國立交通大學

資訊科學與工程研究所

碩 士 論 文

基於晶格化結構之即時角色皮膚形變技術

Real-Time Lattice-based Character Skinning

研 究 生：陳政浩

指導教授：林奕成　博士

中 華 民 國 九十九 年 七 月

基於晶格化結構之即時角色皮膚形變技術

Real-Time Lattice-based Character Skinning

研 究 生：陳政浩　　Student：Cheng-Hao Chen

指導教授：林奕成　　Advisor：Dr. I-Chen Lin

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

基於晶格化結構之即時角色皮膚形變技術

研究生：陳政浩　指導教授：林奕成　博士

國立交通大學

資訊科學與工程研究所

摘要

本論文提出了一個使體網格(Volumetric Mesh)隨著骨骼動畫(Skeleton-driven Animation)即時形變的方法。一般使表面網格(Surface Mesh)隨著骨骼動畫形變的方式為線性混合皮膚形變技術(Linear Blend Skinning)，但是該方法有著數個嚴重的缺點像是需要大量的人工調整。此外，大部分的皮膚形變技術技巧包括線性混合皮膚形變技術都僅著重於主要形變。我們的方法提供一個能夠模擬高、低頻率體效果(Volumetric Effect)例如肌肉隆起與抖動的體形變模型(Deformable Model)，且僅需少量的人工調整。給定一個表面網格，我們將其體素化(Voxelize)來建立一個由方塊組合而成並且包含表面網格的晶格化結構，並套用修改自線性混合皮膚形變技術的晶格平滑皮膚形變技術法(Lattice-based Smooth Skinning)來產生主要形變。接著則套用其他的形變技巧如晶格形狀比對(Lattice Shape Matching)來產生次要形變(Secondary Deformation)的效果。此外，我們的方法可以輕易地實做於圖形處理器上並高效率地執行。

關鍵字：骨骼動畫，次要形變，體形變，皮膚形變技術，形狀比對

Real-Time Lattice-based Character Skinning

Student: Cheng-Hao Chen   Advisor: Dr. I-Chen Lin

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

In this paper we present an approach to deform volumetric meshes for skeleton-driven animations in real time. The standard solution of skeleton-driven surface mesh deformation, linear blend skinning, has several critical drawbacks such as heavy requirement of artist intervention. Furthermore, most skinning approaches including linear blend skinning focus on primary deformations. Our approach provides a volumetric deformable model that can simulate the dynamics of both low and high frequency volumetric effects such as muscle bulging and vibrations with little user intervention. Given a surface mesh, we voxelize the mesh to construct a lattice of cubic cells containing the mesh and then we apply lattice-based smooth skinning method based on linear blend skinning to drive the primary deformation. Other deformable techniques such as lattice shape matching are then applied to the volumetric mesh to generate secondary deformations and other effects. Moreover, our approach can be easily implemented on GPUs and executed with high efficiency.

Keywords: skeleton-driven animation, secondary deformation, volumetric deformation, skinning, shape match

Acknowledgement

兩年的時光匆匆流逝，記得當初上榜的時候，心中的喜悅難以言喻。在研究所期間之所學，希望將來都能夠學以致用。期許自己未來能蛻變成一位獨立、成熟、有想法的人，懷抱著專業知識及一顆熱忱的心接受社會的洗禮。承蒙指導教授林奕成老師的悉心指導，專業獨到的見解及豐富的歷練，使我這一路走來獲益良多，在此致上萬分謝意。此外，也感激楊傳凱教授與林文杰教授兩位口試委員，在百忙之中撥空參與口試，提供諸多寶貴的意見，亦給予我莫大的助益。

另外也要感謝研究所的同學們，感謝你們在課業上及生活上的照料。研究所的日子因你們而美，這些珍貴的回憶深藏我心。

最後，要特別感謝我的家人們，在物質上給我援助，讓我能無後顧之憂的完成學業，在精神上亦給我莫大的鼓勵，謝謝您們！謹以本論文獻給摯愛的家人和所有關心我的朋友。

陳政浩　謹致於
交通大學資訊科學與工程研究所
中華民國九十九年七月

Table of Content

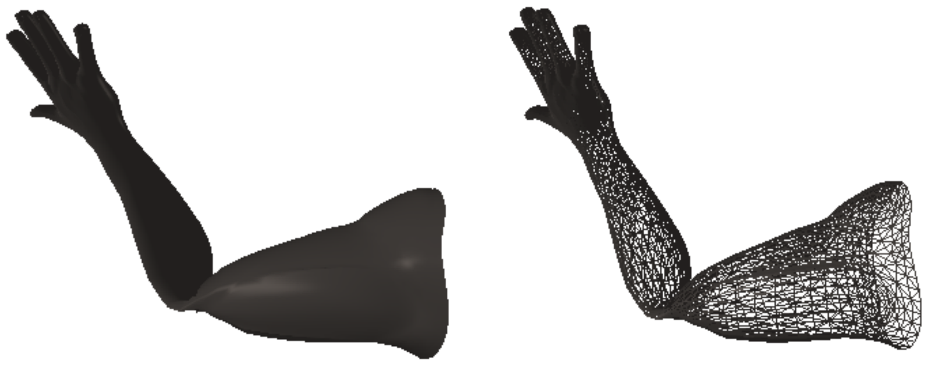
List of Figures

List of Tables

1 Introduction

Skinning and skeleton-driven animation are the technology behind character animation and widely used in many applications like video games. Skinning models define how the geometric shape changes by a function of the skeleton poses and there exists many variations. Skinning can be modeled in a data-driven fashion by using a regression to estimate the shape for a new pose given a set of example pose-and-shape pairs [WPP07]. It can also be modeled procedurally, as in the case of physically-based or anatomy-based approaches. The classic algorithm for skinning is known by many names: Linear Blend Skinning (LBS), Skeletal Subspace Deformation (SSD) [MTLT88]. It is sometimes used not only for skin deformation but also to animate other deformable elements, such as clothes, because it is considerably faster than physically-based cloth simulation [CMT05].

The basic principle of LBS is that transformations of vertices are represented by linearly-blended matrices. This method produces artifacts such as "*candy-wrapper*" effect (Figure 1) in the deformed skin, even if we restrict the skinning transformations to rigid ones. In spite of such shortcomings, linear blending is still the most popular skinning approach because of its simplicity and efficiency.



(a) (b)

Figure 1: A*n example of candy-wrapper effect.*

In other situations, physically accurate skin deformation, which includes muscle bulging, jiggling of fat tissues and other dynamic effects, are desirable. However, many skinning or deformation approaches are often devoid of such secondary deformation effects [OBZH00] due to the small number of degrees of freedom in typical skeletons and the time-consuming manual work required to add physically-realistic secondary deformations.

Our approach begins by applying traditional linear blend skinning method on all particles in our volumetric model to drive the primary deformation. Secondary deformation is then generated by the *lattice shape matching* (LSM) method [RJ07]. We apply rigid shape matching transformations that use regional estimations of rotation and translation for every particle in the cells to provide detailed deformation. In the basic lattice shape matching, increasing the shape matching region size will increase the rigidity, which effectively approximates more rigid bodies. In our case, the shape matching region size is related to the smoothness of mesh. This method provides a simple framework that is robust and capable of generating great secondary deformation effects.

In addition, we define deformable part. A deformable part is a set of particles that are dependent on specific joint or bone. Since some cells' volume may not be preserved during deformation especially the ones near the joints, we apply hierarchical volume preservation through all joint-dependent deformable parts. The volume-preserving algorithm is based on the method proposed by Takamatsu and Kanai [TK09]. Furthermore, the fore-mentioned shape matching regions and deformable parts is automatically computed and could also be user-defined. The mesh could be partially soft and partially rigid with appropriate shape matching regions and mesh parameters defined, and be able to change its material properties dynamically by applying different profiles. It is more physically-realistic than either traditional rigid bodies or soft bodies without complex physics computations. Our approach can be implemented in parallel version, and executed in real-time. Our main contribution includes:

- Lattice-based skinning method based on linear blend skinning with automatic skinning weight computation.
- Enhancement of purely geometric, skeleton-driven animations with physically-realistic secondary deformation.
- A hierarchical volume preservation technique that can reduce

"*candy-wrapper"* effect.

- Our deformable technique is capable of executing in parallel and executed in real-time.

2 Related Work

2.1 Skinning and Mesh Deformation

Skinning techniques are widely used to drive realistic animated characters. Many significant improvements of linear blend skinning are implemented with a variety of compromises between user control, skinning effort, storage requirements, and performance. Pose Space Deformation [LCF00] and Spherical Blend Skinning [KŽ05] are proposed to address well-known artifacts like collapsing joints. Dual Quaternion Skinning [KCŽO07] introduces deformation schemes that develop effective rotation-based interpolations to replace the linear interpolation of frames used by linear blend skinning. Wang et al. [WPP07] proposed a rotational regression method to capture advanced skin deformation such as muscle bulging, and twisting. Instead of directly manipulating surface mesh, Zhou et al. [ZHSL*05] proposed Volumetric Graph Laplacian (VGL) to deform the mesh based on 2D curves, allowing novice users to create pleasing deformations with little effort. Unfortunately, all of above methods focus on the primary deformation of the surface mesh, but do not mention secondary deformations.

Shi et al. [SZTD*08] proposed an example-based approach with surface detail preservation and secondary deformations. However, example-based methods are often inadequate to modern games or animations due to expensive manual works. Von Funck et al. [VFTS07] added elastic secondary deformation to a given primary deformation by the simulation of a small number of user-placed mass-spring sets. To simulate high-frequency dynamics realistically, considerable mass-spring sets need to be tuned correctly. It also requires expensive manual works.

2.2 Alternative Skinning

Other researchers propose to solve skinning artifacts by implementing a different skinning method, for example based on auxiliary curved skeletons [FOKM07]. In some cases, this also allows advanced effects to be animated, such as muscle bulging and skin creasing. The disadvantages include complexity of the GPU implementation and inconsistency with the established skinning pipeline. Another important class of methods is based on generalization of barycentric coordinates [JSW05, JMDG*07, JZVC*08]. This enables the user to deform a character using a simpler mesh. These approaches are mainly appealing in off-line production, where high quality and direct manipulation of the deformations are more important than run-time efficiency.

2.3 Volumetric Deformation

Volumetric deformation can be incorporated with skeleton-driven animations via pre-constructing volumetric elements inside or around the surface mesh, then simulating continuum elasticity in the elements, and finally reconstructing the surface mesh from the volumetric elements through interpolation. Methods in this category include physically-based approaches such as the basic Finite Element Method (FEM) [CGCD*02, MG04], mass-spring systems [BC00, THMG04], and non-physical approaches such as Fast Lattice Shape Matching [RJ07]. Lattice-based shape deformation are widely used to animate embedded geometry [FVT97]; common lattice embeddings such as regular voxel [MG04, JBT04] or body-centered cubic tetrahedral meshes [MBF04] can simplify meshing issues for simulation. Unfortunately, detailed approximations of volumetric deformation are expensive to simulate for many real-time applications. Detailed lattice-based FEM meshes are used in character animation [SNF05], but mostly for offline simulations. Avoiding element recomputation costs is possible using rotated linear element models [MDMJ*05, CGCD*02], but integrating large-deformation dynamics often involves significant costs for algebraic linear system solves.

3 Approach

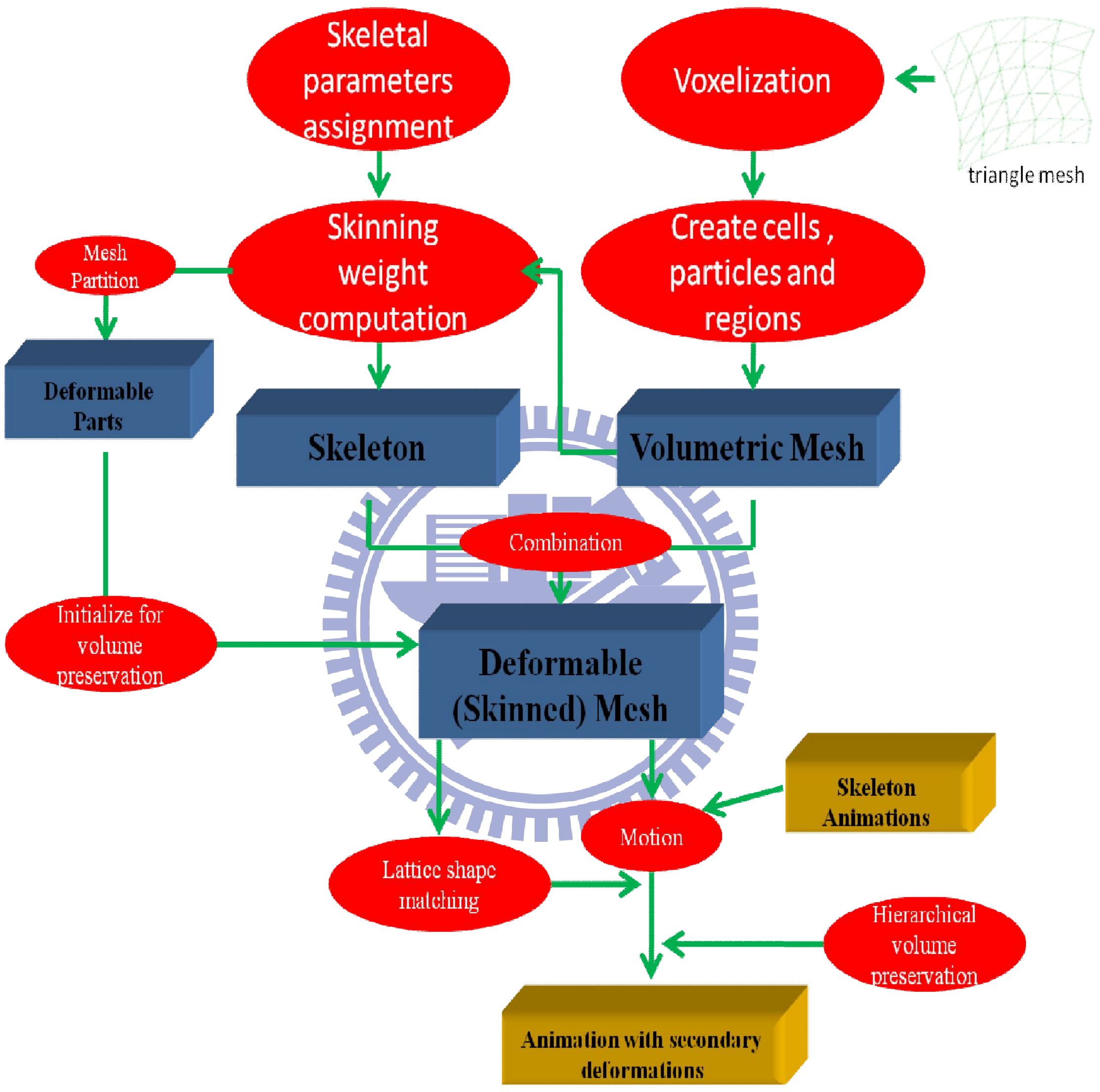
3.1 Overview

The basic linear blend skinning technique plays a critical role in our framework. We apply the linear blend skinning method on particles instead of mesh vertices to drive the mesh. Users can construct a skeleton manually or load an existing skeleton from file. Our method would automatically compute per particle skinning weights through volumetric energy diffusion. After skinning weight computation, we group all particles into several sets called deformable parts based on the joints and bones of the applied skeleton. For each joint-dependent deformable part, we apply hierarchical volume preservation to preserve the volume during deformation. There are many properties such as rigidity, smoothness, and intensity of secondary deformation, which could be adjusted by users. With such properties, physically-realistic effect on the animated character is almost completely generated.

In the next section, we will describe how the lattice shape matching method works in our framework, from the lattice construction stage to the dynamics of the embedded mesh. Given a surface mesh to be deformed, we conservatively voxelize the mesh to construct a lattice of cubic cells containing the mesh. Lattice deformation is controlled by unit point-mass particles placed at the cell corners. The embedded vertices are then deformed using trilinear interpolation of all particles' position. Each particle is associated with a shape matching region comprised of a set of shape matching particles that is given as input to the system. We apply rigid shape matching transforms that use regional estimates of rotation and translation at every particle to provide detailed deformation smoothing without domain boundary artifacts. The lattice shape matching method is used to generate secondary deformation effects because of its elasticity and robustness.

Finally, we will introduce our fast rendering method based on GPU. We organized all particles' data as a linear texture and stored in GPU's global memory, then using a trilinear lattice deformer to reconstruct the surface mesh. Several invariant properties of particle are considered as vertex attributes and stored in vertex buffer. By the computation power of GPUs, we improve the performance drastically with respect to purely CPU-bound computations.

3.2 System Flowchart



Skeletal parameters assignment
Voxelization
triangle mesh
Mesh Partition
Skinning weight computation
Create cells, particles and regions
Deformable Parts
Skeleton
Volumetric Mesh
Combination
Initialize for volume preservation
Deformable (Skinned) Mesh
Skeleton Animations
Motion
Lattice shape matching
Hierarchical volume preservation
Animation with secondary deformations

Figure 2: *System flowchart.*

3.3 Lattice-based Smooth Skinning

We now define the lattice representation of mesh in our framework and present the notations used throughout, and then show how to apply smooth skinning method on the mesh.

3.3.1 Lattice construction

Given a surface mesh to be deformed, we voxelize the mesh to construct a lattice of cubic cells containing the mesh [JBT04]. The surface mesh should be in an appropriate initial pose as Figure 3, since an arbitrary pose may result in inadequate voxelization and possibly fails to build the integration between skeleton and mesh. Figure 4 shows an extreme case of improper initial pose. In such situations, many parts of mesh which should belong to different limbs are connected. All skinning methods will certainly fail in this situation. Moreover, the voxelization level could be assigned by users to generate appropriate result according to the detail of surface meshes.

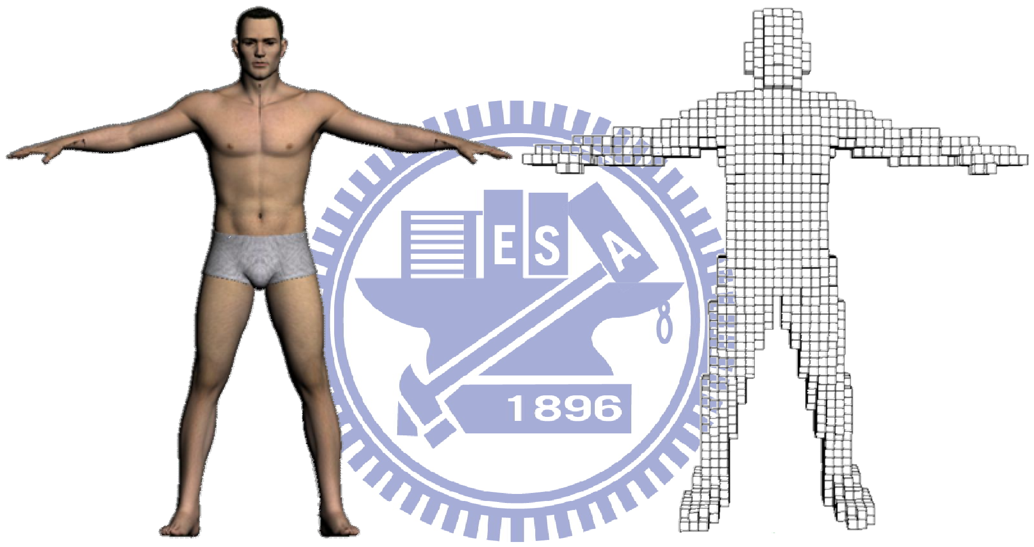


Figure 3: *Appropriate initial pose with reasonable voxelized result.*

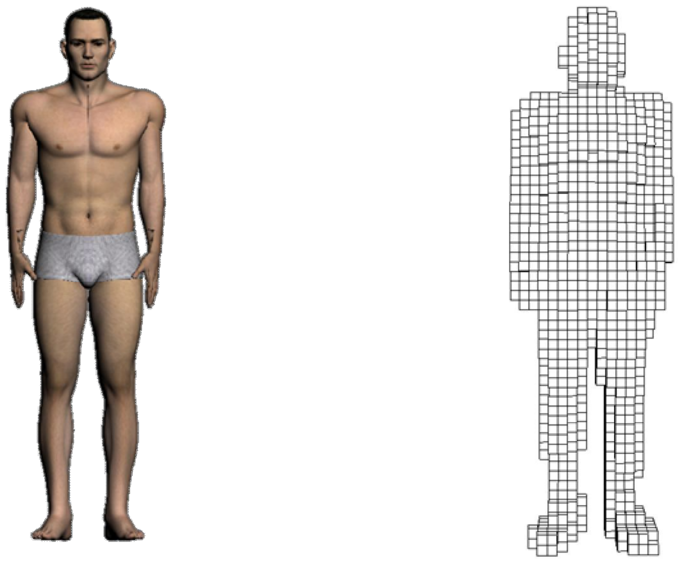


Figure 4: *Inappropriate initial pose and undesirable voxelized result.*

After voxelization, the embedded mesh is then deformed using trilinear interpolation of eight cell vertices positions as shown in Figure 5. Each cell vertex is considered as a particle. Let P denote the set of all particles. For each particle p in P, let its static initial position be $x_p^{\,0}$, its dynamic position be x_p, and its mass be m_p. The dynamics of particles will later be introduced in section 3.4.1. Each particle has its local index represented by a 3-tuple related to a reference particle (usually the particle with index 0). Figure 6 shows an example in 2D case. Let $p_{(\mathrm{x,\ y,\ z})}$ denote a particle with index (x, y, z). We now define the neighbors N_p and adjacencies Adj_p of particle $p_{(\mathrm{x,\ y,\ z})}$:

$$\boldsymbol{N_{p_{(x,y,z)}}} = \{\, \boldsymbol{p_{(a,b,c)}} \mid |\boldsymbol{x}-\boldsymbol{a}| \le \mathbf{1}\,, |\boldsymbol{y}-\boldsymbol{b}| \le \mathbf{1}\,, |\boldsymbol{z}-\boldsymbol{c}| \ \le\ \mathbf{1} \},\ \left|\boldsymbol{N_{p_{(x,y,z)}}}\right| \le \mathbf{26}$$

$$\boldsymbol{Adj_{p_{(x,y,z)}}} = \{\, \boldsymbol{p_{(a\pm 1,b,c)}},\ \boldsymbol{p_{(a,b\pm 1,c)}}, \boldsymbol{p_{(a,b,c\pm 1)}} \},\ \left|\boldsymbol{Adj_{p_{(x,y,z)}}}\right| \le \mathbf{6}$$

That is, the N_p is a set of particles that stay in 3x3x3 cells surrounding p and the Adj_p is a set of particles that have one cell distance away from p. We can know particle-to-particle connectivity easily by comparing the indices, and the connectivity is useful for us to calculate skinning weights (describe later).

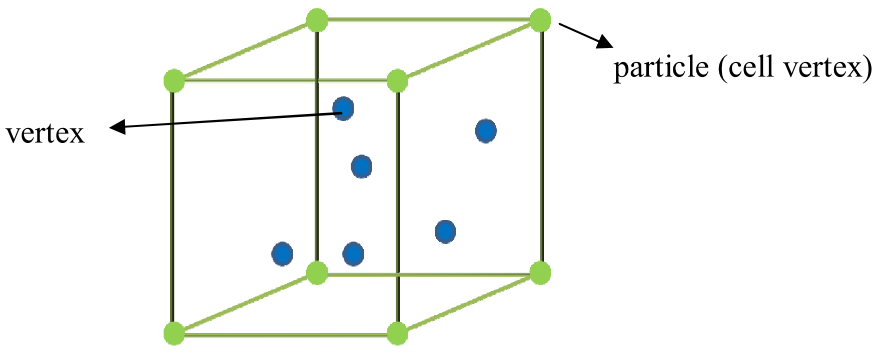


Figure 5: C*ell representation*

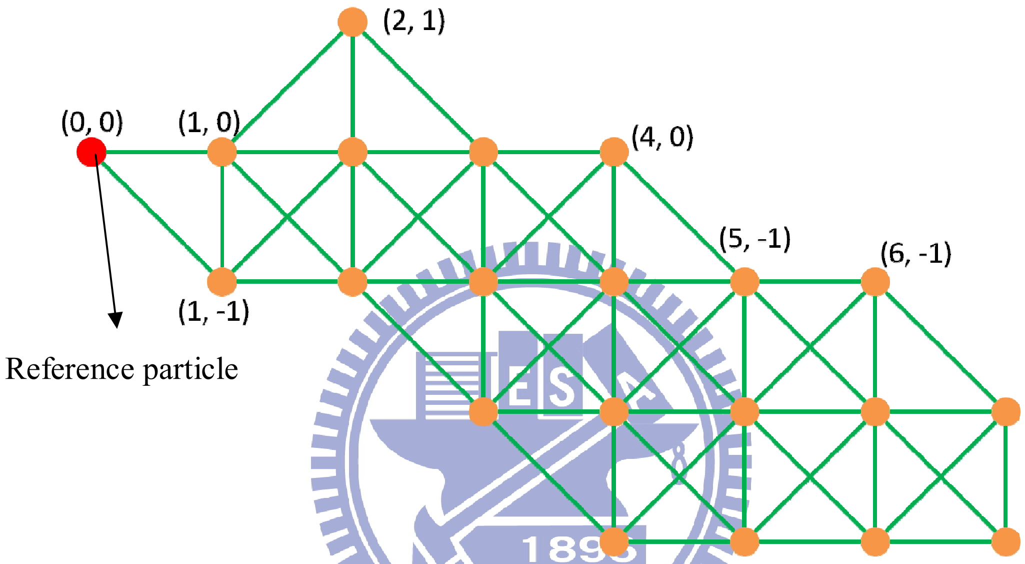


Figure 6: *2D example of particle indexing*

3.3.2 Smooth skinning

In this subsection, we focus on our smooth skinning methods with the voxelized mesh (described in section 3.3.1) adopted from linear blend skinning. A 3D object conforming to this standard consists of skin, a skeleton and vertex weights. The skin is a 3D triangular mesh with no assumed topology or connectivity and the skeleton is a rooted tree. The nodes of the skeleton represent joints and the edges can be interpreted as bones. However, each bone can be easily identified by its attached joint, so the indices of bones are usually considered as indices of joints.

Without loss of generality, transformations of joints in the hierarchy can be assumed to be rigid. In traditional skinning framework [MTLT88], the vertex weights describe the skin-to-skeleton binding (*i.e.*, the amount of influence of individual joints on each vertex). In our case, we consider particle weights instead of vertex weights. Assume that there are k joints in our skeleton. Each joint has an associated local coordinate system in its initial position. The transformation from the initial position of joint $j \in \{ j_1, \ldots, j_k \}$ to its current position could be expressed by a rigid transformation matrix – $T_j \in SE(3)$. We assume that particle p is attached to joints $j_p = \{ j_1, \ldots, j_n \}$ with weights $w_p = \{ w_p^1, \ldots, w_p^n \}$. The indices $j_1, \ldots, j_n$ are integers referring to the joints that influence a given particle; in other words, they are indices into the array of joints. w_p^i represents the influence of joint j_i on particle p. Most skinning applications let n (the number of influencing joints) to be four due to graphics hardware considerations (*e.g. In GLSL, we* store j_p in a *vec4*-typed variable). In addition, the weights are normally assumed to be convex (*i.e.*, $\sum_{i=1}^{n} \boldsymbol{w_i} = \mathbf{1}$ and $\boldsymbol{w_i} \geq 0$). The particle positions x_p in the voxelized mesh deformed by linear blend skinning is then computed as:

$$\boldsymbol{x_p'} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{w_p^i T_{j_i} x_p} \tag{1}$$

That is, transforming particle p by all influencing joint transformations $\boldsymbol{T_{j_i}}$ and taking a weighted average. To gain a better insight into linear blend skinning, we can re-write Equation (1) using the distributivity of matrix-vector multiplication.

$$\boldsymbol{x_p'} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{w_p^i T_{j_i} x_p} = \left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{w_p^i T_{j_i}} \right) \boldsymbol{x_p} \tag{2}$$

The right-hand side of Equation (2) shows that skinning can also be computed by blending the transformations $\boldsymbol{T_{j_i}}$ first and then applying the result to x_p. The transformation blending used in the right hand side of Equation (2) is a direct linear combination of matrices. It is well known that this method is troublesome, because the blended matrix $\sum_{i=1}^{n} \boldsymbol{w_p^i T_{j_i}}$ is not guaranteed to be a rigid transformation, even if all $\boldsymbol{T_{j_i}}$ are rigid (*i.e.*, the linear combination of orthonormal matrices is not necessarily to be orthonormal). This is a well-known linear blend skinning problems – undesired scaling in our transformation matrix. In the extreme case, the blended matrix can even become rank-lacking.

An obvious idea to improve skinning quality is to replace the linearly-blended matrices in Equation (2) with a more sophisticated matrix blending method. Particularly, if we can always produce a rigid transformation, we can expect that no candy-wrapper-like artifacts will occur. To overcome this problem, these transformations $\boldsymbol{T}_{j_i}$ are factorized into rotation $\boldsymbol{R}_{j_i}$ and scale/shear $\boldsymbol{S}_{j_i}$ components by using the polar decomposition [GVL96].

$$\boldsymbol{T}_{j_i} = \boldsymbol{R}_{j_i}\boldsymbol{S}_{j_i} \tag{3}$$

We use cyclic Jacobi iterations to diagonalize $\boldsymbol{T}^{\boldsymbol{T}}\boldsymbol{T} = \boldsymbol{S}^{2} = \boldsymbol{V}\boldsymbol{D}(\boldsymbol{\lambda})\boldsymbol{V}^{\boldsymbol{T}}$ and construct $\boldsymbol{R} = \boldsymbol{T}\boldsymbol{S}^{-1}$. However, doing this natively is inefficient. To solve this limitation, we use the fast polar decomposition technique described in [RJ07]. We store each particle's stretch eigenvectors $\overline{\boldsymbol{V}}$, initialize with $\boldsymbol{V} = \overline{\boldsymbol{V}}$ and then begin Jacobi iterations on $\overline{\mathbf{V}}^{\mathbf{T}}\mathbf{S}^{2}\overline{\mathbf{V}}$ to solve $\boldsymbol{R}$. Finally we build a new transformation $\dot{\boldsymbol{T}}_{j_i}$ by $\boldsymbol{R}_{j_i}$. The Equation (2) can be re-wrote as:

$$\overline{\boldsymbol{x}}_{\boldsymbol{p}} = \left(\sum_{i=1}^{n} \boldsymbol{w}_{\boldsymbol{p}}^{i}\dot{\boldsymbol{T}}_{j_i}\right)\boldsymbol{x}_{\boldsymbol{p}} \tag{4}$$

to ensure that the transformation of all particles are rigid.

3.3.3 Particle skinning weight assignment

Skinning weight assignment techniques fall into two categories those requiring the artists to specify some parameters, such as bone size or a region of influence for a joint, and those using only the mesh surface and bone positions automatically. A recent technique proposed as an automatic algorithm for unguided skeleton mesh called *bone heat* [BP07]. This method aims to model weight assignment as a heat diffusion system on the surface of the mesh; for each bone, the vertices which have that bone as their nearest visible bone are initialized to have a surface temperature inversely proportional to the square of their closest distance to the bone. A diffusion equation is then solved to obtain weight assignment. Our method uses a hybrid approach with the concept of *bone heat* and user-specified mechanism.

The heat diffusion is much easier to be computed through volumetric mesh than surface mesh. Firstly, we treat each bone *j* as a heat source with energy e_j influenced by some user-defined parameters such as bone width, bone length. For each heat source *j*, we compute the directly-influenced particles $\widehat{\boldsymbol{P}}_j$ which are the closest particles to the bone *j* by no more than one cell size. The energy of particles in $\widehat{\boldsymbol{P}}_j$ are assumed to be e_j. Then we construct an undirected simple graph $\boldsymbol{G}$:

$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}), \boldsymbol{V} = \boldsymbol{P}, \mathbf{E} = \left\{(\mathbf{p_1}, \mathbf{p_2}) \middle| \mathbf{p_1} \in \boldsymbol{P}, \mathbf{p_2} \in \boldsymbol{P}, \mathbf{p_2} \in \mathbf{N_{p_1}} \right\}$$

Each particle is considered as a node in $\boldsymbol{G}$, and having edges with its neighbors. Let $\boldsymbol{cost}(\boldsymbol{p_i}, \boldsymbol{p_j})$ denote the cost of edge $(\boldsymbol{p_i}, \boldsymbol{p_j})$, $\boldsymbol{p_i}$ has an index $(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{z_i})$, and $\boldsymbol{p_j}$ has an index $(\boldsymbol{x_j}, \boldsymbol{y_j}, \boldsymbol{z_j})$. The edge cost is

$$\boldsymbol{cost}(\boldsymbol{p_i}, \boldsymbol{p_j}) = \sqrt{(\boldsymbol{x_i} - \boldsymbol{x_j})^2 + (\boldsymbol{y_i} - \boldsymbol{y_j})^2 + (\boldsymbol{z_i} - \boldsymbol{z_j})^2} \tag{5}$$

Figure 7 shows a 2D example graph with edge cost. The cost indicates the decreasing amount of energy transfer from a particle to another. When we apply heat diffusion from the directly-influenced particles $\widehat{\boldsymbol{P}}_j$ to all other particles, a particle's energy is a weighted average of its neighbors'. We compute the energy summation weights by the edge cost. The neighbors' energy summation weights of particle p_i is denoted by $\boldsymbol{ew_{p_i}}$, and computed as:

$$\begin{gathered} \boldsymbol{ew_{p_i}} = \left\{\boldsymbol{ew_{(p_i,p_j)}} \middle| \boldsymbol{p_j} \in \boldsymbol{N_{p_i}}\right\} \\ \boldsymbol{ew_{(p_i,p_j)}} = \frac{\boldsymbol{wt}(\boldsymbol{p_i}, \boldsymbol{p_j})}{\sum_{\boldsymbol{p_k} \in \boldsymbol{N_{p_i}}} \boldsymbol{wt}(\boldsymbol{p_i}, \boldsymbol{p_k})} \\ \boldsymbol{wt}(\boldsymbol{p_i}, \boldsymbol{p_j}) = \frac{\boldsymbol{b_i}}{\boldsymbol{cost}(\boldsymbol{p_i}, \boldsymbol{p_j}) + \mathbf{1}} \end{gathered} \tag{6}$$

, where $\boldsymbol{b_i}$ is bone strength that influence the energy attenuation. Hence, a particle's energy is then computed as:

$$\boldsymbol{e_{p_i}} = \sum_{\boldsymbol{p_k \in N_{p_i}}} \boldsymbol{ew_{(p_i, p_k)} e_{p_k}} \tag{7}$$

The diffusion runs repeatedly until the completion of diffusion process. To obtain the particle weights, we compare the energy from all heat sources for each particle and then normalize them to ensure the weights are convex.

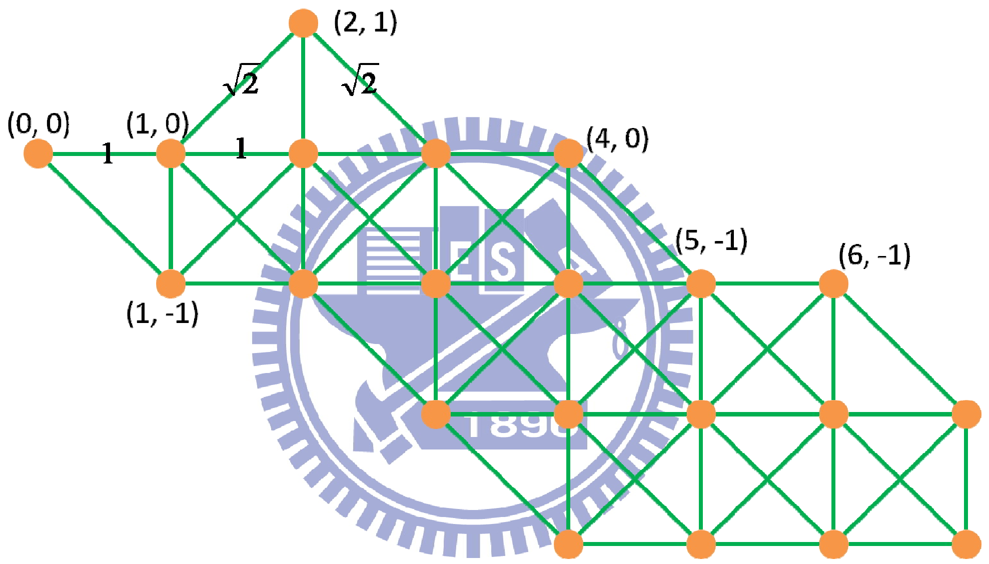


Figure 7: *An example of undirected graph with cost in 2D case.*

After the heat diffusion done, we partition all particles into several deformable parts based on the energy distribution as in Figure 8. *i.e.*, if particle *p* receives the most energy from bone *j*, then particle *p* belongs to bone-dependent deformable part $\boldsymbol{DP_{bone_j}}$. For each bone-dependent deformable part $\boldsymbol{DP_{bone_j}}$, we divide the particles equally based on two joints of bone *j* to build joint-dependent deformable parts like Figure 9:

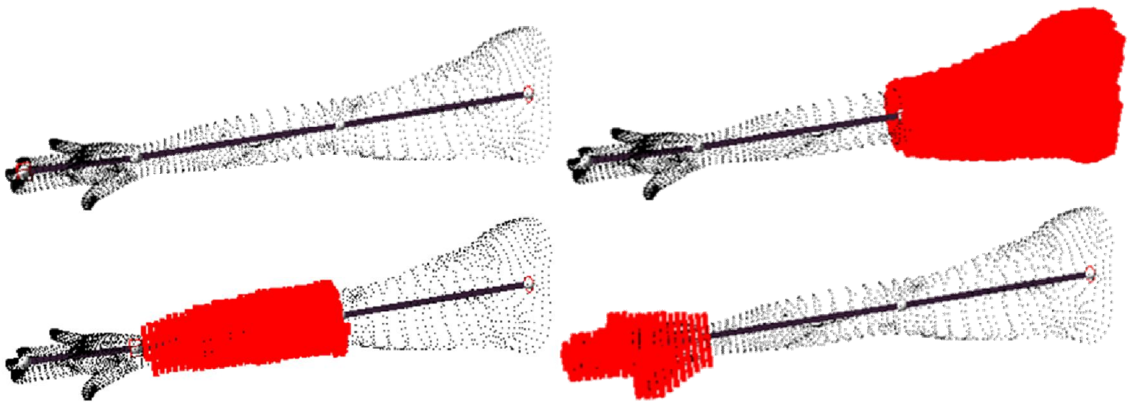


Figure 8: *Red parts are sets of particles called bone-dependent deformable parts.*

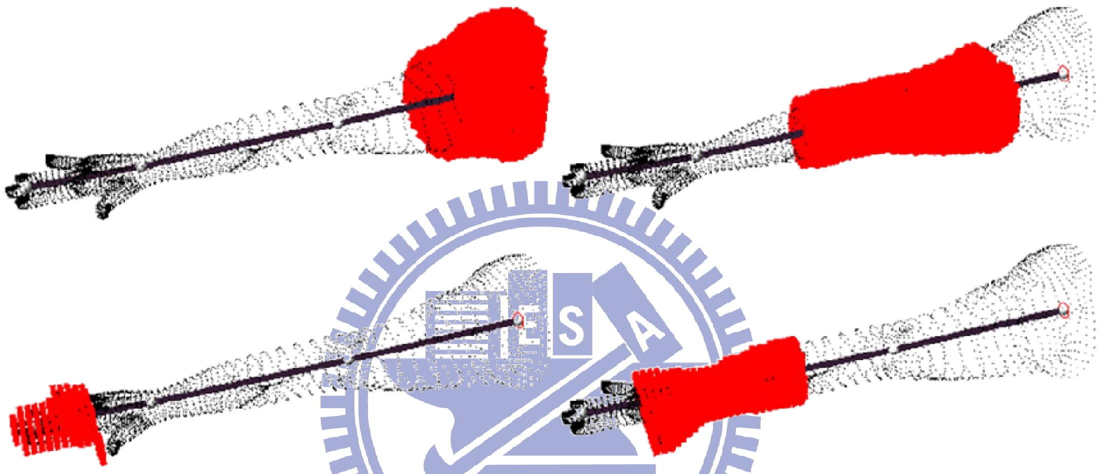


Figure 9: *Red parts are sets of particles called joint-dependent deformable parts.*

The joint deformable parts are basic units for hierarchical volume preservation (describe later). Since deformable parts may contain undesired particles, users can modify undesired particles from one deformable part to another.

Most prior automatic skinning weight algorithms such as *bone heat* apply heat diffusion on surface mesh instead of volumetric mesh. However, heat diffusion usually leads to inaccurate results while diffusing on surface only. Our approach runs heat diffusion on a volumetric mesh through a graph and generates more accurate and more reasonable skinning weights. In fact, our automatic skinning weights computation requires almost no user interventions. Figure 10 presents a skinned human model using our lattice-based smooth skinning method. Since the concept of our lattice-based skinning approach is similar to the basic linear blend skinning, the performance of our method is almost as efficient as linear blend skinning since the number of particles is less than vertices.

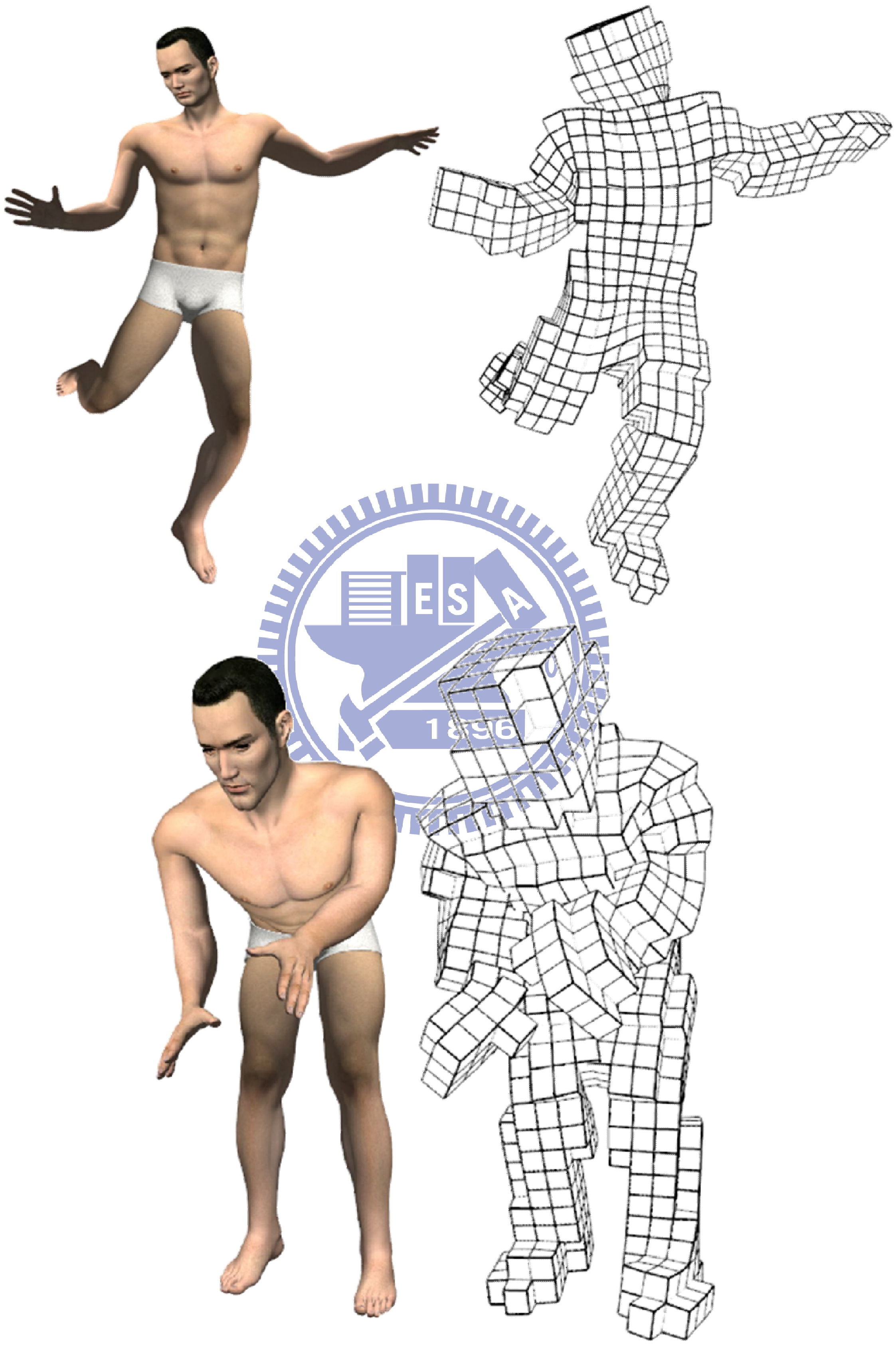


Figure 10: *Lattice-based smooth skinning.*

3.3.4 Volume preservation

Traditional linear blend skinning has some deformation artifacts such as "*candy-wrapper*" effect since it does not address the unnatural volume change. Volume preservation is an important aspect in many application areas, especially in the context of shape deformation. We present a hierarchical approach extended from the method proposed by Takamatsu and Kanai [TK09]. First, we define:

$$\boldsymbol{diff_p} = \left\{ \frac{1}{2}\left(\boldsymbol{x_q^0} - \boldsymbol{x_p^0}\right) \middle| \boldsymbol{q} \in \boldsymbol{Adj_p} \right\} = \left\{ \boldsymbol{d_p^{x+}}, \boldsymbol{d_p^{x-}}, \boldsymbol{d_p^{y+}}, \boldsymbol{d_p^{y-}}, \boldsymbol{d_p^{z+}}, \boldsymbol{d_p^{z-}} \right\}$$

, where $\boldsymbol{diff_p}$ denotes the set of half distance from $\boldsymbol{p}$ to its six adjacencies. Then we could define the volume of each particle p:

$$\begin{aligned} \boldsymbol{Vol(p)} = {} & \boldsymbol{d_p^{x+}} \cdot \left(\boldsymbol{d_p^{y+}} \times \boldsymbol{d_p^{z+}}\right) + \boldsymbol{d_p^{x+}} \cdot \left(\boldsymbol{d_p^{y-}} \times \boldsymbol{d_p^{z-}}\right) \\ & + \boldsymbol{d_p^{x-}} \cdot \left(\boldsymbol{d_p^{y+}} \times \boldsymbol{d_p^{z-}}\right) + \boldsymbol{d_p^{x-}} \cdot \left(\boldsymbol{d_p^{y-}} \times \boldsymbol{d_p^{z+}}\right) \\ & + \boldsymbol{d_p^{z+}} \cdot \left(\boldsymbol{d_p^{y+}} \times \boldsymbol{d_p^{x-}}\right) + \boldsymbol{d_p^{z+}} \cdot \left(\boldsymbol{d_p^{y-}} \times \boldsymbol{d_p^{x+}}\right) \\ & + \boldsymbol{d_p^{z-}} \cdot \left(\boldsymbol{d_p^{y+}} \times \boldsymbol{d_p^{x+}}\right) + \boldsymbol{d_p^{z-}} \cdot \left(\boldsymbol{d_p^{y-}} \times \boldsymbol{d_p^{x-}}\right) \end{aligned} \tag{8}$$

The operator $\cdot$ means dot product and the $\times$ means cross product. We re-write Equation (8) as $\boldsymbol{Vol(p)} = \sum_p \boldsymbol{d_p^x} \cdot \left(\boldsymbol{d_p^y} \times \boldsymbol{d_p^z}\right)$ for convenience. The volume of mesh is then computed as:

$$\boldsymbol{Vol(P)} = \sum_{\boldsymbol{p} \in \boldsymbol{P}} \boldsymbol{Vol(p)} \tag{9}$$

After mesh deformation, all particles $\boldsymbol{P}$ transform to their new positions. Let $\boldsymbol{P}'$ be the deformed particles. We define the particle displacement field $\widehat{\boldsymbol{V}}$:

$$\widehat{\boldsymbol{V}} = \left\{ \widehat{\boldsymbol{v}}_1, \ldots, \widehat{\boldsymbol{v}}_{|\boldsymbol{P}|} \right\} = \left\{ \boldsymbol{s_1 u_1 R_1}, \ldots, \boldsymbol{s_{|P|} u_{|P|} R_{|P|}} \right\}$$

where $\left\{\boldsymbol{R_1}, \ldots, \boldsymbol{R_{|P|}}\right\}$ is a set of particle's rotation. $\left\{\boldsymbol{s_1}, \ldots, \boldsymbol{s_{|P|}}\right\}$ is a set of particle's volume correction scale. For each particle $\boldsymbol{p}$, we define distance to boundary $\boldsymbol{dist_p}$. A boundary particle $\boldsymbol{p}$ means that $|\boldsymbol{N_p}| < \mathbf{26}$ and $\boldsymbol{dist_p} = \mathbf{1}$. For each non-boundary particle $\boldsymbol{q}$, the distance to boundary is computed by:

$$\boldsymbol{dist_q} = \boldsymbol{min}\{\boldsymbol{dist_i}, \boldsymbol{i} \in \boldsymbol{Adj_q}\} + \mathbf{1} \tag{10}$$

The maximum distance to boundary is defined as $\boldsymbol{dist_{max}} = \boldsymbol{max}\{\boldsymbol{dist_i}, \boldsymbol{i} \in \boldsymbol{P}\}$. Since the deformation usually changes largely on surface than in internal region, we determine a volume correction scale factor $\boldsymbol{s_p}$ for each particle $\boldsymbol{p}$ by the distance to boundary from Equation (10):

$$\boldsymbol{s_p} = \mathbf{1} - \frac{\mathbf{1}}{\boldsymbol{dist_{max}}}(\boldsymbol{dist_p} - \mathbf{1}) \tag{11}$$

$\{\boldsymbol{u_1}, \dots, \boldsymbol{u_{|P|}}\}$ is a set of particles' outward vectors. Takamatsu and Kanai [TK09] shows eight patterns for computing outward vectors, which is determined by the existence of adjacent particles as Figure 11:

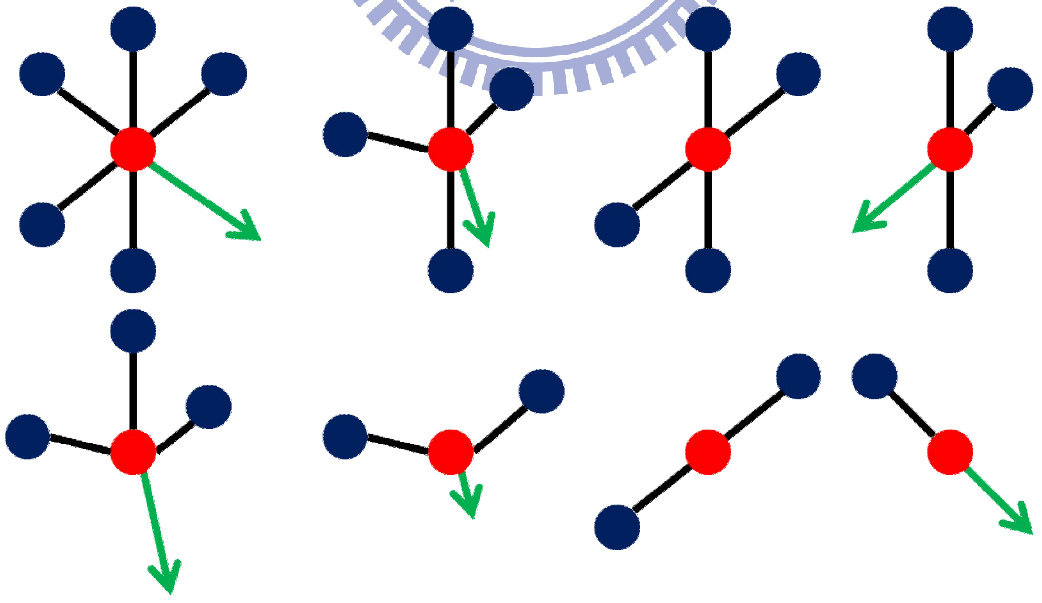


Figure 11: *Eight patterns of outward vectors* (*Green arrows*).

Figure 11 shows how to define outward vectors for boundary particles. For the non-boundary particles and the boundary particles with zero outward vectors, we propagate the known outward vectors outside-in through "small-distance-to-boundary-first" ordering. The outward vector of non-boundary particle $\boldsymbol{q}$ is the average of adjacencies' which have equal or smaller distance to boundary.

With the displacement field calculated, now we can compute the desired mesh volume by the deformed mesh volume $\boldsymbol{Vol(P')}$. Since the volume should be preserved, we can calculate per-particle volume correction vector through a equation:

$$\boldsymbol{Vol(P) = Vol\left(P' + \lambda\hat{V}\right)}$$

(12)

where $\boldsymbol{\lambda\hat{V}}$ is a set of per-particle volume correction vectors. To solve $\boldsymbol{\lambda}$, we define $\boldsymbol{mdiff_p}$ and $\boldsymbol{ndiff_p}$ according to Equation (12):

$$\begin{aligned} \boldsymbol{mdiff_p} &= \left\{\frac{1}{2}\left(x'_q - x'_p\right) \middle| q \in Adj_p\right\} = \left\{m_p^{x+}, m_p^{x-}, m_p^{y+}, m_p^{y-}, m_p^{z+}, m_p^{z-}\right\} \\ \boldsymbol{ndiff_p} &= \left\{\frac{1}{2}\left(\hat{v}_q - \hat{v}_p\right) \middle| q \in Adj_p\right\} = \left\{n_p^{x+}, n_p^{x-}, n_p^{y+}, n_p^{y-}, n_p^{z+}, n_p^{z-}\right\} \end{aligned}$$

and then we can re-write $\boldsymbol{diff_p}$ to:

$$\boldsymbol{diff_p} = \begin{Bmatrix} m_p^{x+} + \lambda n_p^{x+}, m_p^{x-} + \lambda n_p^{x-}, \\ m_p^{y+} + \lambda n_p^{y+}, m_p^{y-} + \lambda n_p^{y-}, \\ m_p^{z+} + \lambda n_p^{z+}, m_p^{z-} + \lambda n_p^{z-} \end{Bmatrix}$$

(13)

Now we use Equation (13) to unfold Equation (12) as following:

$$\boldsymbol{Vol\left(P' + \lambda\hat{V}\right) = \sum_{p \in P'} Vol\left(p' + \lambda\hat{v}_p\right) = \sum_{p \in P'} \left(a_p^0 + a_p^1\lambda + a_p^2\lambda^2 + a_p^3\lambda^3\right)}$$

(14)

where

$$\boldsymbol{a_p^0} = \sum_{p} \boldsymbol{m_p^x} \cdot \left(\boldsymbol{m_p^y} \times \boldsymbol{m_p^z}\right)$$

$$\boldsymbol{a_p^1} = \sum_{p} \boldsymbol{n_p^x} \cdot \left(\boldsymbol{m_p^y} \times \boldsymbol{m_p^z}\right) + \sum_{p} \boldsymbol{m_p^x} \cdot \left(\boldsymbol{n_p^y} \times \boldsymbol{m_p^z}\right) + \sum_{p} \boldsymbol{m_p^x} \cdot \left(\boldsymbol{m_p^y} \times \boldsymbol{n_p^z}\right)$$

$$\boldsymbol{a_p^2} = \sum_{p} \boldsymbol{m_p^x} \cdot \left(\boldsymbol{n_p^y} \times \boldsymbol{n_p^z}\right) + \sum_{p} \boldsymbol{n_p^x} \cdot \left(\boldsymbol{m_p^y} \times \boldsymbol{n_p^z}\right) + \sum_{p} \boldsymbol{n_p^x} \cdot \left(\boldsymbol{n_p^y} \times m_p^z\right)$$

$$\boldsymbol{a_p^3} = \sum_{p} \boldsymbol{n_p^x} \cdot \left(\boldsymbol{n_p^y} \times \boldsymbol{n_p^z}\right) \tag{15}$$

Finally, we obtain a cubic equation with coefficients like:

$$\boldsymbol{c_p^0} = \left(\sum_{p \in P} \boldsymbol{a_p^0}\right) - \boldsymbol{Vol(P)}$$

$$\boldsymbol{c_p^1} = \sum_{p \in P} \boldsymbol{a_p^1}$$

$$\boldsymbol{c_p^2} = \sum_{p \in P} \boldsymbol{a_p^2}$$

$$\boldsymbol{c_p^3} = \sum_{p \in P} \boldsymbol{a_p^3} \tag{16}$$

and then we can solve $\boldsymbol{\lambda}$ to obtain the per-particle volume correction vectors. Since we expect that the variation of mesh volume should be as small as possible, we choose the real solution with the minimum absolute value as our solution.

Once we determined how the particles correct their positions, we apply volume preservation on each joint-dependent deformable part instead of the whole mesh. The volume preservation method [TK09] is a global volume preservation technique. Our approach provides a hierarchical version by applying this method from root joint-dependent deformable part to its all sub-parts. The part hierarchy is built from joint hierarchy of the applied skeleton. Applying this method hierarchically will let it become a local volume preservation method.

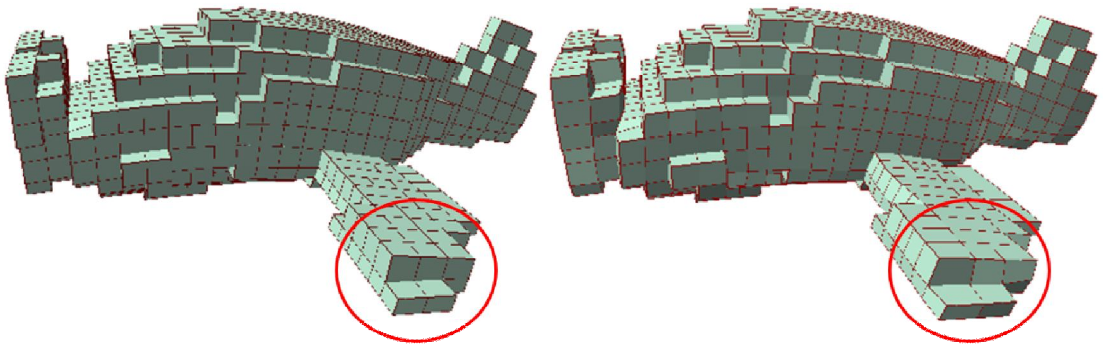


Figure 12: *A mesh pushed by a downward external force. (left) without volume preservation. (right) with volume preservation.*

3.4 Skinning with Secondary Deformation

In this section, we introduce how we combine our lattice-based skinning method and the lattice shape matching. We apply lattice shape matching to generate secondary deformation and lead the body part to be soft and smooth if there is no skeleton binding.

3.4.1 Lattice shape matching

In section 3.3.1, we construct a lattice of cubic cells containing the mesh and create the neighbors N_p for each particle p. Now we define shape matching region for each particle. Each particle p is associated with a shape matching region comprised of a set of shape matching particles, $\boldsymbol{Region_p}$, which for half-width $\hat{\boldsymbol{w}}$ contains p and all particles reachable by traversing not more than $\hat{\boldsymbol{w}}$ neighbors from particle p; *e.g.*, if $\hat{\boldsymbol{w}} = \mathbf{1}$ then $\boldsymbol{Region_p} = \boldsymbol{N_p}$. Here $\hat{\boldsymbol{w}}$ is the region half-width that is given as input to the system. This definition of $\boldsymbol{Region_p}$ allows irregular shape matching region size. In the common case where particles are maximally connected, regions will be cubes with length $\mathbf{2}\hat{\boldsymbol{w}} + \mathbf{1}$. Note that shape matching sets need not be unique, and $\boldsymbol{Region_p} = \boldsymbol{Region_q}$ for $\boldsymbol{p} \neq \boldsymbol{q}$ is common at boundaries. Finally, for each particle p, the set of indices of all shape matching regions to which p belongs is equivalent to $\boldsymbol{Region_p}$. Figure 13 shows an 2D example for shape matching regions with width $\boldsymbol{\hat{w} = 2}$.

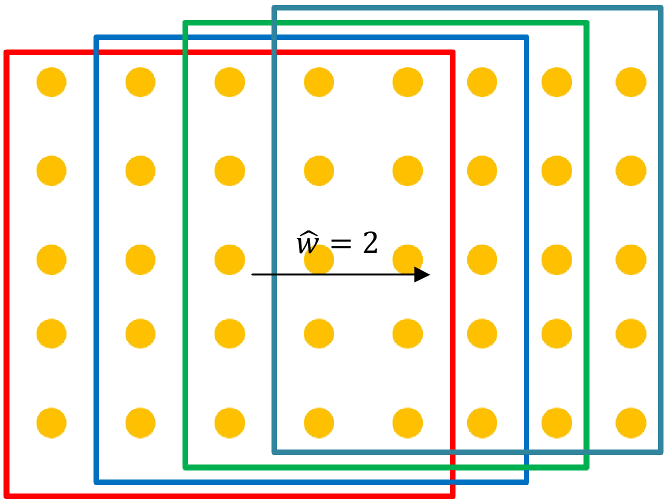


Figure 13: *Each colored square means a region width* $\boldsymbol{\hat{w} = 2}$.

The main lattice shape matching algorithm is proposed by Rivers and James [RJ07]. At each time step, each $\boldsymbol{Region_r}$ finds the best rigid transformation to match the initial particle positions $\boldsymbol{x_p^0, p \epsilon Region_r}$ to their deformed positions $\boldsymbol{x_p, p \epsilon Region_r}$, and then we determine a per-region least-squares rotation $\boldsymbol{\hat{R}_r}$ and translation $\boldsymbol{\hat{t}_r}$ of the rest positions, $\boldsymbol{\tilde{T}_r} = [\boldsymbol{\hat{R}_r} \ \ \boldsymbol{\hat{t}_r}]$. Now we describe calculations for each $\boldsymbol{Region_r}$'s center of mass $\boldsymbol{C_r}$ and least-squares rotation $\boldsymbol{\hat{R}_r}$, which determines the optimal rigid transformation $\boldsymbol{\tilde{T}_r}$, and each particle p's goal position $\boldsymbol{g_p}$. We obtain the deformed center of mass for each $\boldsymbol{Region_r}$ as follows:

$$\boldsymbol{C_r} = \frac{\boldsymbol{1}}{\boldsymbol{M_r}} \sum_{\boldsymbol{p \in Region_r}} \boldsymbol{\tilde{m}_p x_p} \tag{17}$$

where

$$\boldsymbol{M_r} = \sum_{\boldsymbol{p \in Region_r}} \boldsymbol{\tilde{m}_p} \tag{18}$$

is the precomputed effective region mass and

$$\tilde{\boldsymbol{m}}_{\boldsymbol{p}} = \frac{\boldsymbol{m}_{\boldsymbol{p}}}{|\boldsymbol{Region}_{\boldsymbol{p}}|}$$

(19)

is the modified mass of particle *p* which ensures that particles belonging to many regions are not weighted more than others. We estimate the least squares rotation $\hat{\boldsymbol{R}}_{\boldsymbol{r}}$ using the rotational part of

$$\boldsymbol{A}_{\boldsymbol{r}} \equiv \sum_{p \in \boldsymbol{Region}_{\boldsymbol{r}}} \tilde{\boldsymbol{m}}_{\boldsymbol{p}} \left(\boldsymbol{x}_{\boldsymbol{p}} - \boldsymbol{C}_{\boldsymbol{r}}\right)\left(\boldsymbol{x}_{\boldsymbol{p}}^{\boldsymbol{0}} - \boldsymbol{C}_{\boldsymbol{r}}^{\boldsymbol{0}}\right)^{\mathrm{T}}$$

(20)

, where $\boldsymbol{C}_{\boldsymbol{r}}^{\boldsymbol{0}}$ is computed as Equation (17) by replacing $\boldsymbol{x}_{\boldsymbol{p}}$ by $\boldsymbol{x}_{\boldsymbol{p}}^{\boldsymbol{0}}$. We obtain the rotational component $\hat{\boldsymbol{R}}_{\boldsymbol{r}}$ via polar decomposition that is similar to Equation (3). Finally, each region's least square transformation from the rest position $\boldsymbol{x}_{\boldsymbol{p}}^{\boldsymbol{0}}$ is a rotation by $\hat{\boldsymbol{R}}_{\boldsymbol{r}}$ and a translation that moves the rotated $\boldsymbol{C}_{\boldsymbol{r}}^{\boldsymbol{0}}$ to $\boldsymbol{C}_{\boldsymbol{r}}$. The transformation is stored as

$$\tilde{\boldsymbol{T}}_{\boldsymbol{r}} = \left[\hat{\boldsymbol{R}}_{\boldsymbol{r}} \quad \left(\boldsymbol{C}_{\boldsymbol{r}} - \hat{\boldsymbol{R}}_{\boldsymbol{r}}\boldsymbol{C}_{\boldsymbol{r}}^{\boldsymbol{0}}\right)\right]$$

(21)

Each particle *p*'s goal position $\boldsymbol{g}_{\boldsymbol{p}}$ can be calculated as average regional rigid transformation of the particle's rest position, $\boldsymbol{x}_{\boldsymbol{p}}^{\boldsymbol{0}}$:

$$\boldsymbol{g}_{\boldsymbol{p}} = \frac{\boldsymbol{1}}{|\boldsymbol{Region}_{\boldsymbol{p}}|}\left(\sum_{r \in \boldsymbol{Region}_{\boldsymbol{p}}} \tilde{\boldsymbol{T}}_{\boldsymbol{r}}\right)\boldsymbol{x}_{\boldsymbol{p}}^{\boldsymbol{0}}$$

(22)

The particle position $\boldsymbol{x}_{\boldsymbol{p}}$ and velocities $\boldsymbol{v}_{\boldsymbol{p}}$ are updated using the goal position $\boldsymbol{g}_{\boldsymbol{p}}$ and the total external force $\boldsymbol{f}_{\boldsymbol{p}}$:

$$\boldsymbol{v_p(t+h) = v_p(t) + \alpha \frac{g_p(t) - x_p(t)}{h^2} + h \frac{f_p(t)}{m_p}}$$

(23)

$$\boldsymbol{x_p(t+h) = x_p(t) + h v_p(t+h)}$$

(24)

, where $\boldsymbol{h}$ is the simulation time step. The results of pure lattice shape matching are shown in Figure 14:

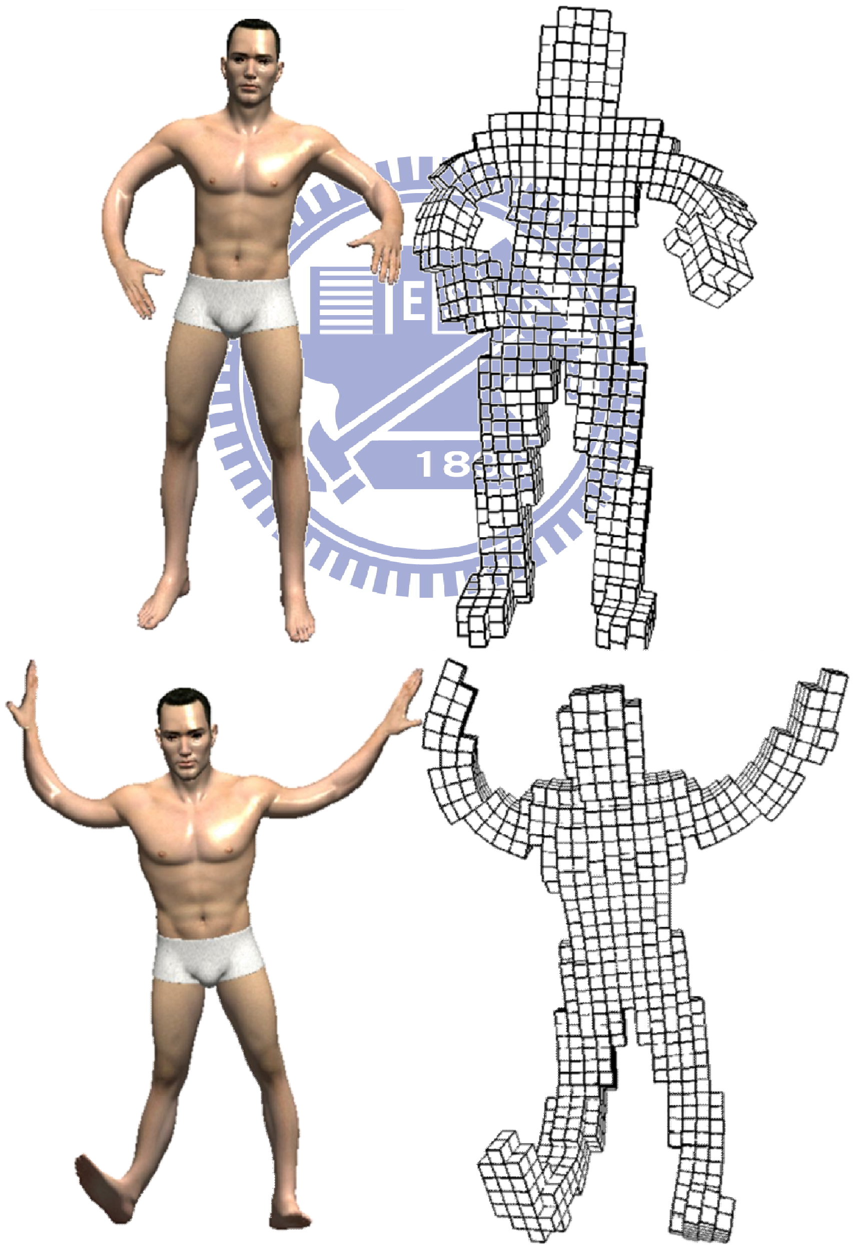


Figure 14: *Deformation by lattice shape matching.*

3.4.2 Combination of Skinning and Lattice Shape Matching

Both the lattice-based skinning and the lattice shape matching update particle positions. In order to generate secondary deformation by lattice shape matching, we use the result of Equation (4) and particle $\boldsymbol{p}$'s dynamic position $\boldsymbol{x_p}$ to obtain

$$\hat{\boldsymbol{x}}_{\boldsymbol{p}} = \boldsymbol{\delta x_p} + (\mathbf{1} - \boldsymbol{\delta})\bar{\boldsymbol{x}}_{\boldsymbol{p}} \tag{25}$$

where $\boldsymbol{\delta}$ is the scale of secondary deformation and $\mathbf{0} \le \boldsymbol{\delta} \le \mathbf{1}$. If $\boldsymbol{\delta} = \mathbf{0}$, then the result is same as lattice-based smooth skinning. The larger the $\boldsymbol{\delta}$ grows, the more obvious the secondary deformation appears. Moreover, users can freely adjust $\boldsymbol{\delta}$ or even switch different $\boldsymbol{\delta}$ profiles during simulation to obtain more realistic effects. The result of Equation (25), $\hat{\boldsymbol{x}}_{\boldsymbol{p}}$, is then applied to the lattice shape matching process. It means that the Equation (20) could be re-wrote as:

$$A_r \equiv \sum_{p \in Region_r} \tilde{m}_p \left(\hat{x}_p - C_r\right)\left(x_p^0 - C_r^0\right)^{\mathrm{T}} \tag{26}$$

At last, we compute the transformation as described in section 3.4.1. At each time step, each particle p will vibrate between x_p and $\bar{x}_p$. Since the goal position g_p will be more and more close to $\bar{x}_p$, x_p will converge toward $\bar{x}_p$ finally. The speed of convergence depends on the damping force of the mesh. The damping force was introduced as an extensive technique in [RJ07]. The original method is proposed by Muller et al. [MHHR06]. As an alternative to global damping of non-rigid motion, this method applies damping on a per-region basis. The estimate of each $Region_r$'s rigid-body velocity is:

$$v_r{}^{damped} = \frac{1}{M_r} \sum_{p \in Region_r} \left(\tilde{m}_p v_p\right)$$

$$I_r = \sum_{p \in Region_r} \tilde{m}_p \tilde{r}_p \tilde{r}_p{}^{\mathrm{T}}$$

$$L_r = \sum_{p \in Region_r} \left(r_p \times \tilde{m}_p v_p \right)$$

(27)

where $r_p = x_p - C_r, p \in Region_r$ and $\tilde{r}_p$ is the 3 by 3 matrix with the property $\tilde{r}_p v_p = r_p \times v_p$. Finally, the damped velocity of particle p is:

$$v_p{}^{damped} = v_p + k_{damping}\left(v_r{}^{damped} + \left(I_r{}^{-1} L_r \right) \times r_p - v_p \right)$$

(28)

where $k_{damping}$ is the damping coefficient.

3.4.3 GPU-accelerated Rendering

For the sake of rendering performance, we use vertex buffer object and a trilinear lattice deformer based on GLSL. The interpolation weight of vertices are considered as vertex attributes and could be store at GPU memory during pre-processing stage. The positions of all particles are stored in a texture and will be updated at each frame. See Figure 15:

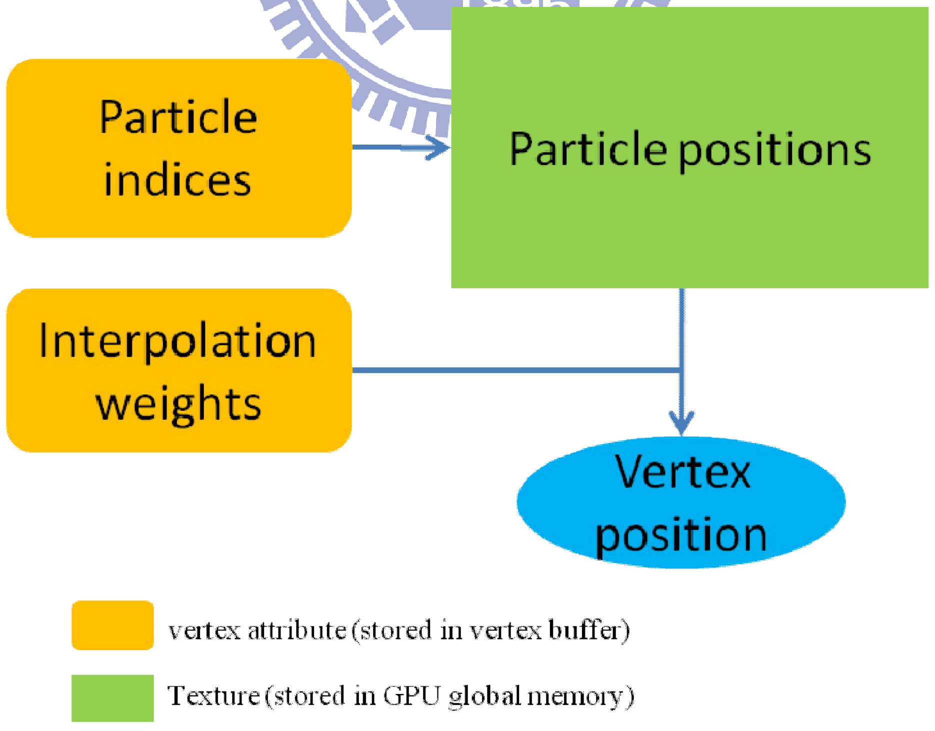

Particle indices
Particle positions
Interpolation weights
Vertex position
vertex attribute (stored in vertex buffer)
Texture (stored in GPU global memory)

Figure 15: *GPU Rendering concept.*

4 Experiment and Result

Our approach is flexible since we provide an interactive environment and many adjustable mesh parameters for users. We test many parameters such as voxelization resolution, α in Equation (23), δ in Equation (25), and etc.

Figure 16 shows our test mesh, skeleton, and mesh-skeleton mapping. All skeleton-driven animations we used come from CMU's BVH motion database.

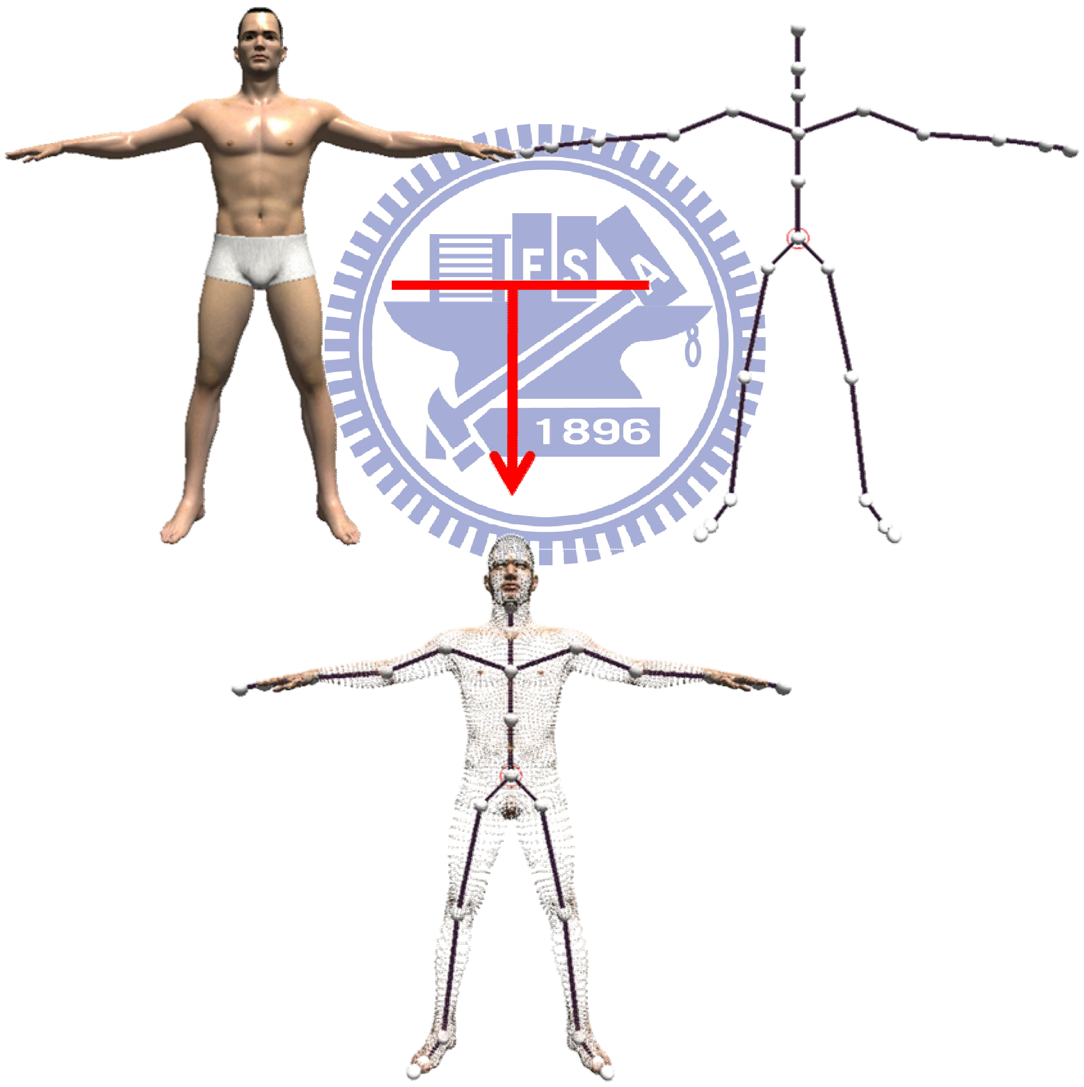


Figure 16: *Our mesh, skeleton and mesh-skeleton mapping.*

Figure 17 and Figure 18 show the results of voxelization resolution test. The voxelization resolution is mainly determined by original surface mesh resolution. Figure 17 chooses an improper resolution (1092 particles) and results in skinning artifacts. For our test surface mesh (28059 vertices), we choose the resolution in Figure 18 (about 1900 particles) to get balance between skinning quality and performance.

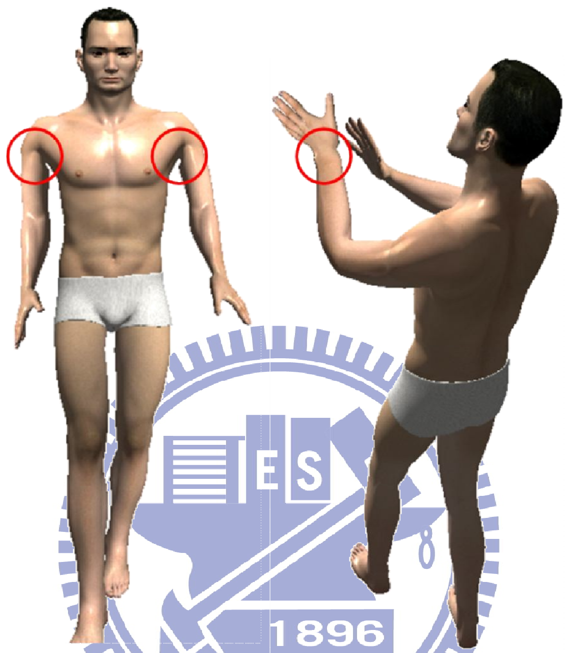


Figure 17: *Skinning artifacts with smaller voxelization resolution (1072 particles).*

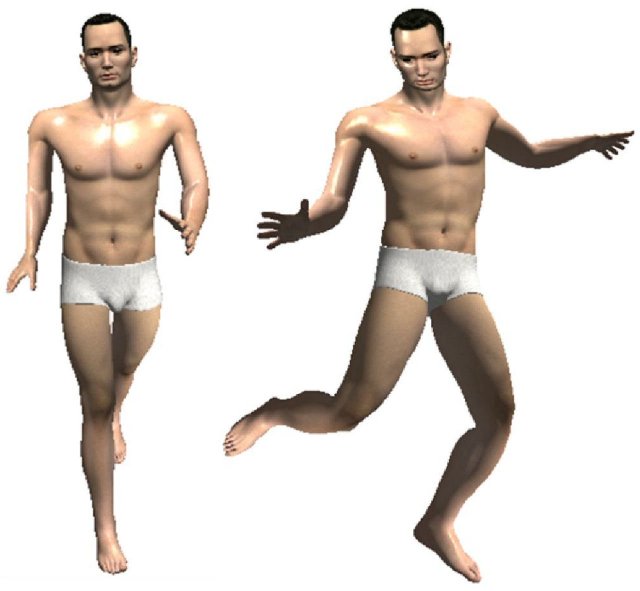


Figure 18: *Less skinning artifacts with larger voxelization resolution (1906 particles).*

Figure 19 shows the character driven by animation with secondary deformation.

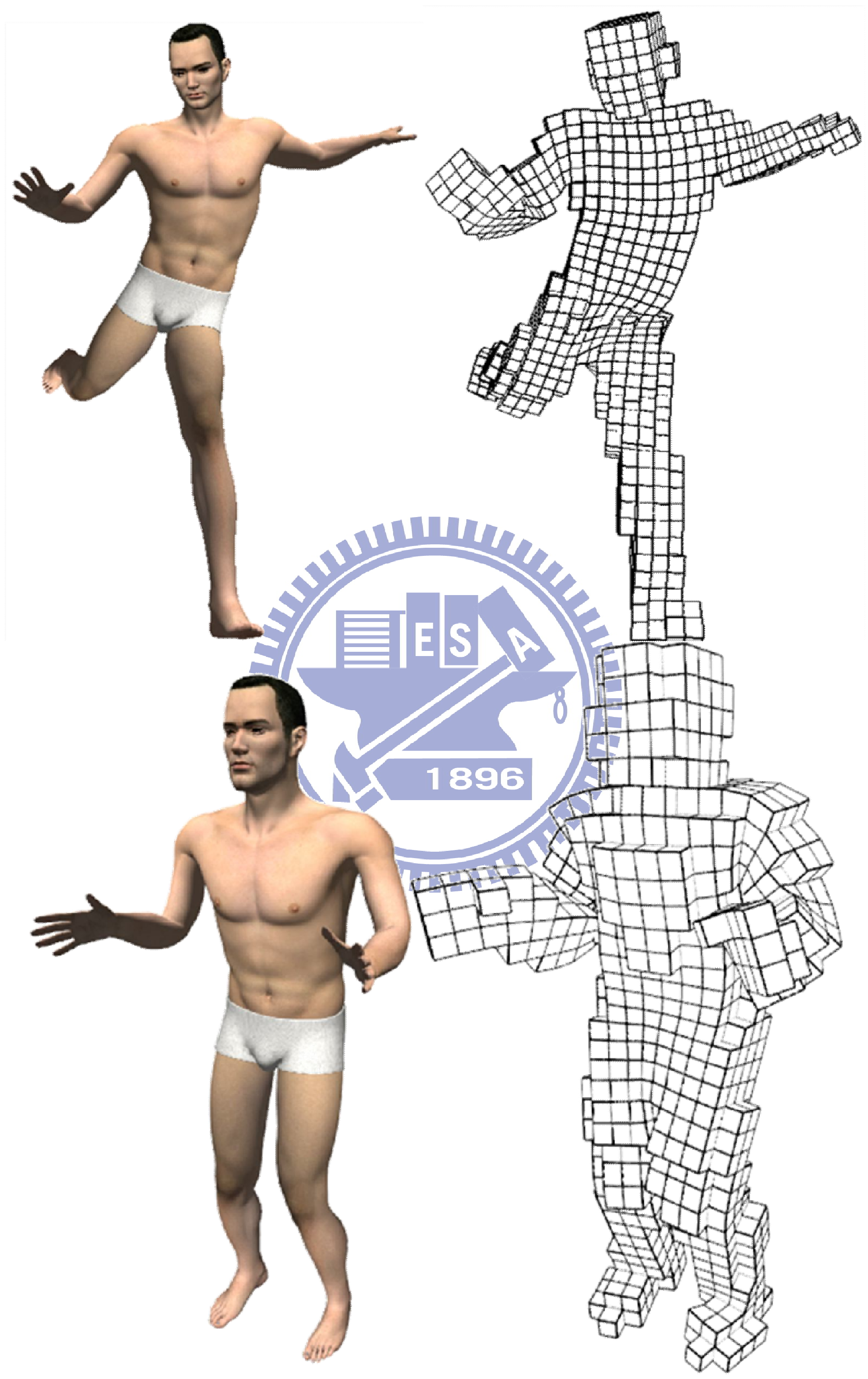


Figure 19: *Character Animation with secondary deformation.*

Now we test the influence of α in Equation (23):

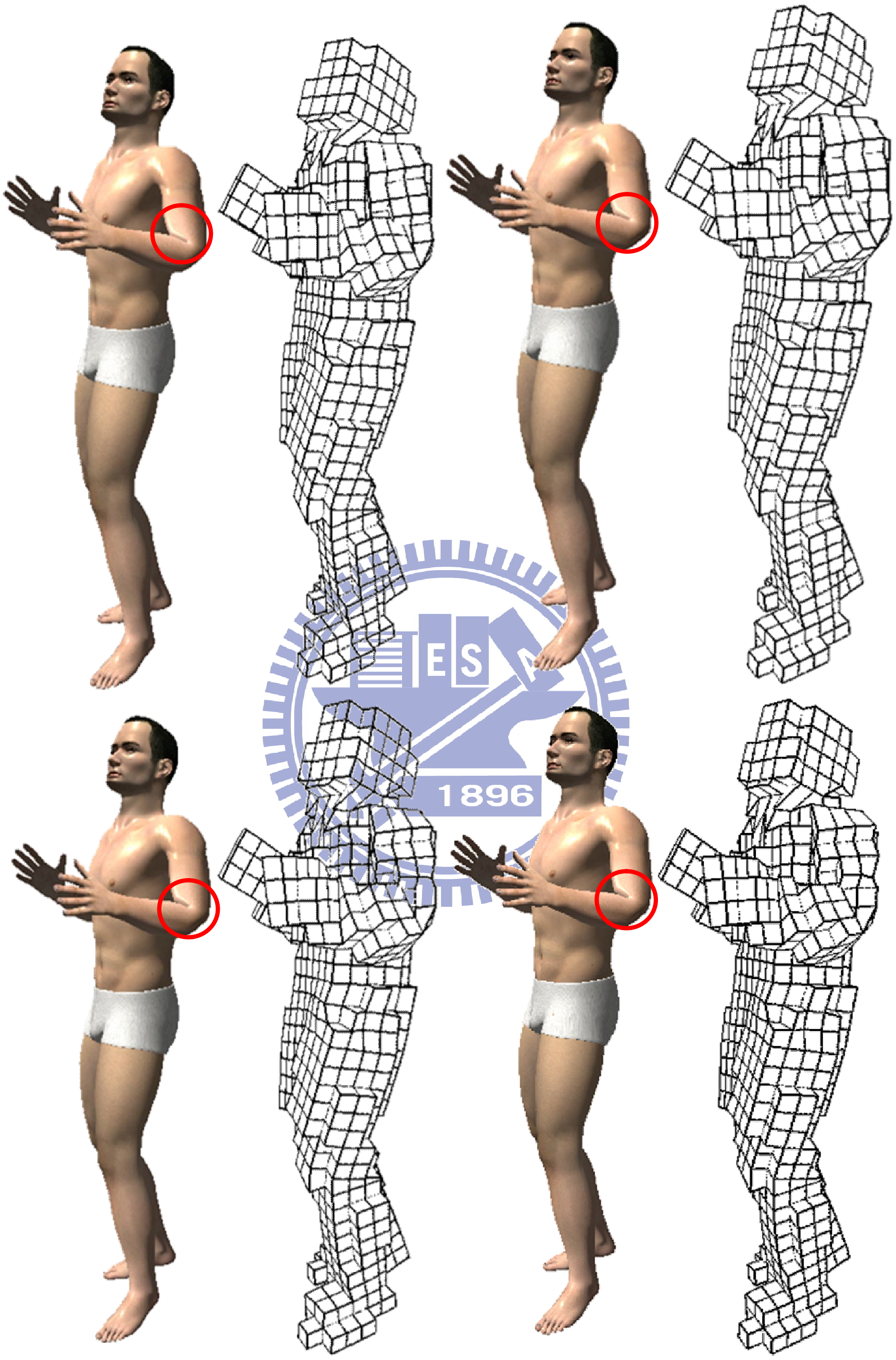


Figure 20: (*left top*) $\alpha = 0.25$, (*right top*) $\alpha = 0.5$, (*left bottom*) $\alpha = 0.75$, (*right bottom*) $\alpha = 1.0$.

Figure 21 uses higher voxelization resolution than Figure 20 for the test of α:

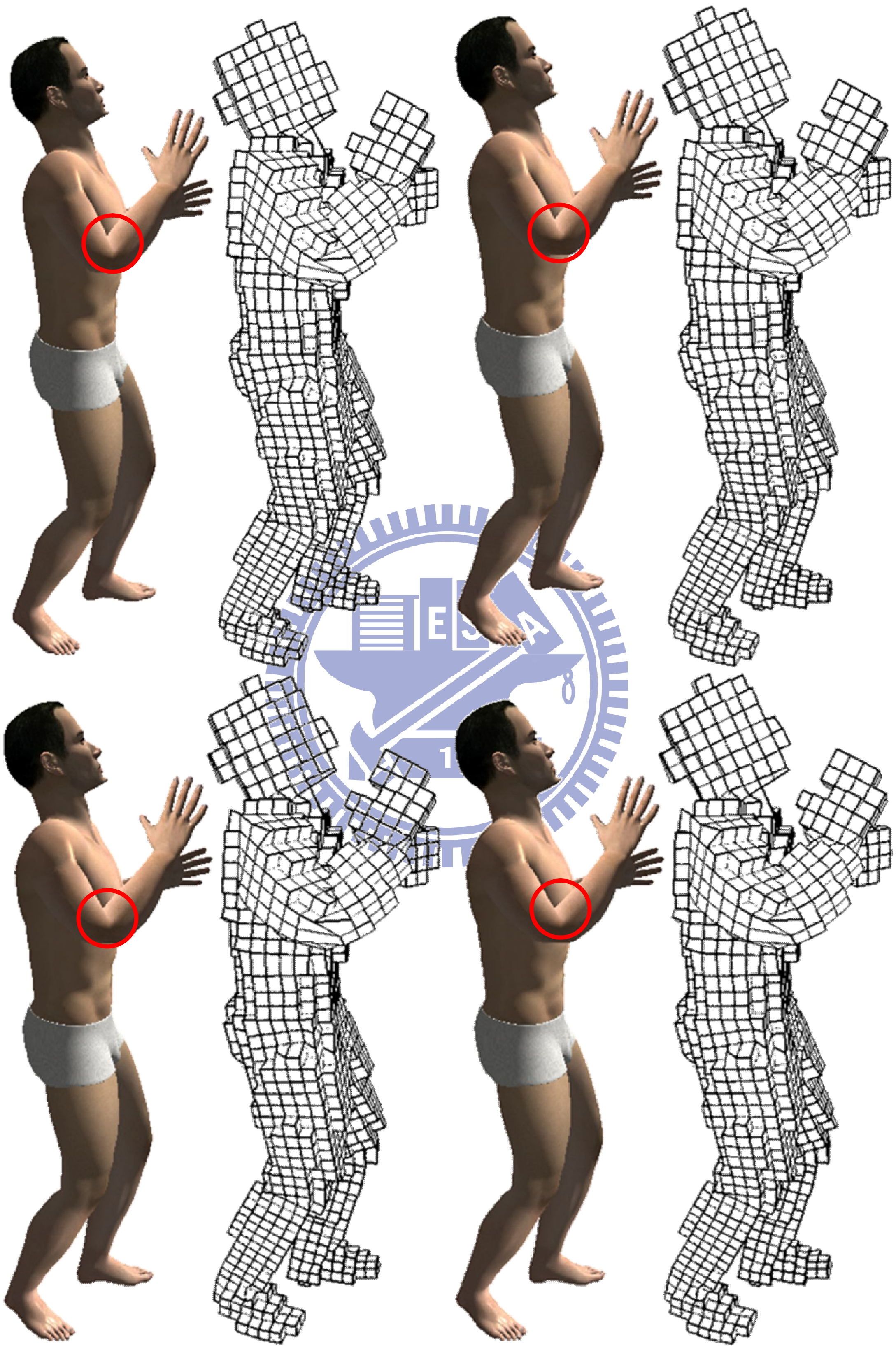


Figure 21: (*left top*) $\alpha = 0.25$, (*right top*) $\alpha = 0.5$, (*left bottom*) $\alpha = 0.75$, (*right bottom*) $\alpha = 1.0$.

Figure 22 uses a larger shape matching region size for the test of α:

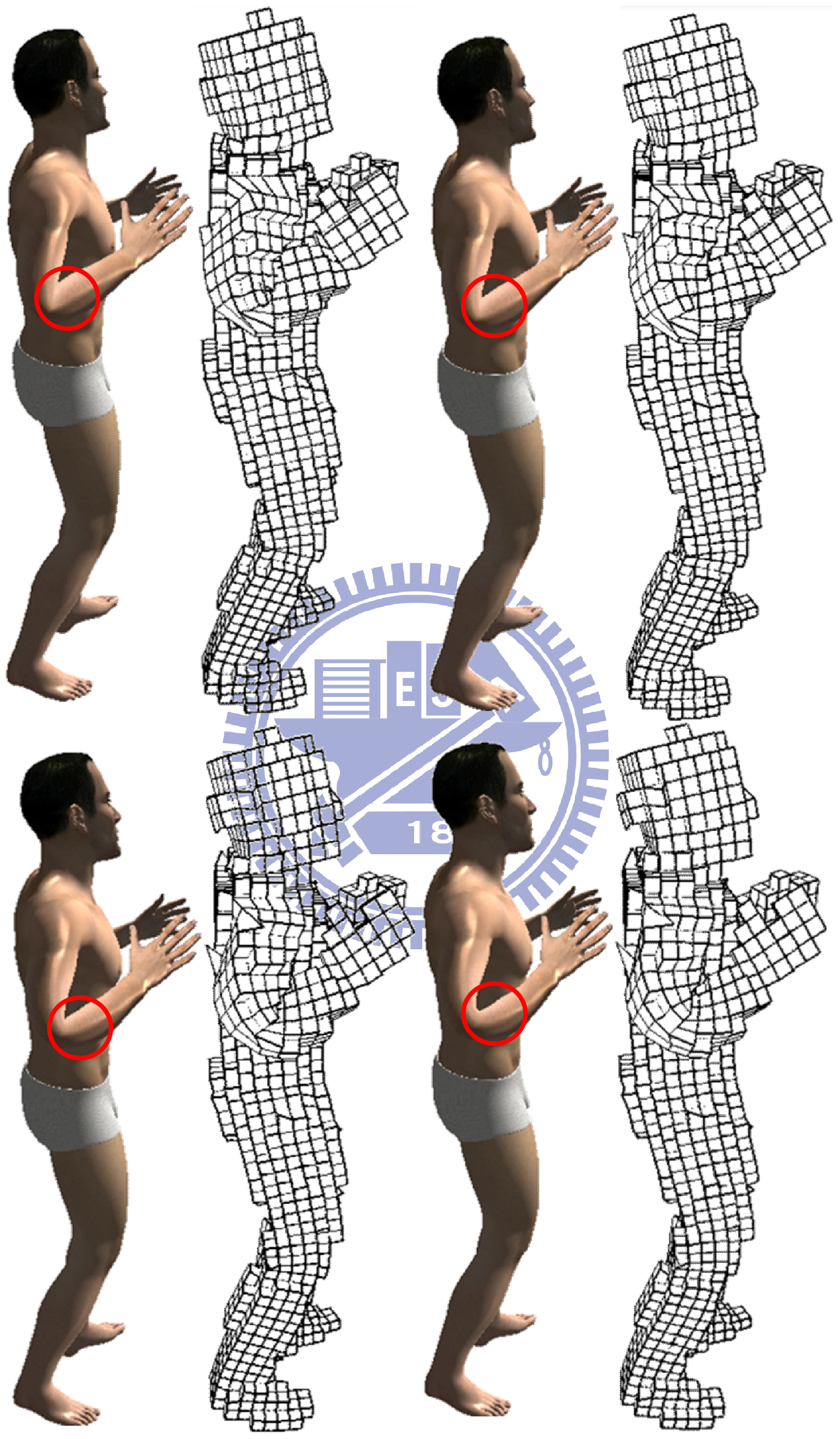


Figure 22: (*left top*) $\alpha = 0.25$, (*right top*) $\alpha = 0.5$, (*left bottom*) $\alpha = 0.75$, (*right bottom*) $\alpha = 1.0$.

As section 3.4.1 mentioned, the shape matching region size influence the smoothness of our mesh since the lattice shape matching method is a regional approach.

Now we show the results of our hierarchical volume preservation method:

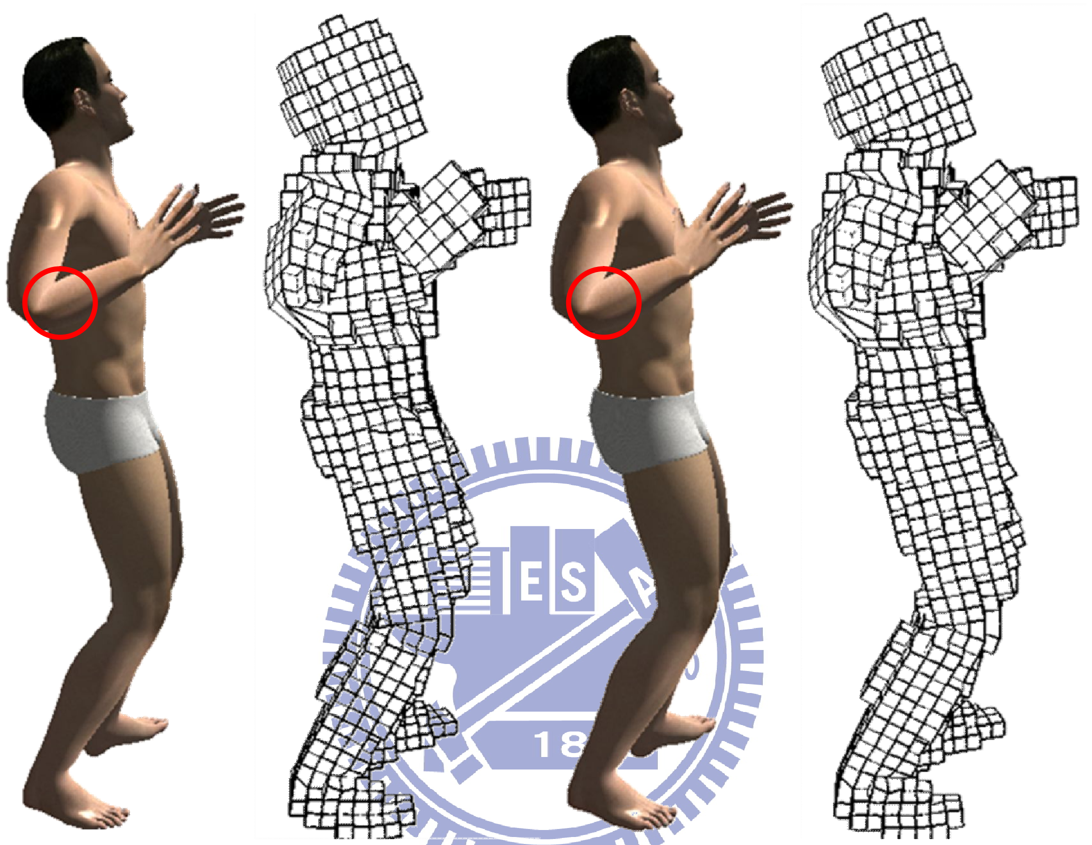


Figure 23: *(left) without hierarchical volume preservation, (right) with hierarchical volume preservation.*

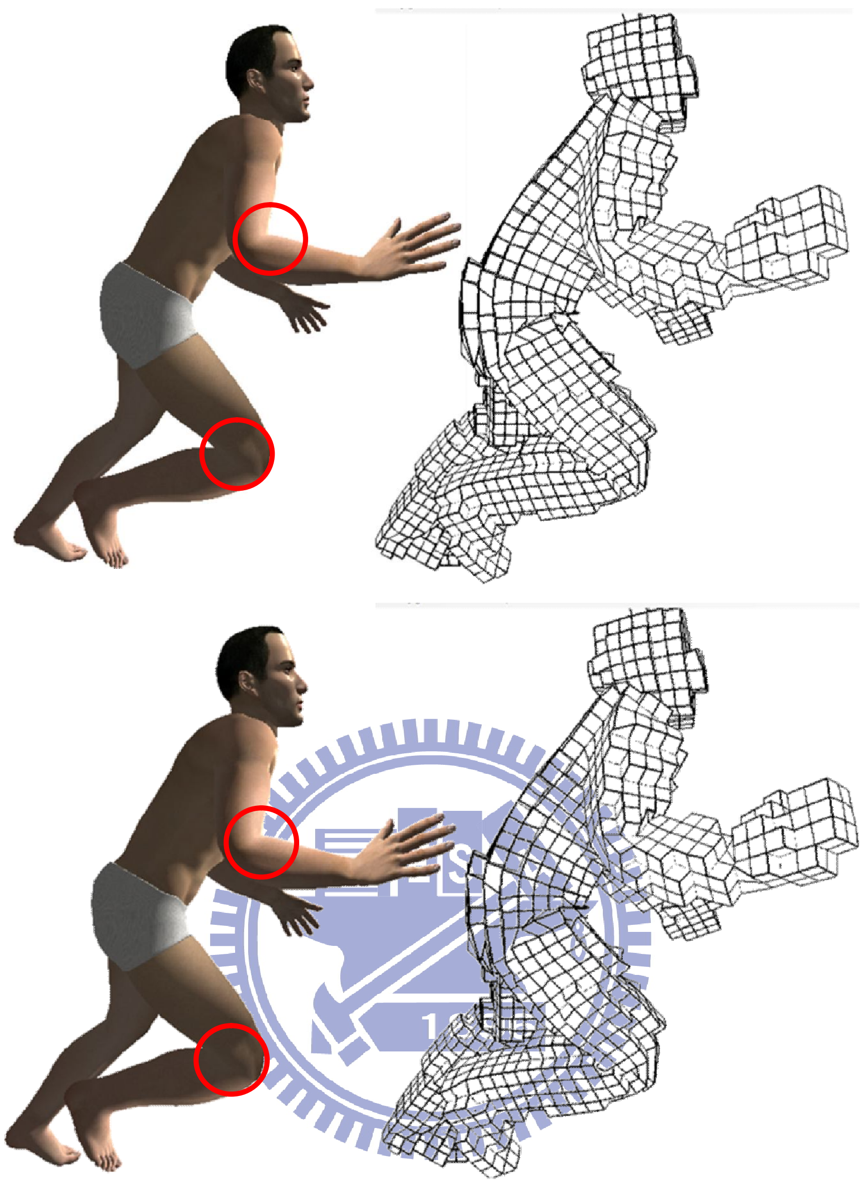
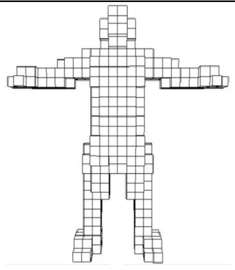
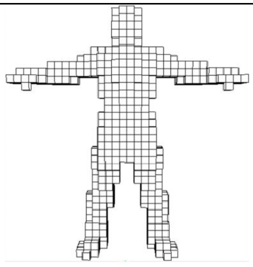
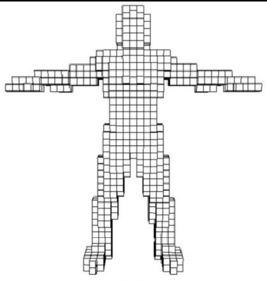
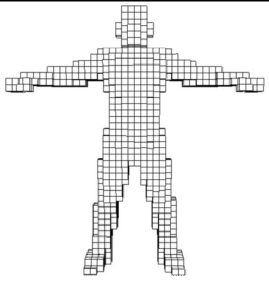


Figure 24: *(top) without hierarchical volume preservation, (bottom) with hierarchical volume preservation.*

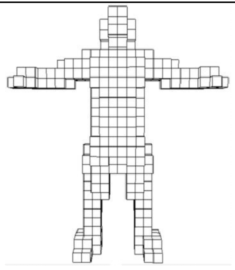
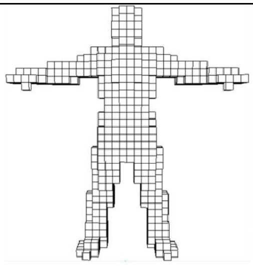
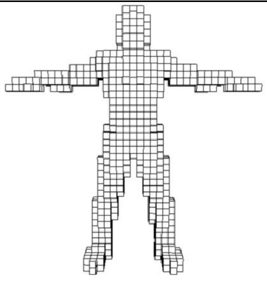
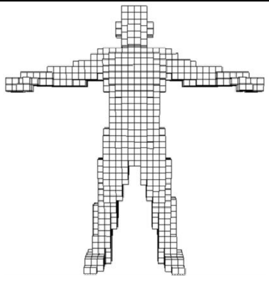
Figure 23 and Figure 24 show the results of volume correction. The volume preservation method is capable of reducing the "*candy-wrapper*" effect resulted from linear blend skinning.

Table 1 and Table 2 show performance tests for different voxelization resolution on different platforms. Our implementation is a CPU-bounded multi-threaded version. We run the test with a fully simulation, including skeleton-driven animation, lattice shape matching (including regional damping) and hierarchical volume preservation.

Property / Mesh	# of cells	# of particles	Average Frame Per Second
	524	1072	148.572
	976	1906	92.717
	1241	2451	61.684
	1902	3058	42.376

Triangle Mesh: 28059 vertices, 55888 triangles α in Equation (29): 0.25 δ in Equation (31): 1.00 Time step: 16ms Number of joints: 38 Region size: 2	CPU: Intel Core 2 Duo P8600 RAM: DDR3-1066 4GB Display: Nvidia GeForce G105M Number of threads: 2 OS: Windows 7 x64 Ultimate

Table 1: *Performance test on laptop.*

Property / Mesh	# of cells	# of particles	Average Frame Per Second
	524	1072	215.294
	976	1906	134.181
	1241	2451	99.436
	1902	3058	81.633

Triangle Mesh: 28059 vertices, 55888 triangles α in Equation (29): 0.25 δ in Equation (31): 1.00 Time step: 16ms Number of joints: 38 Region size: 2	CPU: Intel Core 2 Quad Q6600 RAM: DDR2-800 8GB Display: Nvidia GeForce 8800GT Number of threads: 4 OS: Windows 7 x64 Ultimate

Table 2: *Performance test on desktop computer.*

5 Conclusion and Future Works

We have presented lattice-based smooth skinning and its combination with lattice shape matching. The lattice shape matching method enhances the original skeleton-driven animations with physically-realistic secondary deformation. We provide an interactive environment for users to manipulate the mesh online. Users can change all adjustable mesh parameters during simulation to obtain desirable results. Shi et al. [SZTD* 08] proposed an example-based approach that generates excellent secondary deformation effects. However, it needs volumetric physics-based training examples and results in expensive cost. Our approach requires only skeleton-driven animation and triangle mesh as input data, and then constructs skinned mesh with little user intervention.

For the skeleton-to-mesh binding, we also provide an automatic and reasonable skinning weights computation process for users. Several automatic skinning weight computation methods use the concept of heat diffusion. However, such methods apply heat diffusion with an unreasonable way since the energy diffuse only on surface mesh. About the artifact "*candy-wrapper*" effect of linear blend skinning, we apply a hierarchical volume preservation method to reduce it. Moreover, our approach has excellent performance even on laptop computer.

The most critical issue in our future works is irregular sampling of voxelization. We can use an octree-based approach to achieve irregular sampling. Reducing the resolution of particles or regions in the object interior could speed performance without seriously altering the visual behavior. This method can improve lattice-based skinning quality and performance drastically. Another critical issue in our future works is skinning weight computation. Since our automatic skinning weight computation process only supports positive skinning weight, this restricts the appearance of several advanced deformation effects.

Reference

[BP07] BARAN, I. AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph.* 26, 3 (Jul. 2007), 72.

[BC00] BOURGUIGNON, D., AND CANI, M.-P. 2000. Controlling anisotropy in mass-spring systems. In *Proceedings of Computer Animation and Simulation'00*, 113–123.

[CGCD*02] CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVI´C , Z. 2002. Interactive Skeleton-Driven Dynamic Deformations. ACM Trans. on Graphics 21, 3 (July), 586–593.

[CMT05] CORDIER, F., AND MAGNENAT-THALMANN, N. 2005. A data-driven approach for real-time clothes simulation. *Computer Graphics Forum 24*, 2, 173–183.

[FVT97] FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 1997. Dynamic Free-Form Deformations for Animation Synthesis. IEEE Trans. on Visualization and Computer Graphics 3, 3 (July - September), 201–214.

[FOKM07] FORSTMANN, S., OHYA, J., KROHN-GRIMBERGHE, A., AND MCDOUGALL, R. 2007. Deformation styles for spline-based skeletal animation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, 141–150.

[JBT04] JAMES, D. L., BARBI˘C, J., AND TWIGG, C. D. 2004. Squashing Cubes: Automating Deformable Model Construction for Graphics. In Proc. of the SIGGRAPH 2004 Conference on Sketches & Applications, ACM Press.

[JMDG*07] JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 71.

[JSW05] JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 561–566.

[JZVC*08] JU, T., ZHOU, Q., VAN DE PANNE, M., COHEN-OR, D., AND NEUMANN, U. 2008. Reusable skinning templates using cage-based deformations. *ACM Trans. Graph.* 27, 5 (Dec. 2008), 1-10.

[KŽ05] KAVAN, L. AND ŽÁRA, J. 2005. Spherical blend skinning: a real-time deformation of articulated models. In *Proceedings of the 2005 Symposium on interactive 3D Graphics and Games* (Washington, District of Columbia, April 03 - 06, 2005). I3D '05. ACM, New York, NY, 9-16.

[KCŽO07] KAVAN, L., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2007. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on interactive 3D Graphics and Games* (Seattle, Washington, April 30 - May 02, 2007). I3D '07. ACM, New York, NY, 39-46.

[LCF00] LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and interactive Techniques* International Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 165-172.

[MTLT88] MAGNENAT-THALMANN, N., LAPERRI `ERE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, Canadian Information Processing Society, 26–33.

[MG04] M¨U LLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *Proceedings of GI'04*, 239–246.

[MHHR06] M¨ULLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2006. Position Based Dynamics. In Proc. of Virtual Reality Interactions and Physical Simulations (VRIPhys), 71–80.

[MBF04] MOLINO, N., BAO, Z., AND FEDKIW, R. 2004. A virtual node algorithm for changing mesh topology during simulation. ACM Trans. on Graphics 23, 3 (Aug.), 385–392.

[MDMJ*05] M¨ULLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable Real-Time Deformations. In ACM SIGGRAPH Symposium on Computer Animation, 49–54.

[OBZH00] O'BRIEN, J. F., ZORDAN, V. B., AND HODGINS, J. K. 2000. Combining active and passive simulations for secondary motion. *IEEE Comput. Graph. Appl. 20*, 4, 86–96.

[RJ07] RIVERS, A. R., AND JAMES, D. L. 2007. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. 26*, 3, 82.

[SNF05] SIFAKIS, E., NEVEROV, I., AND FEDKIW, R. 2005. Automatic determination of facial muscle activations from sparse motion capture marker data. ACM Trans. on Graphics 24, 3, 417–425.

[SZTD* 08] SHI, X., ZHOU, K., TONG, Y., DESBRUN, M., BAO, H., AND GUO, B. 2008. Example-based dynamic skinning in real time. In *ACM SIGGRAPH 2008 Papers* (Los Angeles, California, August 11 - 15, 2008). SIGGRAPH '08. ACM, New York, NY, 1-8

[TK09] TAKAMATSU, K. AND KANAI, T. 2009. Volume-preserving LSM deformations. In *ACM SIGGRAPH ASIA 2009 Sketches* (Yokohama, Japan, December 16 - 19, 2009). SIGGRAPH ASIA '09. ACM, New York, NY, 1-1

[THMG04] TESCHNER, M., HEIDELBERGER, B., MULLER, M., AND GROSS, M. 2004. A versatile and robust model for geometrically complex deformable solids. In *Proceedings of CGI'04*, 312–319.

[VFTS07] VON FUNCK, W., THEISEL, H., AND SEIDEL, H.-P. 2007. Elastic secondary deformations by vector field integration. In *Proceedings of SGP'07*, 99–108

[WPP07] WANG, R. Y., PULLI, K., AND POPOVI´C, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3, 73.

[ZHSL*05] ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H. 2005. Large mesh deformation using the volumetric graph Laplacian. *ACM Trans. Graph.* 24, 3 (Jul. 2005), 496-503