Evaluation of Optimised Centres of Rotation Skinning

Paul Maximilian Bittner  
p.bittner@tu-bs.de
Jan-Philipp Tauscher  
tauscher@tu-bs.de
Steve Grogorick  
grogorick@cg.cs.tu-bs.de
Marcus Magnor  
magnor@cg.cs.tu-bs.de
Technische Universität Braunschweig
Mühlenpfordtstr. 23, 38106 Braunschweig

Abstract

In this paper we evaluate the real-time skeletal skinning technique Optimised Centres of Rotation Skinning (CRS) in terms of animation quality and efficiency. Skeletal animation rigidly deforms static meshes, letting them perform rigid movements without requiring additional geometry and make it a core part of movies and video games. Linear Blend Skinning (LBS) is the simplest and most intuitive implementation for appropriate vertex transformation. The popular extension Dual Quaternion Skinning (DQS) resolves the volume loss of LBS but introduces new artefacts. These are addressed by the recent CRS showing promising results. Our source code for LBS, DQS and especially CRS will be made public on our website.

1. Introduction

Digital characters in simulations and video games are expected to perform arbitrary movements, dynamically generated in real-time. It is not feasible to implement these motions by creating individual 3D geometry for every frame of an animation sequence, e.g. a walk cycle. A technique termed skeletal animation (or vertex skinning) allows to dynamically animate static meshes of almost unlimited complexity by only manipulating an underlying skeleton instead of all vertices of the corresponding mesh. Such a skeleton serves as an approximation of the actual mesh object and consists of only a few control points (called bones or joints) that are arranged in a tree. All bones have a weighted influence on surrounding mesh vertices to smoothly blend transformations of adjacent bones when applied to the mesh.

Since the first introduction of the idea to use a tree of bones, i.e., a skeleton [21], many techniques have been proposed to perform the mapping from skeleton to mesh in an appropriate way. The most popular solution is the original approach called Linear Blend Skinning (LBS) as well as more recent ones like Spherical Blend Skinning (SBS) [16] and Dual Quaternion Skinning (DQS) [13]. LBS uses 4×4-matrices to store the transformations of the bones. The final position of a vertex is the convex combination of these matrices with their corresponding weights. This is just a linear interpolation, thus volume is not preserved and may be lost as shown in Figure 1. Therefore, DQS uses dual quaternions instead. Dual quaternions are dual numbers that cause the bends to stretch out.

Figure 1: Example of the candy wrapper artefact due to volume loss of LBS. (Screenshot of implementation in our research rendering framework [3].)

Figure 2: Example of common artefacts produced by popular skinning methods. Due to volume loss in LBS candy wrapper (left) and elbow collapse artefacts (center) appear. Though DQS avoids volume loss, it also introduces new bulging artefacts (right) that cause the bends to stretch out.
tation and translation, just like matrices, while preserving volume around the joints. However, DQS introduces another kind of artefacts called bulging artefacts (see Figure 2). Due to the generalised mathematical formulation of the deformation at any point of the geometry regardless of their local peculiarities in LBS and DQS, a new backward-compatible approach has been proposed [18]. An individual center of rotation (CoR) for each vertex of the mesh is pre-computed to avoid these glitches. This Optimised Centres of Rotation Skinning (CRS) is designed as an extension to SBS and DQS to easily change existing animation pipelines. Le and Hodgins’s examples show clear improvements in reducing animation artefacts in comparison to LBS, SBS and DQS. The goal of this paper is to give an evaluation of CRS compared to LBS and DQS in terms of animation quality and efficiency, the key improvements of the CRS skinning technique. Furthermore, we focus on the necessary pre-computation. Also, detailed explanations of each method outline differences.

We provide our implementation of CRS and its CoR-generation mechanism to the general public on our website.

2. Related Work

In this paper, we focus on direct rigid real-time skeletal techniques to animate meshes in simulations and video games. Magnenat-Thalmann et al. [21] were the first to transform a mesh by using an underlying skeleton. They introduced bones and per-vertex weights. This approach is known as skeleton subspace deformation, joint-dependent local deformation or Linear Blend Skinning (LBS).

Other skeletal techniques [4, 13, 16, 18] are variations of the original LBS. They use bones and vertex weights with different interpolation methods to reduce or remove the volume loss artefacts of LBS. In some cases, Log-matrix Skinning (LMS) [4, 20] chooses inappropriately long trajectories and therefore produces artefacts beyond LBS and will not be taken into consideration for this work. Kavan et al. present Spherical Blend Skinning (SBS) [16] and Dual Quaternion Skinning (DQS) [13]. SBS uses pairs of quaternions and translation vectors instead of transformation matrices to interpolate rotation angles instead of positions, successfully avoiding the volume loss of LBS. However, since DQS is more efficient [12, p. 32][13, p. 45], we will consider that one instead. Nevertheless, SBS introduces custom CoRs. This idea is refined by Le and Hodgins in their more recent publication on CRS [18]. This more recent technique uses pre-computed CoRs for each vertex to avoid the bulging artifacts of SBS and DQS and will also be covered in our evaluation. These CoRs work similarly to curve-skeletons [7, 25, 35]. CRS initially computes LBS and then corrects the volume loss artefacts. Other LBS-correcting techniques make use of additional helper joints [23, 24] to split twists and bends over multiple bones. We do not consider methods like projective skinning[17, 31] or [10, 11, 15] as we specifically focus on the issues that CRS addresses.

Non-real-time skeletal techniques usually provide better results and are a very reasonable choice for non-linear formats like movies. They either use multiple passes to correct errors or are physically based approaches [5, 6, 8, 9, 19, 22, 26, 27, 28]. Vaillant et al. present Implicit skinning [32] that uses DQS and a second pass to resolve its artefacts. They declare implicit skinning to be real-time capable, but since it requires a costly post-computation, it is not a proper solution for video games and simulations. In fact, the performance impact is so dramatic, that using it even for a single character in a video game is intractable. Poisson stitching methods [29, 30, 33, 34] perform a rigid transformation per triangle to reconnect them to a smooth surface, by solving a large Poisson equation in a second pass. They are example based and create new animations from given samples.

3. Techniques

3.1. Linear Blend Skinning

The first method developed for skeletal animation is the most intuitive and serves as base for every other technique developed until today. A skeleton is a tree of n bones. Each bone \( b \in \{0, \ldots, n-1\} \) has a local transformation \( L_b \in \mathbb{R}^{4 \times 4} \) and is associated with the space of its parent bone \( p(b) \). Its transformation in skeleton space \( S_b \) is determined by multiplying \( L_b \) with its parent matrices recursively:

\[
S_b = L_{root} \cdot \ldots \cdot L_{p(p(b))} \cdot L_{p(b)} \cdot L_b.
\]

To create the link from skeleton to mesh, for each bone \( b \), weights \( w_{vb} \in [0, 1] \) are assigned to each vertex \( v \in \mathbb{R}^3 \). As the weights are fractions describing the respective contributions to the transformation of a vertex, multiple bones may affect a single vertex. This way, smooth animations can be performed at the joints like shoulders and knees. Therefore, the weights’ sum per vertex has to be 1. The weight selection is usually done manually by an animation designer.

It is important to consider that assigning a skeleton to a mesh has to be done for an initial pose or state of the mesh as shown in Figure 3. In this so-called bind pose, each bone already has an initial transformation that is represented by its bind matrix \( B_b \). To use arbitrary transformations \( T_b \) for

<table>
<thead>
<tr>
<th>Notation</th>
<th>Vector ( \vec{v} )</th>
<th>Quaternion ( \vec{q} )</th>
<th>Dual Quaternion ( \vec{d} )</th>
<th>Dot Product ( \langle \vec{u}, \vec{v} \rangle )</th>
<th>Matrix ( M )</th>
</tr>
</thead>
</table>

Table 1: Notation table
any bone $b$, this initial transformation has to be undone first by using the inverse bind pose:

$$L_b = T_b \cdot B_b^{-1}.$$  

Vertices are then transformed by linearly blending these bone transformations depending on their weights:

$$LBS(v) = \sum_{b \in \mathcal{B}_v} w_{vb} S_b.$$  

Usually, only a subset of bones $\mathcal{B}_v$ is used for each vertex to reduce memory and computation cost. Typically, $|\mathcal{B}_v| := 4$ is chosen, since vertices are rarely affected by more than four bones. All other weights are then considered to be 0, leading to a shorter form:

$$LBS(v) = \sum_{b \in \mathcal{B}_v} w_{vb} S_b.$$  

It is then necessary to additionally store these bone indices $\mathcal{B}_v$. Computing bone transformations is recommended to be done on the CPU as it is required only once each frame for the whole mesh. The interpolation has to be computed per vertex and can be computed in parallel for all vertices on the GPU.

### 3.2. Dual Quaternion Skinning

DQS addresses the main problem of LBS: the volume loss artefacts. The transformation matrices of LBS do not consider angular positions around the joints. Therefore, DQS replaces them by dual quaternions, that simplify blending rotations that way Figure 4. In contrast to general quaternions, dual quaternions also interpolate translation in addition to rotation. Dual quaternions are dual numbers consisting of two ordinary quaternions

$$\hat{q} = q_0 + \epsilon q_e,$$

where $q_0, q_e \in \mathbb{H}$ are the non-dual and dual part of $\hat{q}$, and $\epsilon$ is the dual unit with $\epsilon^2 = 0$. The algebraic rules for dual quaternions are explained by Kavan et al [13, p. 42].

When $q_e = 0$, the dual quaternion $\hat{q}$ becomes an ordinary quaternion. Therefore dual quaternions can represent three-dimensional rotation ($\hat{r} := r_0$), but unlike ordinary quaternions, they can represent translation, as well. The unit dual quaternion

$$\hat{t} = 1 + \frac{\epsilon}{2}(t_x i + t_y j + t_z k)$$  

expresses a translation by $(t_x, t_y, t_z)$.

Finally, rotation and translation have to be composed into a single dual quaternion, which allows them to encode a complete transformation in 3D-space like a 4x4-Matrix. This is simply done by multiplication.

$$\hat{f} = (1 + \frac{\epsilon}{2}(t_x i + t_y j + t_z k)\hat{r} = \hat{r} + \frac{\epsilon}{2}(t_x i + t_y j + t_z k)\hat{r} (4)$$

This is especially useful for blending bones, where dual quaternions can now replace the default transformation matrices of LBS. Blending all bone transformations is then done the same way as in Equation 1, but with one important difference: Instead of using the default commutative addition for dual quaternions, we have to check their orientation:

$$\hat{a} \oplus \hat{b} = \begin{cases} \hat{a} + \hat{b} & \text{if } \langle \hat{a}, \hat{b} \rangle \geq 0 \\ \hat{a} - \hat{b} & \text{if } \langle \hat{a}, \hat{b} \rangle < 0 \end{cases}$$  

This functions resolves the quaternion antipodality, i.e. $q$ and $-q$ describe the same rotation but their interpolations may differ [14]. Finally, we can derive the final algorithm 1: After computing the dual quaternions for each bone on the CPU, these need to be interpolated per vertex as in LBS (step 1). The resulting dual quaternion has to be normalised to ensure a unit dual quaternion (step 2). Afterwards, it can be converted to a transformation matrix by reversing the previous steps like in Equation 4 (step 4 - 5). The final matrix can then be used to transform the vertex and its normal.
3.3. Optimised Centres of Rotation Skinning

DQS avoids the volume loss artefacts of LBS but since it rotates the vertices around the joints and does not consider their individual distance to these joints, new bulging artefacts are introduced. These artefacts cause the geometry to stretch out from its expected pose as shown in Figure 5 (left). To fix this, Le and Hodgins developed a new skinning method based on DQS that uses an individual center of rotation for each vertex [18]. This CRS computes the CoRs in a pre-processing step from the mesh in rest pose and its skinning weights. At runtime, they are used to measure the error of LBS and correct it with translation. Rotations are blended with Quaternion Linear Interpolation (QLERP) similar to DQS, but with just a single quaternion, since translation is computed independently using CoRs. Computing the CoRs requires the mesh itself and its skinning weights and has two requirements:

- Vertices with similar skinning weights should have similar centres of rotation to ensure similar transformations.
- Rigid transformations must be preserved, i.e. relative distances have to remain constant.

To measure similarity of skinning weights, Le and Hodgins introduce a similarity function $s$, that compares the skinning weights of two vertices. Let $w_v = (w_{v0}, \ldots, w_{vn-1})$ denote the skinning weights vector of vertex $v$, containing the weights for all bones.

$$s(w_p, w_v) = \frac{\sum_{j \neq k} \sum_{p \in \mathcal{V}} w_{pj} w_{pk} w_{v_j} w_{vk} e^{-\frac{(w_{pj}-w_{vk})^2}{\sigma^2}}}{\sum_{j \neq k} \sum_{p \in \mathcal{V}} w_{pj} w_{pk} w_{v_j} w_{vk}}$$  \hspace{1cm} (6)

The term $w_{pj} w_{pk} w_{v_j} w_{vk}$ describes the contribution of the two bones $j$ and $k$ to the similarity and serves itself as a weight for the similarities of each bone pair. The power then describes the actual similarity of both bone weights. Le and Hodgins regard two weights $w_p$ and $w_v$ to be similar iff their contribution ratios are similar:

$$\frac{w_{pj}}{w_{pk}} \approx \frac{w_{v_j}}{w_{vk}} \iff w_{pj} w_{v_k} \approx w_{pk} w_{v_j}$$

Therefore, $w_{pj} w_{v_k} - w_{pk} w_{v_j}$, the distance of both weights measures their similarity. Additionally, this measure can ensure affine weights (their sum equals 1 for each vertex), whereas the sum of weights will not. The parameter $\sigma$ controls the weighting of similarity. Using $\sigma$, we aim to find a translation that eliminates the issues of LBS. This translation describes the offset from the LBS position $LBS(v)v$ to the desired position $R_p v + t$. As $R_p$ is the rotation matrix given by QLERP, $t$ remains the only unknown value. The smallest offset for the whole surface $\Omega$ has to be computed to avoid bulging artefacts.

$$t_p = \arg \min_t \int_{v \in \Omega} s(w_p, w_v) \parallel R_p v + t - LBS(v)v \parallel^2 dv$$  \hspace{1cm} (7)

Simplifying the solution of this linear least squares problem leads to the following form [18]:

$$t_p = LBS(p^*) p^* - R_p p^*$$  \hspace{1cm} (8)

$$p^* = \frac{\int_{v \in \Omega} s(w_p, w_v) v \parallel dv}{\int_{v \in \Omega} s(w_p, w_v) \parallel dv}$$  \hspace{1cm} (9)

Note that $p^*$ is equivalent to the CoR for the rest-pose. Applying LBS leads to the CoR in the deformed target mesh. Equation 9 shows that CoRs only depends on the source vertices and their skinning weights, which allows to pre-compute them without any knowledge about the bone transformations at runtime. As meshes are not continuous surfaces but composed of triangles, the integral over all points of the surface in Equation 9 can be discretised to a sum over all triangles $t_{\alpha, \beta, \gamma} \in T$. The weight $w_l$ and position $v_l$ of triangle $t_{\alpha, \beta, \gamma}$ are then considered to be the average of the corresponding properties of its three vertices $\alpha, \beta, \gamma$. Since the triangles are not equally sized, their contribution is weighted by their area $a_t$. This discretisation leads to an approximation of CoRs:

$$p^*_l = \frac{\sum_{t \in T} a_t s(w_l, w_l) v_l}{\sum_{t \in T} a_t s(w_l, w_l)}$$  \hspace{1cm} (10)
The approximation of the similarity \( s \) may cause numerical errors, which can be avoided by subdividing the mesh, in turn splitting every edge \((i, j)\) with a skinning weight distance \( l = \|w_i - w_j\| > \epsilon \) [18]. Calculating the translation with the CoRs and Equation 8 and rotating with QLERP yields algorithm 2.

4. Implementation

We implemented LBS, DQS and CRS using our custom-built research rendering framework using OpenGL for rendering. Hence, our shaders are written in GLSL.

**Dual Quaternion Skinning** GLSL does not provide in-built support for quaternions and dual quaternions, wherefor we provide this functionality by ourself.

**Optimised Centres of Rotation Skinning** additionally requires the CoRs for skinning. These are uploaded to the GPU as another vertex attribute of the mesh. The implementations of LBS and DQS provide the remaining operations needed for CRS.

4.1. Centers of Rotation Computation

The most challenging part of implementing CRS is computing the CoRs. The naive approach of a direct conversion of Equation 10 to code is computationally expensive. Therefore, we implemented three major optimisations as suggested by [18]:

**Pre-computing** The mesh subdivision, triangle adjacency graph and triangle values \( w_t, a_t, v_t \) are pre-computed because they are independent of the CoR calculation itself.

**Parallelisation** Every CoR can be computed independently of each other. The only memory address that needs to be written to is the CoR position, which is individual for each CoR. We compute the CoRs in intervals of equal length, whereas one thread for each interval is created to compute its CoRs. Algorithm 3 shows the pseudo-code of our pre-computation and parallelisation implementation.

**Breadth-First Search (BFS)** Since the CoR computation is based on similarity of vertex weights, only the triangles of those vertices need to be considered. As such, for each vertex \( a \), all vertices \( b \) with similar skinning weights, i.e., those satisfying \( \|w_a - w_b\| < \omega \), can be queried first. The triangles of those vertices are then used as the starting triangles for BFS on the triangle adjacency graph of the mesh. The BFS continues until the similarity to the next triangle falls below a certain threshold \( \epsilon \). The pseudo-code of our implementation is shown in Algorithm 4.

For comparability, we follow Le and Hodgins’ parameter suggestions for their evaluation that adjust CoR computation \((\epsilon, \sigma, \omega, \epsilon) = (0.1, 0.1, 0.1, 10^{-6})\).

Since the CoRs have to be calculated only once per mesh, they are stored offline.

5. Evaluation

We evaluate the three skinning techniques using the following metrics:

**Animation quality** Reducing animation artefacts, ensuring constant volume and avoiding undesired or unexpected behaviour

**Efficiency** Low impact on runtime performance, e.g., no notable increase of render time, low memory consumption, minimal pre- and post-processing overheads

Since visual performance is the main goal of DQS and CRS we will consider animation quality to be more crucial than efficiency. Furthermore, the CoR computation of CRS will be discussed separately.
Algorithm 4: CoR computation with BFS

**Input:** vertex $v$,

triangle adjacency graph $G$,

vertices with similar skinning weight $N(v) \subseteq V$

**Output:** CoR for $v$

$Q \leftarrow \{ t \mid t \text{ is triangle of } u \in N(v) \}$ \quad // $Q$ is queue

$c_{\text{top}} \leftarrow (0,0,0)$

$c_{\text{bot}} \leftarrow 0$

for $t \in Q$

if $t$ was not visited then

mark $t$ as visited

$\varsigma \leftarrow s(w_v, w_t)$

if $\varsigma \geq \epsilon$ then

$c_{\text{top}} \leftarrow c_{\text{top}} + \varsigma w_v$ \quad // Equation 10

$c_{\text{bot}} \leftarrow c_{\text{bot}} + \varsigma w_t$

append unvisited neighbours of $t$ in $G$ to $Q$

end

end

return $\frac{c_{\text{top}}}{c_{\text{bot}}}$

For testing, we used a cuboid, the wolf from the *Infinity Blade: Adversaries* assets package [2] and the raptor from the “Unreal ARK DevKit” [3].

5.1. Animation Quality

For animation quality evaluation, we will first have a look at the artefacts each skinning model produces and, conversely, avoids. Animation artefacts were the main motivations of the developers of DQS and CRS, whereas DQS was designed to eliminate the volume loss artefact of LBS, while CRS extends DQS to prevent its joint bulging artefact in addition to the volume loss. Due to its simplicity and symmetry, the cuboid is a convenient test mesh for comparing each technique for the same pose in Figure 6. As DQS interpolates rotation as angles around a centre (the joints in this case), volume is preserved in contrast to LBS. CRS uses quaternions for rotation interpolation as well, wherefore it also prevents volume from collapsing most of the time. A problem of CRS are vertices that are influenced by more than two bones [18], where volume loss might still become an issue in a few cases. The joint bulging artefact, illustrated at the right joint in Figure 6, is common for DQS. LBS is not prone to produce this artefact because it linearly interpolates the transformations and not the angles of the rotation. As visible in Figure 6 (bottom), CRS greatly reduces the bulging, but still leaves a little bulge at its outer edge. Especially the inner vertices benefit from their CoRs by not drifting apart.

However, each technique suffers from the rotation limitation. None is able to handle rotations greater than 180°. LBS reverses its own rotation after passing the singularity at that threshold until it reaches its initial state again. DQS and CRS just flip the movement because quaternions always describe a rotation along the shortest angle. Rotations like this are only achievable by splitting a bone into multiple parts and then rotating each of them around an angle smaller than 180° in the same direction. However, this may not only lead to twisting, but also buckling, and might produce errors for some use-cases such as ragdoll physics. Similar to this trick, each of these artefacts can be suppressed by for example using more bones or carefully adjusting skinning weights to be sharper or smoother. In an extreme case, a skeleton containing one bone for each vertex would be ideal and would produce the same motion for each technique, as no interpo-
lation would be necessary at all, but would be impractical and undermine the purpose of skeletal animation.

Apart from theoretical perfection, practical examples are very relevant, since these techniques are used in movie, simulation and games industry. Hence, different animation styles benefit in diverse ways from the proposed techniques. Indeed, CRS might not always be the best method to use, although it produces the fewest and least distracting artefacts. For example, LBS produces a very soft and squishy material when bending (not too far), which might be even a useful application for the volume loss artefact. Notably, comparing frames in Figure 7 shows that each method produces visually indistinguishable results for the attack animation of the wolf mesh. This animation consists of jumping and biting, neither of which involves twisting or strong bending. For such simple movements that do not represent mathematical edge cases, LBS is the most appropriate method, since it is the simplest and fastest solution, as described in the next sections. More obvious examples for this case are meshes with each vertex assigned to exactly one bone. In this case, no interpolation is about to happen, so no advanced techniques are required. The robot mesh in Figure 3 only has ball joints and is hence a practical example of this case.

On average, CRS provides the most appealing motions because it removes most artefacts while deforming the mesh intuitively. DQS solves the volume loss artefacts as well and therefore is a reasonable choice, too, considering that bulging artefacts are not very harmful if a mesh contains enough bones. If the mesh is low polygonal, the weights are very diverse or no extraordinary movements are expected, LBS fits its role convincingly and may be even as usable as the other techniques.

5.2. Efficiency

Efficiency evaluation is divided the two main topics speed and memory. CRS is the only method that requires pre-processing, so it will be discussed separately.

5.2.1 Runtime Performance

We use frames per second (FPS) as a measure for runtime performance. On average Table 2 demonstrates that LBS and DQS show comparable performance. Only CRS shows a small drop in FPS. There are two reasons for the first two to be equally fast:

1. LBS needs 16 additions and 16 multiplications to interpolate two matrices. DQS only computes 8 additions and 8 multiplications instead, notably half of LBS.

2. GPUs have a native hardware support for matrix multiplication. Unfortunately, DQS requires custom functions, making this task notably slower.

On average, CRS provides the most appealing motions because it removes most artefacts while deforming the mesh intuitively. DQS solves the volume loss artefacts as well and therefore is a reasonable choice, too, considering that bulging artefacts are not very harmful if a mesh contains enough bones. If the mesh is low polygonal, the weights are very diverse or no extraordinary movements are expected, LBS fits its role convincingly and may be even as usable as the other techniques.

5.2.2 Memory Consumption

Analysing the GPU memory that each method requires is done by inspecting their respective data structures.

We hypothesise these two effects to cancel each other out, leaving both methods with a similar performance. As CRS executes LBS and an additional quaternion interpolation, it is slower than LBS itself. The post-computing overhead (conversion of quaternions to matrix, etc.) is about the same as in DQS.
greatly benefited from the BFS acceleration. Introducing

<table>
<thead>
<tr>
<th>Model</th>
<th>naive 4 threads</th>
<th>4 threads and BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cuboid</td>
<td>51.4s</td>
<td>16s</td>
</tr>
<tr>
<td>Wolf</td>
<td>25.9s</td>
<td>7.9s</td>
</tr>
<tr>
<td>Raptor</td>
<td>&gt;&gt; 2.5h</td>
<td>≈ 2.5h</td>
</tr>
</tbody>
</table>

Figure 8: CoR computation time per model and ascending optimisations without the subdivision step. Our results for the Cuboid and Wolf correlate roughly to Le’s and Hodgin’s times [18]. The computation of the raptor largely exceeds the expected time demand.

vertices, leading to a total value of eight floats, which is half the space LBS needs for its bones. CRS has to buffer the 4x4-matrix of LBS plus the rotation quaternion, resulting in the largest memory footprint per bone of $16 + 4 = 20$ floats. Additionally, it has to store the CoRs of each vertex, leading to a significantly higher number of $3 + |V|$ additional floats, whereby $V$ denotes the set of all vertices. This impact is depicted in the actual memory required for each mesh, where CRS uses the most by far.

Obviously, CRS is the most expensive skinning method. It does not only need more data per bone than the other techniques but also requires the CoRs of each vertex. This is the most critical part since models tend to contain much more vertices than bones, i.e., $|V| \gg n$. Just like normals for lighting, this serves as an extra attribute of the mesh and may pay off for important characters such as the protagonists in video games. Especially, because modern GPUs often have multiple gigabytes of memory anyway. DQS is obviously the most memory-efficient skinning technique by halving the necessary buffer of the original LBS.

The subdivision step in the computation of CoRs for CRS (Algorithm 3, step 1) splits every edge in the mesh whose skinning weight distance is bigger than the threshold $\epsilon$. Thus, two new triangles are produced whose edges have to be checked as well. The subdivision is only needed to compute the CoRs. Rendering remains unaffected. For the raptor mesh, this process exceeded our 16 GB of main memory, wherefore we had to abort the subdivision. Hence we deactivated it for further tests.

### 5.2.3 Centres of Rotation Computation

Iterating over all faces for each vertex in the naive version leads to increased runtime. Using BFS as an optimisation requires the triangle adjacency graph and knowledge about vertices with similar skinning weights for each vertex. Gathering this information will further increase the total computation time for some models like the cuboid in Figure 8. Both other test meshes, the wolf and the raptor, greatly benefited from the BFS acceleration. Introducing multi-threading decreased the computation time by a factor of about 4 as expected for our 4-core CPU. Note, that the computation time grows quadratically with the number of faces, wherefore the raptor has such a substantially longer computation time compared to the cuboid even though it has fewer vertices. Nevertheless, compared to the Goliath mesh of the original paper, that has about the same number of bones but much more vertices and faces, the computation took much longer than expected, and this even without the sub-division step. This may be caused by a more cache obvious implementation in the original paper, but a detailed comparison is not possible, since their implementation is not open source.

### 6. Conclusion

Incontrovertibly, the visual performance of each method depends largely on the chosen weights and the number of bones. The more diverse the weights are, which is typically the case for low-poly meshes like Figure 7, the more similar the results of each method become. Vice versa, models with smoother weights, common for high-poly meshes, benefit greatly from DQS and even more from CRS.

In terms of efficiency, simplicity and visual performance, DQS is still an all-rounder. It requires only half the memory for the bones while performing as fast as LBS, does not need any additional data and computations and avoids volume loss, the most distracting artefact. Since most animations do not contain intense twists and bends, its joint bulging does not matter much and rarely attracts attention.

CRS main goal is suppressing the joint bulging artefact of DQS. Although this may not work perfect (see Figure 6), it delivers satisfactory results for all tested cases that do not distract or look unnatural. Therefore, CRS is clearly the most visually appealing method.

This good visual performance comes at the cost of a much higher memory footprint for bone data and especially the CoRs. Although memory is more precious for real-time applications, CoRs are only as expensive as normals for a mesh, making them nonetheless affordable, especially for important models. As the number of bones is usually much smaller than the number of vertices, the extra memory used to store the quaternions compared to LBS is negligible.

The non-trivial pre-computation step is more critical because it is not straight-forward to implement and must be executed for each model. Due to its complexity, dedicated libraries could make CRS much more attractive. While our implementation is faster than the naive approach, it can not reach the reported computation times of the closed source reference implementation.

With our source code for the pre-computation available to the public, the necessary time overhead for model enhancement and extra implementation compared to the other methods is affordable.
References


