Spherical Blend Skinning on GPU

Kasper Amstrup Andersen*
Dept. of Computer Science, University of Copenhagen

Figure 1: *Two animated characters using spherical blend skinning on GPU.*

Abstract

Skeletal animation is commonly used for real-time animation of humans, animals, etc. There are several methods available for propagating a skeleton pose to a skin mesh, but unfortunately, one of the most commonly used methods, linear blend skinning, has inconvenient artifacts. It is, however, a simple and very fast method, and it can easily be implemented on a GPU. An improved skinning method, spherical blend skinning, has been proposed. This method requires a rotation center to be computed by solving a linear matrix system, and further, it requires interpolation of quaternions. Thus, this method is more complex and slower than linear blend skinning, but it requires and produces exactly the same input and output as linear blend skinning making it easy to interchange the two methods. In this paper, we propose a method that enables spherical blend skinning to run on a GPU with performance comparable to GPU implementations of linear blend skinning, making spherical blend skinning a possible choice for many real-time applications such as computer games.

The proposed method is implemented in C++, OpenGL [Shreiner et al. 2005] and Cg [NVIDIA 2005] in the OpenTissue framework [OpenTissue 2007]. Shaders are shown in pseudo code general enough to make implementation in other language combinations quite easy.

Keywords: GPU, skeletal animation, skinning.

*e-mail: spreak@spreak.dk

1 Introduction

A method for real-time animation of characters is *skeletal animation* [Kavan and Zara 2003] where a skeleton is used to propagate a pose to a skin mesh, that is, the skeleton is used to deform the skin. The skin is the only thing visualized after being deformed by the skeleton. Some of the advantages of using a GPU for skeletal animation are:

- The GPU is potentially much faster than the CPU to handle streaming input data. The GPU is designed to work on large amounts of (independent) data in parallel.
- Pipeline parallelism. The GPU and the CPU can work in parallel; the CPU can prepare data for the next frame for the GPU while the GPU is drawing the current frame.
- Reduced data transfer. A geometric object can be transferred to GPU memory initially. Then, only a few parameters, necessary to deform the object, can be transferred to the GPU each frame. This is a great advantage since data transfer between CPU and GPU is expensive.
- Memory usage. This is especially important in game development where it can be important to save CPU memory and store data in GPU memory instead.

This paper considers a method, spherical blend skinning (SBS), for deforming a skin mesh given a skeleton, and how this method can be used efficiently on a GPU. SBS is, quality wise, an improvement of the commonly used linear blend

skinning (LBS) method that has several artifacts: A skin being deformed with LBS loses volume as a joint rotation increases, in the worst case collapsing the skin completely. Performance wise, SBS is slower than LBS making an efficient GPU implementation interesting for real-time purposes. In this paper, a GPU implementation of SBS is developed and compared to software and hardware implementation of both SBS and LBS. Comparisons are in terms of both quality and performance.

The paper is structured as follows: Section 2 briefly goes through some of the existing work in the area of GPU skinning methods. Section 3 describes how LBS works, and how SBS attempts to fix its artifacts. Section 4 describes how both LBS and SBS can be implemented efficiently on a GPU, and Section 5 discusses and compares results.

2 Related work

Different skinning methods have existed for a long time, e.g. in the gaming industry, but they are to a great extend unpublished. Linear blend skinning also known as enveloping, Skeleton Subspace Deformation (SSD), Matrix palette skinning etc. is proposed in [Lander 1998]. The artifacts of LBS are described in [Weber 2000]. A linear blend skinning method using the GPU is proposed in [Fernando 2004]. This method uses a vertex shader program, and supports up to 4 bone influences per skin vertex and achieves a very high performance. In [Rhee et al. 2006] another linear blend skinning method is proposed. It uses a fragment program shader and multiple passes to achieve further parallelism. An interesting method for progressive skinning on a GPU is proposed in [Pilgrim et al. 2006]. It uses LBS as the underlying skinning method and uses continuous level of detail to achieve high performance.

In [Kavan and Zara 2005] the cause of the artifacts of LBS is identified, and a solution, spherical blend skinning, using quaternion interpolation is proposed. Other methods that correct the artifacts of LBS have been proposed as well, such as bones blending [Kavan and Zara 2003] which limits the number of bone influences per vertex to two. Another method is proposed in [Mohr and Gleicher 2003] where extra joints are inserted and weights are computed automatically using a heuristic function. In [Wang and Phillips 2002] multi-weight enveloping is proposed where multiple weights are used per vertex. However, the advantage of SBS over these methods is that it works on exactly the same input and produces the same output as LBS and does not have any additional limitations compared to LBS. Another method having this property is Dual Quaternion Skinning (DQS) [Kavan et al. 2007]. Dual Quaternion Skinning has been implemented on a GPU achieving performance comparable to LBS, without its artifacts. The drawback of DQS is that dual quaternions do not exist in most existing graphics libraries yet.

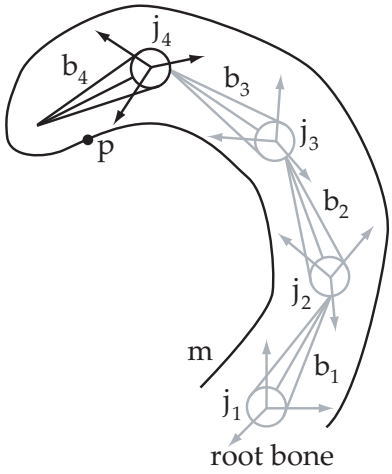

Figure 2: *A skin mesh, m, being deformed by a skeleton consisting of bones, b_i, connected by single joints. Joint j_i is a transform belonging to bone b_i. The vertex, p, is influenced by the bone b_4.*

3 Skinning on a CPU

Skinning is the task of deforming a skin mesh using a skeleton. An example of a skin mesh and a skeleton is shown in Figure 2. A skeleton is a hierarchy of bones [Erleben 2007]. Two bones are connected to each other by a joint consisting of a position and an orientation. A bone has two transforms; a *bone space transform*, B_j^{-1}, and a *pose transform*, P_j. Both transforms can be represented as 4×4 homogeneous transformation matrices:

$$T_j = \begin{pmatrix} T_j^{rot} & T_j^{trans} \\ \bar{0} & 1 \end{pmatrix}$$

where T_j^{rot} is a 3×3 rotation matrix, $\bar{0}$ is a 1×3 zero vector, and T_j^{trans} is a 3×1 vector. The transforms can also be represented as $(quaternion, transform)$ pairs, which is more compact.

The bone space transform places bone j in the World Coordinate System (WCS) origo and aligns its axes with the axes of WCS. The pose transform is the multiplication of the chain of bones from the root bone to bone j for the current skeleton pose. That is, the pose transform moves the bone to a new position. This is shown in Figure 3.

An initial assignment of bones to vertices in the skin mesh is performed in a bind pose, where no parts of the skin mesh are entangled. Here, each vertex in the skin mesh is assigned one or more bones responsible for deforming the vertex. The most simple deformation method allows only single bone influences, and thus a vertex can be deformed with:

$$v' = M_j v_{bindpose} \tag{1}$$

where $M_j = P_j B_j^{-1}$. We denote the matrix, M_j, the *absolute bone transform matrix*.

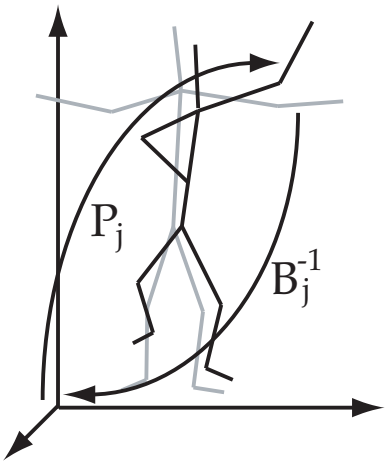

Figure 3: *A bind pose (gray skeleton) and a skeleton in a pose (black skeleton). The transform B_j^{-1} takes a bone down in the WCS origo, and aligns it axes with the axes of the WCS. P_j then transforms the bone into a pose.*

3.1 Linear blend skinning

Linear blend skinning (LBS) is a simple and commonly used skin deformation method. Each vertex in the skin mesh is bound to the skeleton by assigning a set of bones that is responsible for deforming the vertex. This set of bones is called the *bone influences*. Each vertex is also assigned a set of weights, one per bone influence, such that each weight, $0 \leq w_i \leq 1$ and $\sum w_i = 1$. These weights are found experimentally by an animator by creating different skeleton poses and then changing the value of the weights until the skin is deformed accordingly.

To deform a vertex, v, from its initial bind pose, $v_{bindpose}$, a weighted combination of the vertex deformed by its influencing bones is used:

$$v' = \sum_{i=1}^{N} w_i M_{j_i} v_{bindpose} \tag{2}$$

where N is the number of bone influences, and M is an absolute bone transform matrix.

A major problem when using this method is the artifacts, known as "collapsing joints" or "candy wrapper artifacts" [Kavan and Zara 2005], introduced by the deformed skin losing volume as joint rotation increases. The animator has to work around this problem, e.g. by attempting to tune weights, inserting extra bones etc., but this work is tedious and time consuming. LBS and its defects are further discussed in [Weber 2000].

3.2 Spherical blend skinning

The simple weighted combination of transformed positions is changed in spherical blend skinning [Kavan and Zara 2005]. Here, translations and rotations are interpolated independently. The absolute bone transform matrices are converted to $(quaternion, transform)$ pairs. The translation part is interpolation using linear interpolation. The quaternions used for the rotation part can be interpolated by either spherical linear interpolation, or linear interpolation as described in [Dam et al. 1998]. Linear interpolation yields a slightly different result compared to spherical linear interpolation, but it easily generalizes to more than two quaternions making it the preferred choice for SBS [Kavan and Zara 2005]. Linear interpolation of quaternions can be performed with:

$$qlerp(q_1, ...q_n, l_1, ..., l_n) = \frac{l_1 q_1 + ... + l_n q_n}{||l_1 q_1 + ... + l_n q_n||} \tag{3}$$

where q_i is a quaternion and l_i is a weight. It is not possible to simply interpolate quaternions and positions separately and then be done. The problem is that when rotations are interpolated a proper center of rotation is needed. For two transforms, M_1 and M_2, this center is the point, r_c, where $M_1 r_c = M_2 r_c$. That is, r_c must be a point that is constant during interpolation. Thus, all points will be rotated around the point r_c on a circular arc. Finding the rotation center can be done by solving a linear system:

$$(T_s^{rot} - T_t^{rot}) r_c = T_t^{trans} - T_s^{trans} \tag{4}$$

$$s < t, \quad s, t \in \{j_1, ..., j_n\}$$

By stacking all the equations, the system becomes:

$$A r_c = b \tag{5}$$

where A is a $3\binom{n}{2} \times 3$ matrix, r_c is a 3 dimensional vector (the unknown) and b is a $3\binom{n}{2}$ dimensional vector. For more than two bone influences this system is over constrained, making it impossible to solve the system by inverting A. The system can be solved by using Singular Value Decomposition [Erleben et al. 2005] which finds the optimal solution for r_c in the least-squares sense.

When the rotation center, r_c, has been found and the (rotation part) of the absolute bone matrices has been converted to quaternions, that are interpolated with $qlerp$ into a single quaternion Q, a vertex, v, can be transformed with:

$$v' = Q(v - r_c) + \sum_{i=1}^{N} w_i M_{j_i} r_c \tag{6}$$

where N is the number of bone influences. Equation 6 is derived and described in details in [Kavan and Zara 2005].

To speed-up the skinning process, a lookup table can be used to store rotation centers. Rotation centers are vertex independent; they only need to be computed once for each combination of bone influences. Thus, it is sufficient to compute a rotation center the first time a specific bone influence combination is encountered and then use the lookup table each time

the same combination is encountered. Each vertex must be able to perform a lookup using a key that is computed initially, when the skin is loaded. This key is computed, for each vertex in the skin mesh, such that it is possible to identify the influencing bones directly from the key. For a vertex, v_i, the key can be computed as (assuming that the list of bone indices it sorted in increasing order):

$$key(v) = \sum_{i=1}^{N} (j_i + 1) \cdot \text{offset}^{k-j}, \tag{7}$$

The value of offset can be set to 100 meaning that no more than 99 bones are allowed. 1 is added to j_i since indexing (into arrays) in C++/Cg begins at 0. For the list of influences $(4, 8, 10)$, the method returns the integer 50911. The choice of key generation method is arbitrary; the algorithm should, however, return an integer key, since integer comparison is fast. Since the algorithm is run only once, during initialization, performance of the key generation method is not of major importance.

An important property of SBS is that an animators work is unchanged if he is currently using LBS. The problem with many new methods is that they require new tools and a new work-flow for the animator which can require investments in both time and money. With SBS the designer can continue to use his usual tools, and to him SBS will simply be a black box yielding better results than LBS.

4 Skinning on a GPU

LBS is straight forward to implement on a GPU using a vertex program. A GPU vertex, $v_{gpu,lbs}$, supporting up to 4 bone influences can be defined as:

- A 4D vector storing x, y, and z vertex coordinates and a homogeneous coordinate:

$$v = (v_x, v_y, v_z, 1)^T$$

- A 3D vector storing x, y, and z coordinates of the normal belonging to the vertex:

$$n = (n_x, n_y, n_z)^T$$

- A 4D index vector storing bone indices:

$$I = (idx_1, idx_2, idx_3, idx_4)^T$$

  For a vertex with fewer than 4 influences the remaining entries in the vector are set to 0.

- A 4D weight vector storing bone weights:

$$W = (w_1, w_2, w_3, w_4)^T$$

  For a vertex with fewer than 4 influences the remaining entries in the vector are set to 0.

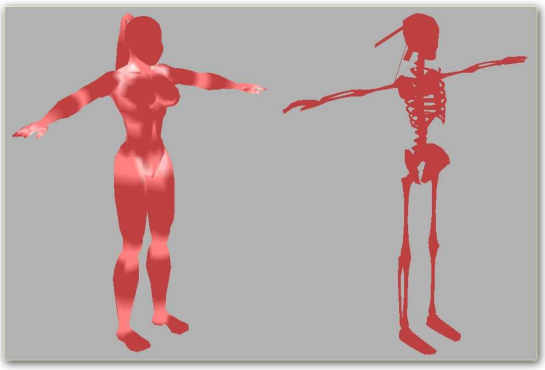

Figure 4: *Two characters (skin meshes) in bind pose. The skin meshes are rendered such that vertices with more influences are brighter. On the character to the left totally white areas have* 4 *influences (shoulder area) while dark areas have only* 1 *influence (shinbone). The character to the right has only single bone influences.*

The limitation of 4 bone influences is sufficient for most characters. It is not possible to vary the size of the index- and weight vectors from vertex to vertex, and it is apparent from Figure 4 that actually only very few vertices in a typical skin mesh have more than 1 influence. This shows that space would be wasted by using too large vectors containing mostly 0 entries.

On a GPU, we distinguish between two types of data, *varying data* and *uniform data*. The varying data is the streaming input data, i.e. vertex data, while the uniform data is data separate from the input stream that do not change between stream elements [NVIDIA 2005]. The skin (vertices and normals), in its bind pose, is transferred to the GPU along with all bone weights and bone indices. This becomes the varying data. The skin can then be deleted from CPU memory. By keeping each skin part in its own buffer on the GPU an appropriate material can be enabled before drawing each part. Some performance is lost here due to the multiple draw calls necessary to draw each skin part, but this is necessary in order for the different skin parts to have different materials.

A skeleton can now be used to deform the skin by transferring the absolute bone transform matrices for a pose to the GPU. Thus, before drawing a frame, a new pose is computed for the skeleton and this pose (the bone matrices) is the only thing transferred to the GPU. This becomes the uniform data. A GPU vertex, v, can now be deformed using a modified version of Equation 2:

$$v' = \sum_{i=0}^{3} W_i \cdot \texttt{bones}[I_i] \cdot v \tag{8}$$

where `bones` is an array containing the absolute bone transform matrices. This method is described in details in [Fernando 2004]. Pseudo code for a vertex shader performing LBS skinning is shown in Appendix A.1.

For SBS to work on a GPU, for each vertex we must be able to compute or lookup a rotation center, and to interpolate quaternions. The rotation centers are most easily computed on the CPU since there are numerous good matrix solvers publicly available here such as [Boost 2007]. Further, a lookup table can easily be used on the CPU as described in Section 3.2. This option is not available on the GPU since it offers no data structures that are shared between vertices. When all rotation centers have been computed they can be transferred in an array to the GPU as uniform data. By keeping with each rotation center in the lookup table an integer m such that the first rotation center to be inserted, r_{c_1}, gets $m = 0$, the second, r_{c_2}, gets $m = 1$ and so on, an array, H, having $H[m] = r_{c_{m+1}}$ can be created and transferred to the GPU. By letting each vertex on the GPU store m as well, it can index into H and fetch the precomputed rotation center.

The other thing needed for SBS to work on a GPU is quaternions. Instead of using absolute bone transform matrices a $(quaternion, translation)$ pair is packed into a 3×3 matrix:

$$C = \begin{pmatrix} q_{v1} & q_{v2} & q_{v3} \\ q_s & 0 & 0 \\ t_x & t_y & t_z \end{pmatrix}$$

An array of C matrices can then be transferred to the GPU as uniform data. A function can then be used to convert each C matrix into a 4×4 absolute bone transform matrix on the GPU. Notice, that using a 3×3 matrix to store rotation and translation reduces the amount of data to be transferred to the GPU compared to transferring 4×4 absolute bone transform matrices.

The varying data is, as for LBS, the skin mesh which is transferred to the GPU initially. For SBS a GPU vertex, $v_{gpu,sbs}$, can be defined as:

- A 4D vector storing x, y, and z vertex coordinates and an index, m, for the center of rotation:

$$v = (v_x, v_y, v_z, m)^T$$

- A 3D vector storing x, y, and z coordinates for the normal belonging to the vertex:

$$n = (n_x, n_y, n_z)^T$$

- A 4D weight vector storing bone indices:

$$I = (idx_1, idx_2, idx_3, idx_4)^T$$

  For a vertex with fewer than 4 influences the remaining entries in the vector are set to 0.

- A 4D weight vector storing bone weights:

$$W = (w_1, w_2, w_3, w_4)^T$$

  For a vertex with fewer than 4 influences the remaining entries in the vector are set to 0.

CPU	GPU
(Compute pose)	Lookup quaternions
Compute rotation centers	Lookup rotation center
Transfer bones	Interpolate quaternions
Transfer rotation centers	Perform skinning using Eq. 9

Table 1: *Work performed by the CPU and the GPU to render a single frame. The CPU must first compute a new pose, transfer the absolute bone transforms to the GPU, and then the GPU is responsible for skinning.*

	Vertices	Triangles	Joints	Bone comb.
Skeleton	2365	4412	23	0
Woman	1844	3342	37	62

Table 2: *The two characters, woman and skeleton, used for evaluating SBS and comparing it to LBS. "Bone combinations" is equivalent to the number of unique rotation centers that are computed, and to all combinations of bone influences.*

A GPU vertex, v, can now be deformed using a modified version of Equation 6:

$$\begin{aligned} v' =& Q \cdot (v_{123} - \texttt{rc[}v_4\texttt{]}) + \\ & \sum_{i=0}^{3} W_i \cdot \texttt{toMatrix(bones[}I_i\texttt{])} \cdot \texttt{rc[}v_4\texttt{]} \end{aligned} \tag{9}$$

where Q is the (up to 4) quaternions interpolated with $qlerp$, `bones` is an array containing the $(quaternion, translation)$ pairs, `rc` is an array containing rotation centers, and `toMatrix` is a function converting a $(quaternion, translation)$ pair to a 4×4 absolute bone transform matrix. A special notation is used in Equation 9: v_{123} is a 3D vector containing the first 3 elements of the 4D vector v.

The work done by the CPU and the GPU when performing SBS skinning is shown in Table 1. Pseudo code for a vertex shader performing SBS skinning is shown in Appendix A.2.

5 Results and evaluation

To test the quality and performance of the SBS algorithm on GPU, two characters from the Cal3D animation library [Cal3D 2007] have been used. The two characters are:

- Skeleton. A character with vertices having only single bone influences.
- Woman. A human character with vertices having multiple bone influence around joints; knees, elbows, shoulders etc.

The characters are shown in Figure 4. The complexities of the two characters are shown in Table 2.

	Skeleton	Woman
LBS	0.0133	0.0102
SBS	0.0144	0.0506
SBS_{LUT}	0.0143	0.0171
$LBSGPU$	0.0006	0.0006
$SBSGPU$	0.0006	0.0358
$SBSGPU_{LUT}$	0.0007	0.0032

Table 3: *Average skinning time (in seconds) taken over a series of 1000 skeleton poses. The "LUT" methods use a lookup table for rotation centers.*

5.1 Performance evaluation

The performance of SBS, with and without lookup table, has been evaluated and compared to LBS. Performance was evaluated on a Barton 2500+ with 1GB ram and a GeForce 6600GT (AGP) graphics card by running a series of 1000 skinning operations and then computing the average time used for a single skinning opertion (without pose computation). Results are shown in Table 3. $LBSGPU$ is the fastest method with the second fastest, $SBSGPU_{LUT}$, being about 5 times slower. $SBSGPU_{LUT}$ is, however, more than 5 times faster than SBS_{LUT}, and thus performance can be gained from using the GPU for SBS. The table also shows that the lookup table is the key to making SBS comparable to LBS, and that there is only a small performance loss when using SBS on a character with only single bone influences.

5.2 Quality evaluation

To evaluate quality of SBS and compare it to LBS, the Woman character have been setup in complex poses to see if the skin mesh has preserved its volume and if artifacts are visible. 3 of the poses are shown in Figures 5, 6, and 7. It is apparent that SBS is better than LBS, but also that SBS is not perfect since some artifacts still occur. This is especially visible in Figure 6. The cause of the artifacts are described in [Kavan et al. 2007].

6 Conclusion and Future Work

We have shown that it is possible to implement the spherical blend skinning method on a GPU. Rotation centers are computed by the CPU and they are transferred to the GPU along with a set of bone transforms represented as $(quaternions, translation)$ pairs, and then the GPU is repsonsible for doing the actual skinning. Spherical blend skinning has been shown to be slower than linear blend skinning running on a GPU, but faster than a pure CPU implementation. The artifacts of linear blend skinning are removed by the spherical blend skinning method. However, the method is not perfect since some artifacts still occur.

Since the changes from LBS are all internal, SBS can be seen

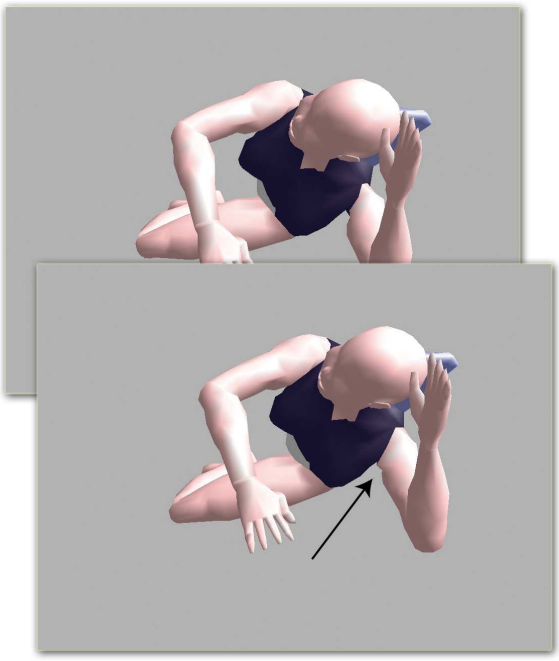

Figure 5: *A character having with an arm being twisted. The back image is the LBS method and here it can be seen that the skin looses volume around the shoulder. The front image is the SBS method, and here the volume is better preserved.*

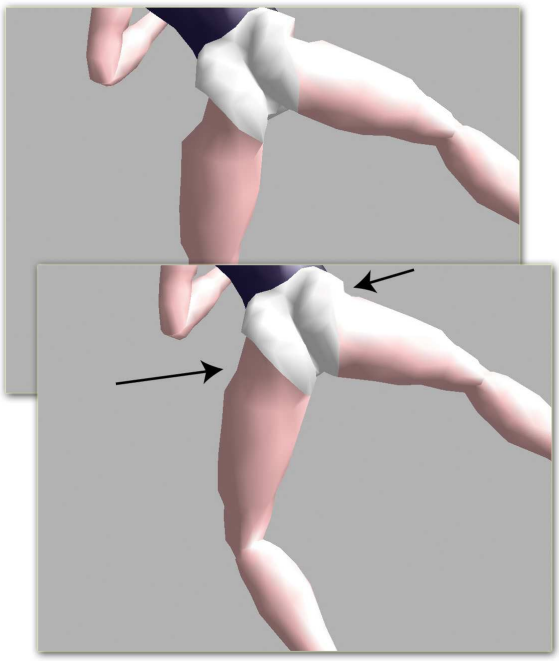

Figure 6: *A kicking leg. The back image is the LBS method and here it can be seen that the skin looses volume around the hip. The front image is the SBS method, and here the volume is better preserved, however, the result is still not perfect.*

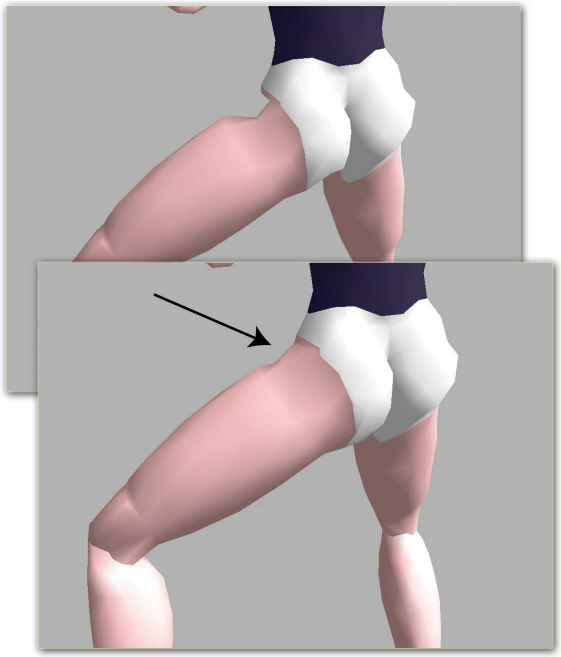

Figure 7: *A character with flexed legs. The back image is the LBS method and here it can be seen that the skin looses volume around the hip. The front image is the SBS method, and here the volume is better preserved.*

as a black box producing better deformations than LBS with no need to change neither input or output. This is a huge advantage for animators, since they do not have to change their work when using spherical blend skinning, and all existing characters can take advantage of spherical blend skinning without any additional work. Only now animators do not have to spend as much time on eliminating the inconvenient artifacts manually.

The SBS method running on a GPU could be improved by removing some of the restrictions it has, such as the limit of 4 bone influences per vertex. Another improvement would be to introduce level of detail (LOD) as proposed in [Pilgrim et al. 2006] to make the method even faster. A collision detection method for SBS has been proposed [Kavan et al. 2006] and it would be interesting to explore if the GPU could be used to support this method as well.

Acknowledgements

Thanks to the authors of the spherical blend skinning article [Kavan and Zara 2005]. Also thanks to the authors of the OpenTissue project for creating an easy to use framework, and to the authors of the Cal3D animation library for making animation data publicly available. Finally, thanks to my supervisor Kenny Erleben.

Obtaining sourcecode

The spherical blend skinning method has been implemented in the OpenTissue framework [OpenTissue 2007]. OpenTissue is open source and publicly available from `http://www.opentissue.org/svn/OpenTissue/`. Currently, the developed skinning methods are placed in the branch `kasper`, but later they will become part of the trunk.

References

BOOST, 2007. Basic linear algebra (ublas). `http://www.boost.org/libs/numeric/ublas/doc/index.htm`.

CAL3D, 2007. Opensource project, 3d character animation library. `http://home.gna.org/cal3d/`.

DAM, E. B., KOCH, M., AND LILLHOLM, M. 1998. *Quaternions, Interpolation and Animation*. Technical Report DIKU-TR-98/5.

ERLEBEN, K., SPORRING, J., HENRIKSEN, K., AND DOHLMANN, H. 2005. *Physics-Based Animation*. Charles River Media.

ERLEBEN, K., 2007. Practical character animation, lecture notes in game animation. DIKU, unpublished.

FERNANDO, R. 2004. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. NVIDIA Corporation.

KAVAN, L., AND ZARA, J. 2003. Real time skin deformation with bones blending. In *WSCG Short Papers proceedings*.

KAVAN, L., AND ZARA, J. 2005. Spherical blend skinning: a real-time deformation of articulated models. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press.

KAVAN, L., O'SULLIVAN, C., AND ZARA, J. 2006. Efficient collision detection for spherical blend skinning. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, ACM Press.

KAVAN, L., COLLINS, S., ZARA, J., AND O'SULLIVAN, C. 2007. Skinning with dual quaternions. In *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press.

LANDER, J. 1998. *Skin Them Bones: Game programming for the Web Generation*. Game Developer Magazine (May).

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM Press.

NVIDIA. 2005. *Cg User's Manual*. NVIDIA Corporation.

OPENTISSUE, 2007. Opensource project, physical based animation and surgery simulation. `http://www.opentissue.org`.

PILGRIM, S. J., AGUADO, A., MITCHELL, K., AND STEED, A. 2006. Progressive skinning for video game character animations. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, ACM Press.

RHEE, T., LEWIS, J. P., AND NEUMANN, U. 2006. Real-time weighted post-space deformation on the gpu. In *EUROGRAPHICS Volume 25, Number 3*, The Eurographics Association and Blackwell Publishing.

SHREINER, D., WOO, M., NEIDER, J., AND DAVIS, T. 2005. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison Wesley.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press.

WEBER, J. 2000. Run-time skin deformation. In *Proceedings of the Game Developers Conference*.

A Vertex shader pseudo code

The following sections show pseudo code for vertex shaders performing GPU skinning. The pseudo code is similar to Cg [NVIDIA 2005] and uses OpenGL [Shreiner et al. 2005] binding semantics. For both LBS and SBS the varying/streaming input has the following form:

```
struct vertex_input {
  float4 position   : POSITION;
  float3 normal     : NORMAL;
  float4 weight     : COLOR0;
  float4 boneIdx    : TEXCOORD0;
};
```

And the output (the input the the fragment program) has the following form:

```
struct vertex_output {
  float4 position   : POSITION;
  float4 color      : COLOR;
};
```

A.1 Linear blend skinning vertex shader

```
   input     : vertex_input IN,
               uniform float4x4 bones[40],
               uniform float4x4 ModelViewIT,
               uniform float4x4 ModelViewProj
   output    : vertex_output OUT
 1 float4 position ← IN.position;
 2 float4 normal ← float4 ( IN.normal.xyz, 0 );
 3 float4 weight ← IN.weight;
 4 float4 boneIdx ← IN.boneIdx;
 5 float4 defV ← float4 ( 0, 0, 0, 0 );
 6 float4 defN ← float4 ( 0, 0, 0, 0 );
 7 for i ← 0 to 3 do
 8 |   float4x4 bone ← bones[boneIdx[i]];
 9 |   defV ← defV + weight[i] * mul ( bone, position );
10 |   defN ← defN + weight[i] * mul ( bone, normal );
11 end
12 OUT.position ← mul ( ModelViewProj, defV );
13 OUT.color ← computeLight( defN, ModelViewIT );
14 return OUT;
```

Pseudocode 1: Vertex Shader pseudo code for LBS skinning. This shader supports up to 40 bones.

A.2 Spherical blend skinning vertex shader

```
   input     : vertex_input IN,
               uniform float3x3 bones[40],
               uniform float3 r_c[65],
               uniform float4x4 ModelViewIT,
               uniform float4x4 ModelViewProj
   output    : vertex_output OUT
 1 float3 rc ← r_c[IN.position.w];
 2 float3 position ← IN.position.xyz;
 3 float3 normal ← IN.normal;
 4 float4 weight ← IN.weight;
 5 float4 boneIdx ← IN.boneIdx;
 6 float3 defV ← float3 ( 0, 0, 0 );
 7 float3 defN ← float3 ( 0, 0, 0 );
 8 float4 Q ← float4 ( 0, 0, 0, 0 );
 9 float3 transV ← float3 ( 0, 0, 0 );
10 float3x3 b ← bones[boneIdx[0]];
11 float4 pivotQuat ← float4 ( b[0][0], b[0][1], b[0][2], b[1][0] );
12 for i ← 0 to 3 do
13 |   float3x3 bone ← bones[boneIdx[i]];
14 |   float4 quat ← float4 ( bone[0][0], bone[0][1], bone[0][2], bone[1][0] );
15 |   float3x3 rot ← quat2Mat( quat );
16 |   float3 tra ← float3 ( bone[2][0], bone[2][1], bone[2][2] );
17 |   transV ← transV + weight[i] * (mul ( rot, rc ) + tra );
18 |   defN ← defN + weight[i] * mul ( rot, normal );
19 |   if i ≠ 0 then
20 |   |   if dot (pivotQuat, quat) < 0 then
21 |   |   |   quat ← -quat;
22 |   |   end
23 |   end
24 |   Q ← Q + weight[i] * quat;
25 end
26 float4 Qnorm ← normalize( Q );
27 float3x3 QnMat ← toMatrix( Qnorm );
28 float3 rotV ← mul ( QnMat, (position-rc) );
29 defV ← rotV + transV;
30 OUT.position ← mul ( ModelViewProj, float4 ( defV, 1 ) );
31 OUT.color ← computeLight( float4 ( defN, 0 ), ModelViewIT );
32 return OUT;
```

Pseudocode 2: Vertex Shader pseudo code for SBS skinning. This shader supports up to 40 bones and 65 rotation centers.