# Fast Skeletal Animation by skinned Arc-Spline based Deformation

Sven Forstmann and Jun Ohya

GITS Faculty, Waseda University, Tokyo, Japan

**Abstract**

*Presented is a novel skeletal animation system for providing high quality geometric deformations in real-time. Each bone of the skeleton is therefore represented by a spline, rather than using conventional matrix rotation. In our approach, each vertex of the animated character can be influenced by a maximum of three spline-curves, which is sufficient for skinned animation. One spline is parameterized by three control points and a possible twist. As opposed to conventional Bézier curves does our arc-spline rely on trigonometric functions for providing better curvatures. The optimized implementation using the OpenGL-shading language shows very promising results for real-time character animation, as even about 1 Million vertices were able to be transformed at interactive 43 frames per second on a GeForce 7800 GTX graphics card.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Splines,Hierarchy and geometric transformations

## 1. Introduction

Animation methods have been of interest in the area of computer graphics almost since the beginning, and many different methods evolved to cover the required deformation issue. The most popular in this area can be generally categorized in skinned skeletal animation, free-form deformation (FFD) and deformation along spline curves. Alternative methods, such as [RS05], [YL05] and [KZ05] allow an easy handling of deformation as they analyze the mesh structure first, but they can not provide real-time performance due to their high complexity.

In case of the basic skinned animation, multiple bones (each of them represented by one transformation matrix) can influence one vertex. This method is the fastest for achieving non-rigid animation and the most common in interactive applications. However, it has some major drawbacks for large scale deformations, as can be seen in Fig.1. In case of FFD, introduced by Sederberg and Parry [TS86], usually a low-resolution control lattice is used to deform the contained geometry. It has recently become quite popular for real-time usage in combination with physical simulation as in [MM05], [SC02] and [Ull04]. FFD is suited well for

operating on underlying nurbs surfaces, as the sharp lattice boundaries do not affect the created triangles directly. However, in real-time, where mostly triangles are used, this will lead to sharp edges where smooth ones are desired (Fig.1). An approach to overcome this situation is the method of S.Capell [SC02], where skinning was used to create smooth transitions near such boundaries. However, to provide high quality twist operations, the lattice would have to have either a very high resolution, or to be subdivided dynamically in order to prevent self-penetration.

We therefore have decided to use a method introduced by K.Singh [KS98], where the animation is driven by spline deformations, since it delivers highest possible quality (see Fig.1) as well as real-time performance. Unlike the more flexible approach of Singh, who applies a spline curvature driven rotation per vertex, we will exploit the spline's Frenet Frames, which can be calculated much faster. As graphics hardware has evolved quickly, it has become possible to shift the complete deformation part to the GPU, giving the CPU more space for other tasks. Due to the GPU's vector oriented design, we are able evaluate three spline deformations almost parallel, resulting in a very efficient computation.

Advantages of our approach are that singularities occurring in matrix skinning can be avoided and computationally expensive multi-segment bones for simulating a smooth deformation become obsolete. They are also impractical for real-time due to the large amount of intermediate matrices that are necessary at each segment. Furthermore, with only three control points and a twist operation, the range of naturally expressible postures is quite large.
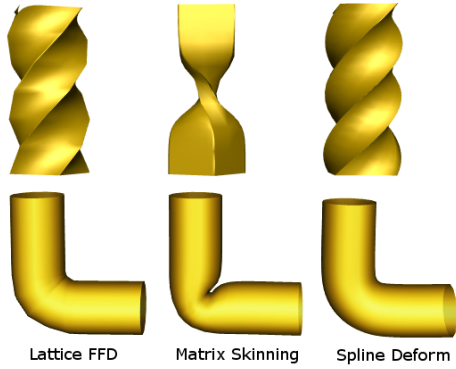


**Figure 1:** *Here a sample showing the most common deformation methods. As for FFD, 7 segments were used.*

## 2. Spline Functions



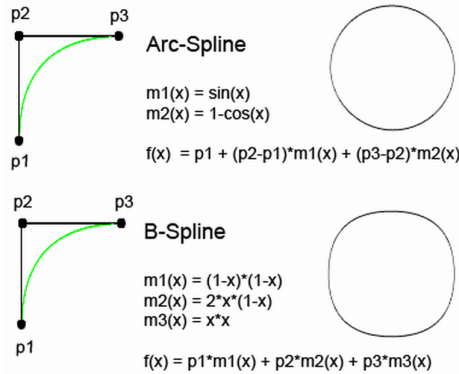**Figure 2:** *Here a comparison between an arc-spline and a Bézier-curve. Note the different circle-shapes, approximated by four splines segments each*

For deciding which type of spline function to choose, two candidates were available: the quadratic Bézier curve and the trigonometric arc-curve, shown in Fig.2. Both curves are bound to three control points, the minimum to define a spline, which is important for high performance.

By comparing the two circles in Fig.2, we can see that the arc-spline provides a natural circle while the Bézier spline's circle is not completely round shaped.

In the following, both curves formulas $f_a$ and $f_b$ with derivations $f_a'$ and $f_b'$ are defined. The input for both formulas are three three-dimensional control vertices $p_{1,2,3}$, the difference vectors $\Delta_{12}$ and $\Delta_{23}$ as well as the variable x. Note that the range of x is depending on the applied function.

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

**Arc curve:**
$$\forall x \in [0..\pi/2]$$
$$f_a(x) = p_1 + \Delta_{12} \cdot sin(x) + \Delta_{23} \cdot (1 - cos(x))$$
$$f_a'(x) = \Delta_{12} \cdot cos(x) + \Delta_{23} \cdot sin(x)$$

**Quadratic Bézier curve:**
$$\forall x \in [0..1]$$
$$f_b(x) = p_1 \cdot (1-x)^2 + p_2 \cdot (2 \cdot x \cdot (1-x)) + p_3 \cdot x^2$$
$$f_b'(x) = p_1 \cdot 2 \cdot (x-1) + p_2 \cdot (2-4x) + p_3 \cdot 2 \cdot x$$

Regarding both formulas, we see that the arc-spline requires much less operations than common Bézier; also can *sin* and *cos* be reused in the arc-splines derivative, which is very advantageous for our desired GPU implementation.
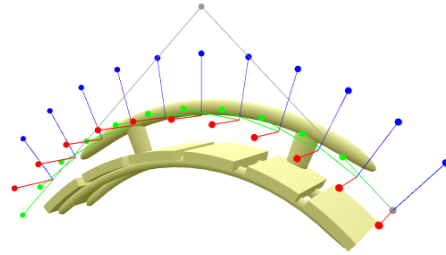
## 3. Spline coordinate-system



**Figure 3:** *The spline coordinate system.*

In order to deform geometry, we are required to create a complete coordinate system around the spline, the so called Frenet Frame, consisting of three basis vectors in each position of the curve. We therefore pre-calculate a general normal vector $b_N$ that is orthogonal to $\Delta_{12}$ and $\Delta_{23}$ and not depending on x as first basis vector. Our second basis vector, the tangent $b_T$, is equal to the derivative of the spline function $f_a$. As for the last vector, the binormal $b_B$, we can simply use the cross-product of $b_N$ and $b_T$. The origin $b_O$ of the coordinate system is given by our initial curve function $f_a$.

$$\forall x \in [0..\pi/2]$$
$$b_T(x) = f_a'(x)$$
$$b_N = \Delta_{12} \times \Delta_{23}$$
$$b_B(x) = b_N \times b_T(x)$$
$$b_O(x) = f_a(x)$$

In Fig.3, we can see such a generated coordinate system with normalized basis vectors. The basis normals and shown in red, tangents in green and binormals in blue.
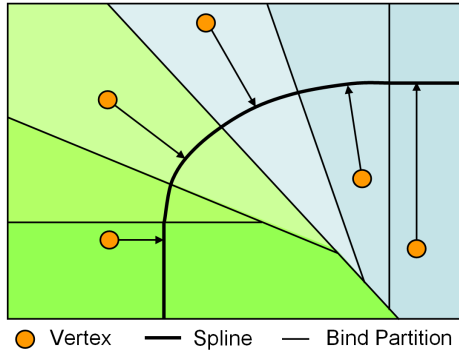
## 4. Bind-pose computation



**Figure 4:** *Space partitioning for the binding process.*

In order to calculate the bind pose, each vertex of the input geometry has to be mapped to a position $x$ on the curve. As this might lead to ambiguities, we decided to use a recursive divide and conquer approach, shown in Fig.4. In the first iteration we start by splitting the space in 2 partitions (green and blue inside the figure) at spline position $x = \pi/4$. By subdividing further, we can exactly determine the position $x$ on the curve together with the relative coordinate position in spline space based on $b_N$, $b_T$, $b_B$ and $b_O$.
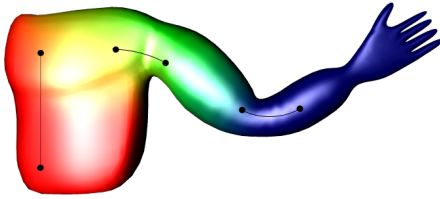
## 5. Spline Mixing



**Figure 5:** *Spline-Skinning: Red,green and blue are representing the weights for the body, the arm and the elbow-spline respectively.*

To achieve a useful result in character-animation, it is necessary to let each vertex of the geometry be influenced weighted by multiple deformations. For assigning the weights per vertex, we chose to use the conventional way by painting the weights on the geometry similar to a texture map. This method has already proven its effectiveness in various 3D authoring tools. An example can be seen in Fig.5, where each of the colors red,green and blue is assigned to a rigged body part. In overlapping areas the weights are mixed, resulting in colorful transitions.

In our implementation, up to three spline outputs $(v_1, v_2, v_3)$ are allowed to influence a single vertex $v_{result}$ as we have to regard the speed constraint. The three weights

$w_{1,2,3}$ for each vertex have to sum up to 1. More specific can we write as following:

$$\forall w_{1,2,3} \in [0..1]$$
$$w_1 + w_2 + w_3 = 1$$
$$v_{result} = v_1 * w_1 + v_2 * w_2 + v_3 * w_3$$

## 6. Shader Optimization

As the GPU built-in vector operations are very efficient for evaluating multiple operations at the same time, we can calculate three splines almost parallel. This results in a speed-up of about twice compared to the serial version. For the implemented version, also additional rotation around the spline axis was added to provide the twist operation.

For combining the three results $v_1$, $v_2$ and $v_3$ to $v_{result}$, we had to modify the mixing-formula, as the floating-point accuracy was not sufficient. In the modified version, we first evaluate the average $v_{middle}$ together with the three difference vectors $\Delta_{v1,2,3}$ before blending them together to the precise final vertex position $v_{result}$.

$$v_{middle} = (v_1 + v_2 + v_3)/3$$
$$\Delta_{v1,2,3} = v_{1,2,3} - v_{middle}$$
$$v_{result} = v_{middle} + \Delta_{v1} * w_1 + \Delta_{v2} * w_2 + \Delta_{v3} * w_3$$

The data that is stored per vertex are three bind-poses and three weights. The spline parameters can be set as vertex program variables and do not need to be stored per vertex. The handling of the vertices normal, binormal and tangent-vectors is a little bit complicated. If all of them are desired, it is recommended to store them in a RGB texture, which can be accessed by the vertex shader on any newer graphics hardware. Each directional vector (normal, binormal and tangent) would occupy one pixel inside the texture. In case that the spline is a straight untwisted line during the binding process, it is not necessary to store binding information; one transform matrix per spline is sufficient as the spline-space is linear.

## 7. Results

For the results, we created various poses to show the flexibility of our approach. In Fig.6, three poses are shown. The bind pose (up), a basic bend deformation (middle) and a combination of bending and twisting (bottom). The underlying skeleton was defined rigid by attaching the arm-spline to the upper control point of the body-spline and the elbow-spline to the right control point of the arm-spline (Fig.5). As for the pose in the middle of Fig.6, even the large scale deformation near the shoulder does not result in self-penetration. All three poses show correct, seamless transitions between deformed and undeformed areas.

In terms of hardware transformation speed, we tested our method by rendering large amounts of models in varying poses. Test system has been a Pentium IV 3.4Ghz with a
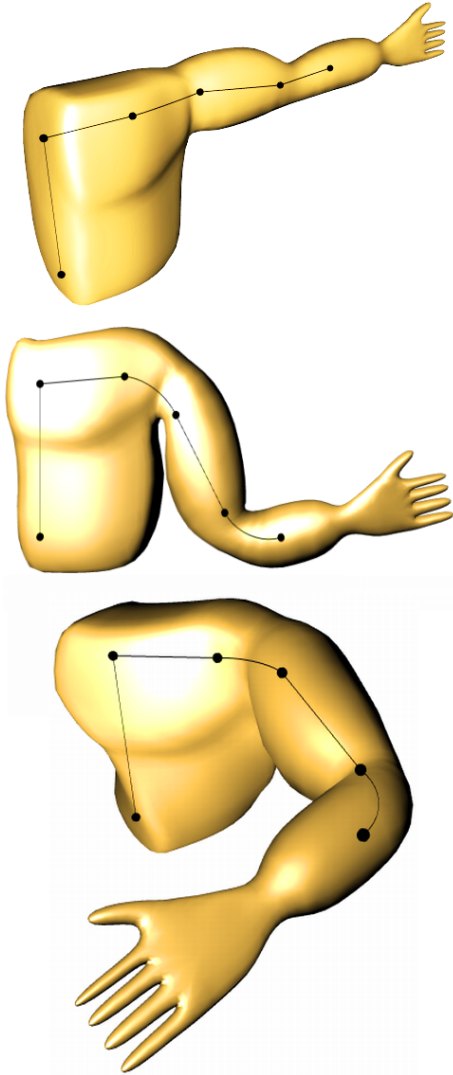
**Figure 7:** *Performance benchmark on a GeForce 7800 GTX graphics card, based on rendering the body-arm model of Fig.6 (12864 vertices) multiple times.*

## 8. Conclusion

We have presented an efficient implementation to achieve high quality skeletal animation in real-time, based on weighted spline-aligned deformation. In contrast to conventional matrix rotation, collapsing geometry in case of large-scale deformations can be avoided and joints can be adjusted more accurately due to additional control points. Our results prove that recent graphics hardware is already able to handle complex spline deformations in real-time; moreover can the achieved results show a very competitive performance which makes our approach feasible for many interactive applications.

## Acknowledgements

## References

[KS98]   K. Singh E. F.: Wires: A geometric deformation technique. In *SIGGRAPH '98* (1998).

[KZ05]   K. Zhou J. H.: Large mesh deformation using the volumetric graph laplacian. In *SIGGRAPH '05* (2005).

[MM05]   M. Mueller B. H.: Meshless deformations based on shape matching. In *SIGGRAPH '05* (2005).

[RS05]   R. Sumner M. Z.: Linear rotation-invariant co-ordinates for meshes. In *SIGGRAPH '05* (2005).

[SC02]   S. Capell S. G.: Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02* (2002).

[TS86]   T. Sederberg S. P.: Free-form deformation of solid geometric models. In *SIGGRAPH '86* (1986).

[Ull04]   Uller M.: Interactive virtual materials. In *CI'04* (2004).

[YL05]   Y. Lipman O.: Mesh-based inverse kinematics. In *SIGGRAPH '05* (2005).

**Figure 6:** *Three poses: The binding pose (up), simple bend operation (middle) and finally bending combined with two twist operations, for the hand and for the body (bottom).*

GeForce 7800 GTX graphics card. The results are presented in Fig.7. Despite complex calculations it is still possible to show a very competitive performance. With our implementation it is possible to transform 1 Million vertices in about 23 Milliseconds, corresponding to 43 frames per second.

This result was achieved by an implementation that stores the binding data as three texture coordinates per vertex. In a second test implementation where normals were transformed as well, the speed decreased by about 25%. However, we believe that a better performance can be achieved by the suggested textu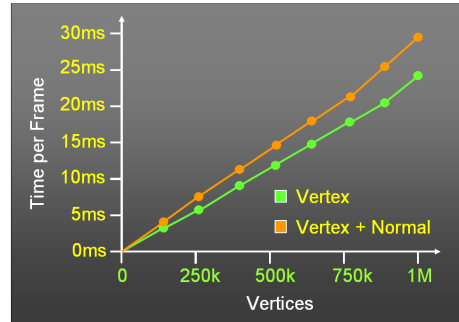re approach as it can save a lot of memory bandwidth.