Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2007)
D. Metaxas and J. Popovic (Editors)

Deformation Styles for Spline-based Skeletal Animation

Sven Forstmann and Jun Ohya [1] Artus Krohn-Grimberghe[2] Ryan McDougall[3]

[1]GITS Faculty, Waseda University, Tokyo, Japan
[2]Karlsruhe University, Karlsruhe, Germany
[3]3D Incorporated, Yokohama, Japan

Abstract

We present a novel skinned skeletal animation system based on spline-aligned deformations for providing high quality and fully designable deformations in real-time. Our ambition is to allow artists the easy creation of abstract, pose-dependent deformation behaviors that might directly be assigned to a large variety of target objects simultaneously. To achieve this goal, we introduce the usage of deformation styles and demonstrate their applicability by our animation system. We therefore enhance spline-skinned skeletal animation with two sweep-based free-form-deformation (FFD) variants. The two FFD variants are pose-dependent, driven by three textures and three curves, which can be designed by the artist. As the three textures are similar to height-maps, their creation is very intuitive. Once designed, the deformation styles can be directly applied to any number of targets for imitating material behaviors of cloth, metal or even muscles. Our GPU based implementation shows promising results for real-time usage, as about 30 Million vertices per second can be animated. The basic spline-skinning even reaches more than twice the speed and gets close to the performance of skeletal subspace deformation (SSD). Furthermore, our method can easily be combined along with other existing deformation techniques as pose space deformation or SSD.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object ModelingHierarchy and geometric transformations I.3.5 [Computer Graphics]: Computational Geometry and Object ModelingSplines I.3.7 [Computer Graphics]: Three-Dimensional Graphics and RealismAnimation

1. Introduction

Character animation is one of the major components in most digital productions, including cinematic productions as well as interactive applications as video games. To cover the required deformation issue, many approaches have been suggested over time, ranging from free-form-deformation (FFD) based techniques over skeletal methods to advanced algorithms, which also take a mesh's topology, physical constraints and even authentic data from laser-scans into account. In the recent years, especially one method has widely found acceptance to cover skeletal animation: the so-called linear blend or matrix skinning, also known as skeletal subspace deformation (SSD). It is the most popular method among all authoring tools and interactive applications. The key of SSD's success easy to explain: it is simple and well-balanced—in terms of quality, speed and implementation complexity.

However, it is far from perfect, since its deformations expose the well-known candy-wrapper effect for twisting operations and collapsing geometry while bending. This issue is very annoying for artists and has inspired many researchers to come up with new solutions and alternative methods. Unfortunately, most inventions have not found their way into practical usage—and the reason is easy to explain: Often, methods are either very complex, so it is hard to accomplish their implementation in a reasonable time, or their computation is simply too demanding. Especially in real-time applications as for example video games, fast evaluations are mandatory. However, also in offline rendering systems, as for cinematic productions, time is an important and non-negligible factor.

We are therefore concerned with a practical solution that provides high quality deformations, easy handling, a fast and stable evaluation, hardware accelerator compliance, an easy

Copyright © 2007 by the Association for Computing Machinery, Inc.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212) 869-0481 or e-mail permissions@acm.org.
SCA 2007, San Diego, California, August 04 - 05, 2007
© 2007 ACM 978-1-59593-624-4/07/0008 $ 5.00

implementation and compliance to existing skeletal animation methods.

A further and maybe even more important issue for an animation system's success is the degree of freedom to adjust a deformation. Here, especially pose-space-deformation (PSD) has made an important contribution to allow artists the design for each pose of an animation individually. However, PSD has a significant restriction: If an artist has enhanced a certain pose—for example by modeling a joints muscle- or cloth-like behavior—it is not possible to reuse this particular behavior for any other joint. Our main goal is therefore to create a system allowing the abstract and reusable design of a joints deformation behavior. This property allows one deformation behavior to be applied immediately to a large number of various targets, reducing an artist's work considerably.

2. Concept and Design

For creating the animation system, we carefully took our design goals into account. As for the foundation of our skinned skeletal animation system, we decided to use spline-aligned deformation [SF98]. It naturally provides high-quality bend and twist deformations without exposing unwanted artifacts like SSD. Another advantage is their fast and stable evaluation—an important property for our desired implementation on the GPU. As for the spline function, we use a special polynomial based spline with variable exponent that is dependent on only three control-points for highest performance.

In order to achieve reusable pose dependent designs, our central aim, we introduce the usage of deformation styles. The idea is to allow the abstract design of a deformation behavior, without requiring any knowledge about the underlying geometry it is applied to. This is one step forward beyond PSD, since PSD always has to be aware of the geometry it is applied to. To solve this problem efficiently, we decided to utilize two sweep-based FFD variants, which are attached to each joint.

The first of our two methods is based on a high-resolution, radial FFD grid, which is wrapped around the spline. It allows the creation of high resolution concentric deformation effects such as for metal, cloth or even muscle bulges. It is driven by three scale textures, which are weighted blended, depending on the spline's pose. The three textures are utilized for frontal, lateral and radial scaling. Our second FFD variant is a rectangular scale envelope that is supposed to allow a simple definition of more general scalings. The artist therefore draws three outlining curves for the frontal, lateral and radial direction. Goals of the second variant include the design of folds to prevent self-intersections near the elbow or the modeling of major lateral bulges for soft-bodies.

3. Contributions

Our skinned skeletal animation system can list the following four contributions:

Reusable deformation-styles: The presented system allows the abstract design of pose-dependent deformation behaviors for the imitation of complex material deformations. Once designed, a style can immediately be applied to an arbitrary number of joints simultaneously. This saves time for the artist during the modeling phase, and may further save memory during run-time, as each style needs to be stored only once.

Fast computation: We achieved a very efficient GPU based implementation of the spline-based skinned skeletal animation system, which outperforms the GPU based implementation described in [FO06] by factor three for the basic spline skinning without deformation-styles. Our algorithm's speed without deformation-styles further gets close to the performance of SSD, which is often referred to as the fastest skinned skeletal animation system. For the case of deformation styles being included, our system still shows a very competitive speed, as the vertex deformation rate remains high at 30 Million vertices per second on our testing system.

Simplicity: Our algorithm is based on simple mathematics and does not contain complex data structures or the requirement of comprehensive mathematical libraries. We assume the implementation to be feasible in a reasonable time without complications. Furthermore, the designed deformation styles cover the complete pose-space of a joint and hence avoid the usage of radial basic functions for the interpolation between certain poses.

Memory usage: In contrast to the GPU-based spline-aligned skeletal animation system [FO06], where each vertex and each normal of the animated mesh were required to be stored three times (once for each spline), our novel algorithm requires them to be stored only once.

4. Related Work

Over time, many methods have appeared to achieve the animation of characters, where the most important methods can be divided into FFD [SP86], SSD [MTLT88], shape blending and spline aligned deformations [SF98]. They form the foundation for many subsequent research approaches and reappeared in countless variants and combinations since their initial invention. In order to increase the deformation quality and realism for skeletal animations, various methods have been suggested.

Methods that directly improve upon SSD are [KZ05] and [KCOZ06]. They change the interpolation domain from matrices to quaternions or even dual quaternions. This successfully avoids effects as collapsing geometry by preserving a high computational speed.

A different and more flexible approach is to use spline

© Association for Computing Machinery, Inc. 2007.

aligned deformations and apply them in a skinned way to character animation. Two methods to achieve this are [FO06] and [YSZ06]. As for [YSZ06], the focus lies especially on the modeling case, as their implementation is a plugin for the commercial software Maya, while [FO06] focusses on the application in real-time systems by extensive usage of the GPU. A related approach in this context is also [CSM05], where curve skeletons are introduced and discussed in general.

A method advancing on FFD is sweep-based FFD [YK06]. It provides the ability to efficiently model radial deformations by allowing the user to edit cross-sections along spline-curves. In the presented system, we utilize two variants of sweep-based FFDs to apply our deformation styles to the geometry. A further sweep-based algorithm is [HYC*05]. The authors use sweep-based deformation to create skinned skeletal animation. To a certain degree, this allows the creation of customized deformations, as the user can model muscles, which are taken into account during the deformation.

To provide more realistic deformations, advanced methods such as [BMGK06], [MG04], [CGC*02] and [PJS06] have been invented. They allow the adjustment of the material stiffness, which directly affects the deformation. Other methods, such as [ZHS*05] construct an inner graph to preserve the mesh's inner volume while deforming.

A method allowing the reuse of deformations is [SP04], where the animation of one mesh may drive the deformation of a another, similar mesh. Different from ours, their method is targeted on reusing the complete deformation, while our method is especially focused on reusing a deformation's behavior.

Example based methods allow the pose-dependent modification of animations. They have been introduced by pose-space-deformation (PSD) [LCF00] and advanced by [TRN06] and [SCFRC01]. PSD basically allows the artist to individualize particular poses, where intermediate poses are calculated by interpolation. In our case, pose-dependent deformations can also be modeled by the artist, but in a different way. Instead of directly modifying a certain vertex in a certain pose, our method targets at a more abstract design, covering all poses at once.

Different from the former approaches are cloth simulations to provide realistic deformations and surface details. A comprehensive overview can be found in [MTV05]. Related to cloth and somehow similar to our method, also [VMT99], [JJT05] and [LC04] allow the design of surface details for the animation. However, different from our design, all three methods apply surface details in direction to the surface normal instead of using FFD, and dependent on the local mesh deformation instead of utilizing the skeleton's pose. A method different from PSD to apply surface details based on the pose, is [DJW*06], where wrinkles are generated procedurally to create cloth-like behaviors.

5. Splines

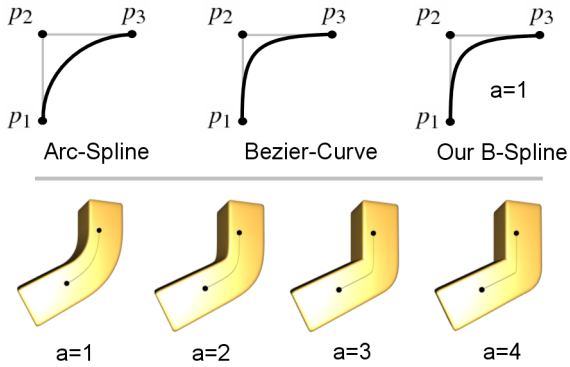

Figure 1: *Spline functions: In the upper row, we compare our short-listed spline functions. The lower row shows the ability of our spline to adjust the stiffness by parameter a.*

In order to achieve spline-aligned deformation, we first have to find a suitable spline-function. For making a decision, we short-listed the trigonometric Arc-spline and the polynomial Bézier-spline. Both can be evaluated very fast, which is important for our GPU-based implementation. In order to provide highest performance, we have further decided to limit the number of control points to three. However, both functions do not naturally allow a modification of the curve stiffness without adding further control points. In our approach, the second control point basically represents a joint's rotation center, and hence, additional control points will complicate the computation. Since we want to keep a simple and easy handling, we remained using three control points $p_{1,2,3}$ and modified the basic Bézier spline function f_b to create a new spline function f_m, providing an additional parameter a for a continuous variable adjustment of the spline's stiffness (Fig.1):

$$\Delta_{12} = p_2 - p_1$$
$$\Delta_{23} = p_3 - p_2$$

$$\forall x \in \mathbb{R} \mid x \in [0,1]$$
$$\forall a \in \mathbb{R} \mid a \geq 2$$

Conventional Bezier curve:

$$f_b(x) = p_1 \cdot (1-x)^2 + p_2 \cdot (2 \cdot x \cdot (1-x)) + p_3 \cdot x^2$$
$$f_b'(x) = p_1 \cdot 2 \cdot (x-1) + p_2 \cdot (2-4x) + p_3 \cdot 2 \cdot x$$

Our modified Bezier curve:

$$f_m(x) = p_1 + \Delta_{12} \cdot (1-(1-x)^a) + \Delta_{23} \cdot x^a$$
$$f_m'(x) = \Delta_{12} \cdot a \cdot (1-x)^{a-1} + \Delta_{23} \cdot a \cdot x^{a-1}$$

5.1. Spline Aligned Deformation

In order to apply our spline as a geometric deformation, it is necessary to construct a complete coordinate-system around

© Association for Computing Machinery, Inc. 2007.

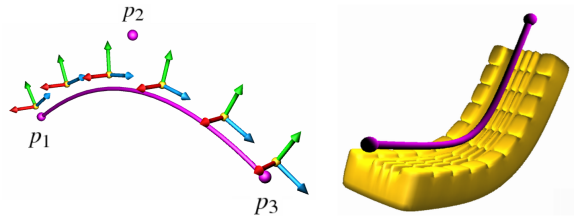

Figure 2: *The spline coordinate system: The left side shows the spline function in violet together with the spline's coordinate system. The spline's tangent is indicated in blue, the normal in red, the binormal in green and the origin of each coordinate system in yellow. On the right-hand side, we can see an example deformation.*

it, the so-called Frenet-frame, as shown in Fig.2. We can compute a complete orthonormal basis b for each position of the spline, consisting of origin b_O, normal b_N, tangent b_T and binormal b_B as follows:

$$
\begin{aligned}
b_T(x) &= f'_m(x) \\
b_N &= \Delta_{12} \times \Delta_{23} \\
b_B(x) &= b_N \times b_T(x) \\
b_O(x) &= f_m(x)
\end{aligned}
$$

$$B = [b_N \mid b_B \mid b_T \mid b_O]$$

$$B = \begin{bmatrix} b_N.x & b_B.x & b_T.x & b_O.x \\ b_N.y & b_B.y & b_T.y & b_O.y \\ b_N.z & b_B.z & b_T.z & b_O.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The origin of the coordinate frame is simply the spline function f_m itself. Then, the tangent is equal to the spline's derivative f'_m, the normal can be pre-calculated as it is perpendicular to the three control points $p_{1,2,3}$ and finally the binormal can be computed as cross-product of the normal and the tangent vector. We can further create the 4x4 transformation matrix B by arranging our four computed vectors as column vectors.

5.2. Spline Binding

Prior to the deformation, we have to map all vertices of our target mesh perpendicular to a definite position x on the spline. We achieve this by utilizing a binary search algorithm, starting at $x = 0.5$ as it is shown in Fig. 3. In order to determine the search direction for x at each step, we can utilize the perpendicular constraint based on the scalar product $\langle \cdot \mid \cdot \rangle$, the vertex v and the plane defined by b_T and b_O as follows:

$$\langle b_T(x) \mid v - b_O(x) \rangle = \begin{cases} < 0, & \text{v lies in front of the plane} \\ = 0, & \text{v lies in the plane (solved)} \\ > 0, & \text{v lies behind the plane} \end{cases}$$

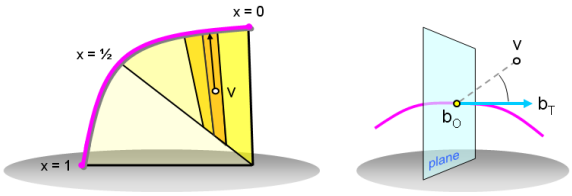

Figure 3: *The binding process: The left side shows the perpendicular mapping of vertex v to the spline by using binary search. The right side shows a way to determine the search direction in each step.*

For further computations, we introduce the spline-basis representation v' of v according to the spline's matrix B:

$$v' = B^{-1} \cdot v$$

6. Deformation Styles

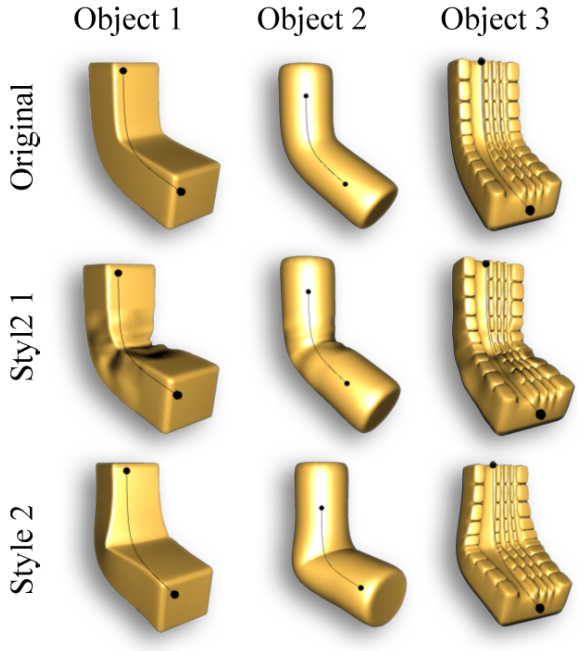

Figure 4: *Deformation Styles: We applied two different deformation styles equally to three objects. Style 1 shows the radial, texture based approach while Style 2 is an example for the rectangular, curve-based method.*

In order to allow a more flexible adjustment of the spline-aligned deformation, we are introducing the use of deformation styles. They allow the pose-dependent modeling of a joint's deformation which can be used to create material behaviors of metal, cloth or muscles. Each style is created from the combination of two pose-dependent FFD variants,

© Association for Computing Machinery, Inc. 2007.

where each variant has its own advantages that cannot be replaced by the other. The first method applies a radial scale envelope (Fig.4, Style 1) while the second method is using a rectangular scale envelope (Fig. 4, Style 2).

6.1. Radial Scale Envelope

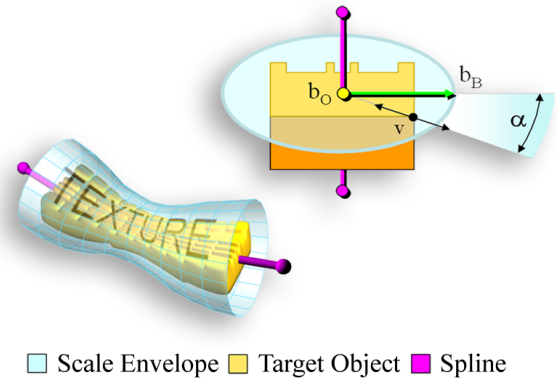

Figure 5: *Radial Scale Envelope: The lower left side shows an example envelope while the upper right side shows the concentric scaling of v in relation to the spline.*

The Radial Scale Envelope is shown in Fig.4 as Style 1 and designed to allow high resolution skin deformation effects such as folds and wrinkles, for example. The algorithm basically applies the deformation by concentrically scaling a vertex v of the target object in relation to the spline origin b_O, as it is shown in Fig.5. The scaling is determined by a radial scale function S_{rs}, which depends on the two-dimensional position (x,α) on the envelopes surface and the pose of the joint. The pose is defined by the joint's twist γ and the joint's bend-angle β, where β is based on the two vectors Δ_{12} and Δ_{23}. The angle α denotes the angle between the vertex v and the spline's binormal b_B in relation to the spline's origin b_O. We define our deformation function D_{rad} to evaluate the deformed vertex as follows.

$$\forall \alpha, \beta \in [0, \pi]$$
$$\forall \gamma \geq 0$$

$$D_{rad}(v') = v' \cdot S_{rs}(x, \alpha, \beta, \gamma)$$

Since the scaling needs to be applied in spline space, we are using v' instead of v. This simplifies our calculation, as the origin in spline-space is b_O and the multiplication of v' by any scalar is equal to scaling v' in relation to b_O.

6.2. Radial Scale Function and Textures

The scale function S_{rs} is the heart of the radial deformation and computes the scaling based on three scale textures, shown in Fig.6, upper row. The first texture T_f is used for frontal scaling, the second for lateral (T_l) and the third for radial (T_r). The weight distribution for the three textures can be seen in Fig.6, lower row. The weight for the frontal texture w_f represented by red, for the lateral w_l by green and for the radial w_r by blue. In order to compute the scale factor for a certain vertex v, we sample all three textures at the texture-coordinate (x,α/π), evaluate the three pose-dependent weights $w_{f,l,r}$ and compute the scaling result S_{rs} as follows.

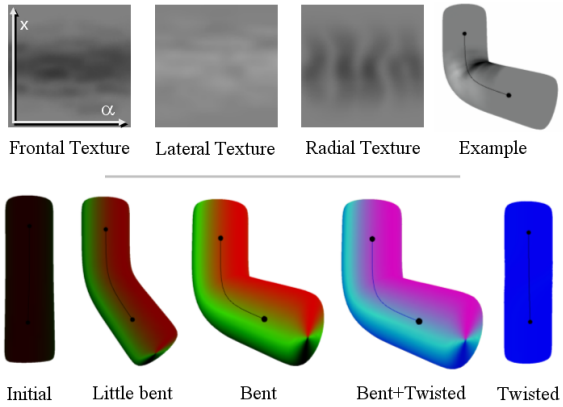

Figure 6: *Scale Textures: The upper row shows the three scale textures that were used to create Style 1 in Fig.4 and an example object where the textures are applied. The lower row shows the pose-dependent weight calculation to apply the three textures, where red corresponds to the weight of the frontal scale texture, green to the lateral and blue to the radial.*

$$\forall w_f, w_l, w_r \in [0, 1]$$
$$\forall w_{sum}, t_f, t_l, t_r, T_f, T_l, T_r \geq 0$$

$$w_f = \max(-\cos(\alpha), 0) \cdot \beta$$
$$w_l = |\sin(\alpha)| \cdot \beta$$
$$w_r = \gamma$$
$$w_{sum} = w_f + w_l + w_r$$

$$t_f = w_f \cdot T_f(x, \alpha/\pi)$$
$$t_l = w_l \cdot T_l(x, \alpha/\pi)$$
$$t_r = w_r \cdot T_r(x, \alpha/\pi)$$

$$S_{rs} = (t_f + t_l + t_r)/\max(1, w_{sum}) + \max(1 - w_{sum}, 0)$$

The idea of this formula is basically to compute the weight for the frontal and the lateral texture based on the angle α and the radial weight based on the twist angle γ. In our formula, w_{sum} represents the sum of all weights, and $t_{f,l,r}$ the weighted texture samples. In order to preserve unity scaling for identity textures at any pose, our formula meets the following condition:

$$\forall x, \alpha \mid T_{f,l,r}(x, \alpha/\pi) = 1 : S_{rs}(x, \alpha, \beta, \gamma) = 1, \forall \beta, \gamma$$

© Association for Computing Machinery, Inc. 2007.

6.3. Rectangular Scale Envelope

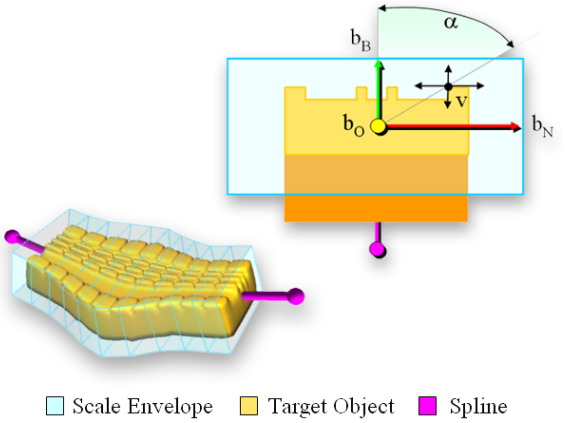

Figure 7: *Rectangular Scale Envelope: The left side shows the application to an example object while the right side shows the scaling of v corresponding to b_B and b_N*

The rectangular scale envelope is shown in Fig.4 as Style 2 and thought to allow the design of the deformations contour. The algorithm basically applies scaling of a vertex v in the two directions b_B and b_N independently, as shown in Fig.7. This is very different from the former radial method that applies a concentric scaling. The purpose of the rectangular approach is the modeling of major frontal folds or lateral effects such as creating bulges. In order to design the deformation, the artist has the freedom to create three functions, which drive the scaling in the frontal, lateral and radial direction.

The scaling is driven by two scaling functions $S_{f,l}$, whereas S_f applies frontal scaling in direction of b_B and S_l the lateral in direction b_N. More specifically, we can write the deformation function D_{rect} as follows:

$$D_{rect}(v') = \begin{bmatrix} v'_x \cdot S_l(x,\beta,\gamma) \\ v'_y \cdot S_f(x,\alpha,\beta,\gamma) \\ v'_z \\ 1 \end{bmatrix}$$

The rectangular scaling is also calculated in spline space, and we therefore use v' instead of v. The spline-space representation v' of v is very handy, as the x-axis in spline space is along b_N and the y-axis along b_B. This allows an easy application of the two scaling functions S_f (frontal) and S_l (lateral). As before, the angle α defines the angle between v and b_B in relation to b_O. The two angles β and γ define the spline's pose.

6.4. Rectangular Scale Functions

For designing the rectangular scale envelope, the artist can define three curves $C_{f,l,r}$, which directly affect the contour of the deformed object in frontal (C_f), lateral (C_l) and radial (C_r) direction. Fig.8, uppper row, shows the three curves that have been used to create Style 2 in Fig.4. The three curves are basically applied to the two scaling functions $S_{f,l}$ as the textures $T_{f,l,r}$ have been applied to S_{rs} before. The only difference is, that we have to separately treat frontal and lateral scaling. We can further simplify the calculation of the lateral weight w'_l as it is not depending on α anymore and write the formula for S_f and S_l as follows.

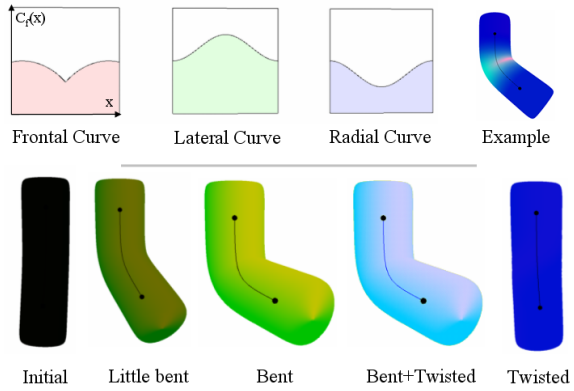

Figure 8: *Rectangular Scaling: The upper row shows the three scale functions that were used to create Style 2 in Fig.4. In the example object on the right, we can see the scale factors while bending. Green represents the lateral and red for the frontal scaling. The lower row shows the pose-dependent weights. Red corresponds to the weight of the frontal scale function, green to the lateral and blue to the radial.*

$$\forall C_f, C_l, C_r, c_f, c_l, c_r, w'_l, w_{sum1}, w_{sum2} \geq 0$$

$$w'_l = \beta$$
$$w_{sum1} = w_f + w_r$$
$$w_{sum2} = w'_l + w_r$$

$$c_f = w_f \cdot C_f(x)$$
$$c_l = w'_l \cdot C_l(x)$$
$$c_r = w_r \cdot C_r(x)$$

$$S_f = (c_f + c_r)/\max(1, w_{sum1}) + \max(1 - w_{sum1}, 0)$$
$$S_l = (c_l + c_r)/\max(1, w_{sum2}) + \max(1 - w_{sum2}, 0)$$

To get an idea of the pose-dependent weights that are used to define the importance of each curve C, we include a couple of example poses in Fig.8, lower row. They demonstrate the weights by color indication. For the computation of S_f and S_l, we introduce three new variables in addition to the previous formula of S_{rs}. The first is w'_l, our new lateral weight, the second w_{sum1} to store the frontal weight and the third w_{sum2} to store the radial weight. As for the radial scaling function S_{rs}, our frontal and lateral scaling functions S_f and S_l also have to preserve unity scaling ($S_{f,l} = 1$) for identity curves ($C_{f,l,r} = 1$).

© Association for Computing Machinery, Inc. 2007.

7. Deformation Styles and Spline Skinning

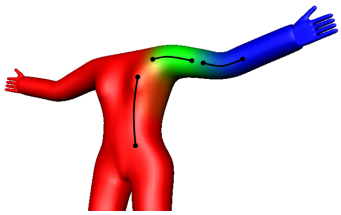

Figure 9: *Spline skinning: The colors indicate the weight for each spline. Red is related to the body spline, green to the shoulder and blue to the elbow.*

To compute our final deformed vertex v_f for n influencing splines B_i, we can simply associate our prior deformations D_{rect} and D_{rad} as follows:

$$v_f = \sum_{i=1}^{n} w_i \cdot B_i \cdot D_{rect}(D_{rad}({B'}_i^{-1} \cdot v))$$

For the formula, we use two different spline bases. The basis of the actual pose is defined by B, while B' defines the basis while binding. We successively apply our radial deformation first, then our rectangular scaling and the spline deformation last, before using the skinning weights w_i for computing the final result. In transition areas where two or more splines meet, styles are blended automatically depending on the skinning weights. For the basic case, in which only spline skinning is used, we get the well-known formula of SSD by pre-computing B''_i:

$$B''_i = B_i \cdot {B'}_i^{-1}$$

$$v_f = \sum_{i=1}^{n} w_i \cdot B''_i \cdot v$$

The skinning weights are defined by the artist, who can paint them onto the surface as in Fig.9. This is already a common technique in most professional modeling tools such as Maya or 3DS Max. To preserve a correct scaling, all weights w_i must sum up to one.

8. Implementation Details

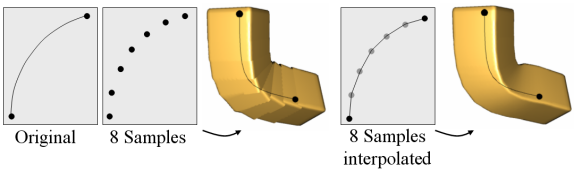

Figure 10: *Spline discretization: Pre-computing all splines is one of the key improvements in our implementation to increase the speed.*

In order to achieve an efficient GPU-based implementation, we had to introduce several optimizations and restrictions to our algorithm. Our first optimization is to separate the spline computation from the per-vertex deformation. In our initial implementation, we were required to evaluate three spline bases per vertex, which was quite demanding. Therefore, we outsourced the computation of the spline basis B to a preprocessing step. There we sample the spline at a fixed number of positions and store the result into three float textures. We require three textures since we need to store the complete basis B consisting of 12 variables at each position, which is equal to three RGBA pixels.

Now, for deforming one vertex, it is sufficient to only sample the pre-computed textures at the splines position x, which is much faster. Here we also profit from the graphic cards texture-filtering feature, as we can directly receive the linearly interpolated spline basis. Without this feature, our spline will show aliasing effects as in Fig.10.

To further improve the speed, we limited the number of influencing spline curves per vertex to three and only apply one deformation style to the most relevant spline. This is sufficient for most cases, as the joint centers of different joints are usually distant enough to prevent errors. As for the deformation, we deform each triangle's vertices and surface normals—but we do not include binormals and tangent vectors. Another issue is the computation accuracy. As recent graphic cards show a weakness to 32-Bit floating point texture filtering, we decided to use 16-Bit textures, which still provides enough accuracy for satisfying results. The filtering of 32-Bit textures is often too slow or even unsupported.

9. Limitations

Even though our method has many advantages in the design of high quality deformations, there are also certain limitations. This first is volume preservation. Since our method is completely dependent on the artist's design, it is up to the artist to design a deformation that seems to preserve the volume or one that models a hollow material and not preserves it. Our second issue is related to PSD. Unlike PSD, where each vertex of a target mesh can freely be modeled pose-dependently, our deformation styles can only affect vertices that are close to the joint they are attached to. Their movement is further defined by the constraints of sweep based FFD. Our last issue regards self intersections. The proposed method does not compute or automatically prevent them—however, the artist may create deformation styles that give the impression of an intersection-free deformation.

10. Results

The first results of our approach can already be seen in Fig.4, where two styles are applied to three objects. Since the objects are very different, we can successfully demonstrate the geometry independence of our method. More appealingly, Fig.11 can demonstrate the effectiveness of our algorithm. We applied a metal-like behavior to a cuboid and created snapshots from various poses.

© Association for Computing Machinery, Inc. 2007.

Figure 11: *Metal: This figure shows the animation of designed metal, which smoothly deforms as the pose changes.*

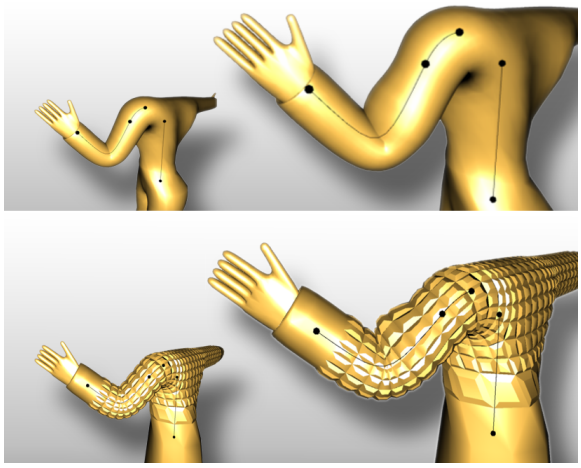

Figure 12: *Muscles: Created muscles can easily be applied to different characters simultaneously.*

In Fig.12, we present the method's ability to design transferable muscle bulges. The left side shows two characters using conventional spline-skinning, while the right side includes our deformation styles. As expected, the surface details of the character in the lower row do not interfere with the applied muscle-style. An example for modeling the deformation of a hollow material can be seen in Fig.13. We improved the realism by displacing the spline origin b_O along the binormal b_B while applying D_{rad} and D_{rect}. The successful prevention of self-intersections by using deformation styles can be seen in Fig.14. It is further an example for designed lateral bulges, whereas all curves are taken from Fig.8. The imitation of cloth is demonstrated by Fig.15, where we clearly can recognize the applied cloth style near the knee region. The textures that have been used in the results were painted using conventional imaging tools. However, for an improved workflow, interactive texture-painting in a WYSIWYG fashion might be advantageous.

For our implementation we used the OpenGL shading language GLSL and the render-to-vertexbuffer technique [Sch06], which can be realized in OpenGL by using the pixel-buffer-object (PBO) extension. The benchmark of our method is displayed in Fig.16, where we tested various algorithm configurations. In case that only basic spline skinning is used, our algorithm gets close to SSD and reaches the pleasing speed of 85 Million vertices per second. If our deformation methods are added step by step, the speed decreases to 30M vertices per second, which is still satisfactory for real-time applications. The speed is not equal for all objects, which might be caused by an implementation issue of our method. We switch the rendering context of the frame-buffer-object (FBO) for each object, which is a relatively expensive operation. However, there is no direct relation to computing the deformation.

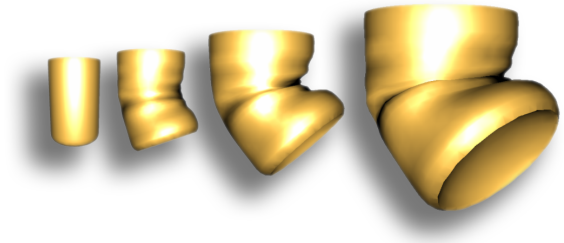

Figure 13: *Hollow materials: Here an animation of crunching an empty can.*

To measure our method's performance more detailed, we created a general timing-breakdown, shown in Table 1. We investigated the timing for two scenes—cuboid and cylinder. For each of the two objects, we also computed the skinning based on three spline-curves, each approximated by 32 samples. The scale texture resolution was 32x64 pixels. As for the timing breakdown, we can see that the major time consumption is caused by our two deformation methods (radial and rect deform) and followed by the spline-skinning (spline deform). Precalculating the spline matrices requires surprisingly less time. The step called *Copy FBO to VBO* is required by the OpenGL architecture and does a complete copy of all vertices from the FBO to the vertex-buffer-object. The final step for rendering the scene is relatively fast, as we did not include complex lighting evaluations. For benchmarking, we created character scenes consisting of about 1 Million vertices to get representative results. In Fig.16, the number in front of each character indicates the number of times it was duplicated to reach 1M vertices.

© Association for Computing Machinery, Inc. 2007.

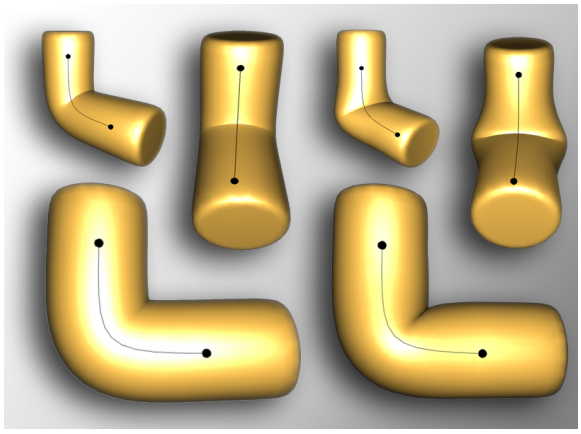

Figure 14: *Self intersections: Demonstrated is the efficient removal of self intersections (right) as well as modeled lateral bulges. The the non-style version is shown on the left.*

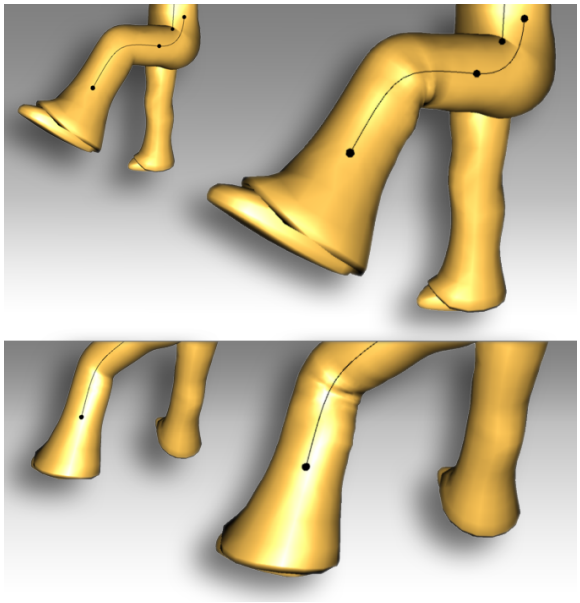

Figure 15: *Cloth: This example shows the algorithm's ability to imitate cloth-like wrinkles.*

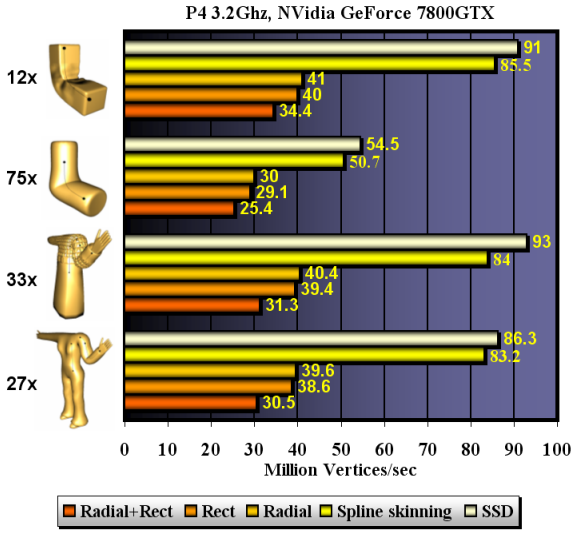

Figure 16: *Benchmark results: Spline skinning indicates basic spline aligned deformation, Radial adds D_{rad}, Rect adds D_{rect} and Rect+Radial adds both.*

Table 1: *Timing breakdown: The table shows the detailed timing of our GPU-based implementation. Cuboid and Cylinder refer to the first and second object in Fig.16.*

Scene	12x Cuboid	75x Cylinder
Vertices Object	12528	92526
Vertices Scene	12 x 12k	75 x 92k
	= 0.94M	= 1.11M
Spline Samples	12 x 3 x 32	75 x 3 x 32
	= 1152	= 7200
Timing (Scene)		
Spline Matrices	0.3 ms	0.3 ms
Deform (Radial)	7.8 ms	8.7 ms
Deform (Rect)	12.6 ms	9.7 ms
Deform (Spline)	5.5 ms	7.9 ms
Copy FBO to VBO	1.9 ms	4.9 ms
Render Scene	3.4 ms	4.5 ms
Total	31.7 ms	36.1 ms
Vertices/sec	34.4M	25.4M

11. Conclusion

In our initial research on deformation styles for character animation, we have presented an efficient implementation by utilizing sweep-based FFD, applied to spline skinning. Deformation styles have the advantage of allowing abstract design of a deformation which can be used over and over again for various animations and characters. In larger productions, this can be very useful, to save artist's time by creating individualized deformations which have to be applied to a multitude of different characters.

Our proposed method can further exhibit an excellent performance, as the final implementation is capable of rendering massive scenes with customized deformations in real-time. The implementation also contributes in the field of spline-based skinned skeletal animation, as our method significantly outperforms previous approaches. Since we apply the basic weight-based skinning method of SSD, integration in existing animation systems can be easily achieved. We believe that the idea of deformation styles can pave the way for further fruitful research and also can have many practical applications.

© Association for Computing Machinery, Inc. 2007.

12. Future work

As our method only shows one way to achieve abstract pose dependent design, further research might come up with different or more general approaches. One idea might be for example the utilization of volumetric scale textures to achieve a more flexible deformation. Another one could be, to pass the deformation further detailed to the surface by utilizing bump- or displacement maps, which is in particular of interest for low-poly models.

References

[BMGK06] BOTSCH M., M.PAULY, GROSS M., KOBBELT L.: Primo: Coupled prisms for intuitive surface modeling. In *Eurographics Symposium in Geometry Processing* (2006), pp. 11–20.

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIC Z.: Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: ACM SIGGRAPH 2002 Papers* (2002), pp. 586–593.

[CSM05] CORNEA N., SILVER D., MIN P.: Curve-skeleton applications. In *Visualization, 2005. VIS 05. IEEE* (2005), pp. 95–102.

[DJW*06] DECAUDIN P., JULIUS D., WITHER J., BOISSIEUX L., SHEFFER A., CANI M.-P.: Virtual garments: A fully geometric approach for clothing design. *Computer Graphics Forum (EG'06 proc.) 25*, 3 (sep 2006), 625–634.

[FO06] FORSTMANN S., OHYA J.: Fast skeletal animation by skinned arc-spline based deformation. In *Eurogrpahics Short-Papers* (2006), pp. 1–4.

[HYC*05] HYUN D.-E., YOON S.-H., CHANG J.-W., SEONG J.-K., KIM M.-S., JÜTTLER B.: Sweep-based human deformation. *The Visual Computer 21*, 8-10 (2005), 542–550.

[JJT05] JING F., JONEJA A., TANG K.: Modeling wrinkles on smooth surfaces for footwear design. *Computer-Aided Design 37*, 8 (2005), 815–823.

[KCOZ06] KAVAN L., COLLINS S., O'SULLIVAN C., ZARA J.: Dual quaternions for rigid transformation blending. Technical report TCD-CS-2006-46, Trinity College Dublin, 2006.

[KZ05] KAVAN L., ZARA J.: Spherical blend skinning: a real-time deformation of articulated models. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), pp. 9–16.

[LC04] LARBOULETTE C., CANI M.-P.: Real-time dynamic wrinkles. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)* (2004), pp. 522–525.

[LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00* (2000), pp. 165–172.

[MG04] MUELLER M., GROSS M.: Interactive virtual materials. In *GI '04: Proceedings of the 2004 conference on Graphics interface* (2004), pp. 239–246.

[MTLT88] MAGNENAT-THALMANN N., LAPERRIERE R., THALMANN D.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface '88* (1988), pp. 26–33.

[MTV05] MAGNENAT-THALMANN, VOLINO: From early draping to haute couture models: 20 years of research. *The Visual Computer 21*, 8–10 (2005), 506–519.

[PJS06] POPA T., JULIUS D., SHEFFER A.: Material-aware mesh deformations. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)* (2006), p. 22.

[SCFRC01] SLOAN P.-P. J., CHARLES F. ROSE I., COHEN M. F.: Shape by example. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), pp. 135–143.

[Sch06] SCHEUERMANN T.: Render to vertex buffer with d3d9. Siggraph 2006 Course 3: GPU Shading and Rendering, 2006.

[SF98] SINGH K., FIUME E.: Wires: a geometric deformation technique. In *SIGGRAPH '98* (1998), pp. 405–414.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. *j-COMP-GRAPHICS 20*, 4 (Aug. 1986), 151–160.

[SP04] SUMNER R. W., POPOVIC J.: Deformation transfer for triangle meshes. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (2004), pp. 399–405.

[TRN06] TAEHYUN RHEE J. L., NEUMANN U.: Real-timeweighted pose-space deformation on the gpu. *Computer Graphics Forum (Eurographics'06 proc.) 25*, 3 (2006), 439–448.

[VMT99] VOLINO P., MAGNENAT-THALMANN: Fast geometrical wrinkles on animated surfaces. In *7th International Conference in Central Europe on Computer Graphics and Visualization* (1999), pp. 55–66.

[YK06] YOON S.-H., KIM M.-S.: Sweep-based freeform deformations. *Computer Graphics Forum (Eurographics'06 proc.) 25*, 3 (2006), 487–496.

[YSZ06] YANG X., SOMASEKHARAN A., ZHANG J. J.: Curve skeleton skinning for human and creature characters: Research articles. *Comput. Animat. Virtual Worlds 17*, 3/4 (2006), 281–292.

[ZHS*05] ZHOU K., HUANG J., SNYDER J., LIU X., BAO H., GUO B., SHUM H.-Y.: Large mesh deformation using the volumetric graph laplacian. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), pp. 496–503.

© Association for Computing Machinery, Inc. 2007.