A non-rigid registration algorithm for compatible skeletonization

Li Wei  
Xiamen University

Wuyi Yu  
Xiamen University

Xin Li  
xinli@cct.lsu.edu  
Louisiana State University

Abstract—We present a registration framework for fitting one skeleton to another. The algorithm matches and transforms closest points iteratively in a non-rigid manner, so that given a set of skeletons, a compatible deforming sequence can be generated automatically. In our optimization, the variables are transformations on skeletal nodes, and the objective function is a weighted combination of three errors: the Data Error, Smoothness error, and Length-Preserving Error. The registration can be efficiently solved using standard L-BFGS non-linear optimizer. We demonstrate effectiveness of our skeleton matching framework on various 3D models.

Index Terms—non-rigid registration, skeleton matching, deformations

I. INTRODUCTION

Curve skeleton integrates geometric and topological characteristics of 3D shapes, therefore, it serves as an important shape descriptor in object comparison, recognition, and retrieval. Shape matching based on skeletons has been shown to perform better than contour or other curve descriptors of the object. However, pruning skeletons in a preprocessing step for subsequent compatible matching is usually labor intensive and not robust. Another challenging issue in matching shape skeletons is that the structure of similar objects may have completely different topology. These difficulties hinder the use of skeleton matching in many applications.

Treating skeletons as graph and directly conducting graph matching have been studied and shown to have extremely high complexity. We propose an easier matching framework for skeletons, based on their geometric information. We treat the skeleton as a space-time object composed of a point set, instead of a graph. Then we deform points of one skeleton to match ones of another skeleton. Non-rigid iterative closest point algorithm is a good choice for our work. The algorithm chooses the correspondence by searching the closest node in 3D space, and applies the suitable transformation at each node in source skeleton. Such an iterative registration scheme can effectively circumvent topological noise in conventional graph matching. However, on the other hand, topology is certainly also an important issue in skeleton matching. Branch points are different from tip points. Therefore, we take into account the valence (topology) information by describing it as an additional dimensional geometric information. Each node has 4 dimensions. Besides its $x$, $y$, $z$ coordinates we add the “degree” dimension, the nodes with the different valences will be penalized while being paired up.

The main contributions include:

- We present a novel and simple registration algorithm for curve skeletons. The matching is efficient and robust against topology noise.
- Our framework can easily adjust the matching behaviors according to different applications.
- Besides its application on shape matching, we explore the use of compatible skeletonization to do shape blending, based on compatible decomposition.

The remainder of this paper is organized as follows. We first review the related literature in Section II, then introduce our algorithms in Section III. Implementation details and some experimental results are shown in Section IV. We show one application of our skeleton matching in Section V, and conclude our paper in Section VI.

II. RELATED WORK

The previous skeleton matching methods which based on the graph or tree representation of the skeleton, are usually time consuming [9]–[11]. These algorithms usually organize each skeleton as an Attributed-Relation Graph (ARG), the similarity between two objects can be measured by matching their ARGs. However, Graph matching is a NP Complete problem. Several approximate approaches have been studied for the matching purpose.

Another idea to match the skeleton graph is by comparing the geodesic paths between skeleton endpoints [3]. Their approach is motivated by the fact that visually similar skeleton graphs may have completely different topological structures. The proposed comparison of geodesic paths between endpoints of skeleton graphs yields correct matching results in such cases.

Zheng and Sharf introduced the notion of consensus skeletons for non-rigid space-time registration of a deforming shape. Instead of basing the registration on point features, which are local and sensitive to noise, the paper adopt the curve skeleton of the shape as a global and descriptive feature for the task. [17]

The idea of using a shape template and deforming a source object to fit the acquired geometries meets most solutions of the space-time registration [1], [4].

For the surface registration, Allen et al. employed an optimization framework to accomplish the match [2]. The paper formulated an optimization problem in which the degrees of
freedom are an affine transformation at each template vertex. The objective function is a weighted combination of three measures: proximity of transformed vertices to the range data, similarity between neighboring transformations, and proximity of sparse markers at corresponding locations on the template and target surface [6], [13], [16].

In normal conditions, the template doesn’t share the same number of vertices or triangles with the source object. Sumner and Popović [14] build a correspondence map between the triangles of the source and the target by specifying a small set of vertex markers. Deformation transfer computes the set of transformations induced by the deformation of the source mesh, maps the transformations through the correspondence from the source to the target, and solves an optimization problem to consistently apply the transformations to the target shape.

The template can also be a skeleton [5], [7], [8], [15]. These techniques use the similar methods. By standard affine iterative closest point registration, correspondences between the points from source and target objects were robustly found and used to deform the limb skeleton surface.

III. ALGORITHM

Our algorithm aims to match skeleton $CM_s$ to $CM_t$. The skeleton is usually represented as a graph, i.e. a set of nodes connected by edges. The degree of a node is defined as the number of edges that share this node. To accomplish the match, we employ an optimization framework. Each node in the source skeleton $N_{si}$ is influenced by a translation $T_i$. We want to find a set of transformations that move all the nodes in $CM_s$ to a deformable $CM_s'$, such that $CM_s'$ matches well with $CM_t$.

The quality of matching is evaluated by three error functions: Data error, smoothness error, and Length-Preserving error. The matching cost is a weighted sum of these three errors. And the optimization framework searches for the transformation with the minimal matching cost.

A. Data Error

An important criterion of a good match is that the source skeleton should be as close as possible to the target skeleton. For evaluating the quality of a matching, we define a data object term $E_d$ as the sum of the squared distance between each node in source skeleton and target skeleton:

$$E_d = \sum_{i=1}^{n} dis^2(T_i(N_{si}), CM_t),$$

Where $n$ is the number of nodes in $CM_s$, and $dis(N, CM)$ function returns the distance from Node $N$ to it’s closest point on $CM$.

Here we use the data structure k-d tree in this function for efficient computation. On the target skeleton, we build a k-d tree, and $dis(N, CM)$ is computed on it.

If we use only space position of the node to create a 3-d tree, the closest point searching may block in a small range, as it shows in Figure 3. We build a 4-d tree, and $dis(N, CM)$ is computed on it.

If we use only space position of the node to create a 3-d tree, the closest point searching may block in a small range, as it shows in Figure 3. We build a 4-d tree, and $dis(N, CM)$ is computed on it.

For this, we add the degree information as the forth dimension. By using this 4-d tree, the node from source skeleton will match with the nearest node with the same degree in target skeleton. The result is showed in Figure 4.
Fig. 3. The skeleton matching result by using 3-d tree. The source skeleton (blue) matches with the target skeleton (red). As shown in the diagram, the end node of foot in source skeleton does not have a good match.

Fig. 4. The node matching result using 4-d tree. The end node of foot in source skeleton matches with the end node of foot in target skeleton.

B. Smoothness Error

Focusing solely on data fitting error does not provide good skeleton matching. The reason is: even we add the degree dimension into the searching tree, most node has the same degree of 2, which results in a problem that several nodes can match to a closest point. So there will be infinitely several nodes match the single closest node at the same time. On the other hand, we also want to penalize that adjacent nodes match to drastically different target nodes, far away from each other.

We therefore introduce the smoothness error, $E_s$, to penalize these cases. By smoothness we do not mean the smoothness of deformed skeleton, but the smoothness of transformation applied to neighboring skeletal nodes. In other words, we wish the translations applied within a local region to be as similar as possible.

This constrain is formulated between every two nodes that share an edge in the source skeleton $CM_s$:

$$E_s = \sum_{\{i,j\}\in \text{Edge}(CM_s)} \|T_i - T_j\|^2. \quad (2)$$

By minimizing $E_s$, we prevent nearby regions of the source skeleton from being matched to disparate parts of the target skeleton.

C. Length-Preserving Error

If the source skeleton has a similar pose with the target skeleton, using $E_d$ and $E_s$ is adequate enough to do optimization. In the more common situation, the position of $CM_s$ and $CM_t$ are not close, the translated source skeleton would not keep the detail information of the original skeleton. For example, if there are many nodes pair with a single closest node, as we translate the nodes to the new position, the length of edges connecting these nodes will significantly change. In other words, we wish the translations applied on the source skeleton are edge-length-preserving.

To formulate this constrain, we apply metric $E_{lp}$ to every edge in $CM_s$:

$$E_{lp} = \sum_{i=1}^{m} \|\text{Length(Edge}_i) - \text{Original}_\text{Length(Edge}_i)\|^2, \quad (3)$$

where $m$ is the number of edges in the source skeleton, the $\text{Length(Edge)}$ function returns the current length of edge, and the $\text{Original}_\text{Length(Edge)}$ function returns the original length of edge.

D. Combination Of Errors

The final objective function $E$ is the weighted sum of the three error functions:

$$E = \alpha E_d + \beta E_s + \gamma E_{lp}$$

We run the optimization iteratively using L-BFGS, which is an unconstrained optimization based on the Quasi-Newton method, until the error converges. The iteration finishes when:

$$\|E_k - E_{k-1}\| \leq 10^{-4}$$

where $k$ is the number of Iterations. As one iteration step finishes, we update the translations onto the source skeleton. The updated skeleton after every iteration will be used for the starting position in the next step. These skeletons generated in the intermediate steps show how the source skeleton iteratively transforms to the target skeleton.

How to select the weights of three errors is very important. The drawback of Data Error is that it is easy to get trapped in local minima. We avoid this problem by setting a lower weight for data error. Meanwhile, we set a higher weight to Smoothness Error. In fact, the weight of the error is also the weight of the gradient of each error. Therefore, if the weight we set is larger than 1.0, L-BFGS optimization will over decline [2].

We set the weights as below:

$$\alpha = 0.1, \beta = 1, \gamma = 1$$
**Data**: A Source Model $CM_s$, and a Target Model $CM_t$.

**Result**: A Transformed $CM_s$, which make the Error function $E = \alpha E_d + \beta E_s + \gamma E_{lp}$ optimization

```
1 begin
2   Define a L-BFGS iterations state
3     Initialization the solution: a set of vector $\{T_i\}$.
4   while iterative process has not converged do
5     Update the $\{T_i\}$ to Source Model
6       for Each Node $N_{si}$ in $CM_s$ do
7           $N_{si}.Position += T_i$
8       end
9     Data Error
10    for Each Node $N_{si}$ in $CM_s$ do
11       Node $N_{ij} = \text{Closest Point}(N_{si})$
12       $E_d \leftarrow \text{SquaredDistance}(N_{si}, N_{ij})$
13   end
14   smoothness error
15   for Each edge $si$ in $CM_s$ do
16       Node $N_{start} = \text{StartNode(edge}_{si})$
17       Node $N_{end} = \text{EndNode(edge}_{si})$
18       $E_s \leftarrow \text{SquareDistance}(P_{start}, P_{end})$
19   end
20   Length Preserving Error
21   for Each edge $si$ in $CM_s$ do
22       edge $si \leftarrow \text{square(length}_{si}.OriginalLength - length}_{si}.CurrentLength)$
23   end
24   Combination of all Errors Weighted
25     $E = \alpha E_d + \beta E_s + \gamma E_{lp}$
26     state.function = $E$
27   for Each Vector in $T_i$ do
28       state.gradient(i) = $\frac{\partial \text{function}(state)}{\partial T_i}$
29   end
30 end
```

**Algorithm 1**: Non-Rigid Skeleton Matching Algorithm

We discuss how the parameters effect the results of registration will in next section. The entire optimization of our algorithm is showed in Algorithm 1.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this section, we show results of our skeleton registration algorithm. Here we present the experiment result of three models: horse, wolf, and human. We create an interface for our algorithm. The interface is written in Coin3d and Qt. A snapshot of the interface is displayed in Figure 5. By using this interface, user can look over the skeletons, set the weights of three errors, and check the result of registration.

The algorithm can also generate consistent skeletons for a deforming motion sequence as we show in Figure 6. As we want to show how the source skeleton deforms to the target skeleton, every step of deformation in the deformation is illustrated. The data error of every node can also be shown by different color of nodes.

![Fig. 5. The skeleton registration Interface](image)

![Fig. 6. The data error, showed by the deforming motion sequence and the color of node.](image)

**TABLE I** summarizes our skeleton data in experiments. We have run our algorithm on a 2.4GHz PC with 1GB of RAM.

<table>
<thead>
<tr>
<th>Model</th>
<th>Skeleton(size)</th>
<th>Num(Node)</th>
<th>Num(Edge)</th>
</tr>
</thead>
<tbody>
<tr>
<td>source horse</td>
<td>63KB</td>
<td>1149</td>
<td>1139</td>
</tr>
<tr>
<td>target horse</td>
<td>65KB</td>
<td>1191</td>
<td>1183</td>
</tr>
<tr>
<td>source wolf</td>
<td>17KB</td>
<td>347</td>
<td>334</td>
</tr>
<tr>
<td>target wolf</td>
<td>18KB</td>
<td>341</td>
<td>339</td>
</tr>
<tr>
<td>source human</td>
<td>26KB</td>
<td>495</td>
<td>402</td>
</tr>
<tr>
<td>target human</td>
<td>26KB</td>
<td>497</td>
<td>489</td>
</tr>
</tbody>
</table>

**TABLE II** shows the iteration step of matching two horse skeletons. In this experiment, we set the parameters:

\[ \alpha = 0.1, \beta = 1, \gamma = 1 \]

as it’s shown in the table, every step of iterations takes no more than 0.5 second. At step 7, iteration comes to converge, and it just takes no more than 3 seconds altogether. The result of registration is shown in Figure 1.

We made experiments on the models of wolf and human. Figure 7 and 8 shows the result of registration.


TABLE II

<table>
<thead>
<tr>
<th>Step</th>
<th>Data</th>
<th>Smoothness</th>
<th>Length-Preserving</th>
<th>Combination</th>
<th>Spend Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.101483</td>
<td>0.00181579</td>
<td>1.39958e-005</td>
<td>0.0900497</td>
<td>0.469</td>
</tr>
<tr>
<td>2</td>
<td>0.0553903</td>
<td>0.000909741</td>
<td>4.34755e-005</td>
<td>0.0492009</td>
<td>0.422</td>
</tr>
<tr>
<td>3</td>
<td>0.030261</td>
<td>0.000423939</td>
<td>7.95143e-005</td>
<td>0.0269497</td>
<td>0.391</td>
</tr>
<tr>
<td>4</td>
<td>0.00986825</td>
<td>9.03821e-005</td>
<td>0.0001329026</td>
<td>0.0082127</td>
<td>0.390</td>
</tr>
<tr>
<td>5</td>
<td>0.00154748</td>
<td>2.05276e-005</td>
<td>0.000155596</td>
<td>0.00153711</td>
<td>0.362</td>
</tr>
<tr>
<td>6</td>
<td>0.0001973068</td>
<td>3.71601e-006</td>
<td>0.000154423</td>
<td>0.000336516</td>
<td>0.421</td>
</tr>
<tr>
<td>7</td>
<td>0.000135033</td>
<td>1.03341e-005</td>
<td>0.00015191</td>
<td>0.00027716</td>
<td>0.453</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.908</td>
</tr>
</tbody>
</table>

Fig. 7. The skeleton registration of wolf

Fig. 8. The skeleton registration of human.

The parameters definitively effect the result of registration. The reason we set $\alpha$ to be a small value is that data error is very localized. If $\alpha$ is a bigger value, like 1.0, the nodes in source skeleton will simply move toward the matching node. We make an experiment by setting the weights as:

$$\alpha = 1, \beta = 1, \gamma = 1$$

The result is show in Figure 9. Although nodes are close to the target skeleton, the smoothness of deformed skeleton is not good.

V. APPLICATION

We apply our skeleton matching in shape blending. Shape blending is widely seen in movie, animation, and computer games. Previous approaches to this problem, including direct vertex interpolation and an interpolation based on edge lengths and angles between edges, tend to produce self intersections. [12] presents a blending process based on compatible star decomposition to avoid self intersections on 2D shapes, with the results of registration of 3D skeletons, we can extended the algorithm to 3D shapes.

Given a shape $M$ and two points $p, q \in M$, we call $p, q$ are visible from each other if $\overline{pq} \subseteq M$, $\overline{pq}$ is the line segment connecting $pq$. All the visible vertices set of $p$ is called the $p$’s visible region $V(p)$. If a decomposition $D(M) = \{M_i\}$ that $\forall M_i \in \{M_i\}$ is a star shape, we call the $D(M)$ is a star decomposition on $M$.

Given two meshes with a correspondence between their vertices, two decompositions of the two meshes are said to be compatible if there is a one-to-one mapping between them such that corresponding pieces are defined by corresponding vertices.

For compatible star decompositions, we also require correspondence between guards of star shapes. Compatible star decomposition can be used in representation of mesh blending between two meshes. It can avoid self intersection problem during the process of shape blending. The details will be described later.

To get a compatible star decompositions, there are two prerequisites we need: the correspondence vertices on source and target meshes, and the correspondence guards. For the second one, the registration of skeletons meets our need. Based on the result of it, we can easily get a compatible star decomposition. For each node $p$ on skeleton, we calculate
the visible region $V_{M_s}(p)$ on source mesh $M_s$, and $V_{M_t}(p)$ on target mesh $M_t$, for compatible consideration, we make the valid visible region $V(p) = V_{M_s}(p) \cap V_{M_t}(p)$. Then we pick a set of skeleton nodes $\{P\}$, that $\bigcup V(p_i) = M_t$, and $V(p_i) \cap V(p_j) = \emptyset, p_i, p_j \in \{P\}$. Figure 10 show a compatible star decomposition on two meshes. Different guardians’ visible regions are colored in different color, and the correspond guardians’ visible regions are colored in a same color.

![Fig. 10. Compatible star decomposition on two meshes. Different guards’ visible region are colored in different colors.](image)

The blending process between two compatible-star-decomposed meshes can be described as below: first choosing a pair of corresponding guards in the two pieces, and representing the vertices of each star piece in spherical coordinates with respect to the chosen guards. Intermediate triangles are generated by linearly interpolating the positions of the corresponding guards and the spherical coordinates of corresponding vertices. The visibility of the two pieces ensures that vertices are sorted by angle around the guard. Linear interpolation of polar coordinates of corresponding vertices preserves the order of the angles. Thus the vertices of each intermediate triangles are also sorted by angle, which means that the process preserve the visibility, so that the triangles won’t produce self intersection.

VI. CONCLUSION

We present an efficient skeleton matching algorithm based on non-rigid iterative closet point registration. Our algorithm treats each skeletal node as a 4-dimensional point, and the transformation is conducted on each node to achieve a global non-rigid deformation from the first skeleton to the second one. The main advantages of our algorithm for skeleton matching includes

1) **Effective and flexible.** It robustly solves the difficult registration problem for complex skeletons, and it provides flexible user control.
2) **Efficient.** The optimization has fast convergence. Usually less then 10 iterations are necessary to get the matching.
3) **Robust against noise.** This non-rigid matching framework is robust against topology noise of 3D models.

Limitations and Future Work. The current method for closest point searching treats the skeleton as a 4-d graph. It works very well when the source skeleton and target skeleton are comparable (similar). When the given skeletons are drastically different, the matching might not be satisfactory. We have not thoroughly investigate it and therefore an effective matching strategy for these types of skeletons needs to be explored further. Although treating the topology structure of skeletal nodes as geometric information improves the robustness of the matching of noisy data, the matching accuracy still get affected. We will also explore for better solutions.

ACKNOWLEDGMENT

The horse models are from Stanford-shape Repository. The Curve-skeleton computation code is from Tamal Dey. This work is partially supported by Louisiana Board of Regents Research Competitiveness Subprogram (RCS) LEQSF(2009-12)-RD-A-06 and PFund: NSF(2009)-PFUND-133.

REFERENCES