EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

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 $\mathbf{p}_i^t$, normal $\mathbf{n}_i^t$, and the tangents $\mathbf{t}_i^t$, $\mathbf{b}_i^t$. 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: $\omega_i[s], \mathbf{M}^t[s] \ \ (\forall s \in \mathcal{B}_i)$

1: $\mathbf{T}_i^t = \sum_{s\in\mathcal{B}_i} \omega_i[s]\,\mathbf{M}^t[s]$ ▷ 12 $N_{\max}$
2: $\mathbf{p}_i^t = \mathbf{T}_i^t \mathbf{p}_i^0$ ▷ 12
3: $\mathbf{n}_i^t = \mathbf{T}_i^t \mathbf{n}_i^0$ ▷ 9
4: $\mathbf{t}_i^t = \mathbf{T}_i^t \mathbf{t}_i^0$ ▷ 9
5: $\mathbf{b}_i^t = \mathbf{T}_i^t \mathbf{b}_i^0$ ▷ 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 3×4 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 $\mathcal{B}^*$ we intend the set of bone indices for vertex i except s_0 i.e.

$$\mathcal{B}_i^* = \mathcal{B}_i \setminus \{s_0\}$$

2. LSB: direct approach

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

Algorithm 2 LBS: direct

Require: $\omega_i[s], \mathbf{M}^t[s], \mathbf{a}[s] \ \ (\forall s \in \mathcal{B}_i)$

1: $\mathbf{T}_i^t = \sum_{s\in\mathcal{B}_i} \omega_i[s]\,\mathbf{M}^t[s]$ ▷ 12 $N_{\max}$
2: $\mathbf{q}_j^t(i) = \mathbf{M}^t[s]\mathbf{p}_i^t \ \ (\forall s \in \mathcal{B}_i)$ ▷ 9 $N_{\max}$
3: $\mathbf{J}_i^t = \mathbf{T}_i^t + \sum_{s\in\mathcal{S}} \mathbf{q}_i^t[s]\ \mathbf{a}_i[s]^T$ ▷ 9 $N_{\max}$
4: $\mathbf{J}_i' = \mathbf{J}_i^{t\,-T}$ ▷ 31
5: $\mathbf{p}_i^t = \mathbf{T}_i^t\,\mathbf{p}_i^0$ ▷ 12
6: $\mathbf{n}_i^t = \mathbf{J}_i' \mathbf{n}_i^0$ ▷ 9
7: $\mathbf{t}_i^t = \mathbf{J}_i^t \mathbf{t}_i^0$ ▷ 9
8: $\mathbf{b}_i^t = \mathbf{J}_i^t \mathbf{b}_i^0$ ▷ 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 $\mathbf{a}[s]$.

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 $\alpha[s]$ and $\beta[s]$ are necessary per every weight function after the first one. Note that in Line 1 we only need the top-left 3×3 submatrix of $\mathbf{T}$.

Algorithm 3 LBS: accurate lighting (our method)

Require: $\omega_i[s]$, $\mathbf{M}^t[s]$ $(\forall s \in \mathcal{B}_i)$, $\alpha_i[s], \beta_i[s]$ $(\forall s \in \mathcal{B}_i^*)$

1: $\mathbf{T}_i^{3\times3} = \sum_{s\in\mathcal{B}_i} \omega_i[s]\,\mathbf{M}^{3\times3}[s]$ ▷ 9 $N_{\max}$
2: $\mathbf{q}_i^t[s] = \mathbf{M}^t[s]\mathbf{p}_i^t$ $(\forall s \in \mathcal{B}_i)$ ▷ 9 $N_{\max}$
3: $\mathbf{q}_i[s]' = \mathbf{q}_i^t[s] - \mathbf{q}_i^t[s_0]$ $(\forall s \in \mathcal{B}_i^*)$ ▷ $3(N_{\max}-1)$
4: $\mathbf{t}_i^t = \mathbf{T}_i^{3\times3}\,\mathbf{t}_i^0 + \sum_{s\in\mathcal{B}_i^*} \alpha_i[s]\mathbf{q}_i^t[s]'$ ▷ $9+3(N_{\max}-1)$
5: $\mathbf{b}_i^t = \mathbf{T}_i^{3\times3}\,\mathbf{b}_i^0 + \sum_{j\in\mathcal{B}_i^*} \beta_i[s]\mathbf{q}_i^t[s]'$ ▷ $9+3(N_{\max}-1)$
6: $\mathbf{n}_i^t = \mathbf{t}_i^t \times \mathbf{b}_i^t$ ▷ 6
7: $\mathbf{p}_i^t = \mathbf{q}_i^t[s_0] + \sum_{s\in\mathcal{B}_i^*} \omega_i[s]\mathbf{q}_i^t[s]'$ ▷ $3(N_{\max}-1)$

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 $\mathcal{Q}(\mathbf{p})$ we denote the application of a dual quaternion to either a position or a vector.

Algorithm 4 DQS: approximated lighting

Require: $\omega_i[s]$, $\mathcal{Q}^t[s]$ $(\forall s \in \mathcal{B}_i)$

1: $\mathcal{Q}^t = \text{mix}[s](\omega_i[s], \mathcal{Q}^t[s])$ ▷ 12 $N_{\max}$
2: $\mathcal{Q}^t$: normalize$(\mathcal{Q}^t)$ ▷ 13
3: $\mathbf{p}_i^t = \mathcal{Q}^t(\mathbf{p}_i^0)$ ▷ 27
4: $\mathbf{n}_i^t = \mathcal{Q}^t(\mathbf{n}_i^0)$ ▷ 18
5: $\mathbf{t}_i^t = \mathcal{Q}^t(\mathbf{t}_i^0)$ ▷ 18
6: $\mathbf{b}_i^t = \mathcal{Q}^t(\mathbf{b}_i^0)$ ▷ 18

5. DQS: accurate lighting (our method)

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

Algorithm 5 DQS: accurate lighting (our)

Require: $\omega_i[s]$, $\mathcal{Q}^t[s]$ $(\forall s \in \mathcal{B}_i)$

1: $\mathcal{Q}^t = \text{mix}[s](\omega_i[s], \mathcal{Q}^t[s])$ ▷ 12 $N_{\max}$
2: $\mathcal{Q}^t$: normalize ▷ 13
3: $\mathbf{p}_i^t = \mathcal{Q}^t(\mathbf{p}_i^0)$ ▷ 27
4: $\mathbf{q}_i^t[s] = \mathcal{Q}^t[s](\mathbf{p}_i^0)$ $(\forall s \in \mathcal{B}_i)$ ▷ 24 $N_{\max}$
5: $\mathbf{q}_i[s]' = \mathbf{q}_i^t[s] - \mathbf{q}_i^t[s_0]$ $(\forall s \in \mathcal{B}_i^*)$ ▷ $3(N_{\max}-1)$
6: $\mathbf{t}_i^t = \mathcal{Q}^t\mathbf{t}_i^0 + \sum_{s\in\mathcal{B}_i^*} \alpha_i[s]\mathbf{q}_i^t[s]'$ ▷ $18+3(N_{\max}-1)$
7: $\mathbf{b}_i^t = \mathcal{Q}^t\mathbf{b}_i^0 + \sum_{s\in\mathcal{B}_i^*} \beta_i[s]\mathbf{q}_i^t[s]'$ ▷ $18+3(N_{\max}-1)$
8: $\mathbf{n}_i^t = \mathbf{t}_i^t \times \mathbf{b}_i^t$ ▷ 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).

References

[TPSH14] TARINI M., PANOZZO D., SORKINE-HORNUNG O.: Accurate and efficient lighting for skinned models. *Computer Graphics Forum (proceedings of EUROGRAPHICS) 33*, 6 (2014). 1