Simplified Muscle Dynamics for Appealing Real-Time Skin Deformation

Keng Siang Lee
School of Computing
National University of Singapore
Singapore

Golam Ashraf
School of Computing
National University of Singapore
Singapore

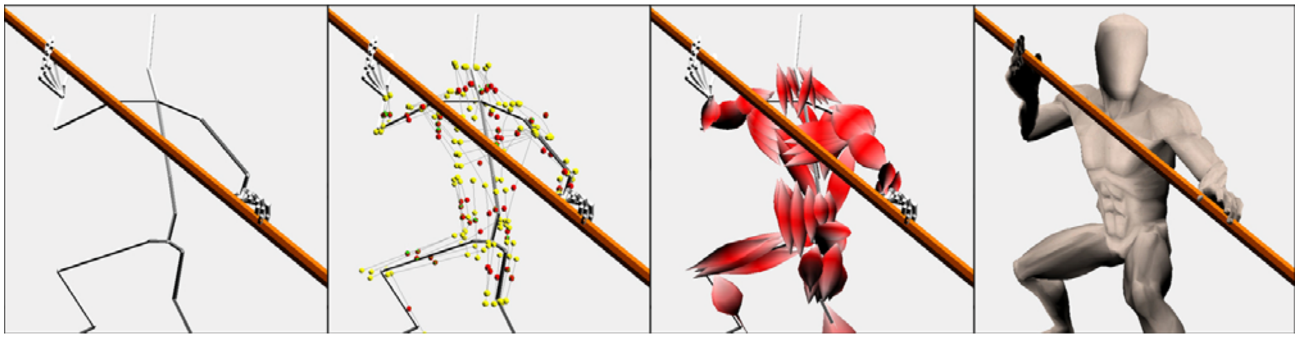

Figure 1: *Snapshots of a real-time virtual martial arts warrior, illustrating quadratic Bézier action curves hooked up with single springs, sinusoidal fusiform muscles and final skin deformation via smooth skinning.*

Abstract - *We propose significant simplifications in muscle modeling and simulation to facilitate real-time anatomical skin deformation for full-body articulated characters. The muscle shape is a function of an animated quadratic Bézier action curve and control rings derived from it. The action curve is uniformly sampled to derive control rings driven by a scaled sinusoidal equation to model fusiform shapes. A single spring is attached between the central control point and the midpoint vector between the extreme control points of the Bézier action curve. Care is taken to stabilize local coordinates for each muscle vertex to enable glitch-free skin deformation. The character's polygonal mesh is smooth-skinned using a two-layered approach: first to the joints, and then to the muscle vertices. Lastly we show how different prominent muscles can be reasonably approximated with the proposed fusiform model. A typical 4000-vertex character skinned with sixty four 72-vertex muscles is able to run on average CPUs at 60-80 fps. Our main contribution is the simplified dynamics driven curved action-axis, which enables economical and expressive muscle animation.*

Keywords: Muscle-based deformation, real-time dynamics, appealing characters.

1 Introduction

Appealing character deformation could play an essential role in believable gaming and virtual world experience. Character traits like sluggishness, power, etc. can be emphasized much better with anatomy-driven deformation than current real-time skeletal deformation of surfaces. Such visual emphasis of traits affords better immersion and connection to virtual characters. However, the high simulation cost of anatomy-driven deformations has limited their application in the real-time domain. On the other hand, video games have been steadily pushing the quality envelop, given rapid advances in hardware and increasing consumer appetite. Realistic and appealing deformation for game characters is therefore a relevant topic for the modern game industry.

Layered anatomical models have achieved body deformation with accurate muscle, bone, fat and skin dynamics. They achieve more robust results than analytical surface deformation models like smooth-skinning. Localized details like muscle bulges, sliding and stretching of skin over bones, loose skin, etc. are better captured with anatomical methods. Some of these features are being adapted to interactive applications, by pre-computing the dynamics simulation, and "playing back" the most appropriate approximation in response to event triggers. Though this alleviates the abovementioned limitations, it still lacks the expressive repertoire of accurate simulations. In some cases, the played-back dynamics response may even violate physical constraints.

In this paper, we address the lack of efficient dynamics methods to achieve anatomy-driven body deformation in real-time. We propose a flexible muscle representation that can be economically driven by single-spring dynamics. We adapt the smooth-skinning metaphor to blend conventional skeleton-driven deformation with our muscle-driven deformation. The resulting muscle-based skin contouring and oscillation (commonly termed as jiggle) can generate significant appeal to real-time characters.

The focus of this paper is not on accurate muscle modeling but on simulating efficient local deformations and oscillations which increase visual appeal without violating significant physical constraints. We also demonstrate how a variety of prominent muscles can be modeled with the same generic representation, aided with intuitive design tools.

2 Previous Works

An in-depth survey on recent multi-layered anatomical models and skin deformation can be found in [9] [2] [10]. We choose to summarize the methods here for completeness, and later present a more focused comparison of our proposals with relevant methods (see Sec. 6).

Multi-layered anatomical models have been organized into skeleton, muscle, fatty tissues and skin layers. The skeleton is first defined using joint design tools. Muscle primitives are then added, with the origin and insertion points of each muscle constrained to the appropriate joints. The skin surface could be handcrafted or "draped" over the muscles using contour detection and implicit functions. Different levels of dynamics and kinematics constraints control the deformation of these models, using procedural volume preservation, elasticity, collision avoidance, and weighted influence functions. The general complexity of these representation and control methods make anatomical models difficult to realize in real-time.

[4] used Free-Form Deformations (FFD) to approximate muscle deformation. [8] applied Dirichlet Free-Form Deformation method (DFFD) for hand deformation. It is generally hard to emulate muscles in complex regions such as the shoulders, since the relationships between FFD deformations and high-dimensional joint rotations are not well defined. Finite Element Mechanics (FEM) models accurately deform volumes defined by connected cells that impart torsion forces on each other using physics of energy transfer [15]. However they are usually too complex for real time simulation.

[11] proposed several surface representations for muscle models that could be deformed with volume preservation formulae. They modeled simple fusiform muscles with ellipsoids and multi-belly sheet-like muscles with multiple instances of these fusiform muscles. [16] used similar volume-preserved cylinders, where each muscle segment's influence on skin vertices was analytically calculated. The fusiform model was further enhanced with spring constraints [10], stable local coordinates [2] and curved action axis [17]. Though these enhancements increase the generality of the fusiform muscle's shape and motion, they add significantly to the computation. Real time performance of these models had been demonstrated using only a few muscles at a time, without any skin deformation. We draw inspiration from these enhancements, and simplify their representation to facilitate fast stable animation for full-body humans (see Sec. 3).

[14] [16] modeled sophisticated elastic skin models that are first deformed by muscle slices, and then skin triangles are adjusted based on spring forces on each edge. Though such elastic models achieve several interesting effects like skin sliding and stretching, the simulation cost is again too high for real time performance. [16] chose to statically bind the skin layer and move it relative to underlying tissue deformation. [7] presented a similar approach in blending handcrafted shapes with skin deformed by skeletal motion. A different approach is adopted in [1] who used per-frame mesh regeneration by sampling the implicit surface envelop of underlying tissue. As [16] pointed out, this approach may have repercussions on faithful texture movement for non-smooth surfaces. We draw inspiration from the unified approach to skin deformation in [16] and [7], and propose a smooth-skinned solution for muscle deformation (see Sec. 3 and 4).

We feel that an expressive and simplified muscle/skin framework is sufficient for current real-time applications. Our goals are to afford easy setup and representation of muscles, minimal semi-automatic pre-computation steps, economical on-the-fly dynamics simulation, and a skin deformation framework that builds on top of existing smooth skinning to joints. Most importantly, it should be easy to blend our methods with existing character production pipelines, so that we can see their rapid incorporation in commercial games.

In this paper, we propose an adaptation to the surveyed musculature modeling methods to facilitate efficient dynamics simulation. First we explain muscle representation using quadratic Bézier curves. We derive local coordinate frames for each cross section of the muscle, before presenting the shape construction with volume preservation. Next we explain the simplified single-spring dynamics representation and simulation mathematics. We then describe how we combine joint and muscle based deformation on the polygonal character mesh using the smooth-skinning metaphor. Lastly, we present the results and a critical analysis of our contributions.

3 Muscle Creation

3.1 Representation Using Quadratic Bézier Action Curve

As proposed by [10] and [1], the action line drives the shape and movement of the associated muscle. It is constrained to skeletal joints and moves with them. Forces and collision detection required to control the movement of individual action line segments may prove expensive for real-time applications. [17] first proposed action curves using cubic rational B-Spline instead. Although their representation is more general, we believe that the overhead involved in calculating C^2 smooth action curves is unnecessary, as muscles generally look alright with C^1 continuity. We propose a quadratic Bézier curve representation for the action curve, to facilitate faster computation and simpler control. The shape of the entire action curve can be changed by just manipulating the middle control point. We chose simplicity in calculation and representation, as this element is a basic building block that needs to be recomputed often (usually 60-100 per character). From our tests, we observe that the quadratic Bézier curves adequately define shape and movement for real-time muscles.

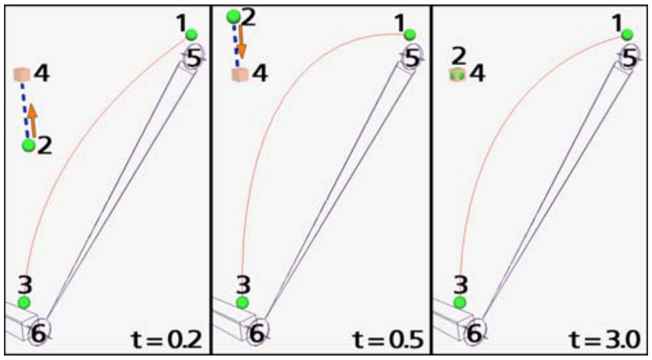

Figure 2: *The three control points P_{org} (1), P_{mid} (2) and P_{ins} (3) of the muscle action curve, the constrained control point P_{const} (4), and the two attachment joints (5 and 6). Notice how P_{mid} moves towards P_{const} when perturbed, causing the Bézier curve to change shape smoothly.*

A quadratic Bézier curve requires three control points (P_0, P_1, P_2) which can be mapped to the muscle origin, mid-point, and insertion point (P_{org}, P_{mid}, P_{ins}), as shown in Fig. 2. During muscle creation, the origin and insertion joints of the skeleton are chosen, and P_{org} and P_{ins} are parented to these 2 joints respectively. These two control points can be in any arbitrary position (usually guided by anatomical information) in their respective parent's coordinate frame. By default, P_{const} is constrained to the midpoint between P_{org} and P_{ins}, but the user can shift this control point to change the muscle curvature. P_{const} retains user-defined offsets when P_{org} and P_{ins} move with the joints (see Fig. 2).

The second control point of the Bézier curve is represented by a freely-hanging dynamic control point P_{mid}, which is attached to P_{const} via a spring. The Bézier action curve is created from the control points (P_{org}, P_{mid} , P_{ins}), so that the muscle moves with soft-body dynamics as the joints (and consequently P_{const}) move in space. The soft body oscillations are especially noticeable immediately after the origin/insertion bones come to rest.

3.2 Construction of Local Coordinate Frame

After creating the action curve, we sample N_{cs} uniformly spaced points along the curve, where N_{cs} is a user specified number. Each of the points represents the center C_i of each cross-sectional slice i of the muscle. We then need to construct the local coordinate frame for each of these cross-sections. We choose to avoid the Frenet frame formulation because of the problems identified in [17]. The normal (calculated as a second derivative) at point p(u) on the curve is sensitive to the curve shape (see Fig. 3a). In other words, the cross-section coordinate frame starts rotating as the action curve changes shape, causing the resultant muscle shape to twist undesirably along its main action curve axis during animation. Furthermore, the normal for p(u) is undefined for a straight line, which may be the initial state of an action curve.

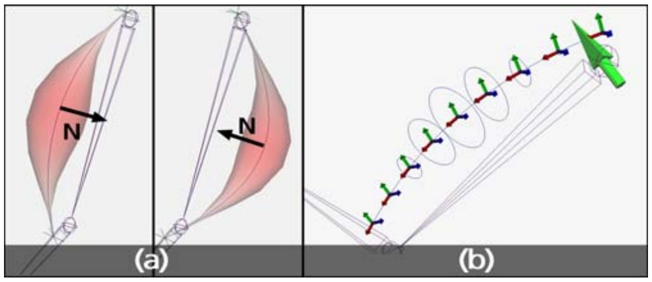

Figure 3: *Stable Local Coordinate Frame. a) Normal inverts due to change in action curvature; b) This problem is solved by using a reference up-vector (big arrow).*

To stabilize the abovementioned problem, we choose a user-defined up-vector v (large arrow in Fig. 3b) to derive the local coordinate frame. We use the tangent of the curve at p(u) (where p(u) = C_i) and the up-vector v for the entire muscle, to construct a local coordinate frame that is independent of the shape of the action curve. First, the normalized curve derivative is calculated at cross-section i (Eqn. 1). Two other orthogonal axes are then calculated.

$$\hat{a}_i = \frac{p'(u)}{|p'(u)|} \qquad [1]$$

$$c_i = \hat{a}_i \times \hat{v} \qquad [2]$$

$$b_i = \hat{c}_i \times \hat{a}_i \qquad [3]$$

The axes of the coordinate frame for cross-section i are then $\hat{a}_i$, $\hat{b}_i$ and $\hat{c}_i$, which correspond to the local x, y and z axes respectively. The up-vector v is set to the joint's local y-axis by default and could be changed interactively via the muscle setup tool. Note that the local coordinate frame of a cross-section will twist by 180 degrees sharply if the tangent vector $\hat{a}_i$ coincides with the up-vector direction. This depends on the range of motion of the muscles as the joints rotate. If such an event is likely to happen, the up-vector should be manually adjusted so that $\hat{a}_i$ will never cross its direction. For example, it can be changed to a direction that is orthogonal to the range of motion for the given muscle.

3.3 Construction/Control of Muscle Shape

With the local coordinate frame defined for each cross-section of the muscle, we can now determine the positions of the muscle vertices in order to define shape. The goal of our formulation is to generate more expressive shapes than existing methods in [11] [16] [10] [17], without adding much computation complexity. We characterize the fusiform muscle with two main properties: bulge size S and taper T. The size parameter controls the girth of the center of the fusiform shape. Apart from the normal volume-preservation related deformation, this parameter is also useful for isometric contraction simulation where the muscle bulge increases without a change in muscle length. The taper parameter controls the curvature at end points, which is useful for approximating tendons.

The bind pose radius r_i of each muscle cross-section is calculated as:

$$r_i = S \sin^T \left(\frac{i-1}{N_{cs}-1}\pi\right) \quad [4]$$

Each muscle cross-section i has a user-defined N_{div} number of muscle vertices it will control. Each of these vertices v_{ij} (where j=1, 2,..., N_{div}) has an offset o_{ij} from the cross-section center C_i. This offset is stored in the local coordinate system of cross-section i. When the muscle is created initially, each of the vertices v_{ij} for cross-section i will get the offset value assigned as $o_{ij} = r_i$. The local coordinate frame of each muscle vertex v_{ij} is a simple addition of its offset o_{ij} with the local coordinate frame of cross-section i. We allow these offsets to be adjusted by the user through our muscle shaping tool implemented in Maya^{TM} in order to sculpt arbitrary shapes easily (see muscle design workflow in Fig. 4). Such an offset-from-center representation also allows for CT scan information of real muscle cross-sections to be incorporated [10] [17]. During simulation, the cross-section offsets are proportionally scaled by volume-preserved shape distortion functions (see Sec 3.4).

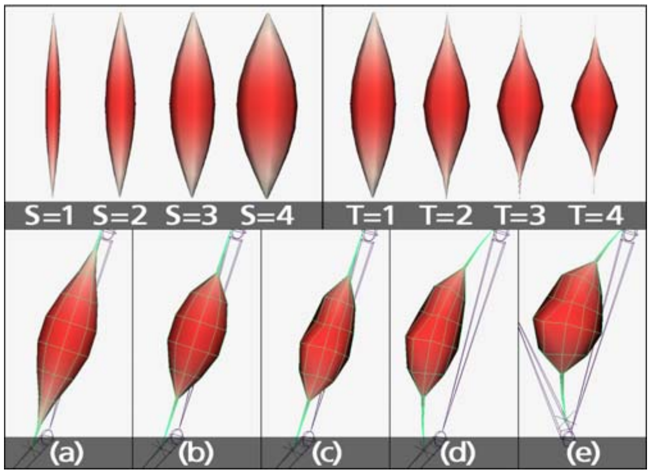

Figure 4: *Expressive shape representation. TOP: Shapes achieved with different size and taper settings; BOTTOM: Muscle design workflow - a) Affix to skeleton; b) Set size, taper and control ring radius; c) Sculpt muscle; d) Tweak action axis curvature; e) Test deformation for joint motion range*

3.4 Muscle Deformation

The basic volume preservation of the muscle is based on an approximation of the fusiform shape by an ellipsoid model [16] [11]. The volume of an ellipsoid is:

$$V = \frac{4}{3}\pi r^2 l \quad [5]$$

where r is the radius of the mid cross-section and l is the length of the ellipsoid. The l in this case represents the Euclidean distance between P_{org} and P_{ins} of the action curve, instead of the actual arc length of the action curve which is computational expensive to calculate. Although this approximation may violate the volume preservation axiom, it can be compensated with a scaling factor (described later in Eqn. 7).

We can calculate each new offset o_{ij}' as l changes, by using this proportional relationship (derived from Eqn. 5):

$$o_{ij}' = \sqrt{\frac{l_{rest}}{l_{new}}}\, o_{ij} \quad [6]$$

where l_{rest} is the rest length of the muscle in bind pose and l_{new} is the new muscle length at a certain time instance.

However, we have observed that the general ellipsoid volume preservation method [11] does not yield visually obvious deformation due to small changes in muscle lengths (typical of humanoid characters). Our aim is to have visually appealing skin deformations, so muscle bulges must be perceptible. Thus we added a new layer of control that allows the user to non-linearly scale the volume of the muscle as the length changes, as shown in Eqn. 7. This is modeled by a scaled sinusoidal function which causes no additional bulge at rest length. In the equations below, B is the bulge multiplier and Δl is the length difference (l_{rest} - l_{new}).

$$o_{ij}' = (1 + B\sin(\frac{\pi}{2}\Delta L))\sqrt{\frac{l_{rest}}{l_{new}}}\, o_{ij} \quad [7]$$

$$\Delta L = \begin{cases} \dfrac{\Delta l}{l_{rest}} & if\ \Delta l > 0 \\ \dfrac{\Delta l}{l_{rest}} \times \dfrac{1}{R} & otherwise \end{cases} \quad [8]$$

We realized that with Eqn. 7, the muscle works well for flexion but thins out significantly when stretched. Thus we introduced another user-defined positive-valued attribute bulge-to-stretch-ratio R to counter this effect. This is taken care of in the second condition in Eqn. 8.

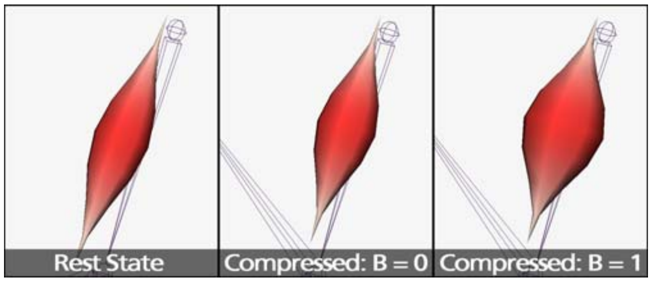

Figure 5: *With and without the bulge multiplier B. The muscle in the middle without the multiplier produces bulges which are mathematically correct but hardly noticeable while the muscle on the right with the multiplier produces visually noticeable bulges.*

3.5 Muscle Dynamics

We now present the dynamics calculations performed on the underlying Bézier action curve. A spring is attached to the control points P_{mid} and P_{const}. This spring produces a force at each time-step Δt that causes P_{mid} to move towards P_{const}. Since P_{mid} is the middle control point which determines the curvature of the Bézier curve, the Bézier curve changes its shape as P_{mid} moves. The force exerted on P_{mid} (Hooke's Law, Eqn. 10) and then its acceleration (Newton's Law, Eqn. 11) is calculated:

$$\Delta x(t) = x_{mid}(t) - x_{const}(t) \qquad [9]$$

$$F_{spring}(t) = -k_s \Delta x(t) \qquad [10]$$

$$\ddot{x}_{mid}(t) = F_{spring}(t) / m_{muscle} \qquad [11]$$

where *x(t)* represents control point position at time *t*, k_s is the user-defined spring constant and m_{muscle} is the user-defined mass of the muscle.

Verlet integration is used to find the position of P_{mid} at the end of each time-step. This method was chosen because it offers more stability at almost the same cost as simpler integration methods such as Euler integration, and is faster than accurate methods such as 4th-order Runge-Kutta. We have used a modified Verlet integration scheme which allows for a simple damping effect:

$$x_{mid}(t+\Delta t) = \ddot{x}_{mid}(t)(\Delta t)^2 - (1-f)x_{mid}(t-\Delta t) + (2-f)x_{mid}(t) \qquad [12]$$

Notice that the velocities of P_{mid} and P_{const} (required by Forward Euler and other conventional integration methods) have been omitted from all the above equations. Velocity is normally used to determine the amount of drag force affecting an object. However, the drag force is approximated by scaling the last two terms in Eqn. 12 with $f \in [0,1]$, a user-specified damping coefficient. This omission of velocity implies fewer calculations per time-step.

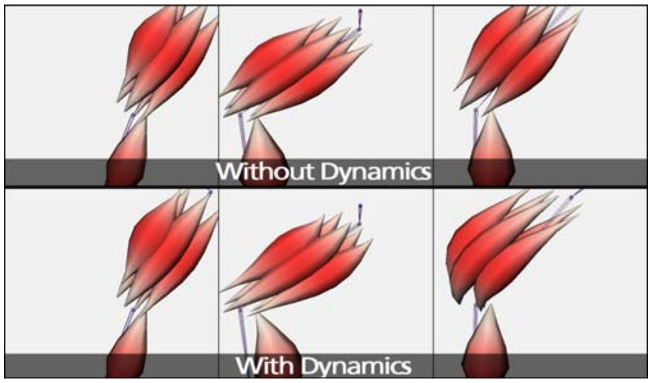

Figure 6: *Single spring dynamics achieve pleasing drag and oscillation for a set of leg muscles*

4 Skin Deformation

A two-layer skinning approach is used to skin the character's polygonal mesh. The first layer is the usual smooth-skinning of mesh vertices to joints, giving basic deformation of the mesh at acceptable cost. The second layer is the smooth skinning of the mesh to vertices of influencing muscles. This allows for addition of local deformation and oscillation to the skin, due to underlying muscle distortion and dynamics. We now look at these two levels of skinning in detail.

The smooth skinning of the mesh to joints is done in the conventional way:

$$v_{joint} = \sum w_i M_i B_i^{-1} v_0 \qquad [13]$$

where v_{joint} is the final skin vertex position in world space due to joint influence, w_i is the weight of joint *i*, M_i is the current world matrix of joint *i*, B_i is the bind pose world matrix of joint *i*, and v_0 is the bind pose skin vertex position in world space. We restrict each skin vertex to be influenced by a maximum of 3 joints, and weight paint tools have been setup in Maya^{TM} to allow for intuitive weights assignment.

Next we need to make an association between the skin vertices and the muscle vertices which influence them. Each skin vertex in bind-pose is associated with the nearest N_m muscle vertices. The muscle-vertex-to-skin-vertex weight is obtained by taking the inverse of the separation distance so that nearer points get more weight. After these computed weights are normalized, the user can assign weights for the association between skin vertices and entire muscles (not muscle vertices). This is done using the same weight assignment tool, exactly following the weight painting metaphor for smooth skinning with bones. This stage explicitly allows users to control the amount of influence from underlying muscles on various parts of the skin mesh.

Each of the skin vertices are affected by the muscle vertices via smooth skinning, similar to Eqn. 13:

$$v_{muscle} = \sum W_i' w_i' M_i' (B_i')^{-1} v_0 \qquad [14]$$

where v_{muscle} is the final smooth-skinned point in world space after muscle deformation, W'_i is the weight of the *i*-th muscle that is affecting the skin vertex, w'_i is the weight of the *i*-th muscle vertex that is affecting the skin vertex and v_0 is the bind pose skin vertex position in world space. M'_i is the current world matrix and B'_i is the bind pose world matrix for each associated muscle vertex. These matrices are derived from the local coordinate frames described in Sec. 3.2 and 3.3.

We pre-multiply $W'_i w'_i$ (Eqn. 14) for the purpose of efficient storage and computation. We also pre-multiply $B_i^{-1} v_0$ (Eqn. 13) and $B'^{-1}_i v_0$ (Eqn. 14) to save repeated matrix multiplications.

The final position of each vertex is calculated by blending the results of Eqn. 13 and 14 together:

$$v_{final} = v_{joint} + v_{muscle} \qquad [15]$$

Note that $\sum w_i + W_i' w_i' = 1$ and we enforce this constraint during weight painting. This ensures that there is no need to explicitly weight v_{joint} and v_{muscle} and normalize v_{final} in Eqn. 15, thus saving expensive division operations. In our demo [6], we allow for scaling of the components in Eqn. 15 for quick feedback on different weight blending effects.

By using the smooth skinning metaphor, we have managed to retain the simplicity and additive nature of skin deformation even after influencing muscles have been added. We have implemented the muscle weight paint tool to closely mimic the joint weight paint tool, to facilitate easy usage by artists who are familiar with existing weight paint workflows. The muscle-vertex-to-skin-vertex weight complexity is purposely hidden from users to protect them from laborious extra specification. By setting reasonable constraints to the number of influence joints and muscle vertices, we ensure that the deformation structure does not bloat unmanageably.

5 Results

A well-defined primary character might contain around 70-80 muscles while a secondary character might contain around 30-40 muscles by simplifying muscle groups. We assign 72 vertices per muscle to achieve reasonable binding results without flickers or sharp edges. Each muscle is attached to only two joints, thus the total number of joints does not affect the performance of the system.

Test A - Muscle Dynamics: We examined how well a maximum of 200 muscles (approximately 2.5 main characters) runs on an average system of 2.6GHz with 2.0GB RAM and GeForce 7800GS graphics card. These muscles were subjected to constant movement for 20 seconds, and the average FPS was recorded (see Table 1). The average frame-rate for 200 muscles is 35fps and this shows that our muscle setup (with no skinning yet) is efficient in delivering real-time dynamics-based movements, for 2-5 characters.

Muscles	**20**	**40**	**60**	**80**	**100**
Avg FPS	267	157	111	85	70
Muscles	**120**	**140**	**160**	**180**	**200**
Avg FPS	59	50	44	39	35

Table 1: *Muscle dynamics performance (without skinning)*

Test B - Skinning: A typical game character is skinned to 21 joints (with each skin vertex affected by a maximum of 3 joints) and 0-80 muscles (with each skin vertex affected by a maximum of 4 muscle vertices). As before, the number of joints does not affect the performance of the system. We used up to 20,000 skin vertices and subjected the character to constant motion for 20 seconds. The results are as follows:

No. of Muscles	**5000 vertices**	**10000 vertices**	**15000 vertices**	**20000 vertices**
0	146	105	68	58
10	92	53	35	29
20	81	48	32	27
30	73	43	30	25
40	67	41	28	24
50	62	38	27	23
60	59	36	26	22
70	57	34	25	21
80	55	33	24	20

Table 2: *Average muscle simulation and skin deformation performance in FPS for different skin mesh resolutions*

The case with 0 muscles serves as a control, to indicate the cost of rendering and skeletal deformation without muscles. As shown in the results, we are able to get real-time performance with 70-80 muscles (with 72 vertices each). Thus we have already achieved efficient results using the CPU alone.

Test C - Appealing Deformation In Real-Time: We have achieved a full-body muscle representation that deforms a character's skin in real time without any significant artifacts, and that too, without any hardware acceleration. The real-time executable demo of an energetic virtual martial artist accompanying this paper [6] comprehensively validates our claims. We created a character with 3671 vertices and skinned it to 21 joints and 64 muscles. All other muscle and influence settings were identical to Test B. We then tweaked the necessary settings such as the dynamics and muscle size, as well as assigned joint/muscle influence weights to the skin vertices. We achieved appealing skin deformation which runs in real-time at about 60-70 fps. We have also simulated diverse character physiques resembling *Hulk, Gollum and Jellyman,* all from the same anatomical model, simply by tweaking global bulge, stiffness, damping and muscle weight-scaling parameters. As shown in Fig. 7, our simplified curved axis representation is able to model the major muscles of a human.

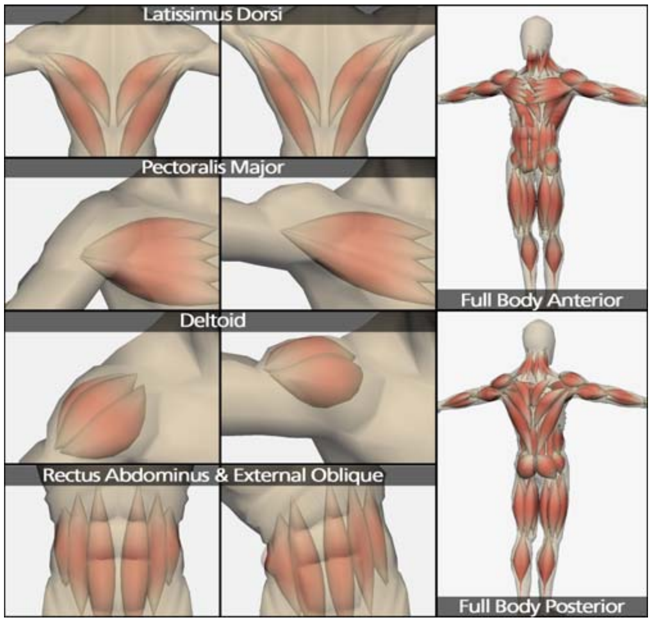

Figure 7: *Approximation of different muscle shapes and deformation with only fusiform muscles*

6 Discussion and Future Work

Quite a few papers have been presented since the late 90's on real-time muscle deformation. Yet, we have not seen their results filtering into commercial games. We feel this is due to several unaddressed limitations: a) The ease of muscle design and parameter specification; b) Stability of simulation and deformation; c) The net complexity of deforming the muscles and skin for a full avatar in real-time. Let us first pick up all the related threads of research that this paper builds on.

We will then discuss exactly how, why and what we simplified. Hopefully, this will shed better light on the value of our work and prove useful to the real-time graphics community.

Related Work: The use of procedural surfaces to model simplified muscles [11] was an exciting step forward from previous Finite Element deformation methods. The ellipsoidal fusiform model seemed perfect for interactive applications, and was naturally enhanced by [10] [2] [17]. Notable enhancements include: non-linear action axis; scaled irregular cross sections for volume preservation; local coordinates to facilitate proper deformation; and a network of 1D mass-spring dynamics. These pieces of work were inspired by the need to model the superficial aesthetics of deforming muscles, without true volume dynamics. An implicit skin calculated from muscle contours [12] was sampled and re-meshed every frame [1]. Our paper has experimented with greater simplifications and we report that things still look good, and perform much faster.

Specifically, we have simplified four aspects: a) Representation of muscle action curve; b) Calculation of local coordinate frames; c) Usage of single mass-spring dynamics; d) Approximation of wrapped skin deformation.

Muscle Representation: We have seen how effectively [11] overlapped a bunch of fusiform shapes to approximate more complex human muscles like the Pectoralis Major. They achieved this with linear action axis. A curved axis could alleviate severe intersections of overlapped fusiforms, where some muscles in the bunch could be slightly "curved over" others. However, we feel that the cubic spline action curve in [17] is an overkill and the piecewise linear segments in [2] may yield discontinuities during dramatic flexion. The quadratic Bézier yields smooth uncomplicated curvature, at a more economical cost than [17]. It also exposes a free hanging center control point that can be conveniently hooked up to a single spring. Lastly, we have achieved more expressive volume deformation and taper with our scaled sinusoidal function.

Local Coordinate Frame at Muscle Vertices: Since the muscle surfaces are being driven by an underlying curve in [2] [17], both pieces of work stress on careful derivation of the local axes at each muscle vertex. Their interest was to position and render the muscle surfaces properly. We additionally use these coordinate frames to implement smooth skinned deformation for the character mesh. We replace two second-order differential equations and a dot-product associated with the rotation minimizing frame in [17] with just two cross-products. By allowing users to make judicious selections of the reference up vector, which is standard practice for a lot of graphical tools, we gain significant performance with predictable stability and minor additional specification. Since the tested skinned character meshes deform smoothly based on the relative transformation of these coordinate frames without any glitches or flips, it proves the stability of our simplified local coordinate frame derivation.

Muscle Mass-Spring Dynamics: Muscles have been traditionally modeled as a set of mass points connected together with springs, where elasticity, curvature and constraint forces could be exerted on each point [10]. We feel that this kind of detail is more relevant to medical simulation, and there is a lot of scope for simplification. The authors themselves pointed out that since a lot of muscles are covered by others, and all of them are covered by the skin, it does not really make sense to have a high degree of accuracy in the muscle representation to reflect the global animation. Taking up this point as a challenge, we decided to use only one mass point, to investigate how badly it affects the net skin deformation. We are pleasantly surprised to see that using standard mass, stiffness and damping settings for all the muscles, we could achieve a desirable degree of lag and oscillation that varies between heavy fats to taut muscular tissue. Similar dynamics settings for muscle bunches result in minimal intersection artifacts during animation, and thus remove the need for collision avoidance. Of course the soft-body effect is more global than realistic tissue constrained between bones and skin, but the low cost of specification, control and simulation is quite attractive. Lastly, we have effectively tried out the cheaper Verlet integration method compared to 4th order Runge Kutta integration [10].

Skin Deformation: The elastic skin membrane model [16] captures skin sliding and stretching with springs between every skin vertex, but is too expensive for real time (0.04fps on SGI R4400 quad CPU). The only muscle-based method that has been reported as real-time [1] (30fps for 14K vertices on R10000 Octane) is based on implicit surface generation from underlying ellipsoidal meta-balls [12]. We remove the online cost of intersecting rays with ellipsoid contours and re-meshing every frame, by borrowing from the metaphor for skeletal deformation. Here, the per-muscle-vertex influence weights are pre-computed at bind pose, and scaled with per-muscle influence weights (user defined or procedurally calculated). Up to 8 matrix multiplications and additions are done per skin-vertex, after updating the local coordinate frames for every muscle vertex. The inherent SIMD properties of both muscle formation and subsequent skin deformation promise easy mapping onto GPUs, something that is not so easily done using the method in [12]. Furthermore, though the implicit surface method is effective for characters with smooth cylindrical segments, there may be problems for meshes with folds and complicated topology. Smooth skinning has been proven to work well with arbitrary topologies and thus is widely adopted in the industry. We have successfully demonstrated for the first time that approximate wrapping of underlying tissue surfaces yields believable results at a fraction of the cost of existing approaches.

In summary, we believe that we have addressed several major limitations that have restricted use of anatomy based skin deformation in real time applications. This is due to a combination of simplified representation, intuitive design workflow, and potential for hardware acceleration. We have demonstrated our system with a full-body game resolution character that performs vigorous motions with pleasing deformations at 60-80 fps on a standard PC with 2-3 GHz CPU and 1-2GB RAM.

As future work, we would like to use real values for muscle mass and stiffness from biomechanics literature, to build a knowledge base of generic musculature. We could try out a sparse network of springs to maintain more localized distortion for prominent muscles. We could borrow from existing weight transfer methods to reduce additional manual labor for painting muscle weights. We are currently working on a modular anatomy-based rigging system, to facilitate the building of arbitrary characters. Lastly, we plan to implement hardware acceleration using the new Shader Model 4.0 architecture.

We are optimistic that this work will be adopted by the real-time content production industry, enabling appealing anatomy-based characters in next generation games and interactive media.

7 References

[1] AUBEL A., BOULIC R., THALMANN D.: Realtime display of virtual humans: Levels of details and impostors. Circuits and Systems for Video Technology, IEEE Transactions on (2000), pp 207–217.

[2] AUBEL A., THALMANN D.: Interactive modeling of the human musculature. Computer Animation (2001), pp 167–255.

[3] ASHRAF G., ZHOU J. Y.: Hardware accelerated skin deformation for animated crowds. Multimedia Modeling (Jan 2007), pp 226–237.

[4] CHADWICK J. E., HAUMANN D. R., PARENT R. E.: Layered construction for deformable animated characters. SIGGRAPH (1989), pp 243–252.

[5] JAMES D., PAI D.: DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. SIGGRAPH (2002), pp 582–585.

[6] LEE K. S., ASHRAF G.: Demo of real-time muscle deformation. http://cg.skeelogy.com/research/simplified-muscle-dynamics.php (2007).

[7] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. SIGGRAPH (2000), pp 165–172.

[8] MOCCOZET L., THALMANN N. M.: Dirichlet free-form deformations and their application to hand simulation. Computer Animation (1997), pp 93.

[9] NEALEN A., MULLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. EUROGRAPHICS (2005), pp 289–301.

[10] NEDEL L. P., THALMANN D.: Real time muscle deformations using mass-spring systems. Computer Graphics International (1998), pp 156–165.

[11] SCHEEPERS F., PARENT R. E., CARLSON W. E., MAY S. F.: Anatomy-based modeling of the human musculature. SIGGRAPH (1997), pp 163–172.

[12] SHEN J., THALMANN D.: Interactive shape design using metaballs and splines. Implicit Surfaces (1995), pp 187–196.

[13] SHEPPARD, J.: Anatomy: A complete guide for artists. Dover Publications (1992).

[14] TURNER R., THALMANN D.: The elastic surface layer model for animated character construction. Computer Graphics International (1993), pp 399–412.

[15] WU X., DOWNES M. S., GOKTEKIN T., TENDICK F.: Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. Computer Graphics Forum (2001), pp 349–358.

[16] WILHELMS J., GELDER A. V.: Anatomically based modeling. SIGGRAPH (1997), pp 173–180.

[17] ZUO L., LI J. T., WANG Z. Q.: Anatomical human musculature modeling for real-time deformation. WSCG (2003).