Dynamic Comics for Hierarchical Abstraction of 3D Animation Data

Myung Geol Choi[1]   Seung-Tak Noh[2]   Taku Komura[1]   Takeo Igarashi[3]

[1]The University of Edinburgh   [2]KAIST   [1]The University of Tokyo
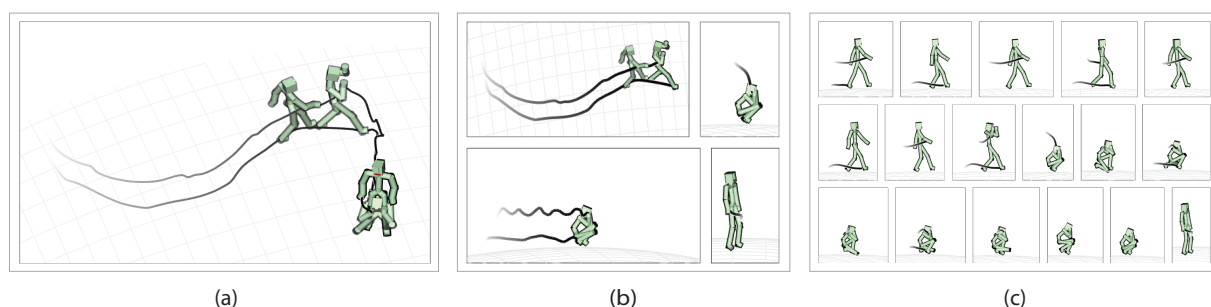

(a)   (b)   (c)

Figure 1: *Three comic sequences which are automatically generated based on the snapshot tree of Figure 2. They describe the same locomotion animation in different perspectives. The left image describes the global traveling path of the subject. The middle comic sequence shows the transition of the locomotion style. The right comic sequence presents the detailed body poses and the number steps in both styles.*

Abstract

Image storyboards of films and videos are useful for quick browsing and automatic video processing. A common approach for producing image storyboards is to display a set of selected key-frames in temporal order, which has been widely used for 2D video data. However, such an approach cannot be applied for 3D animation data because different information is revealed by changing parameters such as the viewing angle and the duration of the animation. Also, the interests of the viewer may be different from person to person. As a result, it is difficult to draw a single image that perfectly abstracts the entire 3D animation data. In this paper, we propose a system that allows users to interactively browse an animation and produce a comic sequence out of it. Each snapshot in the comic optimally visualizes a duration of the original animation, taking into account the geometry and motion of the characters and objects in the scene. This is achieved by a novel algorithm that automatically produces a hierarchy of snapshots from the input animation. Our user interface allows users to arrange the snapshots according to the complexity of the movements by the characters and objects, the duration of the animation and the page area to visualize the comic sequence. Our system is useful for quickly browsing through a large amount of animation data and semi-automatically synthesizing a storyboard from a long sequence of animation.

Categories and Subject Descriptors (according to ACM CCS):   I.3.3 [Computer Graphics]: Picture/Image Generation—Displaying and viewing algorithms I.3.7 [Computer Graphics]: Three-dimensional graphics and realism—Animation

1. Introduction

To comprehend the contents of time-based media data such as 2D video or 3D animation data, the viewers need to watch it as long as the data was recorded. This makes data processing procedures, such as searching and editing, inefficient. For this reason, generating descriptive snapshot im-

ages for a media data has been an interesting research topic in the data visualization field. Considerable literature has been published regarding 2D video data. The primary idea is to select a set of key-frames and display them in time order. This approach has been widely used in many video editor and player programs to aid the users in their data processing.

However, such an approach cannot be applied for 3D animation data because different information is revealed by changing parameters such as the viewing angle and the duration of the animation. Also, the interests of the viewer may be different from person to person. For example, if a user want to know the global traveling path of the animating subject, the abstraction picture should show the subject's trajectory line for the entire animation time with a bird-eye view (see Figure 1 (a)). On the other hand, a frame-by-frame image sequence should be suitable for the user who want to see the changes of body poses during the locomotion (see Figure 1 (c)). Likewise, a single key-frame sequence cannot be enough to provide the user with a comprehensive understanding of the 3D animation.

In this paper, we propose a comic-based abstraction system for 3D animation data. Given a 3D animation data, our system generates a set of different comic sequences in which the shorter sequences tends to present the higher resolution motion, and the longer sequences tends to present the lower resolution motion of the subject. We first capture various 2D snapshots from the 3D animation to prepare the element pictures of the comics. Each snapshot picture describes a different time duration of the animation. Next, by selectively assembling these snapshots, we generate diverse comic sequences that describe animation data in different perspectives. To collect an enough but compact set of element snapshots, we propose a novel segmentation algorithm that automatically builds a tree structure of snapshots. In this tree structure, the leaves of any pruned tree form a set of snapshots that exclusively cover the animation frames without any missing part. This property of the snapshot tree simplifies the problem of assembling a full-frame comic sequences that describe the all frames of the animation. Based on the snapshot tree, we design a user interfaces that enables the user to composite different comic sequence by controlling a simple parameter including the length of comic sequence, the scale of combined time bar and the page area to visualize the comic sequence.

We demonstrate the usability of our method by applying it for different types of motion capture data. And we introduce a comic-guided time bar interface and a browser interface for a motion capture database.

2. Related Works

Since 2D video constitutes the most common time-based media data, their pictorial abstraction has been actively studied. Video data can be considered as a series of 2D im-

ages. By selecting and displaying relatively unique images among them, we can composite a relevant abstraction pictures [BGGU00, Gir03, CGC07]. Goldman et al. [BGSF10] smoothly blended the boundaries of every key-frame image to build a long connected tapestries. These key-frame based approaches have been widely adopted in video editing and playing programs because of their simplicity and suitability for convergence with the time bar interface. However, the user can only guess regarding the missing contents between key-frames based on the surrounding key-frames. To improve this drawback, a moving path of the subject between key-frames can be detected by vision tracking techniques and specified on the picture [BM07, PRAP08, CGRU09]. On the other hand, our approach takes advantages of both continuous and discontinuous descriptions. We generate multiple snapshots for separate parts of the animation, but one snapshot picture depicts the continuous animation within the time duration.

Studies have also investigated the abstraction of 3D animation data. Assa et al. [ACCO05] introduced a systematic procedure to choose a set of key-poses from a motion capture data. These 3D key-poses should be arranged well to be relevant to create a 2D abstraction picture. They introduced several examples including displaying key-poses in 3D space with trajectory lines and compositing a comic style sequence. These final results require manual processing by users. In contrary, our system composites diverse scenes of the animation automatically, and user interaction is required only for exploring the generated scenes. Bouvier-Zappa et al. [BZOP07] augmented local movement of the motion data by attaching cartoon-style signs, including the speed lines and noise wave around the 3D character model. In our case, we display trajectory lines, which are similar to their speed lines. However, our trajectory line can be longer to describe the entire range of the original animation.

For good abstraction of a 3D animation, how information is shown is as important as what is shown. This primarily depends on the properties of the projection camera. If the animation is playing while the user is observing it, the camera may need to follow the moving objects, retaining a good viewing angles. Kown et al. [KL08] and Assa et al. [ACOYL08] optimized the camera movement for single character animation, while Yeh et al. [YLL*12] did so for multi-character animation. Although we also want to describe dynamic motion, we aim to produce a static 2D pictures and the fixed camera property that should be used for each picture. Thus, our problem is more related to studies concerning optimization of the camera angle for a static 3D object. Several feature values have been tested in previous research including the rendered area [VFSH01], silhouette [VBP*09], and surface curvature [LVJ05] of the subject. We are interested in the features that would emphasize the subject's movements rather than the subject's shape. Thus, the areas of key-poses and the length of the trajectory lines could be suitable.

Our comic-base interface is inspired by the McCloud's insight that the comics can abstract a same story in different length of sequences. This property of the comics is well explained in his book, *"Understanding Comics"* [McC94] which gives comprehensive understanding of the comics as a media.

3. Overview

We are interested in abstraction of the moving objects in an animation data. A moving object can be a rigid body, articulated body or free-deformable object. Through the paper, our algorithm will be explained with respect to the articulated body. A rigid body can be considered as an articulated body with a single link, and many free-deformable objects are actually controlled by internal articulated structures. Alternatively, we can build a skeletal structure in a systematic way [TZCO09, BP07, ATC*08]. As for the motion that can not be represented as an articulated structure such as the flow of fluid, we leave them for future work.

Given a 3D animation data, we first collect diverse time durations of the animation and draw snapshot pictures for each of them. This process is done by constructing the snapshot tree in bottom-up way. In Section 4, we first explain the procedure to draw a snapshot picture from a given time duration r of the animation data. Section 5 presents the algorithm for collecting diverse time durations while building the hierarchy. In Section 6, we explain how to assemble a full-frame comics sequences from the snapshot tree and introduce three strategies to explorer between different comic sequences by controlling a simple parameter.

4. Drawing a Snapshot

Given a certain time duration r in the animation data, we first select a set of important components within the duration such as trajectory lines and key-poses of the object. Then we project them on a 2D plane with the viewing angle that best describes the components.

4.1. Trajectory Lines

For an articulated body, we draw trajectory lines for each joint points. To emphasize a twist-rotation of the head and both hands, we add virtual joints at the end points of the nose and both thumbs prior to drawing trajectory lines.

In general, a human body model has more than 25 joints, and it could be too messy if their trajectories are all drawn together in one picture. To make our snapshot picture clearer, we display two representative trajectories. The hypothesis is that human motion data exhibits considerable spatial coherence in many cases [SHP04]. Therefore, joint trajectories could be similar to each other. In addition, because the users are able to view the finer details of the motion through the descendants' snapshots in the snapshot tree, it would not be necessary to put everything in a snapshot and make it too

complex to see. We apply a k-means clustering [HW79] for trajectory lines of every joint with $k = 2$, and we take one joint trajectory line from each group, which is closest to the mean value. The distance between two trajectories are calculated by summing the Euclidean distances between two corresponding 3D points on the trajectory at every frame. The function is defined by

$$tdist(\mathbf{t}_i, \mathbf{t}_j) = \sum_t |(p_t^i - \bar{p}^i) - (p_t^j - \bar{p}^j)| \qquad (1)$$

where p_t^i and p_t^j are the 3D points on ith and jth trajectory, respectively, at the frame time, t. The trajectories are aligned to the origin by subtracting the mean point values $\bar{p}^i$ and $\bar{p}^j$.

4.2. Key-Frame Poses

Previous studies have investigated the selection of a relevant set of key-frames for animation data, especially Assa et al. [ACCO05]. This study introduced a systematic procedure to select a set of key-poses from an articulated body animation. The basic idea of the method is to select the frames that have relatively unique poses compared to their neighbors. We adopt this method and slightly modified it for our purpose. One difference for our snapshot picture is that it does not require densely sampled key-poses because the trajectory lines already present the continuous movement of the object, and the user will be able to view the finer key-frames via the snapshots in lower levels of the snapshot tree. Thus we turned the parameters in Assa's method, so that relatively small number of key-frames were selected.

We also wish to consider the consistency of the key-frames between snapshots that share the same duration of the animation. Suppose that two snapshots are describing the same time duration but show different frames as key-frames. This could be misunderstood by the viewer as a different time duration because we did not specify the exact frame numbers of the key-frames on the image. We first form a key-frame candidate pool by gathering the last frames of all the collected time durations of the animation (The procedure for collecting these durations will be explained in Section 5). For time duration r, we select at least one key-frame from the key-frame pool by using Assa's method. Using the key-frame pool significantly reduces the computational load as well as preserves the consistency of key-frames between snapshots. Therefore we could skip the dimension reduction phase of Assa's method, which was applied for a computational efficiency.

4.3. Viewing Angle

For a given duration r, the viewing angle for its snapshot will be optimized with consideration of all the selected trajectories and key-frame poses. Since the optimization is conducted independently for each duration r_i, the same animation frame, which is in multiple r_i, could be focused with different viewpoints depending on the total range of each snapshot.

To simplify the optimization process, we present the camera angle with the three vectors of eye-point, center-point and up-vector [DS97]. The *eye-point* is the 3D position where the camera is, the *center-point* is the 3D position that the camera looks at, and the *up-vector* is a 3D unit vector that presents the vertical orientation of the camera body. We fix the up-vector to the world vertical axis (going up) to maintain the ground as horizontal in the projected picture. The center-point is set to the geometric center of all the selected components. The eye-point can be represented as a relative vector from the center-point $\mathbf{d} \times \vec{n}$ where $\mathbf{d}$ is the displacement between the eye-point and the center-point, and $\vec{n}$ is the normal vector towards to the eye-point from the center-point. The displacement $\mathbf{d}$ does not affect to the projected image if the camera use the orthographic projection. Otherwise, if we assume that the camera uses perspective projection with a fixed field-of-view value, then we can set $\mathbf{d}$ to the minimum displacement by which all components enter inside the camera frustum. Finally the only variable is the directional unit vector $\vec{n}$.

To find good viewing angles for a 3D object, we can consider any of the various feature values provided by previous work [SLF*11] included the rendered area, surface curvature, silhouette and depth attributes. Because we are more interested in the features that emphasize the movement of the object, we take two features values, the areas of key-poses and the lengths of trajectory lines, and find the $\vec{n}^*$ that maximized these feature values on the projection plane.

$$\vec{n}^* = \arg\max_{\vec{n}} len(\vec{n}) + area(\vec{n}), \quad \vec{n} \in \mathcal{N} \qquad (2)$$

where two functions $len(\vec{n})$ and $area(\vec{n})$ calculate the lengths of all trajectory lines and occupied areas of all objects respectively when they are projected on the plane of normal $\vec{n}$.

The viewing directions calculated independently for each snapshot can flip the global moving direction between snapshots. This can be seen by the user as the object changes its moving direction. To avoid this side effect, we limit the range for viewing normal vectors of all snapshots within a 3D unit vector space $\mathcal{N}$.

To calculate the unit vector space $\mathcal{N}$, we first produce a snapshot covering the entire time duration of the animation r_{entire} and find the optimal viewing direction $\vec{n}^*_{entire}$ without the range limitation $\mathcal{N}$. Then, the range $\mathcal{N}$ is defined to all unit vectors within a 90 degree angular distance from the unit vector $\vec{n}^*_{entire}$. The same range $\mathcal{N}$ is used for every snapshot of the animation data.

Since the optimization can be done in the pre-computation phase, a faster approach would not be necessary. We uniformly sample unit vectors within the range $\mathcal{N}$ with a 10 degree angle step along the latitude and the longitude, and we test all the cases to choose the optimal $\vec{n}^*$ maximizing Eq 11.

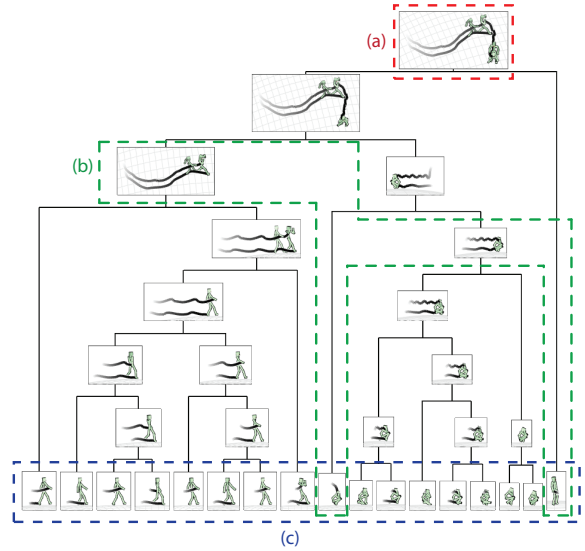Each snapshot picture has different size and different

Figure 2: *Snapshot tree of a locomotion animation. Leaves of any pruned tree form a full-frame comic sequence. (a), (b), and (c) are the node sequences corresponding to the comics (a), (b), and (c) in Figure 1.*

aspect-ratio. However, when they are displayed in a sequence, fitting their heights to the same would be aesthetically beneficial. We can scale each snapshot picture up or down to make them all the same height. From now on, we assume that all snapshot picture have the same height, h.

5. Snapshot Tree

We wish to take the durations that produce simpler (or more readable) snapshots for the viewer. For example, we do not want poses that are too diverse in duration because too many things in a picture would be confusing for the viewers. At the same time, we also need to retain the diversity of the time durations in terms of the length and location of time. To achieve these goals efficiently, we collect the time durations while building a hierarchy of them from the shortest sections to the longer sections in a bottom-up way. Given animation data, we first segment it into short segments that each corresponds to the leaf nodes in the tree structure. Then, we repeatedly merge a pair of consecutive durations to create a longer duration until they all connected as one binary tree.

5.1. Leaf Snapshots

The simplest way to define the shortest durations is to set a single frame as a duration for a snapshot. However, in this case, the number of leaf nodes becomes the same as the total number of frames of the animation. The total tree size could be unnecessarily large for the system. Instead of that, we segment the animation data into short monotonic motions and

set each segment as a duration for a leaf node. A monotonic motion means that the object has beens moved in one way without changing the direction. This kind of movement can be presented in a 2D space without (or with minimal) loss of the original 3D movement, and it will be a more readable snapshot for the viewer.

Since the segmentation of motion data is required in almost all data-driven character animation techniques, there have been various attempts aimed to segment them into monotonic motions [FMJ02, CYI*12]. We employ Choi's method [CYI*12]. It observes the total speed of all joint points and segments the animation data at every point when the speed value falls in a local minima. This approach results in a reasonable approximation, is relatively fast, and simple. If there are multiple subjects in the animation, we also segment at the moments when they contact to each other (see Figure 7). For each segmented duration r_i, one snapshot picture s_i is generated in the same way that was explained in Section 4. These snapshot pictures are set as leaf nodes of the snapshot tree.

5.2. Tree Construction

Now, we have only leaf node snapshots. A longer section can be defined by merging any number of continuous sections corresponding to the leaf nodes. However, we limit the number of nodes to be merged to two because this makes the construction procedure simpler and enables the tree to maintain diversity of the length of the collected durations. Our strategy is to select the best pair of consecutive durations r_i and r_{i+1} among them, and create a longer section $r_{i \oplus i+1}$ by merging them. Then we build a subtree, in which the root node is the new snapshot picture $s_{i \oplus i+1}$ drawn from the time duration $r_{i \oplus i+1}$. The left and right children nodes are the snapshots of s_i and s_{i+1}, respectively. Next, the parent node $s_{i \oplus i+1}$ becomes a candidate to be merged with other nodes instead of both children s_i and s_{i+1}, and we repeatedly merge the two pairs of node until only a single binary tree remains. Finally the root node becomes the snapshot covering the entire range of the animation (see Figure 2).

The criteria for deciding the best node pair to be merged is the complexity of the snapshot picture that will be generated from the merged duration. Higher complexity means less readability, and we want to choose the pair of the least complexity. The complexity function of two durations r_i and r_{i+1} is defined by a weighted sum of three sub-functions, C_{plane}, C_{pose}, and C_{length}.

Plane Complexity The first sub-function in Eq 9, C_{plane} calculates the difference between the two viewing normal vectors n_i and n_{i+1} which is calculated in Section 4.3.

$$C_{plane}(r_i, r_{i+1}) = |acos(n_i \cdot n_{i+1}) - 0.5\pi| \qquad (3)$$

If both viewing normal vectors are significantly different, it means that the components in both sections would not be suitable to be drawn on a single 2D plane. Even if an op-

timal viewing normal vector is recalculated for the merged duration, the resulting snapshot may have some ambiguity. For example, two objects could overlap.

Pose Complexity Another possible reason why a snapshot becomes more complex and less readable is that it contains poses that are too diverse in one picture. To present diverse poses well, the more key-poses will be displayed in the snapshot. However, too many key-poses could make the picture too complex and confusing. We define the pose complexity function by

$$C_{pose}(r_i, r_{i+1}) = \frac{\sum_j \min_k d(f_j, f_k)}{\|r_i\|} + \frac{\sum_k \min_j d(f_j, f_k)}{\|r_{i+1}\|}$$
$$, f_j \in r_i, f_k \in r_{i+1} \qquad (4)$$

where $\|r_i\|$ and $\|r_{i+1}\|$ are the number of frames in both durations r_i and r_{i+1}, respectively. The function $d(f_j, f_k)$ calculates the difference between the jth frame in the section r_i and the kth frame in the section r_{i+1}. To compare two articulate poses, we use the same distance function introduced in [KGP02].

Since r_i and r_{i+1} might have different frame lengths and quite different contents, it is difficult to apply dynamic time warping to find the frame correspondence between them. Instead, we apply a similar method introduced by Wampler et al. [WPP13]. They evaluates the closeness between a motion graph and a short motion segment. Every frame in the duration r_i is compared with the most similar frame in the duration r_{i+1}, and their distance values are summed.

Length Complexity Longer duration could produce more complex snapshot pictures. If the plane complexity and pose complexity are not very distinctive, we choose the pair with the shorter durations. This term also helps to maintain the balance of the binary tree. The function is defined by the difference of the frame numbers of both ranges.

$$C_{length}(r_i, r_{i+1}) = |\|r_i\| - \|r_{i+1}\|| \qquad (5)$$

Before three functions are summed together, their ranges are normalized into $[0, 1]$ by the below equations.

$$C'_{plane}(r_i, r_{i+1}) = C_{plane}(r_i, r_{i+1})/0.5\pi \qquad (6)$$

$$C'_{pose}(r_i, r_{i+1}) = 1 - \exp\left(-\sigma_1 C_{pose}(r_i, r_{i+1})\right) \qquad (7)$$

$$C'_{length}(r_i, r_{i+1}) = 1 - \exp\left(-\sigma_2 C_{length}(r_i, r_{i+1})\right) \qquad (8)$$

The variances σ_1 and σ_2 controls the distributions of the complexity values in the range $[0, 1]$. The values that we decided empirically for our experiments are 0.03 for σ_1 and 1 for σ_2. Finally the complexity function C is defined by

$$C(r_i, r_{i+1}) = w_1 C'_{plane}(r_i, r_{i+1}) + w_2 C'_{pose}(r_i, r_{i+1})$$
$$+ w_3 C'_{length}(r_i, r_{i+1})) \qquad (9)$$

Since we want C'_{length} function affects only when the both animations in r_i and r_{i+1} are very similar, $w_3(= 0.01)$ is set to a relatively small value in our experiments.

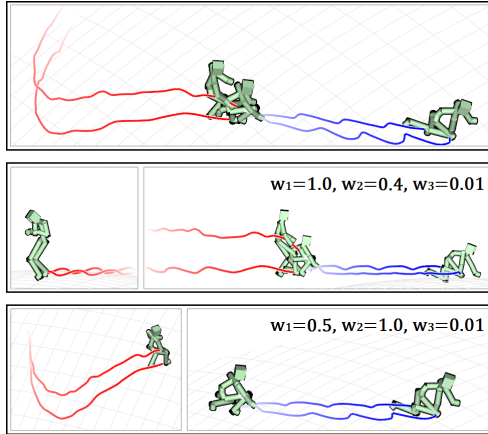When w_1 is greater than w_2, the decision for combining

Figure 3: *Top is the root snapshot for the motion data in which the subject sneaks with turning left (red trajectory), then crawls (blue trajectory). Middle and bottom are full-frame comics of two snapshots generated with different weight values for the combining cost function (Eq 9).*

becomes more sensitive to the view plane complexity, and vise versa. Figure 3 shows the changes of the comics with respect to the change of the w_1 and w_2 via a short motion data example. In the motion data, the subject first turns left while sneaking then crawls forward for a while. The top image in Figure 3 presents the root node of the snapshot tree, in which trajectories are colored red and blue while sneaking and crawling, respectively for the viewers' convenience. The middle image in Figure 3 shows the full-frame comics consisted of two snapshots when the $w_1 (= 1.0)$ is greater than $w_2 (= 0.4)$. Here, the sneaking steps after the left-turn are combined with the crawling steps in the right snapshot because their optimized viewing angles are similar. On the other hand, when the $w_1 (= 0.5)$ is less then $w_2 (= 1.0)$ in the bottom image, all sneaking steps are combined together in the left snapshot while the crawling steps exhibits in the right snapshot.

During our tests with different weight sets and example data, we found that combining similar motion earlier makes us easier to recognize the global movement in the same motion style. Thus, we use the weight $w_1 = 0.5$ and $w_2 = 1.0$ which are same to the bottom in Figure 3 for all the rest experiments. Note that, in either case of the middle or the bottom in Figure 3, the users will be able to view the top snapshot in Figure 3 (the root of the snapshot tree).

In our implementation, one tree node contains a minimal set of data required to draw the snapshot. They are the frame range, optimized viewing angle, joint indices for the selected trajectories, frame indices for the key-frames, and pointers for the left and right children.

5.3. Comic Sequence

The size of a snapshot tree depends on the length of the source animation data. Although the snapshot provides the users with comprehensive understanding of the animation, there are often too many to be displayed all together at once. In addition, because they are not on the linear time line (the same time duration appears in different snapshots), users could misunderstand the temporal order between the pictures. To avoid this mess, the snapshots should be selectively shown to the user in a carefully designed order.

For example, if we display only the snapshots of the leaf nodes in the depth first search order, that sequence shows the animation in the correct order without any missing or overlapped portions of time. We call this the *full-frame comics*.

6. Assembling Full-Frame Sequence

We define the *full-frame comics* as a comic sequence which covers the all frames of the animation without overlapped portions of time. The full-frame comics allows the user to read the sequence straightly without any confusion in the flow of the animation. Thus, we preserves that the displayed comic sequence to the user always retains a full-frame comics form.

Our snapshot tree makes it easier to composite a full-frame comics. When any random intermediate node is collapsed (or detached with all its children and becomes a leaf node itself), the sequence of leaf nodes in the depth first search visiting order always becomes a full-frame comics (see Figure 2 (a), (b), and (c)). This is obvious because the range covered by an intermediate is the same as the ranges covered by both children nodes. Therefore, the user can composite a different full-frame comics by collapsing and expanding any intermediate node. However, if the size of the tree becomes larger, the this could be a tedious task for the user. In the next sections, we introduce three strategies to assemble between different full-frame comics with simple parameters.

6.1. By Combining Complexity

The complexity of combining (see Eq 9) two children nodes tends to be lower when they are in lower levels of the tree. Lower combining complexity means that there are less differences between the parent snapshot and both children snapshots; they may have similar viewing angles, and key poses. Therefore, if the combining complexity of any intermediate node is lower than a certain threshold, then we can collapse the node by default to show a more compact abstraction for the user. In addition, when the user want to see a shorter full-frame comics, we can collapse an intermediate node one by one from of the least combining complexity. On the other hand, if a longer full-frame comic sequence is required by user to observe more details of the animation, we can expand an collapsed intermediate node one by one from of the most combining complexity.

To demonstrate the usability of the method, we built a comic-based motion data browser with 42 motion data files. One motion file was displayed per row for the full-frame comic. The user was able to change the length of each comic sequence by dragging the mouse left or right. The demonstration is shown in the supplementary video.

6.2. By Time Bar

In the most video data processing programs, selected key-frame images are shown on the time bar to aid the user's frame search and other data processing tasks. When the time bar is scaled down and the space is insufficient for all key-frames, they simply drop a key-frame out one by one from the less important one. By using our snapshot tree, we can design a similar interface for the 3D animation data. However, in our case, instead of dropping a snapshot, we can replace a pair of snapshots with their parent snapshot when the time bar is scaled down. The shorter full-frame comics still describe the entire animation with out any missing part, but it occupies less space to fit into the shorter time bar.

The widths of each snapshot vary depending on the size of the contents but not the frame numbers covered by them. This yields the mismatch between the location of a snapshot on the time bar and the actual frames covered by the snapshot, though the total length of the comics is fitted to the length of the time bar. To adjust these mismatches, a different strategy is required for the selection of the node to be collapsed. When the scaling of the time bar is large enough, we just display the longest full-frame comics which is usually the leaf node sequences of fully expanded tree. When the size of time bar is decreased, we first check the degree of mismatch between the decreased time bar and the current full-frame comics $\mathcal{S}$. The function is defined by

$$mismatch(\mathcal{S}) = \sum_i \left(\frac{\|r_i\|}{\|r_{entire}\|} - \frac{width(s_i)}{b} \right)^2 \qquad (10)$$
$$\mathcal{S} = \{r_1, r_2, ..., r_n\}$$

where, $\|r_i\|$ is the total frame number in the duration r_j and $\|r_{entire}\|$ is the total frame number of the entire animation. The function $width(s_i)$ returns the pixel width of the snapshot picture s_i and b is the pixel with of the current time bar. Next, we test the mismatch degrees of every full-frame comics $\mathcal{S}'_j$ that consists of one less number of snapshots than the current comics $\mathcal{S}$ to find the best matched comics $\mathcal{S}'_q$ among them.

$$q = \arg\max_j mismatch(\mathcal{S}'_j) \qquad (11)$$
$$\mathcal{S}' = \{r'_1, r'_2, ..., r'_{n-1}\}$$

If $mismatch(\mathcal{S}'_q)$ is less than $mismatch(\mathcal{S})$, then $\mathcal{S}$ is replaced with $\mathcal{S}'_q$. Otherwise, nothing is done. We can apply a similar strategy for the case when the time bar size is increased. After replacing the full-frame comics, the widths of each snapshot can be slightly adjusted for the finer match-
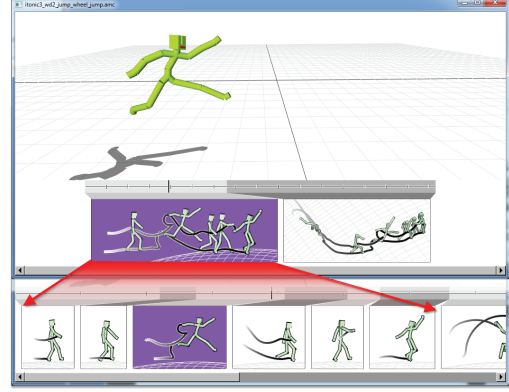

Figure 4: *Comic-guided time bar. For the short time bar (upper), the full-frame comics is assembled with two snapshot which each covers almost half frames of the animation. For the long time bar (lower), the full-frame comics describes detailed postures in the more separated pictures. The user can watch each part of the motion in an optimized points-of-view without adjusting camera angle.*

ing. One advantage of this interface is that the viewer can recognize the relative speeds of motion of each snapshot. For example, if a snapshot contains a long trajectory line but is occupying only a short length of frames on the time bar, it means that the motion is faster than usual. With the comic-guided time bar, the user can avoid the cumbersomeness of adjusting the camera angle as the subject moves because the snapshots in the comics already show each part of the motion with an optimized viewing angle (see Figure 4).

6.3. By Window Area

In a real comic books, the panels are displayed in multiple rows, so that they fill up the 2D space of the book paper. The multiple-row arrangement would also be effective for the use of the 2D window area on the computer system. Our snapshot tree allows design of a suitable full-frame comics which is not too large or small for the given space. The area can be specified by a user input (resizing window area) or the type of displaying device (e.g. cell phone, tablet and etc.).

Given an area, we first count the possible number of rows. It can be calculated simply by dividing the height of the area by the fixed snapshot height h. Then, we measure the suitable length of the comics for the area by multiplying the number of rows to the width of the given area. By applying the same method introduced in Section 6.1, the best fitted full-frame comics can be designed from the tree. To decide the wrapping points, we test every possible case that divides the comics into the same number of subsequences for the number of rows and choose the case minimizing error value, which is evaluated by summing the squared marginal or excess lengths at every row (see Figure 5).
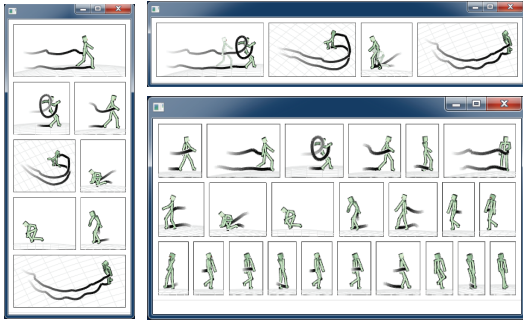
Figure 5: *Three full-frame comics for the same animation. They consists of different numbers of pictures to be fitted into different areas but still cover the all frames of the animation.*
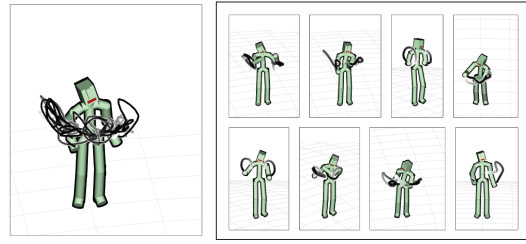

Figure 6: *Two full-frame comics for a hand gesture motion. In the left comics the entire animation is described in one snapshot picture. In the right comics, the motion is divided into eight gestures; then, each is described in a separated snapshot picture.*

7. User Study

To evaluate the preference for our method, we conducted a simple user study with our comic-based motion data browser which is introduced in Section 6.1. The participants were ten graduated students who were not familiar with human motion data. As a comparison target, we implemented a key-frame based abstraction method based on the approach presented by Assa et al. [ACCO05]. This method only collects a given number of key-frames from motion data, and does not suggest any viewing angles for them. This is likely no a perfect comparison target for our method. However, we failed to find a more reasonable previous work to be compared with ours. We set the number of key-frames to be the same number of leaf nodes in the snapshot tree corresponding to motion data, and the camera angles were calculated by applying the same method explained in Section 4.3. To encourage our participants to actively use both interfaces, we asked them to search several example motions such as "Spiral Walk" and "Crawl with Left-turn" by browsing the motion database through both user interfaces. After completing the three tasks for each interface, nine of ten participants answered that they obtained more information from the comic based interface than the key-frame based interface. They credited the trajectory lines and viewing angles dynamically changing with respect to the frame ranges covered by each snapshot. On the other hand, one participant answered that the task with our interface was as difficult as the task with key-frame based interface. We speculate that there would be a difference between individuals depending on how familiar they are with 3D animation data.

8. Discussion and Limitations

We introduced generating a pictorial abstraction for the 3D animation data in which multiple snapshots taken with diverse perspectives were generated and presented as a full-frame comics form for the user. Our user interface enabled the users to explorer between full-frame comics interactively. One drawback of the our method is that a snapshot covering a long time duration becomes too complex an image for the viewer when the subject in the animation has stayed in the same position changing its pose. For example Figure 6 (a) shows a snapshot that describes a long hand gesture motion. From this snapshot, the viewer may recognize that the subjects' hands have been moved a lot, but it is almost impossible to follow the exact movement paths. For aesthetic reasons, a trajectory line that is too complex may be replaced with a simplified cartoon-style speed line [JR05].
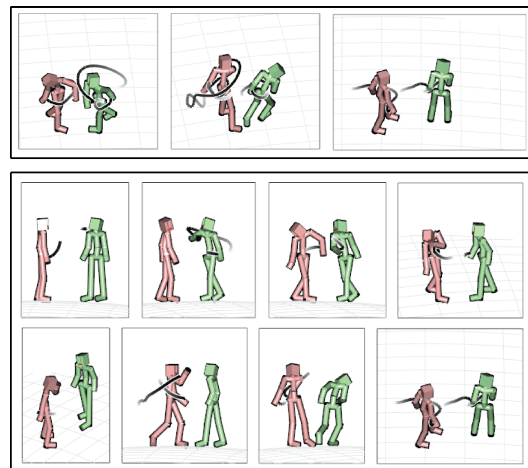

Figure 7: *Two full-frame comics are generated from the same fighting animation. The user can direct comic sequences akin to real comics by using our method. In the upper comics the actions are depicted more dynamically while in the lower comics the actions are more descriptive.*

Our approach can be applied for a multi-subject animation data, in which the motions of all subjects are highly synchronized (see Figure 7). However, it may not be suitable for the case that subjects are far from each other or their motion not synchronized well without interaction. In this case, the monotonic motion segmentation would not work

well with asynchronous motions. In addition, only a telephoto view will be used for every snapshot to take widely distributed subjects together in one picture. In this case, it could be better to generate multiple snapshot trees for each subject group involved in a same event. We can consider to adopt the approach introduced by Yeh and his colleagues. They presented a method to detect group interactions in a multi-character animation data [YLL*12]. A new comic assembling technique may be required to arrange the snapshots from two trees.

In the future, we plan to develop a system for non-experts to generate a high quality narrative comic sequence akin to real comic books. Figure 7 shows an example of two full-frame comics generated from the same fighting scene. The upper comics is more dynamic while the lower one is more descriptive. A cheap and easy motion capture system such as Kinect [KIN] will allow collection of the necessary motion data as the quality of final pictures are not sensitive to the quality of the motion data. To aid user's direction of more full-frame comics, we can adopt the approaches developed for generating comics from a movie [HLC*06] and 3D game play [SRL06].

References

[ACCO05] ASSA J., CASPI Y., COHEN-OR D.: Action synopsis: pose selection and illustration. *ACM Trans. Graph. SIGGRAPH* (2005), 667–676. 2, 3, 8

[ACOYL08] ASSA J., COHEN-OR D., YEH I., LEE T.: Motion overview of human actions. *ACM Trans. Graph. SIGGRAPH 27*, 5 (2008). 2

[ATC*08] AU O. K.-C., TAI C.-L., CHU H.-K., COHEN-OR D., LEE T.-Y.: Skeleton extraction by mesh contraction. *ACM Trans. Graph. SIGGRAPH* (2008), 44:1–4:10. 3

[BGGU00] BORECZKY J., GIRGENSOHN A., GOLOVCHINSKY G., UCHIHASHI S.: An interactive comic book presentation for exploring video. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2000), pp. 185–192. 2

[BGSF10] BARNES C., GOLDMAN D. B., SHECHTMAN E., FINKELSTEIN A.: Video tapestries with continuous temporal zoom. *ACM Trans. Graph. SIGGRAPH 29*, 3 (2010). 2

[BM07] BENNETT E., MCMILLAN L.: Computational time-lapse video. *ACM Trans. Graph. SIGGRAPH* (2007), 102. 2

[BP07] BARAN I., POPOVIĆ J.: Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3 (2007). 3

[BZOP07] BOUVIER-ZAPPA S., OSTROMOUKHOV V., POULIN P.: Motion cues for illustration of skeletal motion capture data. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering* (2007), p. 140. 2

[CGC07] CALIC J., GIBSON D., CAMPBELL N.: Efficient layout of comic-like video summaries. *Circuits and Systems for Video Technology, IEEE Transactions on 17*, 7 (2007), 931–936. 2

[CGRU09] CAGLIOTI V., GIUSTI A., RIVA A., UBERTI M.: Drawing motion without understanding it. In *Advances in Visual Computing*, no. 5875 in Lecture Notes in Computer Science. 2009, pp. 147–156. 2

[CYI*12] CHOI M. G., YANG K., IGARASHI T., MITANI J., LEE J.: Retrieval and visualization of human motion data via

stick figures. *Computer Graphics Forum, Pacific Graphics 31*, 7pt1 (2012), 2057–065. 5

[DS97] DAVE SHREINER GRAHAM SELLERS J. K. B. L.-K.: *The Opengl Programming Guide: The Official Guide to Learning Opengl, Chapter 3 Viewing*. 1997. 4

[FMJ02] FOD A., MATARIĆ M. J., JENKINS O. C.: Automated derivation of primitives for movement classification. *Auton. Robots 12* (2002), 39–54. 5

[Gir03] GIRGENSOHN A.: A fast layout algorithm for visual video summaries. In *Proceedings of International Conference on Multimedia and Expo* (2003), vol. 2, pp. II–77–80 vol.2. 2

[HLC*06] HWANG W.-I., LEE P.-J., CHUN B.-K., RYU D.-S., CHO H.-G.: Cinema comics: Cartoon generation from video stream. In *Proceedings of GRAPP* (2006), pp. 299–304. 9

[HW79] HARTIGAN J., WONG M.: A k-means clustering algorithm. *JR Stat. Soc., Ser. C 28* (1979), 100–108. 3

[JR05] JOSHI A., RHEINGANS P.: Illustration-inspired techniques for visualizing time-varying data. In *Proceedings of IEEE Visualization, 2005. VIS 05* (2005), pp. 679–686. 8

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Trans. Graph. SIGGRAPH 21*, 3 (2002), 473–482. 5

[KIN] Microshot Corp. Redmond WA, Kinect for Xbox 360. 9

[KL08] KWON J.-Y., LEE I.-K.: Determination of camera parameters for character motions using motion area. *Vis. Comput. 24*, 7 (2008), 475–483. 2

[LVJ05] LEE C. H., VARSHNEY A., JACOBS D. W.: Mesh saliency. *ACM Trans. Graph. 24*, 3 (2005), 659–666. 2

[McC94] MCCLOUD S.: *Understanding Comics:The Invisible Art, Chapter 3 Blooding in the Gutter*. 1994. 3

[PRAP08] PRITCH Y., RAV-ACHA A., PELEG S.: Nonchronological video synopsis and indexing. *Pattern Analysis and Machine Intelligence 30*, 11 (2008), 1971–1984. 2

[SHP04] SAFONOVA A., HODGINS J. K., POLLARD N. S.: Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph. SIGGRAPH* (2004), 514–521. 3

[SLF*11] SECORD A., LU J., FINKELSTEIN A., SINGH M., NEALEN A.: Perceptual models of viewpoint preference. *ACM Trans. Graph. 30*, 5 (2011), 109:1–09:12. 4

[SRL06] SHAMIR A., RUBINSTEIN M., LEVINBOIM T.: Generating comics from 3D interactive computer graphics. *IEEE Computer Graphics and Applications 26*, 3 (2006), 53–61. 9

[TZCO09] TAGLIASACCHI A., ZHANG H., COHEN-OR D.: Curve skeleton extraction from incomplete point cloud. *ACM Trans. Graph. 28*, 3 (2009), 71:1–1:9. 3

[VBP*09] VIEIRA T., BORDIGNON A., PEIXOTO A., TAVARES G., LOPES H., VELHO L., LEWINER T.: Learning good views through intelligent galleries. *Computer Graphics Forum 28*, 2 (2009), 717–726. 2

[VFSH01] VAZQUEZ P.-P., FEIXAS M., SBERT M., HEIDRICH W.: Viewpoint selection using viewpoint entropy. In *Proceedings of the Vision Modeling and Visualization Conference* (2001), VMV '01, pp. 273–280. 2

[WPP13] WAMPLER K., POPOVIĆ J., POPOVIĆ Z.: Animal locomotion controllers from scratch. *Computer Graphics Forum 32*, 2pt2 (2013), 153–62. 5

[YLL*12] YEH I.-C., LIN W.-C., LEE T.-Y., HAN H.-J., LEE J., KIM M.: Social-event-driven camera control for multicharacter animations. *IEEE Transactions on Visualization and Computer Graphics 18*, 9 (2012), 1496–1510. 2, 9