Noname manuscript No.
(will be inserted by the editor)

Automatic Cage Construction for Retargetted Muscle Fitting

Xiaosong Yang · Jian Chang · Richard Southern · Jian J Zhang

Received: date / Accepted: date

Abstract The animation of realistic characters necessitates the construction of complicated anatomical structures such as muscles, which allow subtle shape variation of the character's outer surface to be displayed believably. Unfortunately despite numerous efforts, the modelling of muscle structures is still left for an animator who has to painstakingly build up piece by piece, making it a very tedious process. What is even more frustrating is the animator has to build the same muscle structure for every new character. We propose a muscle retargeting technique to help an animator to automatically construct a muscle structure by reusing an already built and tested model (the template model). Our method defines a spatial transfer between the template model and a new model based on the skin surface and the rigging structure. To ensure that the retargeted muscle is tightly packed inside a new character, we define a novel spatial optimization based on spherical parameterization. Our method requires no manual input, meaning that an animator does not require anatomical knowledge to create realistic accurate musculature models.

Xiaosong Yang
National Centre for Computer Animation, Bournemouth University, United Kingdom
E-mail: xyang@bmth.ac.uk

Jian Chang
National Centre for Computer Animation, Bournemouth University, United Kingdom
E-mail: jchang@bmth.ac.uk

Richard Southern
National Centre for Computer Animation, Bournemouth University, United Kingdom
E-mail: rsouthern@bmth.ac.uk

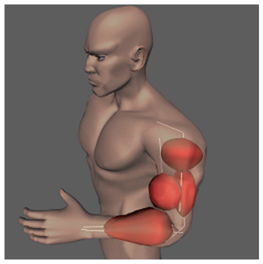
Jian J Zhang
National Centre for Computer Animation, Bournemouth University, United Kingdom
E-mail: jzhang@bmth.ac.uk

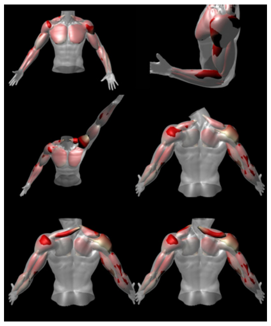
Keywords Muscle modelling · Character animation

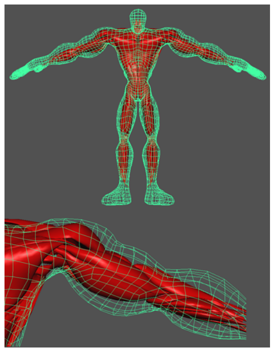
1 Introduction

In animation production, creating realistic character animation is a very complicated process which is time-consuming and laborious. While some of the work requires professional artistic skills, most of the time will be spent on repetitive tasks such as skin weight painting, muscle modelling, and detailed tweaking of skin deformation frame by frame. In the last two decades many attempts have been made to automate this process to save artists from having to perform this tedious task. Researchers have tackled most of the processes in character animation, such as character modelling (skin shape), rigging [1–3], muscle modelling and deformation (skinning) [4–7]. Several techniques [8–11] have been published in the last two decades on the topic of muscle deformation. Most of these focused on the physical properties of muscle, trying to mimic their functionality in real human and animal bodies. Some of these methods have been adopted in medical visualization and surgery simulation. However these approaches are seldom used in animation production as they involve excessive computation, and do not offer significant visual improvement. The production of realistic muscle deformation still requires significant manual work by animators.

Muscle modelling involves not only modelling the shape of each muscle, but also their placement and grouping, which influences the deformation of the skin layer. Initially only very simple muscle shapes [12] are introduced, such as the fusiform, multi-belly and bi-cubic patch muscles. Tools for muscle modelling have been integrated into Autodesk Maya, a professional modelling package, since 2008. Mimicking complicated human musculature from basic shapes depends entirely on the animator's creativity. To automate


(a) Pratscher [13]


(b) Yang [14]


(c) 75 feature points used for Seo's method [3]

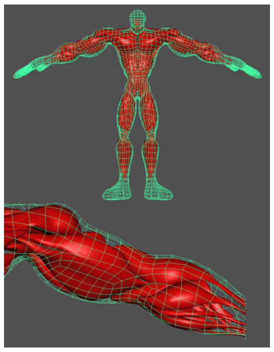

(d) our result

Fig. 1 Comparison of similar works on muscle modelling

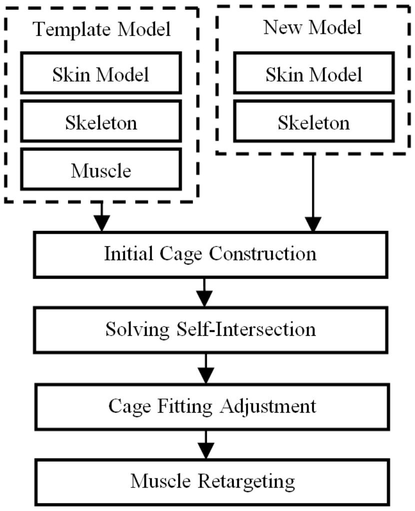


Fig. 2 System structure

this procedure, Pratscher et al. [13] presented an "outside-in" muscle modelling technique which builds muscle structures from the analysis of the skin shape. However the muscles constructed are still too simplistic (Figure 1(a)) to create realistic muscle deformation. Later work by Yang [14] based on Constrained Delaunay Triangulation created more complicated musculature (Figure 1(b)), but the results are still far from realistic.

For each new project, an animator will typically need to model skin, skeletal and muscular structure from scratch. Unfortunately there is no process by which these complex assets can be efficiently re-used on later projects. This costly procedure has led to concept of muscle retargeting, whereby an existing muscle setup, consisting of the models and controls, is deformed to match a new character. This level of automation would naturally provide a significant saving to production costs. However since the new character may look very different from the original model, simple affine transformations cannot produce a useful result.

In order to achieve high quality muscle transfer, a representation of the internal space of the original character must be transformed to fit into the new character. If we consider the skin surface as a cage wrapping around the muscle structure, this retargeting task can be seen as *cage based space deformation*. Seo [3] presented a technique based on this principle to retarget both the original skeleton and muscles to a new character. However since the mapping is based on a simplified body segmentation, the retargeted muscles are not guaranteed to closely fit the new skin model (as in Figure 1(c)).

Based on the same retargeting idea, we present a new muscle modelling method which starts from a template model including the skeleton structure, skin surface and a pack of muscle structures. Given a new character model, our method builds a volume mapping between the template skin and the new skin model. The muscles from the template model, which are embedded in this volume, are deformed to match the new model. Since our template muscle structures originate from a real human body, the retargeted muscles will closely resemble the actual human anatomy. Our novel volume mapping technique optimizes the way the cage fits the new skin shape (as showed in Figure 1(d)), meaning that the mapped muscles can be packed tightly inside the skin surface. In this way, the animator can begin the process of muscle deformation with a high quality muscle representation. Figure 2 shows the system framework.

In Section 2 we introduce related work on cage modelling and deformation. In Section 3 we discuss how we build the retargeting cage from the skeletal structure, and how we adjust the cage to fit more closely with the new skin shape. In Section 4 we describe a method to prevent cage self-intersection, and in Section 6 we discuss our experimental results.

2 Related works

The process of retargeting muscular structure from one character into another is effectively a 3D space mapping problem solved by volume deformation. In early work [15–18] a lattice was used to deform the embedded shape. However, since the lattice involved a limited number of control points, the deformation appears rigid. This lead to many publications considering a dense polyhedron cage which gives the animator more flexible control over the deformation.

Mean Value Coordinates were first introduced by [19] for 2D polygons and then extended to 3D [20–22]. Ju et al. [20] presented an efficient MVC method which encloses the deformable object within a triangular cage surface. Joshi et al. [23] presented Harmonic coordinates [24] which supports non-negative coordinates, capable of handling highly concave cages. Lipman et al. [25] presented Green coordinates [26] which preserves the deformed object shape during cage deformation. However since the retargeted shape may extrude outside the cage, it is not suitable to our muscle retargeting problem, as the deformed muscles may intersect the skin surface.

An alternative approach to space deformation are techniques based on scattered-data interpolation [27, 28]. However since we already have a skin mesh which envelopes the space to be mapped, using only a few feature points on the surface without considering the existing topology is not appropriate to this problem.

As an alternative to the surface cage, the volume cage [29] embeds deformable objects into a simplified tetrahedron mesh. The control mesh complexity increases the required computation significantly. In the case of muscle retargeting, it is difficult to find a consistent method to locate the inside vertex in the two different character models. For these reasons we use a surface cage in this paper.

As with other space mapping methods, we use a cage to build up the spatial transfer between the template model and the new skin model. However in this specific muscle retargeting problem, we not only need the spatial transfer to be smooth, but also the retargeted muscles should be tightly packed inside the new skin model, otherwise the muscle deformation may produce an unexpected result.

Several methods have been presented for cage construction from meshes. Xian and co-workers [30] present an efficient method based on simplifying an enveloped voxelization of the polygon mesh. However, the cage created is highly dependent on the shape of the original mesh, making it is impossible to enforce a consistent topological structure on different meshes. Ben-Chen et al. [31] generate the envelope based on a sampling set of the vertices from original mesh, and no topological information from the original mesh is considered in the process. It may create cages with different topology when some parts of the original mesh are in a very close distance. The method of Tao [6] constructs and fits a cage to the input geometry, but requires manual input to resolve surface self-intersection.

Seo and co-workers [3] presented a retargeting method to transfer both the skeleton rig and muscles to a new model. Their approach requires the animator to manually specify point correspondences between the source and target model. The cage is then constructed based on surface patch segmentation. While our method uses the same spatial deformation method, our methods differ in how the cage is constructed.

The approach in [3] is very general, able to map muscle between characters of arbitrary topology. However, this generality is largely useless in the animation setting, where the vast majority of muscled characters are humanoid or quadrupeds. This generality necessitates the manual specification of feature point correspondences which is an exceedingly tedious task and exposes the technique to the possibility of user error. In the human example used in [3], about 50 feature points were manually specified, which creates a cage which is too simple for high quality muscle retargeting.

Our method exploits the topological structure of humanoid characters defined by the skeleton rig, and builds the cage of a user specified density automatically. The cage (Figure 11(b)) used for Figure 1(d) consists of 508 automatically generated points which provides an accurate fit of the muscles within the skin, but would be impossible to specify manually.

In this paper we will build up the cage from the skeleton structure. In order to solve the self-intersection problem, we introduce a sphere mapping based optimization method to adjust the location of vertices on the cage such that they do not penetrate the skin surface while ensuring that muscles are tightly packed under the skin surface. This ensures the generation of a valid cage (Figure 11(b)).

3 Initial cage construction

To construct an appropriate mapping of the muscle structure into a new character model, the deformation cage should have the following properties:

1. The cages for the template model and the new character should have the same topology and mesh configuration. For example, the feature vertices on the belly should be kept at the same position on both models. Otherwise muscles may be mapped to the wrong position, or even skewed or twisted.
2. The cage must be sufficiently simple, and must not involve skin details. The cage is meant to wrap around muscles rather than unrelated details on skin surface, such as scales, clothing or even antlers.
3. The cage should be complex enough to represent the shapes of skin bulges implying underlying muscles.

We present a new cage construction algorithm designed for muscle retargeting which adheres to these desired properties based on the information contained within the embedded skeleton. As long as both characters which have the same skeleton structure we can guarantee topology consistency between these two cages. As all the vertices of the cage are computed from the intersection between the skin surface and a ray cast from an inside joint, if the skin mesh is a closed mesh, the cage will not be affected by any outside skin surface details. The complexity of the cage mesh will be controlled by the user through three sets of parameters. The user can choose to build up very dense cage models to increase the mapping accuracy. We will show how these parameters will affect the fitting of the muscles in the final experiment section.

The skeletal structure inside a human body is very complicated, making it very hard to design a simple algorithm to construct the cage based only on the template cross-sections (as in [6]). The cage at the pelvis and chest must be specially designed. We present an algorithm to construct the cage segment by segment based on the topological structure of the skeleton. By choosing the same density parameters for all cage segments, we can ensure the cage segments will be seamlessly integrated into a final closed mesh.

As an animator arrives at the stage of muscle modelling, he/she should have a character skin model and a rigged skeleton structure both ready at the rest pose. Using these as the only input, in the following sections we present the process whereby each cage segment can be constructed.

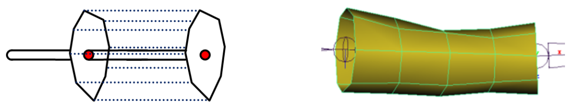


Fig. 3 The cage cross-section at the linear linkage joint

3.1 Linear Linkage

If a joint has only one parent and one child, such as the elbow and knee, we will build up the cage cross-sections as in Figure 3.

Just like in [6], we build up the cross-section polygon at the linear linkage joint. The number of vertices for this polygon is defined by the density parameter d_c, while the number of cross sections along the bone is specified by a second density parameter d_b. In Figure 3 d_c and d_b are 8 and 5 respectively. Both parameters can be selected by animator considering the complexity of the skin mesh in this area.

3.2 Bifurcation linkage (joint with three links)

A bifurcation normally occurs at a character's root joint. Figure 4 shows a frontal view of the cage wrapping around this joint.

The cage construction at the bifurcation joint is particularly complicated. Suppose J_r is the root joint, J_1, J_2, J_3

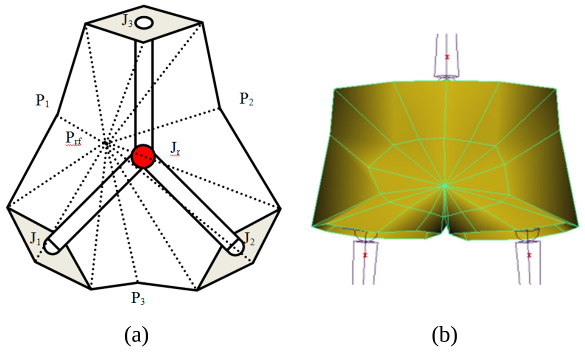


Fig. 4 Cage segment at the bifurcation joint

are three linked joints. From J_r a ray is casted in the direction perpendicular to the plane $< J_1, J_2, J_3 >$, the intersection on the skin surface in the front direction is P_{rf}. For the joint structure in a human character, J_1, J_2, J_3 are all linear linkage joints. We can build the cage cross-section at these three joints as in Figure 3. We will cast three rays from J_r at the middle of the angle between adjacent joints $< J_1, J_2 >$, $< J_1, J_3 >$, $< J_3, J_2 >$, the intersection with the skin surface will be P_1, P_2, P_3. So the front side of this cage section can be constructed by linking P_{rf} with all the front sides of the cross-section at J_1, J_2, J_3 and P_1, P_2, P_3 as in Figure 4(a). This figure only shows the situation in which the cross-section polygon at a linear linkage joint has four vertices, and the front side has only one ring of vertices. We can add several rings around P_{rf} to make sure the front side has higher density of vertices. The number of rings is defined by the third density parameter d_r which is 2 in Figure 4(b).

3.3 Cross linkage (joint with four links)

The joint at the chest has the cross linkage as in Figure 5. Normally in a character model, $< J_1, J_2, J_3, J_4 >$ roughly lie on the same plane. So our cage can be constructed in the same way as bifurcation linkage. Project the centre joint J_r in the normal direction of this plane, intersect with the skin surface with P_{rf}. Casting four rays from J_r in the direction half the angle between two adjacent joints $< J_1, J_2 >$, $< J_2, J_3 >$,$< J_3, J_4 >$,$< J_4, J_1 >$, intersected with the skin surface with $< P_1, P_2, P_3, P_4 >$. Linking these vertices as in Figure 5(a) will form the front part of this cage.

We can construct the back part in the same manner. As with the bifurcation case, we can also form another ring of vertices around P_{rf} to produce a more dense cage model as in Figure 5(b).

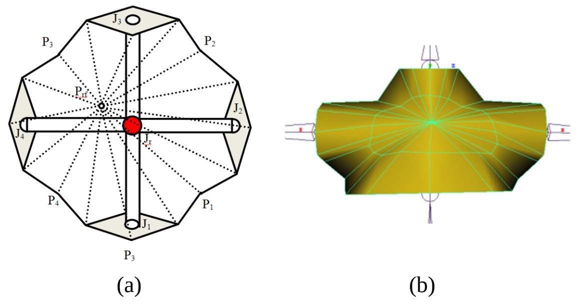


Fig. 5 Cage segment at the cross joint

3.4 Ending joint

For the special purpose of muscle modelling, we will not consider the head (including facial musculature), hands or feet in our system. These three parts are represented with an ending joint at the wrist, ankle and the centre of the head, at which we construct a cap for the cage as in Figure 6. From the cross-section vertices, we cast rays in the direction of the last bone, intersecting with skin at $< P_1, P_2, P_3, P_4 >$; cast another ray from J_r in the same direction, find P_5. Using these five points we form a cap to close up the cage. The

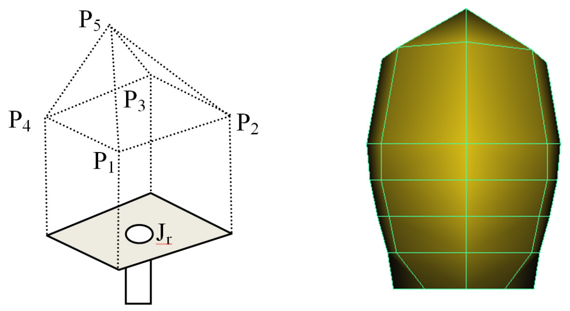


Fig. 6 The cap at the ending joint (head)

above four methods cover all possible joint configurations of humanoid characters. Rules for cage construction of skeletal structures with alternative topology, for example an octopus, could be deduced using a similar methodology.

Following these procedures, we can construct the cage for the whole body. Figure 9(c) shows the cage constructed from the skin and skeleton model in our template (Figure 9(a)).

4 Self-intersection problem

As with [6], our cage construction algorithm also suffers from the self-intersection problem (Figure 10(a)), which can cause a serious artefact as a result of performing muscle retargeting. Tao [6] proposes that this problem can be manually corrected. However, our experience suggests that it is almost impossible to interactively tidy the messy polygon as in Figure 10(a).

We analysed the cage segment by segment and found that some of these problems can be easily solved, such as the case in Figure 10(b), which we call the over-shooting problem. As all the vertices of the cage result from the intersection between a shooting ray and the skin surface, in some cases the ray may intersect the incorrect part of the skin mesh. In Figure 10(b) a ray is cast from the right pelvis joint to the left with the half angle between the two adjacent joints, but intersects with the left leg. To solve this over-shooting problem, i.e. to find a correct angle for the ray shooting, we use a trail and error method to cast several rays around the half angle ray. If the distances between the joint and the intersection points on the skin mesh have large difference, we can identify this error. Figure 10(c) shows the fixing result.

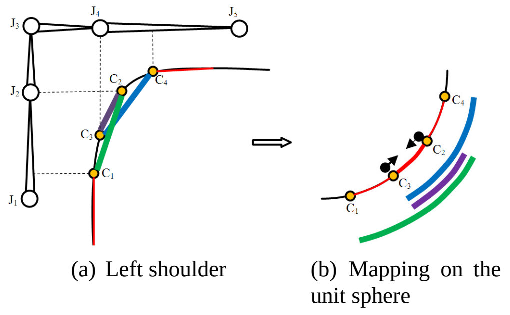


Fig. 7 Using spherical parameterization to solve cage intersection.

A straightforward approach to self-intersection detection is to evaluate the orientation of each cage face. However, for characters with complex geometry it is difficult to specify a criterion to robustly detect face flipping. In Figure 7(a) we demonstrate a common self-intersection example at the shoulder joint. Large dihedral angles between adjacent cage faces may not indicate an intersection (see P_5 in Figure 6). Another approach could be to compare the local cage normal with an associated local skin surface normal, but finding the skin surface corresponding to the cage vertices may be difficult on a complicated 3D surface.

Our novel solution to the self-intersection problem is to solve it in parameter space. Fold-over is detected by “pinning” the cage vertices to the parameterized skin surface (Figure 7(b)) and minimizing the area which the cage covers in parameter space. As we can assume a spherical topology, a spherical parameterization technique is used. This is explained in more detail in the following section.

4.1 Spherical parameterization

As previously stated, a cage generated by the method in Section 3 may have self-intersections which jeopardise the process of retargeting the muscles to the new skin mesh. To prevent self-intersection and cage flipping errors, we represent the skin mesh by a spherical parameterization as in Figure 10(e). We establish a one-to-one mapping between mesh vertices and surface points on a unit sphere. After that, all cage vertices are nailed to the exact position on the skin faces, and transformed to the sphere surface following each individual face. By doing this, we reinterpret our initial problem of relocating the cage points along the mesh to rectifying the self-intersections as moving their corresponding points on the sphere to rectify the self-intersections. While mesh geometry may vary, the shape of the unit sphere remains unchanged, providing a unified approach to tackling this problem.

As in [3], a precondition for our algorithm is that the input geometry represents a genus zero surface, i.e. topologically equivalent to a sphere.

Given a skin mesh M represented as $< v_i, e_{ij}, f_{ijk} >$ where $< v_i >$ is the vertices set, $< e_{ij} >$ is the edge set and $< f_{ijk} >$ is the face set, its spherical parameterization presents a 3D mapping:

$$S : M \left\langle v_i, e_{ij}, f_{ijk} \right\rangle \to M^s \left\langle v_i^s, e_{ij}^s, f_{ijk}^s \right\rangle \quad (1)$$

where v_i^s is the corresponding set of vertices on a unit sphere. We adopt the Barycentric Spherical Parameterization method [32] to establish the one-to-one mapping relation. For an arbitrary cage vertex v_l^c, we use the barycentric coordinates of the point with respect to the mesh triangle f_{ijk} it belongs to and approximate its corrsphonding point v_l^{cs} on the sphere as interpolation of vertices of spherical triangle f_{ijk}^s with the same set of barycentric coordinates.

Similarly, we can map a cage triangle onto the unit sphere as a spherical patch by the same mapping function S. For a cage triangle T, we have it's patch area on the sphere as A_T. It is noted that when there is a self-intersection on the cage, the counterparts of overlapped triangles remain overlapped on the unit sphere. This gives us a convenient way of checking if the cage has self-intersections. If the summation of areas of all corresponding patches of cage triangles is greater than the sphere area, the self-intersection happens. Therefore, if we want to rectify the self-intersection, we can relocate the cage vertices to alter the area of their corresponding patches, so that

$$\sum A_T = 4\pi \quad (2)$$

The area of an individual corresponding patch can be estimated from the summation of all the spherical triangles f_{ijk}^s it covers. We can redefine the problem as an optimization problem by minimizing an energy function relating to the area as

$$min\ E_s = \sum A_T \quad (3)$$

This optimization is more robust to the numerical error introduced by the discretization. As the spherical parameterization fails to preserve the area, the area of some overlapped cage triangles is relatively small (e.g. the area of arm or leg) and will cause slow convergence when we try to solve the optimization problem. To get around of this, we simply replace the summation of A_T in equation 3 with a weighted summation, where the weight is defined as $S^{-1}(A_T)/A_T$, where S^{-1} is the function to find the area of the patch after inverse spherical mapping. Basically it is the ratio between the areas of original 3D triangle and the spherical patch.

The optimization problem in Equation 3 can be used to rectify cage self-intersection. However, the output sometimes includes degenerate cage triangles with zero area, which are as harmful as self-intersection for the process of muscle retargeting. A penalty is added to the overall energy function to eliminate the degeneration triangle. We define

$$E_A = \sum \left(\frac{S^{-1}(A_T) - \frac{1}{N}\sum S^{-1}(A_T)}{\frac{1}{N}\sum S^{-1}(A_T)} \right)^2 \quad (4)$$

With N is the total number of the cage triangles, and $\frac{1}{N}\sum S^{-1}(A_T)$ is the average area of the projection of A_T back to the mesh.

We expect the new cage point to be close to the associated joint. We measure the distance of the initial cage point and its new position with the angle of their casting ray, θ_i. The cosine of the angle can be calculated by the dot product of the two vectors which start from the joint position and end at the corresponding mesh vertices. Therefore, we have

$$E_R = \sum cos^2 \theta_i \quad (5)$$

With the above three energy functions, we can specify the new location of cage vertices to rectify the self-intersection and degeneration problems by the following optimization problem:

$$min\ E_s + k_A E_A + k_R E_R \quad (6)$$

where k_A and k_R are the associated weight. In our experiments, k_A is set to $\frac{4\pi}{N}\sum S^{-1}(A_T)$, and k_R is set to $\frac{1}{N}\sum S^{-1}(A_T)$. This is a non-linear optimization problem, which can take a long time to converge when the number of cage vertices is relatively large.

As an alternative, we decide to use a greedy algorithm to solve the above optimization problem, which finds a feasible (rather than an optimal) solution to rectify the cage self-intersection.

Using the criteria of the area on the spherical parameterization, we determine whether a cage is self-intersecting. Similarly, for an individual cage vertex v_i^c, we determine which points are associated with the intersection. By moving the vertex on the mesh to a new position with a small perturbation, it becomes $v_i^c + \delta$. If the associated energy term E_S decreases for a certain displacement δ, then this vertex has contribution to the self-intersection and is labelled as an intersection point.

In our greedy algorithm, firstly, on the cage we label the intersection vertices which lying inside other patches on the sphere surface. Then, for each intersection points, we provide a set of candidates for its new position by casting a ray with some random deflection from the initial ray onto the mesh. The differences of the overall energy $E = E_s + k_A E_A + k_R E_R$ of the initial position and those candidates are computed. We then cache the initial vertex and its candidate which has the lowest overall energy and the energy decrease resulting from moving the vertex to its new position. Repeating the above process, if we find a larger drop of the overall energy, we then cache the pair of vertices (the cage vertex and its candidate). When all the intersection vertices and their candidates are visited, we have a cached vertex and its candidate. By moving this vertex to its candidate position, we can have the largest drop of the overall energy. Then we finish a loop with one cage vertex updated. We then start the loop again until the overall energy converges or the loop reaches a presented number. Figure 10(d) shows the cage self-intersection at the shoulder part of the fat model (Figure 11(e)), 6 intersection cage vertices (Figure 10(f)) are detected from the spherical parameterization. All 6 intersections are fixed from the first iteration. Figure 10(g) shows the final fixed result.

5 Cage based muscle retargeting

After building up a valid cage we can map the interior muscle structure into the new model. Many methods have been proposed to perform cage based deformation, such as MVC[20,21], PMVC[22], Harmonic coordinates[23] and Green Coordinates[25]. Because the cage for character models normally doesn't have large reflex angles (as shown in Figure 9(c)) and both cages are built from the same T-pose, these methods performed quite similarly in our experiments. As a result, we use the quicker, more efficient Mean Value Coordinates approach to perform the cage based deformation. Our cage may not entirely encapsulate all the muscles, and some muscles may lie outside of the cage. As Mean Value Coordinates are continuous and tolerant of negative coordinates, this did not cause any problems in our examples.

Table 1 Greedy algorithm to solve self-intersection

```
1. Label the intersection cage vertices;
2. For first intersection vertex, compute the value of overall
   energy, $E$. Set $E_{min}$ with the value of $E$
3. For each intersection vertex;
   a. Identify a set of random mesh points as candidates;
   b.For each candidate, compute the value of overall energy,
   $E$; if $E < E_{min}$, update $E_{min}$ to $E$, cache this intersection
   vertex and this candidate;
4. Update the cached intersection vertex to the cached candi-
   date.
5. If not converge or reach maximum loop number, go to step
   2.
```

6 Experiments

We implement this muscle retargeting method into an Autodesk Maya plugin. Cage construction is developed using PyMEL for maya; because MVC involves very heavy computation for our large mesh, this module is implemented using C++ Maya API. Figure 8 shows how it works in Maya with our python code. Our prepared template character model includes skin, skeleton and muscle structure. Given any new character model, the user can easily build up the cage and map the muscle into the new skin model. The self-intersection module of our system was developed separately. Since we use the Barycentric Spherical Parameterization method from [33] which only supplied Matlab code, we develop an input and output interface to transfer data between Maya and Matlab.

Figure 9 shows the template model which includes the skin mesh, skeleton structure (Figure 9(a)), the muscle models (Figure 9(b)), and the cage (Figure 9(c)). To test our method, we choose an extremely muscular model (Figure 11(a)) and a fat character (Figure 11(e)). The cage is showed in Figure 11(b) and 11(f). Figure 11(c) and 11(g) show the retargeted muscles.

The tightness of fitting of the muscle models inside the skin mesh is essential for muscle animation. In the template model, the muscle structure is tightly packed inside the skin model. In order to measure how tightly the transferred musculature fits inside the new skin model, we use cage based deformation to map the template skin on to the new skin model. If these two skin models are close enough, then we can say the retargeted muscle structure is also "close" to the new skin model. Here we adopt the method from [34] to measure the closeness between the two skin meshes. Figure 11(d), 11(h), 12(d) show this experiment, in which we compare the mapped template skin to a new character skin model. Since the mapping only involves the torso, arm and legs, the head, feet and hand are not considered in this computation. It uses rainbow colour from blue to red representing the distance from 0 to maximum. Table 2 shows the result from all our experiments. All cages are created from (8, 4, 2) density parameters, including 508 vertices and 568 faces, the muscle model include 76561 vertices, 78943 faces, the size means the diagonal distance of the bounding box. The mean distance is less than 1% of the mesh size, indicating a very good fitting. The fat model presents a big different figure from the template model. Using the same density cage will induce larger mapping errors than other experiment. Choosing a denser cage would improve the result.

In another experiment we evaluate how cage density affects mapping fitness. If we adopt a very dense cage mesh, the muscle structure fits more tightly into the new skin surface. However this will increase the complexity of computation. Figure 12 shows the result of this experiment. For cross-section density d_c, we choose the polygon with 4, 6, 8, 10, 12, 14, 16 sides, bone section d_b with $2,4,\cdots,16$, number of rings d_r at fan joints is $1,2,\cdots,16$. From Figure 12, we can see, the fitness of the mapping is increased along with density of the cage. However when the density reaches 8, there is little further improvement. In practice we normally choose $(8,8,4)$ as the cage density parameters.

There is an indirect relationship between cage density and the occurrance of self–intersection. A low density cage seldom results in this problem, but for $d_c \geq 8$ or $d_r \geq 4$ we notice self–intersection appearing with more frequency at the bifurcation linkage and cross linkage. For example, an $(8,8,4)$ cage for the muscular model in Figure 11(a), 38 intersection vertices are detected. Although all the intersections are fixed in the first three iterations of our algorithm, this process does incur additional computation.

Table 2 also shows the running time of our system. The time of constructing cage is largely depends on the size of the mesh, in particular the intersection computation between casting ray and mesh faces. Our template model takes the longest time. The muscle mapping times are almost the same, because the MVC computation only relies on the complexity of the cage and the mapped model. The cage in all experiments use have the same density and the same large muscle models.

7 Conclusions and Discussion

In this paper we presented a new muscle retargeting method to help an animator to quickly construct character muscle structures. Based on this detailed muscle model, the animator can produce whole body animation using any existing muscle deformation techniques. Since all stages of the process are completely automatic requiring no user input, this method will largely improve the efficiency of the animation production.

In our current implementation muscle retargetting is only supported for humanoid characters. The skeletal structure of quadrupedal characters has the same topological structure as humanoids, and our method can be easily extended to support these characters.

Currently the distribution of cross-sections still follows regular space sampling. To further increase the fitness of this retargeting, we should adjust the cage to create cross-sections according to important feature points on the new character model. This would give a better fit to the new skin model if, for example, a muscle bulge was located at a different position on the limb, and may improve the retargeted muscle for new character.

In this paper, we limit input surfaces to genus 0 as this is a requirement of the spherical parameterization method used to resolve the cage self–intersection problem. This limitation is satisfactory for most humanoid characters which topologically conform to a tree structure. The topology of the cage is only based on the skeleton structure: it has no direct link with the topology of the skin mesh. Extending our algorithm to robustly deal with loops in the skeleton structure (for example, a torus) is an area we hope to explore in future work.

Acknowledgements The template model is from the Ultimate Human (http://www.cgcharacter.com/ultimatehuman.html), other mesh data used in this paper were made available by 'MakeHuman' (http://www. makehuman. org/blog/index.php).

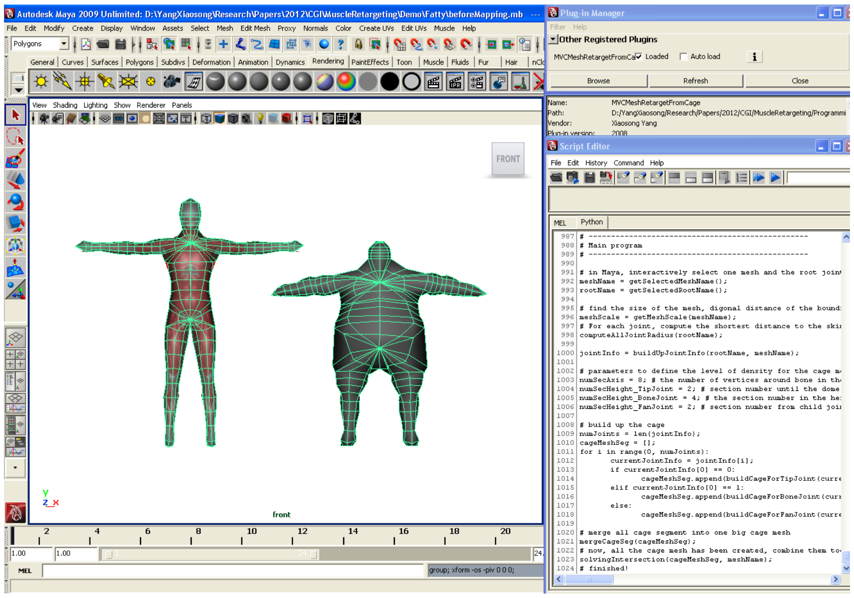
References

1. Ilya Baran and Jovan Popovic. 2007. Automatic rigging and animation of 3D characters. ACM Trans. Graph. 26, 3, 72.
2. Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. 2008. Skeleton extraction by mesh contraction. ACM Trans. Graph. 27, 3, 1-10.
3. Jaewoo Seo, Yeongho Seol, Daehyeon Wi, Younghui Kim, and Junyong Noh. 2010. Rigging transfer. Comput. Animat. Virtual Worlds 21, 3 (May 2010), 375-386
4. Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan, 2008, Geometric skinning with approximate dual quaternion blending. ACM Trans. Graph., 27(4): p. 1-23.
5. Xiaohan Shi, Kun Zhou, Yiying Tong, Mathieu Desbrun, Hujun Bao, and Baining Guo. 2008, Example-based dynamic skinning in real time. ACM Trans. Graph., 27(3): p. 1-8.
6. Tao Ju, Qian-Yi Zhou, Michiel van de Panne, Daniel Cohen-Or, and Ulrich Neumann, 2008, Reusable skinning templates using cage-based deformations. ACM Trans. Graph., 27(5): p. 1-10.
7. Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan, 2007, Skinning with dual quaternions, in Proceedings of the 2007 symposium on Interactive 3D graphics and games. ACM: Seattle, Washington.
8. Sang Il Park and Jessica K. Hodgins, 2008, Data-driven modeling of skin and muscle deformation. ACM Trans. Graph., 27(3): p. 1-6.
9. Min Hong, Sunhwa Jung, Min-Hyung Choi, and Samuel Welch, 2005, Fast volume preservation for realistic muscle deformation, in ACM SIGGRAPH 2005 Sketches, ACM: Los Angeles, California.
10. Xiaosong Yang, Jian Chang, and Jian J. Zhang, 2007, Animating the Human Muscle Structure. Computing in Science and Eng., 9(5): p. 39-45.
11. Robson Lemos, Marcelo Epstein, Walter Herzog, and Brian Wyvill, 2001, Realistic Skeletal Muscle Deformation Using Finite Element Analysis, in Proceedings of the XIV Brazilian Symposium on Computer Graphics and Image Processing. IEEE Computer Society.
12. Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, Stephen F. May, 1997, Anatomy-based modelling of the human musculature, in Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co.

13. Michael Pratscher, Patrick Coleman, Joe Laszlo, and Karan Singh, 2005, Outside-in anatomy based character rigging, in Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation. ACM: Los Angeles, California.
14. Xiaosong Yang and Jian J. Zhang, Automatic muscle generation for character skin deformation. Comput. Animat. Virtual Worlds, 17(3-4): p. 293-303.
15. Thomas W. Sederberg and Scott R. Parry, 1986, Free-form deformation of solid geometric models. SIGGRAPH Comput. Graph., 20(4): p. 151-160.
16. Sabine Coquillart, 1990, Extended free-form deformation: a sculpturing tool for 3D geometric modeling. SIGGRAPH Comput. Graph., 24(4): p. 187-196.
17. Ron MacCracken and Kenneth I. Joy, 1996, Free-form deformations with lattices of arbitrary topology, in Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM.
18. Kazuya G. Kobayashi and Katsutoshi Ootsubo, 2003, t-FFD: free-form deformation by using triangular mesh, in Proceedings of the eighth ACM symposium on Solid modeling and applications. ACM: Seattle, Washington, USA.
19. Michael S. Floater, 2003, Mean value coordinates. Comput. Aided Geom. Des., 20(1): p. 19-27.
20. Tao Ju, Scott Schaefer, and Joe Warren, 2005, Mean value coordinates for closed triangular meshes, ACM Trans. Graph., 24(3): p. 561-566.
21. Michael S. Floater, Geza Kos, and Martin Reimers, 2005, Mean value coordinates in 3D. Comput. Aided Geom. Des., 22(7): p. 623-631.
22. Yaron Lipman, Johannes Kopf, Daniel Cohen-Or, and David Levin, 2007, GPU-assisted positive mean value coordinates for mesh deformations. Proceedings of the fifth Eurographics symposium on Geometry processing, 123. Eurographics Association.
23. Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki, 2007, Harmonic coordinates for character articulation. ACM Trans. Graph., 26(3): p. 71.
24. Mirela Ben-Chen, Ofir Weber, and Craig Gotsman, 2009, Variational harmonic maps for space deformation. ACM Trans. Graph. , 28(3): p. 1-11.
25. Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008, Green Coordinates. ACM Trans. Graph., 27(3): p. 1-10.
26. Lu Chen, Jin Huang, Hanqiu Sun, and Hujun Bao, 2010, Technical Section: Cage-based deformation transfer. Comput. Graph. ,34(2): p. 107-118.
27. Nikita Kojekine , Vladimir Savchenko , Mikhail Senin , Ichiro Hagiwara, 2002, Real-time 3D Deformations by Means of Compactly Supported Radial Basis Functions. in Short papers proceedings of Eurographics.
28. Mario Botsch , Leif Kobbelt, 2005, Real-time shape editing using radial basis functions. Computer Graphics Forum, p. 611-621.
29. Jin Huang, Lu Chen, Xinguo Liu, and Hujun Bao, 2009, Efficient mesh deformation using tetrahedron control mesh. Comput. Aided Geom. Des., 26(6): p. 617-626.
30. Chuhua Xian; Hongwei Lin; Shuming Gao, 2009, Automatic generation of coarse bounding cages from dense meshes. in IEEE International Conference on Shape Modeling and Applications. Beijing.
31. Mirela Ben-Chen, Ofir Weber, and Craig Gotsman, 2009, Spatial deformation transfer, in Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM: New Orleans, Louisiana.
32. Craig Gotsman, Xianfeng Gu, and Alla Sheffer, 2003, Fundamentals of spherical parameterization for 3D meshes. ACM Trans. Graph. 22, 3 (July 2003), 358-363.
33. Shadi Saba, Barycentric Spherical parameterization of genus-0 3D Meshes. [cited 2011 16th Feburary, 2011]; Available from: http://www.cs.technion.ac.il/ shadis/.
34. Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno, 1998, Metro: Measuring Error on Simplified Surfaces. Computer Graphics Forum, 17(2): p. 167-174.

Table 2 The measurement of fitting of our experiments.

model	vertices	faces	size	mean	variance	Time for creating cage (s)	Time for muscle mapping (s)
Template (Figure 9)	14652	15200	28.049			3.7	
Muscular (Figure 11(a))	5314	5312	29.652	0.0631	0.0039	1.76	83.21
Fat (Figure 11(e))	713	1422	25.578	0.1486	0.0109	0.39	89.06
Chubby (Figure 12(a))	5654	5610	43.231	0.1228	0.0094	1.28	85.82

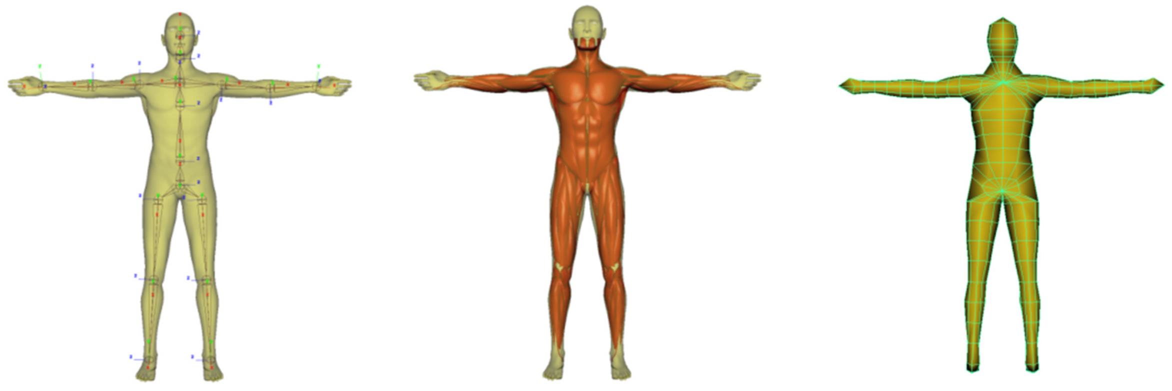

Autodesk Maya 2009 Unlimited: D:\YangXiaosong\Research\Papers\2012\CGI\MuscleRetargeting\Demo\Fatty\beforeMapping.mb ---
File Edit Modify Create Display Window Assets Select Mesh Edit Mesh Proxy Normals Color Create UVs Edit UVs Muscle Help
Polygons
General Curves Surfaces Polygons Subdivs Deformation Animation Dynamics Rendering PaintEffects Toon Muscle Fluids Fur Hair nCl
View Shading Lighting Show Renderer Panels
FRONT
front
MEL
Plug-in Manager
Filter Help
Other Registered Plugins
Browse
Refresh
Close
Script Editor
File Edit History Command Help
MEL
Python

MVCMeshRetargetFromCa ✓ Loaded ☐ Auto load

Name: MVCMeshRetargetFromCage
Path: D:/YangXiaosong/Research/Papers/2012/CGI/MuscleRetargeting/Programmi
Vendor: Xiaosong Yang
Plug-in version: 2008

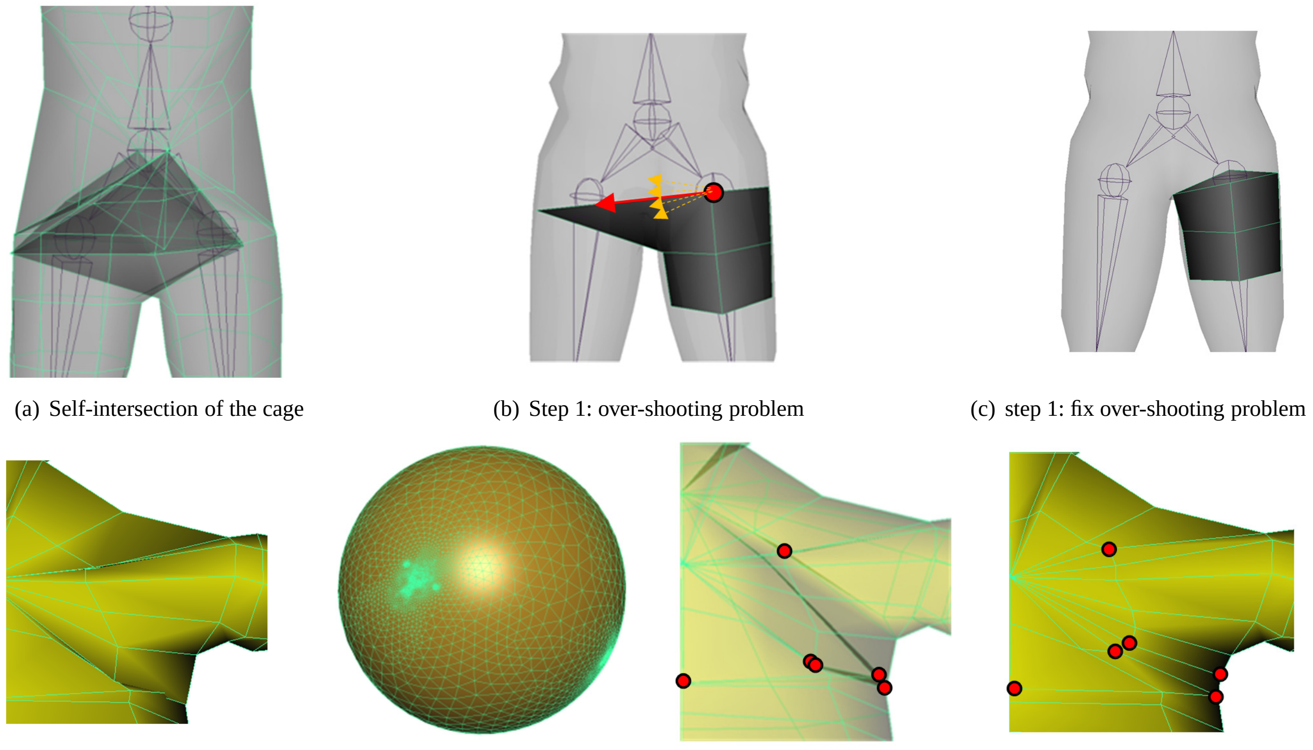
```
# ---------------------------------------------
# Main program
# ---------------------------------------------

# in Maya, interactively select one mesh and the root joint
meshName = getSelectedMeshName();
rootName = getSelectedRootName();

# find the size of the mesh, digonal distance of the bound
meshScale = getMeshScale(meshName);
# For each joint, compute the shortest distance to the ski
computeAllJointRadius(rootName);

jointInfo = buildUpJointInfo(rootName, meshName);

# parameters to define the level of density for the cage m
numSecAxis = 8; # the number of vertices around bone in th
numSecHeight_TipJoint = 2; # section number until the dome
numSecHeight_BoneJoint = 4; # the section number in the he
numSecHeight_FanJoint = 2; # section number from child joi

# build up the cage
numJoints = len(jointInfo);
cageMeshSeg = [];
for i in range(0, numJoints):
        currentJointInfo = jointInfo[i];
        if currentJointInfo[0] == 0:
                cageMeshSeg.append(buildCageForTipJoint(curre
        elif currentJointInfo[0] == 1:
                cageMeshSeg.append(buildCageForBoneJoint(curr
        else:
                cageMeshSeg.append(buildCageForFanJoint(curre

# merge all cage segment into one big cage mesh
mergeCageSeg(cageMeshSeg);
# now, all the cage mesh has been created, combine them to
solvingIntersection(cageMeshSeg, meshName);
# finished!
```

Fig. 8 System interface



(a) Skin and skeleton structure (b) Muscle structure (c) Cage modele

Fig. 9 Template model



(a) Self-intersection of the cage

(b) Step 1: over-shooting problem

(c) step 1: fix over-shooting problem

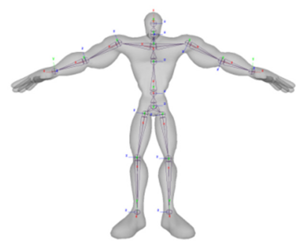
(d) step 2: before fix

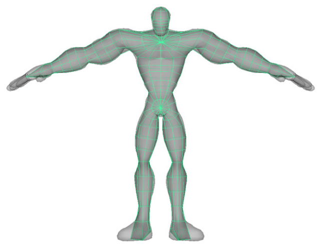
(e) spherical mapping of the skin mesh

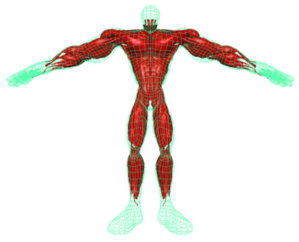
(f) step 2: 6 intersection vertices labelled

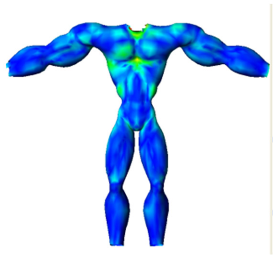
(g) step 2: after fixing with spherical mapping

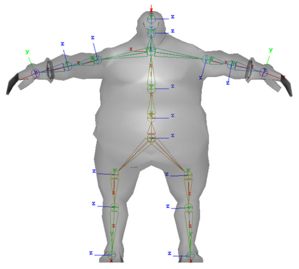
Fig. 10 Solving the self-intersection problem

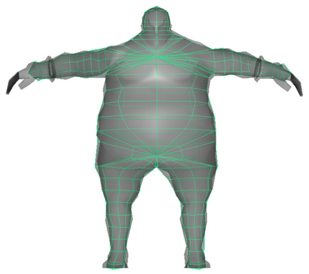

(a) skin and skeleton for a muscular model

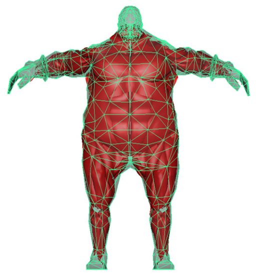

(b) cage structure

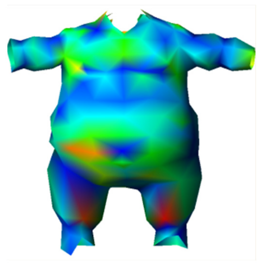

(c) the retargeted muscles

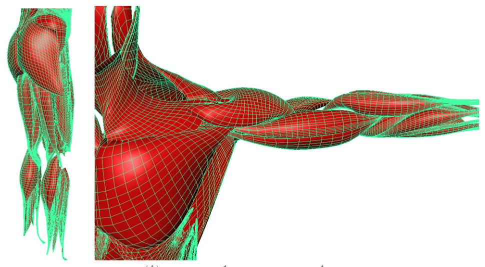

(d) fitness


(e) skin and skeleton for a fat model


(f) cage structure


(g) the retargeted muscles


(h) fitness


(i) template muscle

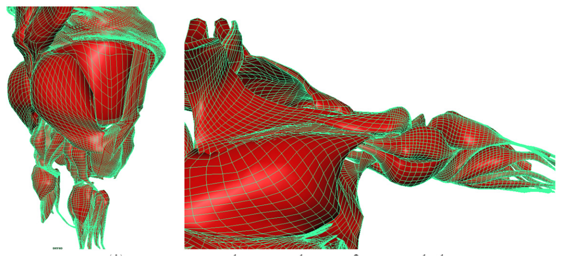
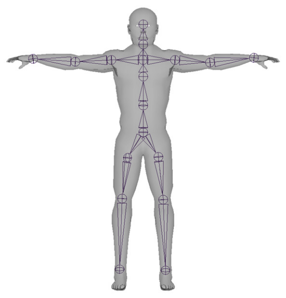
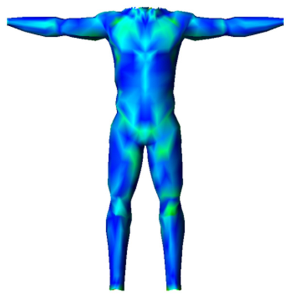

(j) retargeted muscle to fat model

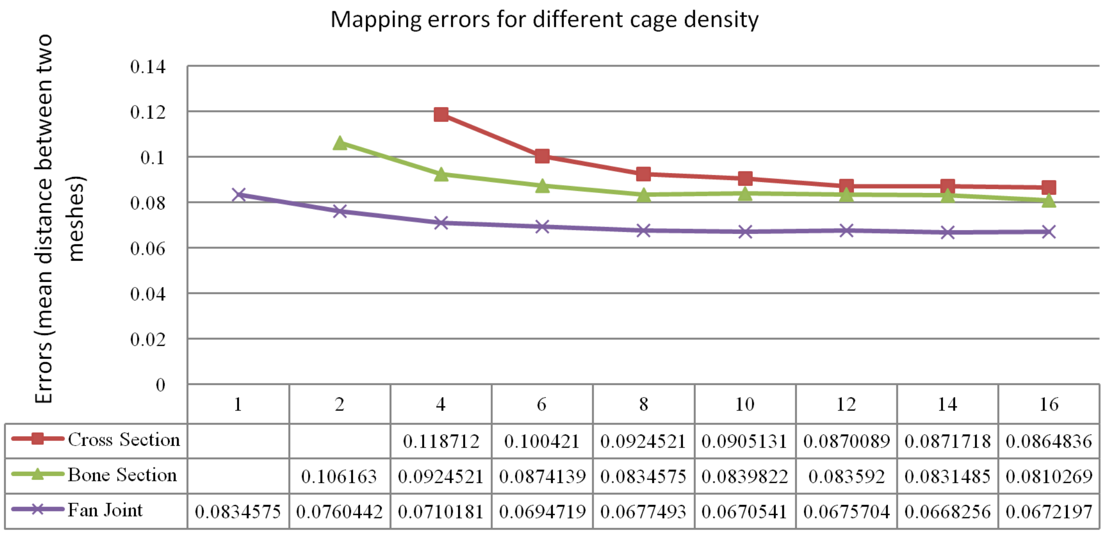
Fig. 11 Two experiments: retarget muscle to a muscular and a fatty model


(a) skin skeleton structure

(b) cage structure

(c) the retargeted muscles


(d) fitness for density (8, 4, 2)



	1	2	4	6	8	10	12	14	16
🟥 Cross Section			0.118712	0.100421	0.0924521	0.0905131	0.0870089	0.0871718	0.0864836
▲ Bone Section		0.106163	0.0924521	0.0874139	0.0834575	0.0839822	0.083592	0.0831485	0.0810269
✕ Fan Joint	0.0834575	0.0760442	0.0710181	0.0694719	0.0677493	0.0670541	0.0675704	0.0668256	0.0672197

Fig. 12 How the cage density affects the mapping fitness