Data-Driven Detailed Hair Animation for Game Characters

Chenlei Wu
University of Tokyo

Takashi Kanai
University of Tokyo

Abstract

We propose a data-driven method to realize high quality detailed hair animations in interactive applications like games. By devising an error metric method to evaluate hair animation similarities, we take hair features into consideration as much as possible. We also propose a novel database construction algorithm based on Secondary Motion Graph (SMG). Our algorithm can improve the efficiency of such graphs to reduce redundant data, and also achieve visually-smooth connection of two animation clips while taking into consideration their future motions. The costs for the run-time process using our SMG are relatively low, allowing real-time interactive operations.

Keywords: Data-driven, Interactive applications, Hair animation, Game characters, Secondary Motion Graph.

1. Introduction

Decades ago, in interactive applications like video games, the hair of characters was usually represented as three-dimensional surfaces (e.g. NURBS) with alpha mapping, which looks very unnatural. Friction, collision, and electrostatic forces occurring among hair strands are very complex, and the features of hair strands like coarse surface or small and irregular radius [1] made realistic hair simulation a very hard task. Moreover, the limited resources in games made it almost impossible to represent realistic hair animations at that time.

Many methods are proposed to solve this problem, like cylinders or strip structures [2].

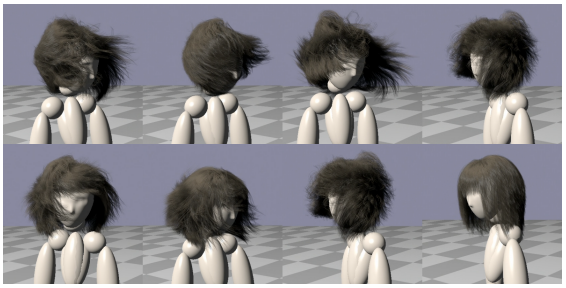


Figure 1: Our method can achieve detailed hair animation of game characters at over 100 fps. The quality of animation is very high and realistic.

These methods can accelerate calculation time for strand-hair simulation and achieve better results. However, the resultant motions are still too coarse and not satisfactory. To achieve visually-satisfying effects, methods that simulate each strand independently [3, 4] are the main trend at the moment. These methods are extremely high-cost and thus time-consuming for real-time interactive applications.

In this paper, we propose a data-driven method to approximate detailed hair animation in interactive applications such as games (in Figure 1). Taking into consideration only the specific features of the game characters, only limited motions (walk, turn left/right, jump, etc.) need to be operated, we can say that actually-appearing animations are somehow predictable. Using data-driven and subspace techniques [5, 6, 7], a database containing hair animation clips is constructed in advance to accelerate run-time process using such a pre-computed database with very low cost.

Some specific problems of this kind of approach include the size of database tends to be

huge if details are sought, and influence of continuous motions like inertia forces cannot be ignored to avoid visual sense of incompatibility. To address these issues, we propose a novel database construction algorithm based on Secondary Motion Graph. Our algorithm can improve the efficiency of such graphs to reduce redundant data, and also achieve visually-smooth connection of two animation clips while considering their future motions.

2. Related Work

Presently, although the quality of video game graphics is very high, a variety of methods are proposed to address the high-cost problem of hair simulation. The most common method is to set a few strands as guide hair and apply interpolation for the rest as shown in [8]. Chai et al. proposed a reduced hair model method [9]to accelerate run-time simulation by selecting effective guide hairs and interpolation weights from training data. But still, the resulting speed varies in the number of guide hairs and the simulation method. Also, the run-time cost for the simulation is still too high for applications like games. Our method differs from Chai et al.'s method that we mainly focus on reducing the calculation cost and have no reliance on simulator in the run-time process.

Other methods like parallel computing on GPU proposed by Han et al. [10] accelerate simulation time. However, both parallel computing methods and interpolation methods cannot deal with hair-hair collisions very well, which plays a very important role in the appearance of animation hair.

Data-driven and subspace method are used in many places to address the run-time calculation cost problem (e.g. fluids [11], deformations [12]). The main concept of data-driven method is to build up a database to support real-time calculation. James and Fatahalian [5] proposed a method to deal with predictable motions for interactive applications. The Impulse-Response Function (IRF) and Impulse Palettes can represent a specific motion sequence, and by combining several motions and recalling the pre-computed data of the selected motion, realistic and complicated deformation can be realized interactively. However, this method focuses on motions for single instant impulses, and cannot realize complex motions. The switch from one impulse to another also lacks accuracy and cannot demonstrate inertia left by previous actions very well.

Guan et al. [6] proposed a method to take hair parameters (length, softness, etc.) into consideration by using multi-linear algebra to factorize hair appearance from the learnt database. This method can realize hair animation in real-time while adjusting hair parameters. However, like James et al's method, Guan et al's method does not attach importance to the inertia force between motions.

Kim et al. [7] combined IRF with a Motion Graph (explained in Section 3.1) as an extension to James' method to deal with real-time cloth animation for complicated motion sequences. The cloth's motions along the body are stored in a Secondary Motion Graph (SMG), in which each node corresponds to a body's animation clip. By expanding the graph from the position where the maximum cloth state's connection error occurs, the inertia effect left by previous motions can be preserved to some extent. This method can realize more complex motions than James's method in [5], and realizes satisfactory cloth animation in interactive applications.

Kim et al.'s method is promising, but as very large databases are considered, the quality of the constructed database becomes important. Stanton et al. applies Kim et al.'s method to fluid in [13]. By detecting the most common actions player tends to take, Stanton et al's method focuses on how to improve the efficiency of the constructed SMG containing important data.

Considering that hair is also an accessory on a character's body, which moves along the body's motion, Kim et al.'s method provides new ideas for how to deal with hair animations in interactive applications. However, the special features of hair make it difficult to directly applying this method. We thus tried to find a way to promote the quality of the database (based on SMG).

3. Motion Graph and Secondary Motion Graph

In this section, we describe the Motion Graph as our input and the Secondary Motion Graph

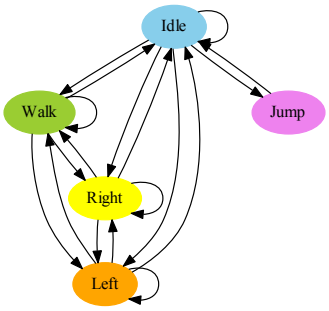


Figure 2: Motion Graph containing five motions: Idle, walk, turn-left, turn-right, jump.

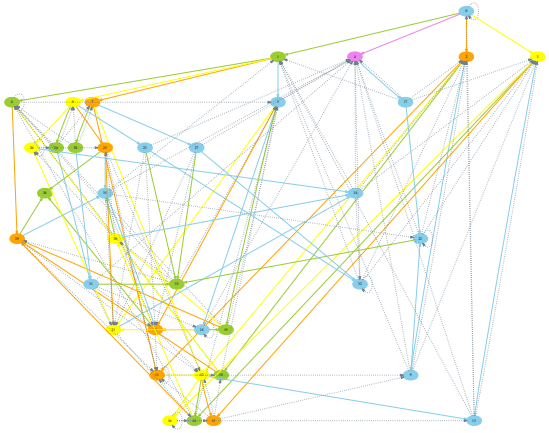


Figure 3: Generated Secondary Motion Graph with threshold $\lambda = 5.5$.

as our output. We also briefly describe Impulse Response Function used in our method.

3.1. Motion Graph

In our method, a primary Motion Graph (MG) is used as input. It can be constructed by a method like [14], and its motions are manually adjusted by software tools such as Blender or Maya. The character's body motion sequences are contained in this graph (Figure 2). Each node represents a specific motion (walk, jump, etc.) constructed by many frames of body poses. Transition restrictions are added as graph's edge. The information we need here from MG is the movement of the head along body.

3.2. Secondary Motion Graph

A Secondary Motion Graph (SMG) is a graph storing data on hair motions corresponding to motion graph nodes. The next motion a user wants to take is in general unknown and unpredictable. Although the inertia or collision forces on body's motion need not be considered, the animations of hair will become very unnatural if they are dealt with in the same way as the body, because of the lack of these force's effect.

To represent other parts attached to the body naturally, we need to construct a SMG (see Figure 3) to store the corresponding hair motions of these parts. A *SMG node* contains,

1. Corresponding MG node ID (MGID),
2. Motion sequence data (IRF) with the same number of frames as its motion graph node,
3. Parent node's ID (which is the previous motion it corresponded to, i.e. the switch it comes from),
4. Children nodes' ID.

The constructed SMG contains more nodes than the original motion graph. That means, a motion of body might correspond to several different hair motion nodes, depending on the motion taken previously. Especially in games, a relax state (e.g. the idle motion) is considered as the default state when no command is given. This relax motion might be related to more SMG nodes than other motions.

A *switch* is a directional link between two SMG nodes. Two motion sequences are connected from the start to end SMG nodes of a switch. Note that a switch and the one with its reverse direction may have totally different motion sequences. There are also self-looping switches from a node to an identical node.

3.3. Representation of Motion Sequences

The motion sequence data in each SMG node is stored in a structure analogous to IRF (Impulse Response Function) database [5]. A matrix is constructed containing the whole motion sequence data. The row of a matrix is frames, and the column is vertex coordinates of strands, which are pre-localized. For such a matrix, singular value decomposition is applied for the compression. A compressed motion sequence is then represented as a set of basis vectors whose dimensions are far less than the number of frames.

4. Construction of Secondary Motion Graph for Hair

To obtain smooth and natural hair animation results when commands are given by the user, we have to construct a Secondary Motion Graph (SMG) along the primary motion graph. In addition, to represent the detail inertia forces passed from the previous motions, we need to expand the SMG to a satisfied level to contain the required details. For example, if a character starts from the Idle-state, do a walk-motion and come back to Idle-state again, the hair's motion for the later Idle-state will be different from the start one due to inertia force. There will also be two SMG nodes stored in the graph related to the same MG node (Idle).

What we need to pay attention to is how to select and save important motion clips needed, to make the graph more *sufficient* (i.e. contain more important data as well as be smaller in size at the same time).

4.1. Hair's Shape Historgram

To select sufficient motion data, we have to find motions that are close to each other. A detailed hair model contains thousands of strands moving separately. One strenuous motion can cause different strands alternating with each other, which makes the distance among strand vertices very large from other states, yet the shape of hair seems to be similar (as shown in Figure 5).

The error metric method Kim et al. used in [7] is simply L0 or L1 Euclidean distance among vertices. For hair, this method cannot identify similar states or motions correctly. Given that there are far more strand vertices than cloth, and that a small amount of deviated hair should not affect the entire look too much, a good way to evaluate the similarities of two hair motions is needed.

Here, we introduce the *shape histogram* (similar with [15]) as demonstrated in Figure 4a to evaluate the similarities between hair states. First of all, the hair states are all localized to the center by multiplying the head motion's inverse (which is known from the MG) when IRF is constructed. The space around the localized hair model is divided into many grids, and the number of vertices in each grid is calculated as the histogram's value in each bin. The non-zero bins in the histograms for both two hair states are used as valid bins to calculate the error (as shown in Figure 4c).

The average of the absolute difference between the histogram of two bins is used as an error between two current hair states. To evaluate whole sequences, we also use the average value of the error of every frame as the *merge error* $\epsilon(a, b)$ for two nodes a, b of SMG,

$$\epsilon(a,b) = \frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{n_j}\sum_{i=1}^{n_j}\left|H_a^j(i) - H_b^j(i)\right|\right), \quad (1)$$

where N is the number of frames, n_j is the number of valid bins for frame j, and $H_a^j(i)$ is a histgram value of node a, i-th bin, and frame j. Note that for a pair of nodes in SMG, N is a constant value, but n_j might change per frame.

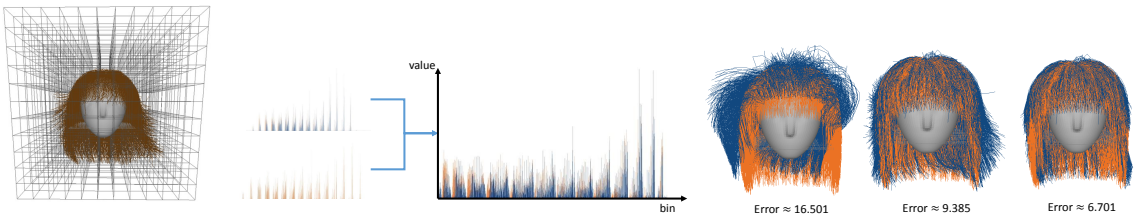
The calculated merge-errors are used to decide whether the current two motions are similar enough to merge, or from which node the next expansion is to be applied, which will be described in a later subsection.

4.2. Constructing SMG for Hair

Unlike Kim et al's method [7] which only considers instant state error when a switch occurs from one motion to another, we want to take the whole sequence of future motions into consideration. To consider such a future motion sequence, we propose here a novel construction algorithm so that more smooth connection between two motion sequences can be established. The procedure to construct a SMG consists of the following steps: 1. Initialization, 2. Expansion, 3. Merge (and expansion), and 4. Addition of response sequence nodes. We will describe our algorithm step by step. In Appendix A, we also show the pseudo code of our complete algorithm.

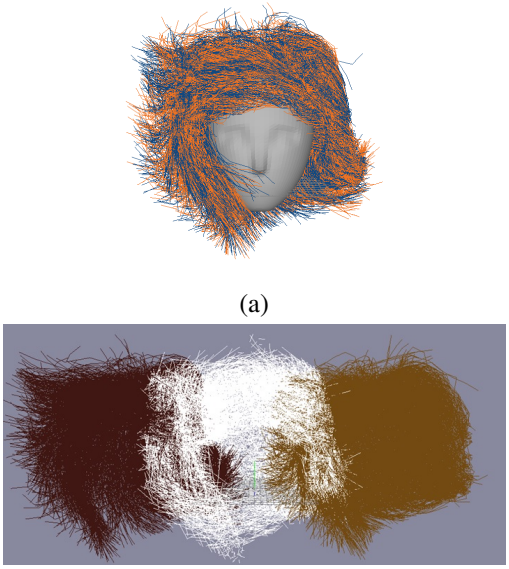
To help understand our algorithm, we will use a MG in Figure 2 as an example. There are five nodes (Idle, Walk, Left, Right and Jump) and 18 switches including self-looping (orange allows, e.g. Idle to Idle) in this graph.

Initialization. First, a start node (e.g. Idle) is selected, and all switches in MG are traversed in a breath-first manner to construct an initial tree of SMG as shown in Figure 6a. In this tree,



(a) (b) (c)

Figure 4: (4a) Bounding box is applied to hair model, number of vertices fallen into each grid is counted as the bin value in the shape histogram. (4b) The valid bin in two shape histograms is the error between these two hair states. (4c) Larger deformations hold larger error values.



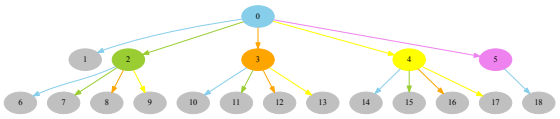
(a)

(b)

Figure 5: In 5a, two hair poses are very similar, but actually, the distance between vertex pairs (as shown in white in 5b) is very large. This is because collision and friction occurring in hair strands made them deviate to each other yet the whole shape is the same.

nodes of the same color have the same MG node id (MGID).

There are two kinds of nodes contained in this tree. One is the formal SMG nodes and the other is the so called *ghost nodes*. The formal SMG nodes are nodes that are actually used during real-time process. In contrast, ghost nodes are potential nodes for SMG. They are stored to determine whether the SMG's accuracy is enough, and will be added as formal SMG nodes if needed. Introducing ghost nodes is the key to our algorithm, since they are considered candi-



(a)

SMG Nodes	0	2	3	4	5	
	1	6	7	8	9	10
Ghost Nodes	10	11	12	13	14	15
	16	17	18			

(b)

Figure 6: (a) Initialized SMG. (b) SMG nodes list (dark color) and ghost nodes list (gray color).

dates for the *future* motions which can be actually taken.

In the initial tree construction, we set the simulation results for MG nodes which are traversed for the first time to be SMG nodes, and results from MG nodes already been calculated to be ghost nodes.

Expansion. Next, we apply the expansion of the graph from ghost nodes so that errors between two motion sequences decrease. For each ghost node g, the errors to the nodes satisfying the following three criteria are calculated using Equation (1):

1. Nodes that come from the same switch, and holds the same MGID.
2. Ancestor nodes with the same MGID.
3. Nodes with the same MGID if no node is satisfied with conditions 1. and 2.

We then select the minimum error ϵ_{min} (hereafter called *to-merge error*) among errors calculated above, and the corresponding node h (hereafter called *to-merge node*) which is the most

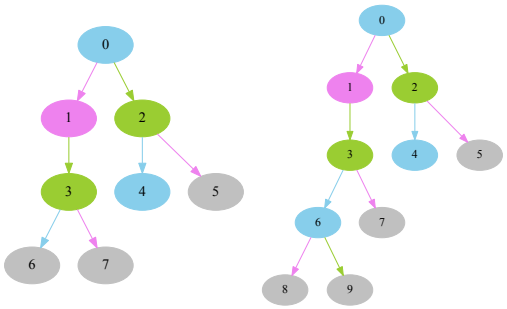


Figure 7: (a) Graph before expansion. If node No.6 holds the largest error, expansion occurs at No.6 as in (b).

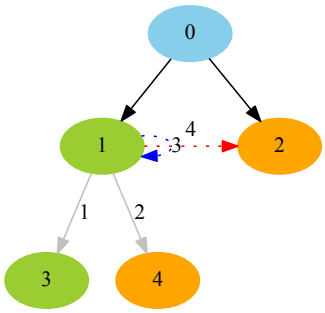


Figure 8: Merge applies to turn the father node's child pointer to the to-merge non-ghost node. If node 3's to-merge node is 1, then the child node of node No.1 (edge 1) will change to 1 (edge 3), and No.3 wil be deleted from SMG.

similar motion. As an example in Figure 7, for a ghost node No.6, No.4 (from condition 1.) and No.0 (from condition 2.) are selected as candidates, and No.4 is finally selected as a to-merge error node.

We now process the expansion step as follows: We first prepare a heap $\mathcal{E}$ and for each ghost node a triplet (g, h, ϵ_{min}) is stored. We next take out a ghost node from $\mathcal{E}$ with the maximum key using delete-max operation and set it to a formal SMG node. We then traverse switches from a new SMG node, create ghost nodes, calculate to-merge errors, and store them to $\mathcal{E}$. This process is repeated until the maximum key is less than a threshold λ. Figure 7 shows how expansion process works.

Merge (+Expansion). When the expansion process is completed, the to-merge error of all keys in a heap should be less than a threshold λ. Next, we merge each pair of two nodes g and h of error ϵ_{min} to decrease the size of SMG. Two nodes to be merged have similar motions.

To execute the merge operation, we continue to use a heap $\mathcal{E}$ created in the expansion process. For each ghost node g of a heap, the merge is applied if a to-merge node h is the formal SMG node. That is, g is discarded and its parent node point is switched to h. If h is a ghost node, merge operation cannot be applied since two nodes are both ghost nodes. In this case, we first set h to the formal SMG node, and apply expansion operation to a new SMG node. Then, we apply merge operation to g described above.

When this process is completed, the remaining ghost nodes are all discarded. Figure 8 shows a SMG after a merge operation has finished.

Adding Response Sequence Node. In the Expansion step, threshold is used. Such a threshold given by user will directly decide the size of the graph. A larger value will cause more detailed motions to be lost, while a smaller value will lead to a huge graph. To reduce the graph to a reasonable size and obtain smooth results at the same time, we consider introducing the idea of IRF [5], and preserving the most easy-to-notice details. As an example, a MG node "Idle" is the most common state for the character, the motion holds the largest number of SMG nodes, and extra details are added to expand these motions.

As shown in Figure 9, we add a longer and cycled motion sequence of "Idle" as a response sequence node (RS node) to the graph. RS nodes differ from SMG nodes as follows: 1. length of node, 2. relationship with other nodes, 3. how they are displayed.

Here, SMG nodes of "Idle" that 1. points to a self-loop node, 2. points to a node of "Idle", 3. is a self-loop node, are considered for merging. We then merge each of those nodes to an appropriate position in the corresponded RS node. This position is where the most similar motion sequence ends, so the node can turn to response sequences with less incompatibility.

To determine an appropriate position in the corresponded RS node, we prepare a matrix form of errors (*error matrix*) as shown in Figure 10. In this figure, the row indicates frames of RS node,

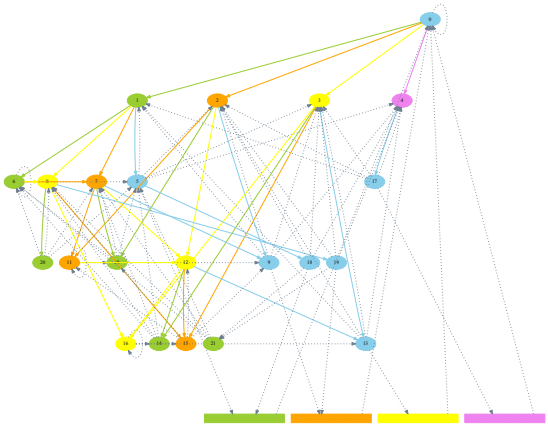


Figure 9: Response nodes shown in the bottom of the graph represents different response motions caused by inertia forces left from previous motions. Although the hair motions contained in response nodes are different, they all relate to the same body motion (Idle state).

and the column shows frames of a SMG node to be merged. Dissimilarity between every frame in RS node and SMG node is then stored as a grayscale color in each element of a matrix where a lighter color holds a larger shape histogram error as described in Equation (1).

In the error matrix, a motion sequence to be selected is represented as a right downward line (a red line in Figure 10). Among such lines, we select one with the smallest variance of shape histogram errors on the line. This ensures that two motions move in a similar strand, and also are usually with small errors. One advantage of the above method is *threshold-independent*, that is, an adequate position in the RS node is determined without using the threshold value.

5. Run-time Process

After the construction of SMG, we use it as a database to support runtime process. As SMG is based on motion graph, only motions contained in graph can be displayed. We compute vertex coordinates of hair in the current SMG node and its current frame by the linear combination of the basis vectors and weights, as the head position changes. The hair motion data contained in IRF does not include displacement information, and only contains deformation data for hair in

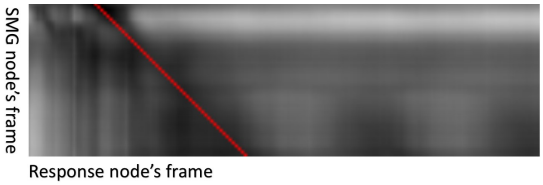


Figure 10: Dissimilarities between every frame in RS node and SMG node are stored. Lighter part holds larger errors, and darker part holds smaller error. The width of the Figure is the RS node length, and the height is the SMG node to be paired. The red line in the Figure is the sequence where the smallest variance exists.

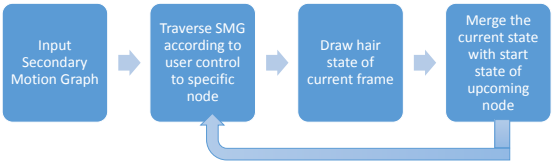


Figure 11: Flow chart for run-time process.

local coordinates. As hair moves together with the scalp (or head), we only need to transfer the hair model together with head joint displacement information.

We now set a constraint to a motion graph about the change from the current node to the next node: Although an action command is given from user during moving, the character continues moving until it reaches the switchable point (end of current node). Then, a motion under execution changes, and the character moves from the current node to the next node (see runtime process in Figure 11).

At the same time, the hair node in SMG will switch to the child node according to the ID of the next motion. As the transition constraints are added to the motion graph, no illegal switch occurs, and the corresponding node will also not be in the child node list.

The switch in MG can always find the corresponding SMG node since a SMG is an infinite loop graph. However, if the response sequence node exists, when a motion switches to the idle state and keeps looping (as a SMG node switches to response sequence nodes), the node is selected and the animation will start from the start frame

which the previous SMG node is pointing. The error between the response node and node of the next motion can be ignored, because the child node of the response node before merging SMG has already been merged to the idle node.

5.1. Connection Penalty

As we use ghost nodes instead of an instant state error, the distortion which occurred while switching from one node to another is larger than Kim et al.'s method [7]. However, by blending two nodes, we can achieve transfer from one node to another without unobvious errors within several frames, even if their start and end states are quite different.

We simply apply linear blending to connect two nodes. As the motion graph node continues to its end, the error must happen between the end of the first node and the beginning of the next node. We save the end state of the previous node as penalty occurring with the switch as distances between vertex positions of such two nodes, and add this penalty back to the start sequence of the next frame (for a certain blending length), as shown in Figure 12. Note that the blending length must be smaller than the node length, so the penalty will not be passed to the next motion.

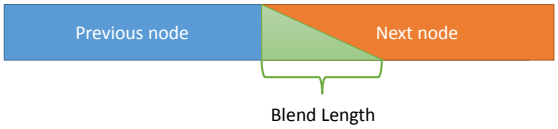


Figure 12: Blend occurs between two nodes, the green part is the penalty added to the next node, within several frames (blend length).

6. Results and Discussion

In our experiment, five motion-captured motions containing Idle, Walk, Left, Right, and Jump from CMU motion capture database [16] with 18 switches are used for the primary motion graph. We use a hair model containing 4134 guide hairs and 41340 vertices with Maya's nHair system as black box simulator to calculate hair motions. Hair-hair collisions and hair-head collisions are calculated, but hair-body collisions are not implemented.

For a reasonable threshold 5.5, 10662 frames of valid motion sequences (about 5.1 GB) are simulated. From such raw data, 2220 frames of motion sequences (about 460MB after SVD) are contained in the final SMG. Table 1 shows the statistical results for constructing SMG set with several threshold values. As the threshold increases, the graph size becomes smaller (1576 frames for threshold 6.5 and 1251 frames for threshold 8.0), but we can still obtain visually-satisfying results by adding RS nodes. Compared to Kim et al.'s method, the size of our graph increases more efficiently while threshold decreases, and more natural-looking results with the same graph size can be obtained, because the effect caused by inertia forces (or future error) are taken into consideration (see the attached video). Less redundant data are contained in the graph with our method.

In the run-time process, we use AMD's TressFX hair system as renderer. Over 100 fps (with CPU Intel Core i7-4770 and GPU NVIDIA GeForce GTX TITAN) can be reached while CPU usage is under 20% and memory usage is about 512MB, without any GPU acceleration or parallel computing.

Limitation. As a limitation, the data compression ratio is not satisfied, since the col/row rate of the matrix for SVD is large. To achieve detailed hair animations, we have to use large amount of hair vertices. Although in the runtime process, the rendering time is saved, the memory cost is unavoidable. For interactive applications, single-strand interpolation could be combined to achieve low memory cost. It is possible to apply Chai et al. [9]'s reduced hair method to obtain the most representative guide hairs for each IRF (as long as the number of guide hair strands for each IRFs is the same), and recover detailed animation by interpolation.

7. Conclusion

We proposed a novel data-driven approach to realize detailed hair animation for real-time applications like games, with very little computation cost during the run-time process. The L0 or L1 error metric between vertex pairs is meaningless for hair, because detailed hair animations

λ	#Frames Sim.	#Nodes Expanded	#Frames SMG	#Nodes SMG	#Nodes [7]
5.5	10662	155	2220	37	-
6.5	6896	105	1576	26	-
8.0	5318	79	1363	22	37
10.0	4227	62	978	15	23

Table 1: Statistical results of our SMG construction algorithm. From left to right: Threshold value (λ), Total number of simulated frames (#Frames Sim.), Number of nodes after expanded operation (#Nodes Expanded), Total number of frames in SMG (#Frame SMG), Number of nodes in SMG (#Nodes), Number of nodes in SMG by Kim et al.'s method with similar visual quality (#Nodes [7]).

require large amounts of hair, and the error of each strand can be very large even though the total shape of hair is quite similar.

We propose a method using hair shape histogram to recognize important (meaningful) animation data. By storing the most important data, we can construct a secondary motion graph with less redundancy. The threshold value varies largely if the relationship between vertex pairs are considered, however, by using our error metric of shape histogram, we can obtain a comparatively stable threshold that varies slightly when the number of strands or primary motion graph changes.

As blending is applied when switching from one node to another, connection error is not as important for hair animation as cloth. By considering total sequence errors instead of instant connection errors, we can achieve more natural results when switching among motions because inertia forces are taken into consideration.

Acknowledgements

We would like to thank Prof. Taku Komura and Prof. Tomohiko Mukai for their precious advices, and Hiromu Ozaki for helping with the code debugging.

A. Our SMG Construction Algorithm

Algorithm 1 shows the pseudo code of our SMG construction algorithm.

References

[1] Clarence R Robbins. *Chemical and Physical Behavior of Human Hair*, volume 4. Springer, 2002.

[2] Chuan Koon Koh and Zhiyong Huang. Real-time animation of human hair modeled in strips. In *Proc. Eurographics Workshop on Computer Animation and Simulation*, pages 101–110, Vienna, 2000. Springer Vienna.

[3] Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frédéric Leroy, and Jean-Luc Lévêque. Superhelices for predicting the dynamics of natural hair. *ACM Trans. Graph.*, 25(3):1180–1187, July 2006.

[4] Andrew Selle, Michael Lentine, and Ronald Fedkiw. A mass spring model for hair simulation. *ACM Trans. Graph.*, 27(3):64:1–64:11, August 2008.

[5] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph.*, 22(3):879–887, July 2003.

[6] Peng Guan, Leonid Sigal, Valeria Reznitskaya, and Jessica K. Hodgins. Multilinear data-driven dynamic hair model with efficient hair-body collision handling. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 295–304, Aire-la-Ville, Switzerland, 2012. Eurographics Association.

[7] Doyub Kim, Woojong Koh, Rahul Narain, Kayvon Fatahalian, Adrien Treuille, and

Algorithm 1 Construct SMG

Require: Motion Graph $\mathbb{M}$, Hair Model
  // 1. Initialization
  Select a start node i
  for all Node j of $\mathbb{M}$ from i in a breath-first manner **do**
    if traversed at the first time **then**
      Add j to a SMG $\mathbb{S}$ as a SMG node
    else
      Add j to $\mathbb{S}$ as a ghost node
    end if
  end for
  // 2. Expansion
  Heap $\mathcal{E} \leftarrow 0$
  for all Ghost node g in $\mathbb{S}$ **do**
    Compute ϵ_{min} and the corresponding node h
    Store a triplet (g,h,ϵ_{min}) to $\mathcal{E}$
  end for
  while $g = \mathcal{E}$.delmax() > λ **do**
    Set g to a SMG node i
    for all switch (i, j) in node i **do**
      Compute ϵ_{min} and h for j
      Add node j to $\mathbb{S}$ as a ghost node
      Store a triplet (j,h,ϵ_{min}) to $\mathcal{E}$
      Set g to a SMG node i
    end for
  end while
  // 3. Merge
  while $g = \mathcal{E}$.delmax() is not nil **do**
    if h is a SMG node **then**
      Change a switch from a parent of g to h
      Discard g
    else if h is a ghost node **then**
      Set h to a SMG node i
      for all switch (i, j) in node i **do**
        Compute ϵ_{min} and h for j
        Add node j to $\mathbb{S}$ as a ghost node
        Store a triplet (j,h,ϵ_{min}) to $\mathcal{E}$
      end for
      Discard g
    end if
  end while
  // 4. Add Response Sequence Node
  Add response sequence node i
  for all Node j of the same MGID in i in $\mathbb{S}$ **do**
    Compute a position in i
  end for

James F. O'Brien. Near-exhaustive precomputation of secondary cloth effects. *ACM Trans. Graph.*, 32(4):87:1–87:8, July 2013.

[8] Hubert Nguyen and William Donnelly. Hair animation and rendering in the nalu demo. *GPU Gems*, 2:361–380, 2005.

[9] Menglei Chai, Changxi Zheng, and Kun Zhou. A reduced model for interactive hairs. *ACM Trans. Graph.*, 33(4):124:1–124:11, July 2014.

[10] Dongsoo Han and Takahiro Harada. Real-time hair simulation with efficient hair style preservation. In *Workshop on Virtual Reality Interaction and Physical Simulation*, pages 45–51. The Eurographics Association, 2012.

[11] Theodore Kim and John Delaney. Subspace fluid re-simulation. *ACM Trans. Graph.*, 32(4):62:1–62:9, July 2013.

[12] Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. Subspace clothing simulation using adaptive bases. *ACM Trans. Graph.*, 33(4):105:1–105:9, July 2014.

[13] Matt Stanton, Ben Humberston, Brandon Kase, James F. O'Brien, Kayvon Fatahalian, and Adrien Treuille. Self-refining games using player analytics. *ACM Trans. Graph.*, 33(4):73:1–73:9, July 2014.

[14] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, July 2002.

[15] Mihael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl. 3d shape histograms for similarity search and classification in spatial databases. In *Proceedings of the 6th International Symposium on Advances in Spatial Databases*, SSD '99, pages 207–226, London, UK, UK, 1999. Springer-Verlag.

[16] CMU graphics lab. motion capture database. http://mocap.cs.cmu.edu/.