames then is 1 3 4 5 6 7 ... The next selected local index is 0 again. The same procedure is executed and (0,1) is added to the sorted dictionary. The remaining sequence of frames then is 3 4 5 6 7 8 ... Now again 0 is selected as the local index. There are two keys in the dictionary lower or equal to zero: 0 and 0. Therefore, the local index is increased by two to compute the global index. The new global index would be 2, but this index is already contained in the sorted dictionary as value. Therefore, it needs to be increased by one. 3 is not contained in the list and therefore is the global index.

INDEX LIST DURING KEY PROBE ITERATION	SORTED DICTONARY	OPERATION
↓ **(0,0)**		
{0, 1, 2, 3, 4, 5, ...}	{}	Remove 0, Insert (0,0)
↓ **(1,1)**	0 ≤ 1→ +1	
{1, 2, 3, 4, 5, 6, ...}	{(0,0)}	Remove 2, Insert (1,2)
↓ **(0,0)**	0 ≤ 0→ +1	
{1, 3, 4, 5, 6, ...}	{(0,0),(1,2)}	Remove 1, Insert (0,1)
↓ **(0,0)**	0 ≤ 0, 0 ≤ 0→ +2; 2 = 2 → +1	
{3, 4, 5, 6, ...}	{(0,0),(0,1),(1,2)}	Remove 3, Insert (0,3)
{4, 5, 6, ...}	{(0,0),(0,1),(0,3),(1,2)}	

Figure 4.2: The computation of the sorted dictionary containing the global indices of the frames inside the animation.

The function ComputeKeyIndex in its basic structure can be viewed in Formula 4.2. It iterates over all non-key frames and selects the ith frame each iteration. This frame

is temporarily added to the key matrix and removed from the non-key matrix. Then the reconstruction error is computed with the method of squared error. This is the part with the main logic of the algorithm. First of all, the weight matrix is computed $W = emp_{nkey} * temp^{\dagger}_{key}$. It consists of the non-key matrix and the pseudo inverse of the key matrix. The pseudo inverse is needed because the matrix may not be square. For non-square matrices, the pseudo inverse is needed as described by Golub et al. [GVL96]. The *Moore-Penrose pseudo inverse* can be computed with *math.net* with a QR factorization. This is a memory and computation intensive process. In most cases the rows of the matrix are linear independent because there is a large number of well distributed points over space and time, and only few key frames are selected. This allows to utilize a special case of the pseudo inverse. If the rows are linear independent a right inverse $A^{+} = A^{t} * (A * A^{t})^{-1}$ can be computed that is identical with the pseudo inverse. If the columns are linear independent a left inverse can be computed that is identical with the pseudo inverse. The linear independency cannot be guaranteed. If the rank of the matrix is lower than the row count, at least two rows are linear dependent. Computing the rank every time would also decrease the performance drastically. Therefore, measures to improve the chance that the rows are linear independent have been taken. If the rows are not linear independent *math.net* would throw an error during the computation of the right inverse. This error can be caught and then the Moore-Penrose pseudo inverse is computed. Tests with realistic data sets have shown that this case never occurred so far. The computed weight matrix then is multiplied with the key matrix. This equates a reconstruction of the non-key matrix. Afterwards, the matrices are element-wise subtracted and the Frobenius norm is calculated. The Frobenius norm computes the element-wise square and adds the squared elements together. The remainder of the algorithm is a search for the index with the least error. It simply updates the key index if the error was smaller.

4.3 Optimization

To overcome software and hardware limitations that prohibit the direct applicability on cell data, optimizations were needed in a first step. Another reason was, to enable an adequate computation time for better user experience. Optimization was inevitable to provide an adequate level of usability concerning the computation time.

Split Up: Addressing the already mentioned memory problem, the cell data is split up by the ingredient type. In the key frame selection step of the *Key Probe* technique each ingredient type is considered as a stand-alone animation. In the basic algorithm during the least squares optimization the key frame introducing the least error is selected. With the split-up approach, the errors for all groups are computed and added together. The key frame with the lowest error sum along all ingredient groups is then selected. In other words, the key frame that would reconstruct the entirety of all ingredient animations and therefore the whole animation is chosen. Without this split-up, an execution on the given test data set would have not been possible. This optimization could affect the results based on the group size of each ingredient type. Groups that have more protein

Algorithm 4.2: ComputeKeyIndex

Input: Matrix with non-key frames $\mathbf{A_{nkey}} = (a_{nm})$, matrix with already selected key frames $\mathbf{A_{key}} = (a_{km})$
Output: integer **keyIndex**

```
1  for i ← 0 to n-1 do
2      double minError = Integer.MaxValue();
3      int keyIndex = −1;
4      double[] keyFrame = A_nkey.getRow(i);
5      mat temp_nkey = A_nkey.removeRow(keyFrame);
6      mat temp_key = A_key.addRow(keyFrame);
7      double error = ||temp_nkey − temp_nkey ∗ temp_key^† ∗ temp_key||^2;
8      if error < minError then
9          error = minError;
10         keyIndex = i;
11     end
12 end
13 return keyIndex;
```

instances can introduce a higher error simply because more elements of the matrix can deviate. Therefore, the computed error of the groups is weighted by their number.

Projection: A projection in a 2D space would reduce the data to two third. This also reduces the computation time. The disadvantage is that the third dimension would not be recognized. This would lead to ignore major transformations in this dimension. The view point would have to be adjusted to the concrete data set and animation to avoid this case. This is not always possible as the animation resides in 3D space and transitions in all directions are possible. For this reason, this optimization was not applied in the final version.

Elimination of not Important Data: In cell data sets, often cytoplasm is contained as a surrounding for the structure of interest. This data is not needed in all animations and is often hidden by the user. As it is not rendered it should also not be present in the computation of the key frames. This helps to eliminate duplicate key frames due to transformation of the cytoplasm that is not rendered and therefore not visible. The split up of the data into ingredient types for computations enables easy selection of which ingredient types should be used for the computation. Only those who are rendered or of interest should be considered in the key frame selection. With this optimization, not only computation time is reduced, but key frames have a higher expressiveness.

Temporal Clustering: An animation could have many frames to be visually fluid. This leads to a lot of data that is almost the same except for minor changes. For this reason, a temporal clustering as a preprocessing step is introduced. The user can specify a maximum distance of atoms in consequent frames. During the construction of the

data structures this distance is evaluated. If any atom has a smaller distance than the specified one the consequent frame is discarded. This reduces the temporal resolution to frames, which hold valuable data for the user and reduces the computation time drastically. Another approach would be to let the animation only compute a certain number of frames and so reduces computation time of frames too. This would equate to a preceding uniform key frame selection and may tamper the produced results.

Spatial Clustering: A spatial clustering of the protein instances in each frame was considered too. It was not applicable because atoms of the same cluster would change over time and become different clusters. This hinders the tracking of clusters and *Key Probe* would not be applicable. The clustering algorithm would need to be changed over time too. Clusters at the beginning may be dense in space and then their atoms transit into curves. This would introduce special requirements for the clustering algorithm, which would need to be adaptive in time and space.

Spatial Grouping: Like a grouping by ingredient type a spatial grouping was investigated for optimization reasons. A spatial grouping would mean to divide the space into 3D sub-regions and compute the error for each sub-region. Then take the key frame with the least error for all sub-regions. This idea was not implemented due to the same problem as in spatial clustering. If atoms transit from one region to the next the number of elements in the frames of a group varies. *Key Probe* works on matrices and the row length of matrices cannot vary. If the moving elements would be replaced with zeros this would distort the selected key frames. The value from the last frame could be used to avoid this distortion. With this strategy, the performance would decrease instead of improving. The key frame would have to be computed for each sub-region and each sub-region would have the same size as the complete animation. If there are no transitions between the sub-regions this approach could be a valid optimization strategy. Each sub-region could be viewed as a separate animation.

Early Termination of Error Computation: An improvement implemented for the grouping was the early termination of error computation. During the key frame selection, the error of each ingredient type of a key frame is computed and summed up. If the current sum is greater than the lowest error already found, the computation of the error for the remaining groups could be omitted.

Visibility Culling: An approach to improve the visual results of the key frame extraction method, is to take the visibility of the protein instances into account. Protein instances that are culled would not be considered in the error computation. Therefore, only visible elements of the matrix would be considered as change and as a consequence only visible change would result in key frames. This optimization technique influences the computation time only slightly. The reason for this is that all computations and matrix sizes stay the same, besides an element-wise matrix multiplication with a matrix containing zeros and ones at the end of error computation. The calculation of the visibility of each element is considered a preprocessing step, same as the animation computation and therefore does not influence the computation time.

4.4 Implementation of the Hybrid Frame Visualization

The Hybrid Frame Visualization consists of an animated 3D view and a sequence of images. The animation was already implemented and is displayed in the 3D view of the *CellView* framework, developed by Le Muzic et al. [LMAPV15]. The sequence of frames is implemented with a *Unity* ListView. The elements are instantiated *GameObjects* of a *HybridFrame Prefab* holding a texture and a script. A *GameObject* in *Unity* is a plain object that can have attached logic with scripts and appearance like UI elements or 3D meshes. A *Prefab* is a *GameObject* that can be predefined and then for example be instantiated during runtime via code. The *HybridFrame Prefab* has attached a texture to show the frame and a controller for handling the clicks and starting the animation. After the computation of the key frames or when clicking on the "Update Layer" button, marked with 2 in Figure 3.2, a dynamic growing list is updated. If the list is too small, new instances of the *HybridFrame Prefab* are created and initialized with the start and stop key frame. Existing elements will be updated with the new start and stop key frames. If the list previously grew larger than the current selected number of key frames, elements will be disabled to be invisible in the UI. For rendering to the textures of the *HybridFrame Prefab*s a co-routine is used. A co-routine in *Unity* is initialized with a method returning an enumerable. After the start of the co-routine, the method is executed until the next occurrence of a "yield return" statement is found or until its end is reached, between each invocation of the rendering pipeline. This allows to implement logic without an update method, which is quite convenient for rendering different frames to textures. This is needed, because after each frame the animation needs to be updated. This update is done by the *CellView* framework inside an update method of a *GameObject*. Other solutions would be to adapt the update method of the *CellView* framework to render to the textures of *HybridFrame Prefabs*, or to create another *GameObject*, which saves the state and updates the *GameObject* holding the animation state. These solutions are not as elegant and maintainable as our solution with the co-routine and were therefore not implemented.

4.5 Implementation Environment

The hybrid visualization of a narrative sequence of animation states was implemented inside the *CellView* framework. *CellView* is a tool developed by Le Muzic et al. [LMAPV15]. It allows the real-time visualization of large biomolecular datasets at an atomic level. This is achieved with various acceleration techniques, like a dynamic level of detail for the atoms. The *CellView* framework was developed with the *Unity3D* game engine. *Unity* supports programming languages like JavaScript and C#. For *CellView* and this work C# was used with the support of the Visual Studio Development Environment. For matrix computations, the library *math.net* was used. This is an opensource library that allows advanced matrix computations. It is not limited to 4×4 matrices, like the *Unity3D* functions. *math.net* offers excellent performance for most matrix operations. Matrix factorization of very large matrices is expensive nevertheless, which is inevitable

as hardware limits are reached.

4.6 General Applicability

The *CellView* framework has many branches. These branches may also provide different types of animations. To allow other branches the use of our hybrid visualization approach, we have implemented it independent from the current animation implementation in *CellView*. To achieve this, we developed an easy to implement interface that provides the data for the key frame computation. This data consists of a list of frames. A frame holds a list of all points in the point cloud. Each point is described by a vector with four elements. The interface also returns the index of the last frame in the animation. For the temporal clustering a distance is needed. This distance is provided by the interface too. Besides this data, the interface also allows simple control over the rendered animation. The currently rendered interval of the animation can be set with a start and an end frame. The developers who want to use our hybrid visualization, just have to implement this interface. To make the hybrid visualization available in the 3D view, we created a *Prefab*, which can be dragged and dropped via the editor. A *Prefab* was needed due to the complex nesting of UI elements inside the *Unity* editor. After creating the hybrid visualization *GameObject* from the *Prefab*, the developers must add their implementation of the interface, providing the data, to the "HybridVis" script of the *GameObject*. With this approach, utilizing a *Prefab* the users can also decide where they want to place the hybrid visualization. They can place it at the top, at the bottom, at the center or any arbitrary position needed. A step by step manual is given in the Enumeration A in the appendix. The needed assets are listed in Listing A in the appendix.

CHAPTER 5

Evaluation

In order to support the users in perceiving molecular transitions, we implemented a Hybrid Visualization. This hybrid visualization consists of a 3D explorer and a sequence of frames. The 3D explorer is animated and allows an in-depth analysis of the animation. The sequence gives a fast overview of the animation and allows direct access of specific sub-sequences, which are shown in the 3D view after clicking on a key frame. While this form of interaction is a useful feature and the combination helps to convey the transitions in molecular data, the usefulness of the hybrid visualization is majorly dependent on the selected key frames. For this reason, the focus of the evaluation of the visualization lies on the selection of key frames.

In the first part of this chapter, the visualization will be evaluated by comparing the results of the key frame extraction method to uniform sampled key frames. The second part is focused on the optimization techniques and compares their performance. For the complete evaluation, a dataset created with the *Cellpack* tool, which was developed by Johnson et al. [JAAA+15], was used. This dataset contains a human blood serum surrounding a HIV virus and is the same dataset as used by other CellView developments for evaluation, like the work of Reisacher [Rei16] or Sorger et al. [SMK+16]. It contains 20630 individual protein instances that are set in motion in different animations.

5.1 Comparison of Key Frames

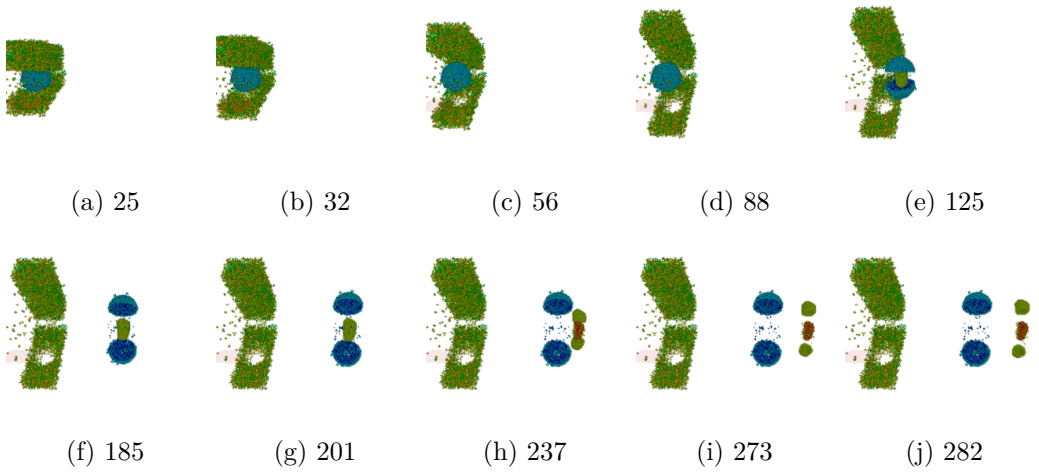
To evaluate the expressiveness of the key frames, we compare them with uniformly sampled frames and discuss the results supported by screenshots. The target of this evaluation is, to verify how well the key frames of our adapted *Key Probe* method describe the animated molecular data. An evaluation and discussion of *Key Probe* for animated meshes and motion capture datasets was done by its authors, Huang et al. [HCHY05]. We compare to uniform sampled key frames because there does not exist a database for key frame extraction methods on our data yet. Creating such a database would be out of

the scope of this work. The extracted key frames of three different animations will be discussed. Each animation will be computed at 30 FPS with a duration of ten seconds. The optimization techniques used are the Early Termination of Error Computation, Temporal Clustering and sometimes the Elimination of not Important Data.

The first animation shows an exploded view of the cytoplasm and the HIV. This transition allows a view in the inner structure of the HIV. To achieve this, each layer is split into two parts and the parts are then tilted outwards. In this animation, the cytoplasm was considered during the key frame computation. The key frames extracted with *Key Probe* are shown in Figure 5.1, the uniformly sampled key frames in Figure 5.2. As in all further examples, ten key frames were extracted. The first three key frames extracted with *Key Probe* show the parting of the cytoplasm. The fourth frame shows exactly the finishing of the parting of the cytoplasm. The next two frames show the parting of the membrane. The seventh frame again shows exactly the finishing of this parting. In the next two frames, the parting of the capsid is shown. This parting is finished with the last frame. The uniform sampled frames already start at the fourth frame with the parting of the membrane. Therefore, they give the impression that the parting of the membrane already starts while the cytoplasm is still parting, which is not the case. The remainder of the uniform sampled frames is very similar to the frames extracted with *Key Probe*.

The name of the second animation is "Spherechart". It allows a quantification of the protein instances too. Its focus lies on the inner protein instances of the HIV structure. It tries to better preserve the relation with the physical correct structure and trades for this benefit the efficiency of quantitative comparison. The extracted key frames that were computed by *Key Probe* are shown in Figure 5.3 and uniformly sampled key frames are shown in Figure 5.4. The first five key frames extracted with *Key Probe* focus on the large change of the cytoplasm around the HIV virus. The sixth frame depicts the end of this change. The next two frames are focused on the parting of the membrane. Here the end is omitted due to the selection of too less key frames. Then the transformation of the capsid is shown. In the uniform sampled key frames the large change of the cytoplasm is only depicted with three frames. Its end is at the fourth frame. The next frames show the parting of the membrane. Here the double inverting of the top half from upside-down to again upside-up is better depicted. The transformation of the capsid is not directly shown. Therefore, the movement of the single parts is shown. The end frame is almost identical.

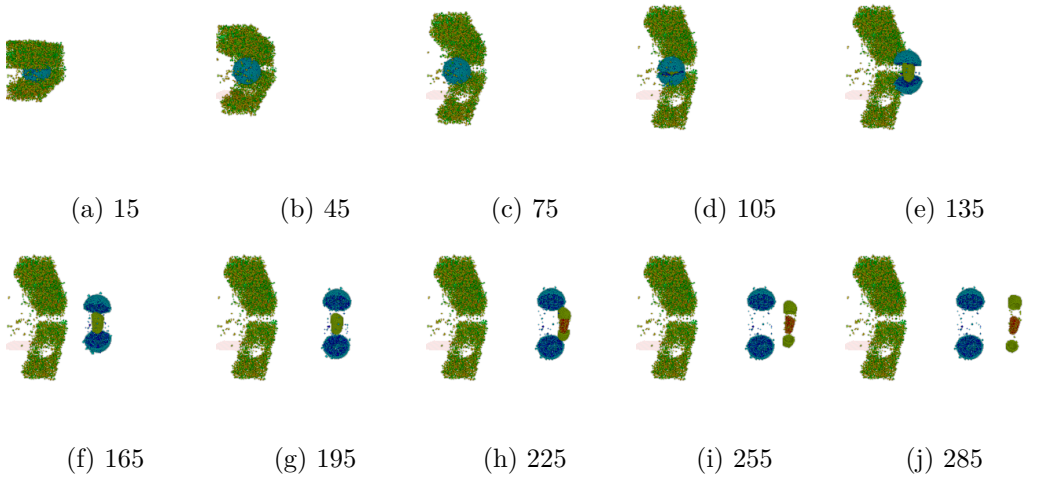
The third animation is called "Barchart" and depicts the transition of a physically correct state of protein instances to an abstract representation. It is used to quantify the contained protein instances and set them in a relation with their positions in the physically correct model. The key frames extracted with *Key Probe* are shown in Figure 5.5, the uniformly sampled key frames in Figure 5.6. In the "Barchart" animation *Key Probe* emphasizes the change of the membrane and depicts it in a step by step manner. For this reason, the movement of the capsid is almost omitted. Only the last key frame indicates that these structures move too. If more key frames would be sampled, these inner structures would be the next key frames selected. Compared to uniformly sampled



(a) 25 (b) 32 (c) 56 (d) 88 (e) 125

(f) 185 (g) 201 (h) 237 (i) 273 (j) 282

Figure 5.1: Extracted key frames for the "Explosion" animation with *Key Probe*.



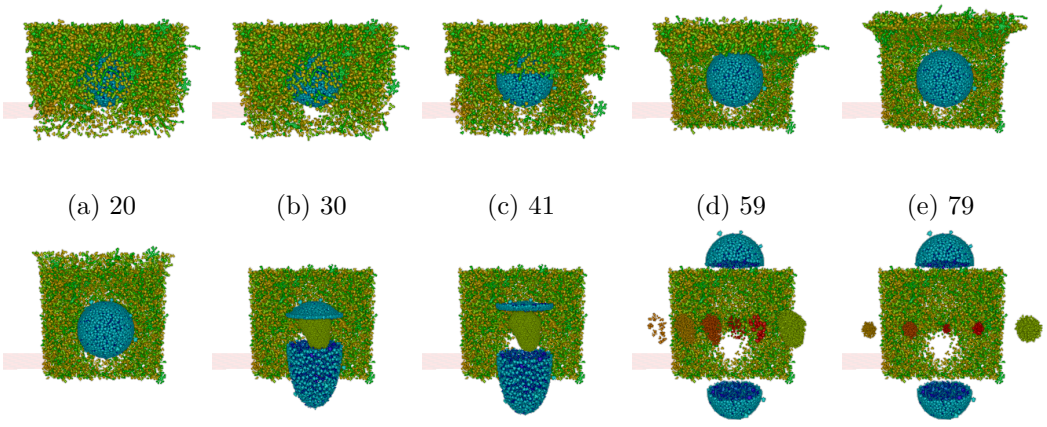
(a) 15 (b) 45 (c) 75 (d) 105 (e) 135

(f) 165 (g) 195 (h) 225 (i) 255 (j) 285

Figure 5.2: Extracted key frames for the "Explosion" animation uniformly sampled.

frames, the transformation of the capsid is better represented by *Key Probe*, because the step by step start of movement of the different protein instances could be better observed. In the uniformly sampled frames the transformation of the membrane is only depicted with five key frames and the step by step movement is omitted. Therefore, the transformation of the capsid is shown with three key frames and significantly better displayed than by *Key Probe*. The end shows two identical frames. Such redundancy is not possible with *Key Probe* and the space could be better used for depicting the transformation of the layers of the membrane.

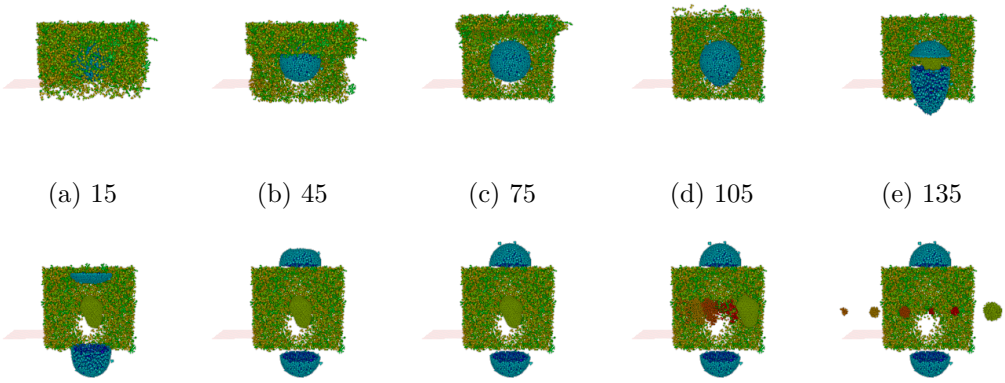
For the "Barchart" animation the Elimination of not Important Data was used and the cytoplasm around the HIV structure was ignored during the computation of *Key Probe*.



(a) 20 (b) 30 (c) 41 (d) 59 (e) 79

(f) 95 (g) 140 (h) 147 (i) 265 (j) 283

Figure 5.3: Extracted key frames for the "Spherechart" animation with *Key Probe*.

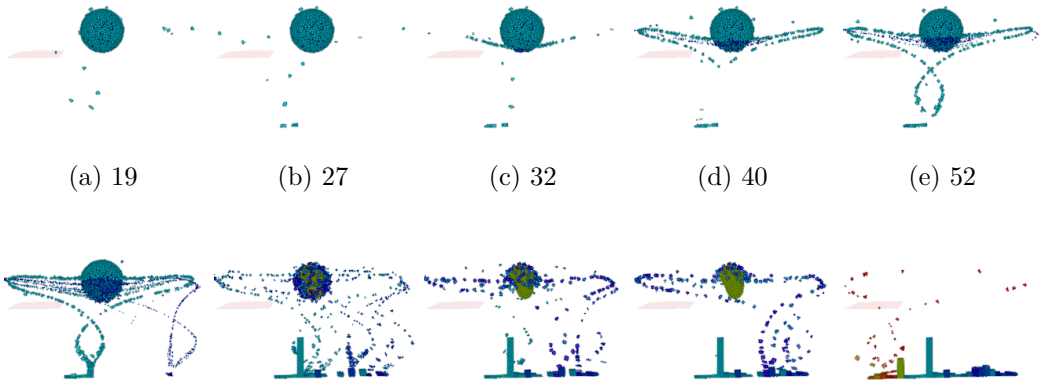


(a) 15 (b) 45 (c) 75 (d) 105 (e) 135

(f) 165 (g) 195 (h) 225 (i) 255 (j) 285

Figure 5.4: Extracted key frames for the "Spherechart" animation uniformly sampled.

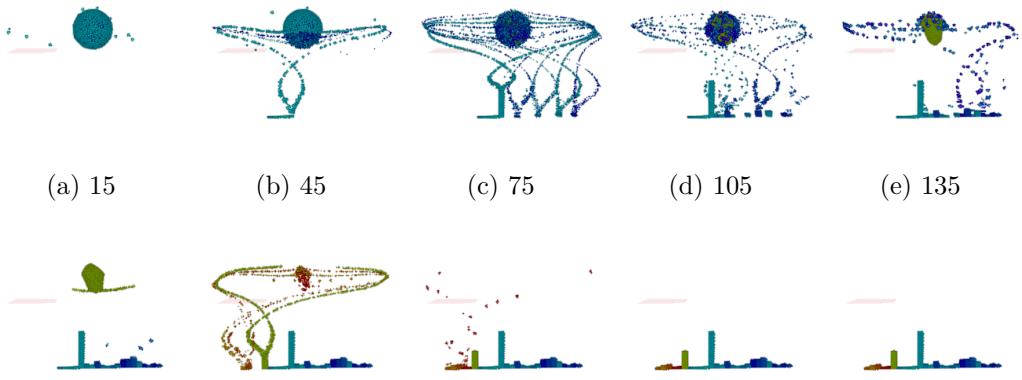
KeyProbe is good at detecting the finished movement of objects inside an animation. This ability could also be described as "cut detection". While there are no real cuts, because the animation is simulated, finishing and starting of sub-animations inside the animation can so be extracted. This ability is especially shown in the first example of the transition from the physically correct model to an exploded view. *KeyProbe* is susceptible for large groups of moving objects. These are stronger weighted compared to small groups till there are enough frames sampled from these large groups. Then the next smaller groups are considered. This is especially a problem if only view key frames are selected and there are many different groups with different sizes. If enough key frames are selected, this problem is neglectable. The same applies, if a large group would be eliminated. For example, in Figure 5.7 the "Spherechart" animation is almost perfectly described by the



(a) 19 (b) 27 (c) 32 (d) 40 (e) 52

(f) 109 (g) 124 (h) 133 (i) 142 (j) 222

Figure 5.5: Extracted key frames for the transition to a bar chart with *Key Probe*.



(a) 15 (b) 45 (c) 75 (d) 105 (e) 135

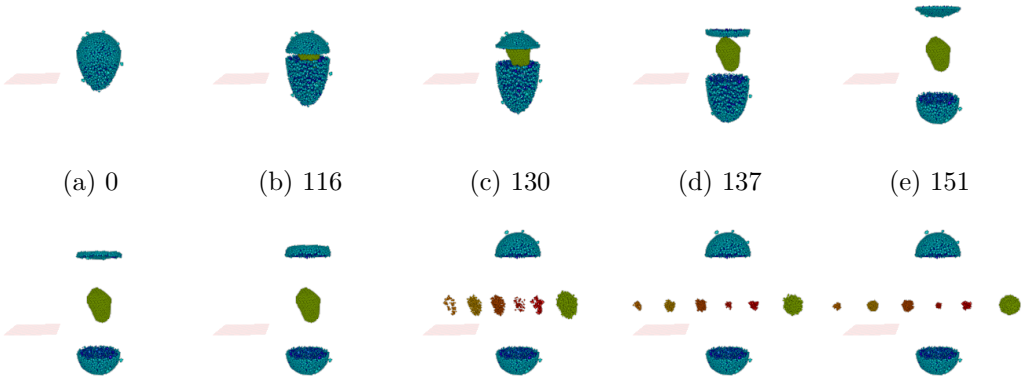
(f) 165 (g) 195 (h) 225 (i) 255 (j) 285

Figure 5.6: Extracted key frames for the transition to a bar chart uniformly sampled.

key frames because the cytoplasm was ignored. Therefore, enough key frames are selected to depict the animation.

5.2 Optimization

For the various optimization techniques presented in Chapter 4.3, we evaluated the timings and the results. In most cases the results are not heavily influenced, or not influenced at all, by an applied optimization technique. The following optimization techniques will be compared: Projection, Elimination of not Important Data (Elim), Temporal Clustering (TempClust), Early Termination of Error Computation (EarlyErrorTerm).



(a) 0 (b) 116 (c) 130 (d) 137 (e) 151

(f) 172 (g) 179 (h) 186 (i) 269 (j) 283

Figure 5.7: Extracted key frames for the transition to a sphere chart. The cytoplasm was eliminated this time.

The timings in this section were computed with an i5-2500k at a clock speed of 4.2 Ghz and 8 GiB of RAM at a clock speed of 1333 Mhz. The software limitations are introduced due to a 32-bit process of the *Unity3D* engine and its limitations for .net 3.5, which does not allow arrays larger than 2 GiB as documented by Microsoft [Mic].

Table 5.1 shows the time for extracting the key frames with a respective optimization technique. The last row shows the performance of the combination of Early Termination of Error Computation and Temporal Clustering. This specific case is the standard case used for the comparison, while Projection is deactivated due to its disadvantages and Elimination of not Important Data is not always possible. The animations used for the calculations of this table are "Spherechart" and "Barchart". They were computed at 30 FPS over a duration of ten seconds for the mentioned standard case and temporal clustering optimization. For the other optimization techniques, projection, elimination of not important data and early error termination, the FPS were reduced to 3. This is equal to a uniformly pre-sampling. As the computation without temporal clustering would take hours, this reduction was applied.

The results show, as already mentioned, that the optimization techniques have a low impact on the extracted key frames. However, projection and the elimination of not important data refocus the algorithm to different aspects of the transition and therefore the extracted key frames change. The timings show, that each optimization technique reduces the computation time efficiently. The best optimization technique is temporal clustering, as without it, computation would last several hours and was therefore not compared. The second-best technique is the elimination of not important data. It reduces the computation time about ~20-30%, depending on the animation. Projection reduces the computation instead of the expected one-third only by ~25%. This is still very good,

but as already mentioned influences the results in way, that they cannot be used. If the animation is project-able in 2D then this optimization technique could be added to the final stack. Early error termination can be used always, as it does not influence the results at all. It reduces the computation by ~10-20%, which is still pretty good. Best results are achieved with the combination of early error termination and temporal clustering. In comparison to only using temporal clustering, the combination reduces the calculation time about 10%.

Name	Animation(FPS)	Time(ms)	Frames
Elim	Barchart(3)	55371.1	1,2,3,4,5,11,12,13,14,23
Elim	Spherechart(3)	61051.2	10,13,14,16,18,20,21,25,26,28
Projection	Barchart(3)	63597.0	2,3,4,5,11,12,13,14,15,23
Projection	Spherechart(3)	68335.2	1,3,4,6,7,9,13,14,21,27
EarlyErrorTerm	Spherechart(3)	69089.0	2,3,4,6,7,9,13,14,25,27
EarlyErrorTerm	Barchart(3)	72964.3	1,2,3,4,5,11,12,13,14,23
Without	Barchart(3)	81024.9	1,2,3,4,5,11,12,13,14,23
Without	Spherechart(3)	89564.8	2,3,4,6,7,9,13,14,25,27
TempClust (max. distance 250) and EarlyErrorTerm	Spherechart(30)	332601.7	20,30,41,59,79,95,140,147,265,283
TempClust (max. distance 250)	Spherechart(30)	373378.3	20,30,41,47,68,79,93,147,154,261
TempClust (max. distance 3000) and EarlyErrorTerm	Barchart(30)	387671.9	46,52,55,61,67,70,73,79,82,222
TempClust (max. distance 3000)	Barchart(30)	424088.9	19,27,36,40,43,55,112,130,136,222

Table 5.1: Performance of different optimization techniques in milliseconds at 30 FPS and 3 FPS with a duration of 10 seconds.

CHAPTER 6

Conclusion

To convey the transitions between different states in spatio-temporal molecular data, an animation can be used. However, the problem with such a visualization technique is, that it is fleeting. In an animation, a frame is visualized and immediately replaced by the next frame. This produces a fluid transition but hinders comparison of transition states. To compensate this problem, we used a sequence of key frames that describe the state transitions best. This type of visualization is non-fleeting, but has the disadvantage that the relation between images and states might get lost or be unclear. We combined both visualization types in a way that the advantages of both can be exploited but the disadvantages are alleviated. The concrete implementation consists of an animated 3D view with a bottom panel for a sequence of images. These images are key frames of the animation that describe the transition best. To select expressive key frames, an object based key frame extraction method was utilized. The name of the extraction method is *Key Probe*, which was proposed by Huang et al. [HCHY05]. We adapted *Key Probe* to operate on point cloud molecular data. This step was needed, because *Key Probe* was designed for animated meshes and motion captures, which consist of few hundred 3D vertices, compared to molecular data, which consists of thousands of 3D vertices. The high number of vertices exceeds memory limitations during matrix computations of *Key Probe*. Therefore, the adaption includes optimization techniques to make *Key Probe* applicable on standard hardware with the software limitations of a 32-bit system. Further optimization techniques were introduced to reduce computation time. We have evaluated the expressiveness of the extracted key frames. The evaluation demonstrates the overall usefulness of our approach and the expressiveness of the key frames. The key frames work as a navigation tool inside the animation too. The user gets an overview and can explore details of a specific sub-sequence of the transition by clicking on a key frame. This interaction plays the animation in the 3D view from the selected key frame to the next key frame.

In future, there are still many challenges for the visual exploration of spatio-temporal

molecular datasets. For example, the optimization of the key frame extraction algorithm. When new datasets and animations become available, other optimization techniques, like spatial clustering, could be implemented with an adaptive clustering approach. Temporal clustering could be made adaptive too. This would make the clustering more independent from the type of animation. Another improvement would be to implement the extraction algorithm on the GPU. This is especially challenging, as compute shaders, which would be used for such an implementation, are limited in size. Matrices of our size would not fit directly in the data structures available on the GPU. Additionally, matrix operations are not completely parallelizable. While matrix multiplications or element wise operations like adding, subtracting or squaring would be parallelizable, other steps in the extraction method, like the computation of an inverse or the sum of all elements, are not easily parallelizable. The computation would also require many compute shader calls with synchronization between them. This would reduce the performance again. If a GPU implementation would be possible, this would probably enable real time computation of key frames. The user could interactively select the focus of the key frame computation. For example, this focus could be a specific subset of protein instances or a sub-sequence inside the transition that needs additional exploration. The last example would only be needed for more complex transitions over a longer duration. We have developed a Hybrid Frame Visualization for spatio-temporal molecular data and showed the applicability of *Key Probe*, an object-based key frame extraction method, on this kind of data. The yielded method could be used for other large-scale spatio-temporal point cloud data sets, which opens interesting opportunities for future applications.

APPENDIX A

Manual and Assets

1. Add an empty game object to the scene.
2. Add the script “HybridFrameVis” Script in Scripts/HybridFrameVis/HybridFrameVis.cs to the game object.
3. Add the HybridFrameVisPrefab to the “Canvas” in the scene.
4. Go down to HybridFrameVisPrefab -> ScrollView2 -> ScrollViewControlMask2 -> ScrollViewContent2
5. Add the “ScrollViewContent2” to the created game object in the script as the item “List Content Panel”.
6. Ensure that as “Hybrid Vis List Item” in the Script of the created game object the “HybridVisListItemPrefab” is selected. Otherwise select this.
7. Fill in the main camera for your scene under “Main Camera GO” and the Camera you want to use for rendering the Textures of the hybrid visualization in “Render Texture Camera GO”. Note: these cameras must not be the same.
8. Implement the abstract class “HybridFrameVisAnimationManager”.
9. Add this implementation to a game object and add this game object to the created game object in the “Hybrid Frame Vis” script under “Animation Manager GO”
10. Press Play, generate your frames and press “Compute Key Frames” in the HybridFrameVis game object. The computed key frames should be shown in the HybridFrameVisPrefab inside the Canvas.
11. Furthermore, you can compute the key frame layers with the specific button. In the field “Layer”, you can select the layer and with the button “Update Layer” this layer will be shown.

- Assets/Scripts/HybridFrameVis/HybridFrame.cs – holds the data for a hybrid frame (start frame, end frame, . . .)
- Assets/Scripts/HybridFrameVis/HybridFrameVis.cs – business logic of the hybrid frame visualization (compute key frames, render textures, ...)
- Assets/Scripts/HybridFrameVis/HybridFrameAnimationFrame.cs – holds the data for a frame of the animation (positions)
- Assets/Scripts/HybridFrameVis/HybridVisAnimationManager.cs – abstract class which HybridFrameVis.cs utilizes to get the frames of the animation and set current frames.
- Assets/Scripts/HybridFrameVis/HybridVisItemController.cs – Controller for the Prefab "HybridVisItemPrefab", controls animation via HybridVisAnimationManager
- Assets/Editor/HybridFrameVisEditor.cs – buttons to call business logic from HybridFrameVis.cs
- Assets/Resources/Prefabs/HybridFrameVisPrefab – The GUI of the hybrid frame vis (showing key frames, which are clickable)
- Assets/Resources/Prefabs/HybridVisListItemPrefab – One item of the GUI of the hybrid frame vis (showing one key frames, which is clickable)

Bibliography

[AAG03] Natalia Andrienko, Gennady Andrienko, and Peter Gatalsky. Exploratory spatio-temporal visualization: an analytical review. *Journal of Visual Languages & Computing*, 14(6):503 – 541, 2003.

[AMM+07] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. Visualizing time-oriented data—a systematic view. *Computers & Graphics*, 31(3):401 – 409, 2007.

[BM10] Stefan Bruckner and Torsten Moller. Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1468–1476, 2010.

[BMS08] Andreas Baak, Meinard Müeller, and Hans-Peter Seidel. An efficient algorithm for keyframe-based motion retrieval in the presence of temporal deformations. In *Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval*, pages 451–458, 2008.

[BWK+13] Jürgen Bernard, Nils Wilhelm, Björn Krüger, Thorsten May, Tobias Schreck, and Jörn Kohlhammer. Motionexplorer: Exploratory search in human motion capture data based on hierarchical aggregation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2257–2266, 2013.

[CDMB+07] Paolo Compieta, Sergio Di Martino, Michela Bertolotto, Filomena Ferrucci, and Tahar Kechadi. Exploratory spatio-temporal data mining and visualization. *Journal of Visual Languages & Computing*, 18(3):255 – 279, 2007.

[Dol07] Helmut Doleisch. Simvis: Interactive visual analysis of large and time-dependent 3d simulation data. In *Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is Yet to Come*, pages 712–720, 2007.

[GVL96] Gene H Golub and Charles F Van Loan. Matrix computations. johns hopkins studies in the mathematical sciences, 1996.

[GWY+11] Hanqi Guo, Zuchao Wang, Bowen Yu, Huijing Zhao, and Xiaoru Yuan. Tripvista: Triple perspective visual trajectory analytics and its application on microscopic traffic data at a road intersection. In *2011 IEEE Pacific Visualization Symposium*, pages 163–170, 2011.

[HCHY05] Ke-Sen Huang, Chun-Fa Chang, Yu-Yao Hsu, and Shi-Nine Yang. Key probe: a technique for animation keyframe extraction. *The Visual Computer*, 21(8):532–541, 2005.

[HHS08] Peng Huang, Adrian Hilton, and Jonathan Starck. Automatic 3d video summarization: Key frame extraction from self-similarity. In *International Symposium on 3D Data Processing, Visualization and Transmission*, 2008.

[JAAA+15] Graham T Johnson, Ludovic Autin, Mostafa Al-Alusi, David S Goodsell, Michel F Sanner, and Arthur J Olson. cellpack: a virtual mesoscope to model and visualize structural systems biology. *Nature methods*, 12(1):85–91, 2015.

[KWS+14] Artem Konev, Jürgen Waser, Bernhard Sadransky, Daniel Cornel, Rui AP Perdigao, Zsolt Horváth, and M Eduard Gröller. Run watchers: Automatic simulation-based decision support in flood management. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1873–1882, 2014.

[LHM14] Isaac Liao, Wei-Hsien Hsu, and Kwan-Liu Ma. *Storytelling via Navigation: A Novel Approach to Animation for Scientific Visualization*, chapter 1, pages 1–14. Springer International Publishing, 2014.

[LLWC08] Tong-Yee Lee, Chao-Hung Lin, Yu-Shuen Wang, and Tai-Guang Chen. Animation key-frame extraction and simplification using deformation analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(4):478–486, April 2008.

[LMAPV15] Mathieu Le Muzic, Ludovic Autin, Julius Parulek, and Ivan Viola. cellview: a tool for illustrative and multi-scale rendering of large biomolecular datasets. In *VCBM*, pages 61–70, 2015.

[LOT05] Shiyu Li, Masahiro Okuda, and Shinichi Takahashi. Embedded key-frame extraction for cg animation by frame decimation. In *2005 IEEE International Conference on Multimedia and Expo*, pages 1404–1407, 2005.

[LS08] Aidong Lu and Han-Wei Shen. Interactive storyboard for overall time-varying data visualization. In *2008 IEEE Pacific Visualization Symposium*, pages 143–150, 2008.

[Ma03] Kwan-Liu Ma. Visualizing time-varying volume data. *Computing in Science & Engineering*, 5(2):34–42, 2003.

[MGA+08] Miriah Meyer, Luke J Gosink, John C Anderson, E Wes Bethel, and Kenneth I Joy. Query-driven visualization of time-varying adaptive mesh refinement data. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1715–1722, 2008.

[Mic] Microsoft. 64-bit applications .net framwork 3.5. https://msdn.microsoft.com/en-us/library/ms241064(v=vs.90).aspx. [Online, accessed=10.02.2017].

[Rü] Christoph Rüegg. Maximum matrix size. https://discuss.mathdotnet.com/t/maximum-matrix-size/64/2. [Online, accessed=05.02.2017].

[RD14] Rico Richter and Jürgen Döllner. Concepts and techniques for integration, analysis and visualization of massive 3d point clouds. *Computers, Environment and Urban Systems*, 45:114 – 124, 2014.

[Rei16] Matthias Reisacher. Cellpathway: a simulation tool for illustrative visualization of biochemical networks. Master's thesis, Technical University of Vienna, Faculty of Informatics, 2016.

[RM13] Sébastien Rufiange and Michael J McGuffin. Diffani: Visualizing dynamic graphs with a hybrid of difference maps and animation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2556–2565, 2013.

[RPS01] Freek Reinders, Frits H. Post, and Hans J.W. Spoelder. Visualization of time-dependent data with feature tracking and event detection. *The Visual Computer*, 17(1):55–71, 2001.

[SMK+16] Johannes Sorger, Peter Mindek, Tobias Klein, Graham Johnson, and Ivan Viola. Illustrative transitions in molecular visualization via forward and inverse abstraction transform. In *Eurographics Workshop on Visual Computing for Biology and Medicine (VCBM)*, pages 21–30, 2016.

[TMB02] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: can it facilitate? *International Journal of Human-Computer Studies*, 57(4):247 – 262, 2002.

[WCPB12] Wathsala Widanagamaachchi, Cameron Christensen, Valerio Pascucci, and Peer-Timo Bremer. Interactive exploration of large-scale time-varying data using dynamic tracking graphs. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 9–17, 2012.

[WKS+14] Jürgen Waser, Artem Konev, Bernhard Sadransky, Zsolt Horváth, H Ribičić, Robert Carnecky, P Kluding, and Benjamin Schindler. Many plans: Multidimensional ensembles for visual decision support in flood management. *Computer Graphics Forum / Proceedings of Eurovis 2014*, 33(3):281–290, 2014.

[YA06] Toshihiko Yamasaki and Kiyoharu Aizawa. Motion segmentation of 3d video using modified shape distribution. In *2006 IEEE International Conference on Multimedia and Expo*, pages 1909–1912, 2006.

[ZRHM98] Yueting Zhuang, Yong Rui, Thomas S Huang, and Sharad Mehrotra. Adaptive key frame extraction using unsupervised clustering. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pages 866–870 vol.1, 1998.