Accurate and Efficient Lighting for Skinned Models
Supplemental

Marco Tarini 1,2, Daniele Panozzo 3, Olga Sorkine-Hornung 3
1 Università dell’Insubria, Varese 2 ISTI-CNR Pisa 3 ETH Zurich

Abstract
We analyze the operation count and the bandwidth requirements of our algorithm [TPSH14] compared with existing approaches. These results are summarized in Table 1 of the paper [TPSH14].

1. LBS: approximated solution

We analyze the per-vertex cost of updating the normals during a Linear Blend Skinning (LBS) deformation. We assume that each algorithm computes, for each vertex of each animation frame $t$, its position $p_i^t$, normal $n_i^t$, and the tangents $t_i^t$, $b_i^t$. The exact operation count depends on the instruction set implemented in the specific GPU hardware. We use the number of multiply-and-add floating point operations, but different choices (e.g. vector operations) will result in similar conclusions. The number of operations is reported on the right of each line of pseudocode.

Algorithm 1 LBS: Approximated lighting

Require: $\omega_i[s], M_i^t[s] \ (\forall s \in B_i)$
1: $T'_i = \sum_{s \in B_i} \omega_i[s] M_i^t[s]$ \quad $\triangleright 12 N_{max}$
2: $p'_i = T'_i p_i^0$ \quad $\triangleright 12$
3: $n'_i = T'_i n_i^0$ \quad $\triangleright 9$
4: $t'_i = T'_i t_i^0$ \quad $\triangleright 9$
5: $b'_i = T'_i b_i^0$ \quad $\triangleright 9$

The approximated solution (Eq. 3, see paper) first blends the transformation matrices (Line 1), and then uses the blended matrix to transform the vertex coordinates, the normal and the two tangential vectors (Lines 2-5). The blending is performed only over $3 \times 3$ matrices, including the translational part. The transformation of the positions in Line 2 requires both the rotation and the translational part of the LBS transformation, while the other quantities require only the rotational part and are thus cheaper to evaluate.

In the following, by $B_i^*$ we intend the set of bone indices for vertex $i$ except $s_0$ i.e.
$$B_i^* = B_i \setminus \{s_0\}$$

2. LBS: direct approach

Here we evaluate the necessary number of operations for a direct computation of Eq. (2) (see paper).

Algorithm 2 LBS: direct

Require: $\omega_i[s], M_i^t[s], a_i[s] \ (\forall s \in B_i)$
1: $T'_i = \sum_{s \in B_i} \omega_i[s] M_i^t[s]$ \quad $\triangleright 12 \ N_{max}$
2: $q'_i(i) = M'_i[s] p_i^t \ (\forall s \in B_i)$ \quad $\triangleright 9 \ N_{max}$
3: $J'_i = T'_i + \sum_{s \in S} q'_i(a_i[s]) T'$ \quad $\triangleright 9 \ N_{max}$
4: $J'_i = J'_i^{-T}$ \quad $\triangleright 31$
5: $p'_i = J'_i p_i^0$ \quad $\triangleright 12$
6: $n'_i = J'_i n_i^0$ \quad $\triangleright 9$
7: $t'_i = J'_i t_i^0$ \quad $\triangleright 9$
8: $b'_i = J'_i b_i^0$ \quad $\triangleright 9$

A naive implementation requires to evaluate the correct Jacobian of LBS (Lines 1-3) and invert it (Line 4). Apart from being expensive, this method also requires to send 3 additional floats per weight function to the shader for the $a_i[s]$.

3. LBS: accurate lighting (our method)

Our algorithm avoids an explicit computation of the Jacobian and its inverse. This makes it considerably more efficient than the previous algorithm and also reduces the amount of information passed to the shader, since only the $\omega_i[s]$ and $q_i[s]$ are necessary per weight function after the first one. Note that in Line 1 we only need the top-left $3 \times 3$ submatrix of $T$. 

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and Blackwell Publishing Ltd. Published by Blackwell Publishing, 9601 Garsington Road, Oxford OX4 2DQ, UK and 150 Main Street, Malden, MA 02148, USA.
Algorithm 3 LBS: accurate lighting (our method)

Require: $\omega_s$, $M_i$ (\(\forall s \in B_i\)), $\omega_s$, $b_i$ (\(\forall s \in B_i\))
1. $T^{3 \times 3} = \sum_{s \in B_i} \omega_s M_i^{3 \times 3}$\(\omega_s\) \(\triangleright 9 N_{max}\)
2. $q_i = M_i^{3 \times 3} p_i$ (\(\forall s \in B_i\)) \(\triangleright 9 N_{max}\)
3. $q_i = q_i - q_i[3]_0$ (\(\forall s \in B_i\)) \(\triangleright 3 (N_{max} - 1)\)
4. $t_i = T^{3 \times 3} t_i + \sum_{s \in B_i} \omega_s q_i[1]_s$ \(\triangleright 9 + 3 (N_{max} - 1)\)
5. $b_i = T^{3 \times 3} b_i + \sum_{s \in B_i} \omega_s q_i[0]_s$ \(\triangleright 9 + 3 (N_{max} - 1)\)
6. $n_i = t_i \times b_i$ \(\triangleright 6\)
7. $p_i = q_i[0]_s + \sum_{s \in B_i} \omega_s q_i[3]_s$ \(\triangleright 3 (N_{max} - 1)\)

Algorithm 4 DQS: approximated lighting

Require: $\omega_s$, $q_i$ (\(\forall s \in B_i\))
1. $Q' = \sum_{s \in B_i} \omega_s q_i$ \(\triangleright 12 N_{max}\)
2. $Q'$: normalize($Q'$) \(\triangleright 13\)
3. $p_i' = Q'(p_i)$ \(\triangleright 27\)
4. $n_i' = Q'(n_i)$ \(\triangleright 18\)
5. $t_i' = Q'(t_i)$ \(\triangleright 18\)
6. $b_i' = Q'(b_i)$ \(\triangleright 18\)

Algorithm 5 DQS: accurate lighting (our)

Require: $\omega_s$, $q_i$ (\(\forall s \in B_i\))
1. $Q' = \sum_{s \in B_i} \omega_s q_i$ \(\triangleright 12 N_{max}\)
2. $Q'$: normalize \(\triangleright 13\)
3. $p_i' = Q'(p_i)$ \(\triangleright 27\)
4. $q_i[0]_s = Q'(q_i[0]_s)$ (\(\forall s \in B_i\)) \(\triangleright 24 N_{max}\)
5. $q_i[3]_s = q_i[3]_s - q_i[3]_0$ (\(\forall s \in B_i\)) \(\triangleright 3 (N_{max} - 1)\)
6. $t_i' = Q' t_i + \sum_{s \in B_i} \omega_s q_i[1]_s$ \(\triangleright 18 + 3 (N_{max} - 1)\)
7. $b_i' = Q' b_i + \sum_{s \in B_i} \omega_s q_i[0]_s$ \(\triangleright 18 + 3 (N_{max} - 1)\)
8. $n_i' = t_i' \times b_i'$ \(\triangleright 6\)

In line 4, applying a quaternion to positions costs 3 fewer operations because only differences between the results are used, leading to a small optimization (see attached code).

4. DQS: standard approximation

It is beyond the scope of this appendix to detail the operation cost of standard Dual Quaternion Skinning, and the reader is referred to the attached code. We will assume that the blending (“mix” in the pseudo-code) of $x$ dual quaternions costs $12x$ (because before linearly interpolating the 8 scalars we must check whether each addend quaternion must be flipped); dual quaternion normalization costs 13 operations; applying a dual quaternion to a vector costs 18 operations and to a position 27 operations. With $Q(p)$ we denote the application of a dual quaternion to either a position or a vector.

5. DQS: accurate lighting (our method)

The adaptation of our algorithm to the case of DQS works as follows:

References