High-Quality Object-Space Dynamic Ambient Occlusion for Characters using Bi-level Regression

Binh Huy Le
SEED - Electronic Arts
Los Angeles, CA
ble@ea.com

Henrik Halen
SEED - Electronic Arts
Los Angeles, CA
hhalen@ea.com

Carlos Gonzalez-Ochoa
SEED - Electronic Arts
Los Angeles, CA
cgoa@ea.com

JP Lewis
SEED - Electronic Arts
Los Angeles, CA
noisebrain@gmail.com

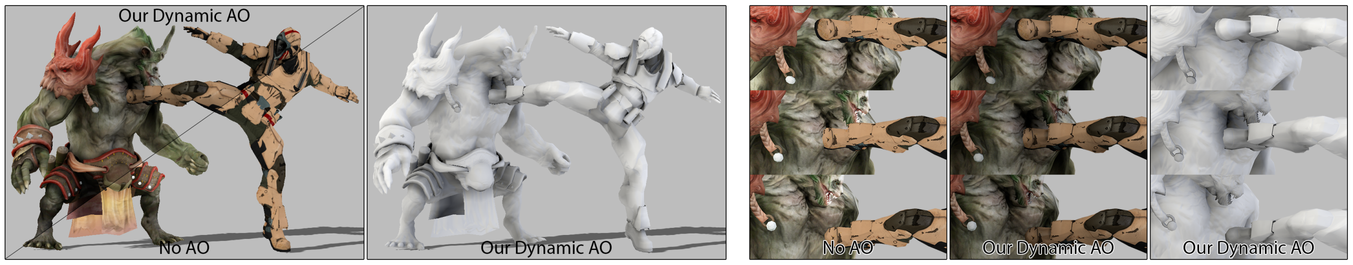


Figure 1: Our Dynamic Ambient Occlusion (AO) model produces global illumination effects for character animation. Notice how the effects change with different poses (right).

ABSTRACT

The widely used ambient occlusion (AO) technique provides an approximation of some global illumination effects and is efficient enough for use in real-time applications. Because it relies on computing the visibility from each point on a surface, AO computation is expensive for dynamically deforming objects, such as characters in particular. In this paper, we describe an algorithm for producing high-quality dynamically changing AO for characters. Our fundamental idea is to factorize the AO computation into a coarse-scale component in which visibility is determined by approximating spheres, and a fine-scale component that leverages a skinning-like algorithm for efficiency, with both components trained in a regression against ground-truth AO values. The resulting algorithm accommodates interactions with external objects and generalizes without requiring carefully constructed training data. Extensive comparisons illustrate the capabilities and advantages of our algorithm.

CCS CONCEPTS

• **Computing methodologies** → **Animation**; **Rendering**; *Supervised learning by regression*; • **Mathematics of computing** → *Nonconvex optimization.*

KEYWORDS

rendering, ambient occlusion, skinning, skeletal animation

ACM Reference Format:
Binh Huy Le, Henrik Halen, Carlos Gonzalez-Ochoa, and JP Lewis. 2019. High-Quality Object-Space Dynamic Ambient Occlusion for Characters using Bi-level Regression. In *Symposium on Interactive 3D Graphics and Games (I3D '19), May 21–23, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3306131.3317029

1 INTRODUCTION

Realistic images require global illumination effects. Direct computation of global illumination is not possible for real-time applications, however, since single images of complex scenes can take minutes or hours to compute [Christensen and Jarosz 2016; Dutre et al. 2006]. Hardware support for ray tracing may address this problem in the future [RTX 2018], but existing solutions fall short of the performance needed to render complete complex scenes in real time. As such the use of hardware assisted ray tracing will likely be focused towards effects such as refraction, order-independent transparency, and complex shadows that are difficult or impossible to achieve with existing real-time rendering techniques. Another recent idea is to use machine learning techniques to filter and inpaint incomplete and high-variance samples from monte-carlo rendering that has not yet run to convergence [Bako et al. 2017; Chaitanya et al. 2017]. While extremely promising, on current hardware these techniques still fall short of the performance needed to produce production-quality complex scenes in real-time.

The cost of global illumination has resulted in the development of techniques, such as shadow mapping [Williams 1978], horizon mapping [Max 1988], precomputed radiance transfer [Sloan et al. 2002], and others, that cheaply approximate some of the first order effects of global illumination. Among these, ambient occlusion (AO) [Christensen 2002; Landis 2002b] is a proven technique that is widely used in games. The AO value for every point in a 3D scene is defined as its exposure to the ambient lighting. Equivalently, the AO value at a point can be computed from the amount of shadow from all other points casting on it. Under the assumption of static geometry, the AO value can be *precomputed* and stored in a texture for use in a real-time shader, where the value simply attenuates the indirect lighting component.

Computing AO for animated objects such as characters is expensive. This has led to screen-space ambient occlusion (SSAO) methods, which efficiently approximate AO using only screen-space (e.g. deferred shading) information available at run-time, thereby allowing AO for dynamic objects. However, SSAO is a further approximation on AO, and is limited to relatively short-range effects that can be computed from screen-space information. As a result, while games and other interactive experiences often have very realistic background elements such as vehicles, the characters (which are often the main point of interest) fall short.

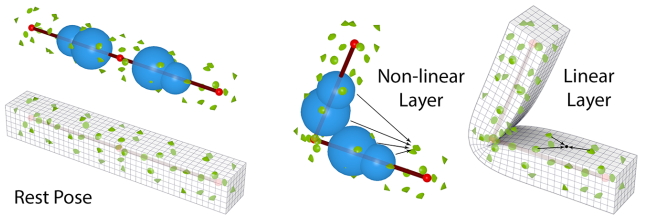


Figure 2: Illustration of our AO model with two layers: the non-linear layer first transforms proxy spheres (blue) and key points (green) by the current skeleton pose (red), and then, the linear layer interpolates the per-vertex AO from values at key points.

In this paper, we compute dynamic AO in object space, using a custom machine learning regression approach.

Our overall idea is to factorize the problem into a nonlinear, coarse resolution component that captures the nonlinear articulation effects of the body, and a higher-resolution linear interpolation that leverages the low-dimensional computation and hardware support of a skinning model. Our decomposition resembles some radiosity approaches that decompose the scene into emitters (similar to our course-resolution component) and receivers (analogous to our high-resolution component) [Arikan et al. 2005; Silvennoinen and Lehtinen 2017]. Fig. 2 gives an overview of our model.

Our contributions include:

- AO model: our model is highly compatible with de-facto skinning model (Linear Blend Skinning). This gives both flexibility and good generalization. We can train parts of the model individually and combine them later.
- New non-linear kernel functions: our solution is differentiable through its entire useful range, while closely approximating the closed-form solution.
- Training/optimization: our robust initialization utilizes skinning weights to handle multi-part models. We use caching to obtain fast value and gradient calculations.
- Unlike some machine learning models, our method provides interpretability that enables easy manipulations such as character-to-character interactions.
- As shown in the evaluation, our solution is fast, robust, and has better generalization than previous character-specific methods.

2 RELATED WORK

While precomputed (baked) AO [Kavan et al. 2011; Landis 2002a; Larsson and Halen 2009; Zhukov et al. 1998] is often an acceptable approximation for *static* objects, it fails for moving objects. In particular, precomputed AO cannot produce dynamic AO effects on characters, such as the dynamically changing occlusion around the elbow or knees as they bend, or the effect of holding a hand close to the body.

The main existing solution for dynamic AO is screen space ambient occlusion (SSAO) [Mittring 2007]. SSAO rasterizes the scene to a depth buffer and uses this as a surrogate for the original geometry for the purpose of AO computation. For each pixel in this depth buffer, SSAO also discards the shadowing effect of distant pixels to reduce the computation. SSAO is widely used because it is independent of scene complexity and handles dynamic geometry. However, because of these two strategies, SSAO typically has artifacts as shown in Figs. 3 and 9. SSAO only generates shadows locally, e.g. near the creases, and these shadows fall off too quickly with distance. SSAO can also discard the shadow contribution of geometry hidden from the camera view.

Holden et al. [2016] and Nalbach et al. [2017] formulated AO as a regression from screen-space input data to the ground-truth AO (and other effects [Nalbach et al. 2017]). Since the AO values used to train the regression can be precomputed using high-quality global-illumination, the results improve upon SSAO. Effects that cannot be deduced purely from screen-space information are still difficult to handle in these methods however. Unfortunately characters prominently feature one such effect: occlusions caused by limbs. Consider the case where an arm crosses in front of the torso, as seen for example in Figs. 8 and 9. Depending on the distance between the arm and torso, AO may or may not be required, but this is difficult to entirely deduce from only local screen-space information. Given the importance of characters, methods that handle this problem are required.

Kontkanen and Aila [2006] were the first to develop a dynamic AO method targeted to characters. They formulate the problem as a linear regression from pose to the AO value. This method is both particularly simple and efficient, however, the restriction to linear regression may limit the types effects that are obtained.

Kirk and Arikan [2007] introduced another dynamic AO method for characters. This method moves beyond the linearity restriction by using a "piecewise linear" collection of locally linear models. More specifically, they cluster poses using k-Means, then compress the representation for each pose using PCA, and represent AO as a function of the pose. A moving least squares step is used to smooth the boundaries in pose space between the local linear models. The paper shows effective AO effects on characters. As is acknowledged in the paper, this method requires careful tuning of several types of parameters. Artifacts result if too few clusters are used, or if the retained dimensionality of the PCA is too small, whereas choosing these values too large results in excess memory use (and to a lesser extent, unneeded computation). The number of clusters needed may vary depending on the type of motion, and one must ensure that the motion sample used for clustering contains the range of poses that will be encountered online. Further, the results of the k-Means algorithm depend on its initialization (which is usually random) [Arthur and Vassilvitskii 2007], so several iterations of trial-and-error tuning may be required in the worst case.

In summary, previous research has shown the viability of custom dynamic AO algorithms for characters, but there are trade-offs in terms of the range of effects that can be obtained or the amount of parameter tuning required. Our method generates very high quality results with very limited parameter tuning. It also has no requirement for training data that anticipates the poses to be encountered at runtime, as it successfully generalizes from very generic training data.

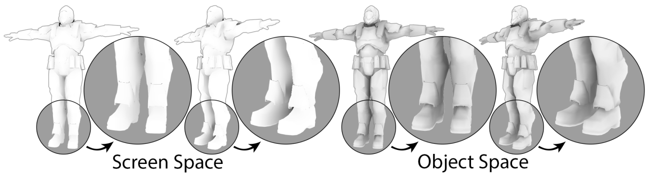


Figure 3: The artifacts of SSAO (left) compared to our object space AO (right). Notice the shadow of SSAO on the leg and foot changes with the view point as SSAO does not retain the geometry of the object. The shadows of SSAO quickly drop to zero as we go further from creases. SSAO results are rendered by Maya 2018's viewport 2.0 with DirectX 11[1]. The rendering resolution is 1024×1024 and the radius of the sampling area is 64.

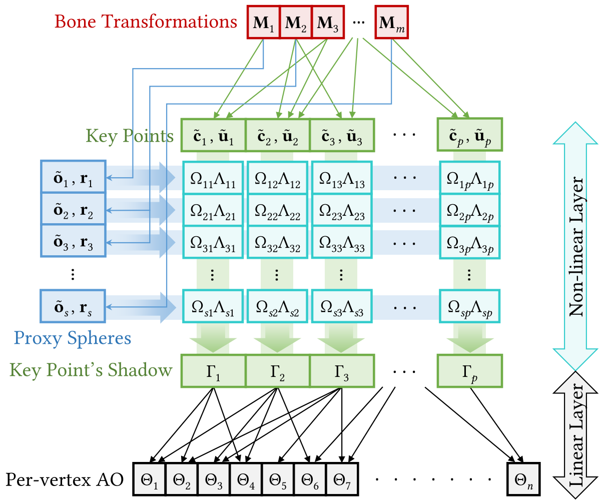

Bone Transformations
$\mathbf{M}_1$ $\mathbf{M}_2$ $\mathbf{M}_3$ ··· $\mathbf{M}_m$
Key Points
$\tilde{\mathbf{c}}_1, \tilde{\mathbf{u}}_1$ $\tilde{\mathbf{c}}_2, \tilde{\mathbf{u}}_2$ $\tilde{\mathbf{c}}_3, \tilde{\mathbf{u}}_3$ ··· $\tilde{\mathbf{c}}_p, \tilde{\mathbf{u}}_p$
$\tilde{\mathbf{o}}_1, \mathbf{r}_1$ $\tilde{\mathbf{o}}_2, \mathbf{r}_2$ $\tilde{\mathbf{o}}_3, \mathbf{r}_3$ $\tilde{\mathbf{o}}_s, \mathbf{r}_s$
$\Omega_{11}\Lambda_{11}$ $\Omega_{12}\Lambda_{12}$ $\Omega_{13}\Lambda_{13}$ $\Omega_{1p}\Lambda_{1p}$
$\Omega_{21}\Lambda_{21}$ $\Omega_{22}\Lambda_{22}$ $\Omega_{23}\Lambda_{23}$ $\Omega_{2p}\Lambda_{2p}$
$\Omega_{31}\Lambda_{31}$ $\Omega_{32}\Lambda_{32}$ $\Omega_{33}\Lambda_{33}$ $\Omega_{3p}\Lambda_{3p}$
$\Omega_{s1}\Lambda_{s1}$ $\Omega_{s2}\Lambda_{s2}$ $\Omega_{s3}\Lambda_{s3}$ $\Omega_{sp}\Lambda_{sp}$
Proxy Spheres
Key Point's Shadow
Γ_1 Γ_2 Γ_3 ··· Γ_p
Per-vertex AO
Θ_1 Θ_2 Θ_3 Θ_4 Θ_5 Θ_6 Θ_7 ······· Θ_n
Non-linear Layer
Linear Layer

Figure 4: The pipeline of our computing model, where the transformations of proxy spheres, the transformations of key points, and the per-vertex AO interpolations are controlled by fitting parameters.

3 COMPUTING MODEL

The input of our computing model is the skeleton pose $\mathbb{M}$ with m rigid bones. The transformation of bone $j = 1..m$ is denoted by the matrix $\mathbf{M}_j = \left[\mathbf{R}_j | \mathbf{t}_j\right]$ where $\mathbf{R}_j \in \mathbb{R}^{3\times3}$ is the rotation matrix and $\mathbf{t}_j \in \mathbb{R}^3$ is the translation vector.

Our computing model includes two layers as illustrated in Fig. 2:

- The dense, non-linear layer (§3.1) computes the transformed proxy spheres and key points (with their normals) from their rest pose. Then, all proxy spheres (shadow emitters) cast shadow on every key point (shadow receivers). The dense combination (Cartesian product) of proxy spheres and key points captures the global interaction between parts of the model when its pose changes. This step outputs a low resolution occlusion map stored at each key point.
- The sparse, linear layer (§3.2) up-samples the low resolution occlusion map to the whole model, i.e. per-vertex AO output, where the AO at each vertex can be linearly inferred from some nearest key points.

The parameters of our computing model (illustrated in Fig. 4) will be optimized by the model fitting (§4). They include:

- s proxy spheres, where s can be directly or indirectly controlled by the user. Proxy sphere $h = 1..s$ is rigid bounded to bone $b(h)$, i.e. sphere h will be transformed by $\mathbf{M}_{b(h)}$ only. The center of h at the rest pose is $\mathbf{o}_h \in \mathbb{R}^3$. The radius of h is $r_j \in \mathbb{R}$, which is unchanged during animation.
- p key points, where p is set by the user. At the rest pose, the position and normal of key point $k = 1..p$ are $\mathbf{c}_k \in \mathbb{R}^3$ and $\mathbf{u}_k \in \mathbb{R}^3$, respectively. Key point k is smoothly bound to m bones, i.e. it will be transformed by a linear blending of all $\{\mathbf{M}_j | \forall j = 1..m\}$. Let $w_k \in \mathbb{R}^m$ be the blending weights of key point k, where w_{kj} denotes the weight w.r.t. bone j. w_k is sparse and affine.
- n linear regression weights and biases, where n is the number of vertices of the model. The weights and bias of vertex $i = 1..n$ are $\alpha_i \in \mathbb{R}^p$ and $\beta_i \in \mathbb{R}$, respectively, where α_{ik} denotes the weight w.r.t. key point k. α_i is sparse, non-negative, and soft-constrained affine. The number of non-zero values, n_{nz}, is defined by the users.

The output of our model is n per-vertex AO values, where $\Theta_i \in \mathbb{R}, 0 \leq \Theta_i \leq 1$ is the AO value at vertex $i = 1..n$.

3.1 Non-linear Layer

This layer is inspired by the classical idea of approximating AO from sphere proxies with modifications to reduce stress on the run-time model and parameter optimization. Our design philosophy avoids discontinuous, branching geometry intersections (intersect/no intersect) and replaces them with continuous, differentiable, non-branching approximations.

We first compute the transformation of skeleton pose $\mathbb{M} = \{\mathbf{M}_j | j = 1..m\}$ on each proxy sphere $h = 1..s$ and each key point $k = 1..p$ by using Eq. (1), where the tilde with super script $(\tilde{\bullet}_h^{\mathbb{M}})$ denotes the transformed position from the rest pose $\bullet$, and normalize($\bullet$) denotes the vector normalization function. We recall that each proxy

[1]https://www.autodesk.com/products/maya

sphere is rigid bound to one bone. Therefore, we need neither blending transformations nor changing its radius.

$$\tilde{\mathbf{o}}_h^{\mathbb{M}} = \mathbf{R}_{b(h)}\mathbf{o}_h + \mathbf{t}_{b(h)} \tag{1a}$$

$$\tilde{\mathbf{c}}_k^{\mathbb{M}} = \sum_{j=1}^{m} w_{kj}(\mathbf{R}_j\mathbf{c}_k + \mathbf{t}_j) \tag{1b}$$

$$\tilde{\mathbf{u}}_k^{\mathbb{M}} = \text{normalize}\left(\sum_{j=1}^{m} w_{kj}\mathbf{R}_j\mathbf{u}_k\right) \tag{1c}$$

Then, we compute the shadow of each proxy sphere h casting on key point k as the product $\Omega_{hk}^{\mathbb{M}}\Lambda_{hk}^{\mathbb{M}}$, where $\Omega_{hk}^{\mathbb{M}}$ estimates the normalized solid angle that sphere h covers, i.e. solid angle scaled down by 2π, and $\Lambda_{hk}^{\mathbb{M}}$ estimates its visibility ratio, i.e. ratio of h inside the hemisphere angle defined by $\tilde{\mathbf{u}}_k^{\mathbb{M}}$. The normalization aims to scale $\Omega_{hk}^{\mathbb{M}}\Lambda_{hk}^{\mathbb{M}}$ in the range of $[0, 1]$. The intuition of $\Omega_{hk}^{\mathbb{M}}$ and $\Lambda_{hk}^{\mathbb{M}}$ is illustrated in Fig. 5.

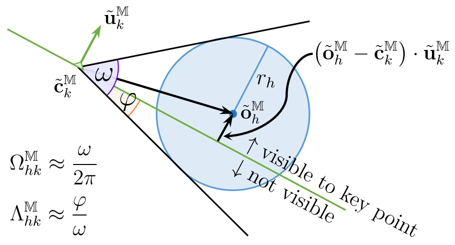

$\tilde{\mathbf{u}}_k^{\mathbb{M}}$
$\tilde{\mathbf{c}}_k^{\mathbb{M}}$
ω
φ
r_h
$\tilde{\mathbf{o}}_h^{\mathbb{M}}$
$(\tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}}) \cdot \tilde{\mathbf{u}}_k^{\mathbb{M}}$
↑ visible to key point
↓ not visible
$\Omega_{hk}^{\mathbb{M}} \approx \frac{\omega}{2\pi}$
$\Lambda_{hk}^{\mathbb{M}} \approx \frac{\varphi}{\omega}$

Figure 5: Illustration of the normalized solid angle $\Omega_{hk}^{\mathbb{M}}$ and the visibility ratio $\Lambda_{hk}^{\mathbb{M}}$ at key point k with respect to proxy sphere h. $\tilde{\mathbf{c}}_k$ and $\tilde{\mathbf{u}}_k$ are the position and the normal of the key point, respectively. $\tilde{\mathbf{o}}_h^{\mathbb{M}}$ and r_h are the center and the radius of the proxy sphere, respectively.

3.1.1 *Normalized Solid Angle.*

$\Omega_{hk}^{\mathbb{M}}$ is computed with Eq. (2), where $\delta_{hk}^{\mathbb{M}}$ is the ratio between the distance from key point $\tilde{\mathbf{c}}_k^{\mathbb{M}}$ to center $\tilde{\mathbf{o}}_h^{\mathbb{M}}$ and the sphere radius r_h. Notice that as we use simple spheres and points (with normals), $\Omega_{hk}^{\mathbb{M}}$ can be computed in closed-form [Sloan et al. 2007]: $\Omega^* = 1 - \sqrt{1 - 1/\delta_{hk}^{\mathbb{M}\,2}}$. However, this function is C^1 discontinuous at 1 and undefined with $\delta_{hk}^{\mathbb{M}} < 1$, i.e. key point k is inside proxy sphere h. Ω^* will not be convenient for the model training. For this reason, we created $\Omega_{hk}^{\mathbb{M}}$ by first plotting Ω^*, then searching for an activation function that has a form like Ω^*, and finally tuning coefficients to match with Ω^*. The function $\Omega_{hk}^{\mathbb{M}}$ here is a softsign function that passes close to the point $(1, 1)$.

$$\Omega_{hk}^{\mathbb{M}} = \frac{3}{4} - \frac{3\left(\delta_{hk}^{\mathbb{M}} - 1\right)}{1 + 4\left|\delta_{hk}^{\mathbb{M}} - 1\right|} \tag{2a}$$

$$\text{where: } \delta_{hk}^{\mathbb{M}} = \frac{\left\|\tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}}\right\|_2}{r_h} \tag{2b}$$

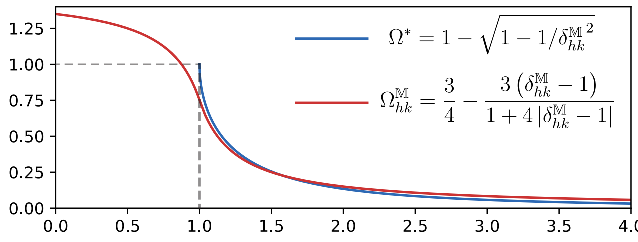


Figure 6: Our smooth normalized solid angle function $\Omega_{hk}^{\mathbb{M}}$ (red) can approximate the closed-form function Ω^* (blue) and extrapolate to the range of $0 \leq \delta_{hk}^{\mathbb{M}} < 1$.

3.1.2 *Visibility Ratio.*

$\Lambda_{hk}^{\mathbb{M}}$ is estimated in Eq. (3) from the signed projection length of the distance vector $\tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}}$ on the normal $\tilde{\mathbf{u}}_k$. Referring to the illustration in Fig. 5, we can see that proxy sphere h is totally visible ($\Omega = 1$) if the signed projection length $\left(\tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}}\right) \cdot \tilde{\mathbf{u}}_k^{\mathbb{M}} \geq r_h$, i.e. $\xi_{hk}^{\mathbb{M}} \geq 1$, and h is totally not visible ($\Omega = 0$) if $\left(\tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}}\right) \cdot \tilde{\mathbf{u}}_k^{\mathbb{M}} \leq -r_h$, i.e. $\xi_{hk}^{\mathbb{M}} \leq -1$. Because this visibility function is discontinuous, we instead use visibility ratio function $\Lambda_{hk}^{\mathbb{M}}$ as a sigmoid (algebraic) function. Our $\Lambda_{hk}^{\mathbb{M}}$ is plotted in Fig. 7. While there is a closed-form solution of this visibility problem in [Mazonka 2012], that solution is C^1 discontinuous at $\left(\tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}}\right) \cdot \tilde{\mathbf{u}}_k^{\mathbb{M}} = \pm r_h$.

$$\Lambda_{hk}^{\mathbb{M}} = \frac{1}{2} + \frac{\xi_{hk}^{\mathbb{M}}}{\sqrt{1 + 4\xi_{hk}^{\mathbb{M}\,2}}} \tag{3a}$$

$$\text{where: } \xi_{hk}^{\mathbb{M}} = \frac{\left(\tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}}\right) \cdot \tilde{\mathbf{u}}_k^{\mathbb{M}}}{r_h} \tag{3b}$$

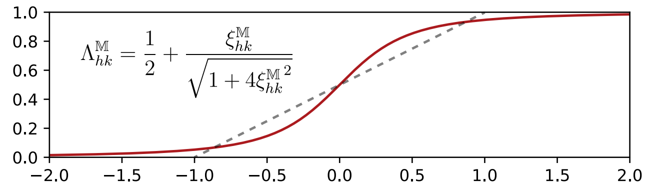


Figure 7: Our visibility ratio function is a smooth function that passes close to $(\xi_{hk}^{\mathbb{M}} = -1, \Lambda_{hk}^{\mathbb{M}} = 0)$ and $(\xi_{hk}^{\mathbb{M}} = 1, \Lambda_{hk}^{\mathbb{M}} = 1)$.

3.1.3 *Sparse Shadow Value.*

$\Gamma_k^{\mathbb{M}}$ at key point k, i.e. the output of the non-linear layer, is computed by adding contributions from all proxy spheres with a γ-norm function (Eq. (4)), where the parameter $\gamma > 1$ is set by the users. The γ-norm reduces double shadowing, i.e. spheres with overlapping solid angles will cast shadow twice, by emphasizing contributions from spheres with most shadow, i.e. sparsifying the contribution vector similar to the maximum norm ($\gamma = \infty$). Our γ-norm approximation significantly reduces the computation cost compared to the traditional geometry-based multi-pass technique [Bunnell 2005], i.e. our complexity is linear with the number of spheres compared to the quadratic order (two passes to remove double shadowing) or cubic order (three passes to add triple shadowing back).

$$\Gamma_k^{\mathbb{M}} = \left(\sum_{h=1}^{s} \left(\Omega_{hk}^{\mathbb{M}} \Lambda_{hk}^{\mathbb{M}} \right)^{\gamma} \right)^{\frac{1}{\gamma}} \tag{4}$$

3.2 Linear Layer

The AO value of vertex i at pose $\mathbb{M}$ is the linear combination of shadow values at key points $\Gamma_k^{\mathbb{M}}$ followed by remapping back to the range of $[0, 1]$ (Eq. (5)). Note that depending on the pose $\mathbb{M}$ and the parameters, the actual value of $\Theta_i^{\mathbb{M}}$ could go out of the range $[0, 1]$. For rendering, we could clamp this value at the final step. We recall that the weight vector α_i is sparse, non-negative, and soft-constrained affine, i.e. $\sum_{k=1}^{p} \alpha_{ik} \approx 1$. This soft affinity constraint is handled in the optimization step (§4.2). A regression bias β_i is added to encode local detail at the vertex:

$$\Theta_i^{\mathbb{M}} = 1 - \left(\sum_{k=1}^{p} \alpha_{ik} \Gamma_k^{\mathbb{M}} + \beta_i \right) \tag{5}$$

Algorithm 1 summarizes our AO computing steps. All **foreach** loops can be parallelized.

ALGORITHM 1: Compute Per-vertex Ambient Occlusion

```
input      : $\mathbb{M} = \{\mathbf{M}_j | j = 1..m\}$, where $\mathbf{M}_j = [\mathbf{R}_j | \mathbf{t}_j]$.
parameters : $\{\mathbf{o}_h, r_h | h = 1..s\}$, $\{\mathbf{c}_k, \mathbf{u}_k | k = 1..p\}$,
             $\{w_{kj} | k = 1..p, j = 1..m\}$, $\{\alpha_{ik}, \beta_i | i = 1..n, k = 1..p\}$.
output     : $\{\Theta_i^{\mathbb{M}} | i = 1..n\}$.
1  foreach proxy sphere h do
2      Compute center $\tilde{\mathbf{o}}_h^{\mathbb{M}}$ ;                              // Eq. (1a)
3  end
4  foreach key point k do
5      Compute position $\tilde{\mathbf{c}}_k^{\mathbb{M}}$ and normal $\tilde{\mathbf{u}}_k^{\mathbb{M}}$ ;     // Eqs. (1b) and (1c)
6      Compute shadow $\Gamma_k^{\mathbb{M}}$ ;                              // Eqs. (2), (3) and (4)
7  end
8  foreach vertex i do
9      Compute AO value $\Theta_i^{\mathbb{M}}$ ;                              // Eq. (5)
10 end
```

4 MODEL FITTING

Given a skinning model of a character, we find parameters of the AO model (described in §3) by generating sample poses, ray tracing the ground truth AO, and then optimizing the model parameters to minimize sum of squared difference with the ground truth AO.

We used uniform, per-joint sampling and single-bounce, GPU-raytraced AO as detailed in Appendix A. This uniform sampling is general without any special knowledge about the desired target animation. Per-joint sampling contains no combination of different joint rotations, i.e. we do not rotate more than one joint at once, which keeps the size of our training data manageable. However, our model fitting can work with any off the shelf pose sampling and ground truth AO rendering such as artist-made poses or multi-bounce ray tracing.

The skinning deformation model (LBS) might generate self intersecting output geometry. In that case, vertices at the self intersections are rendered black, i.e. they are totally covered from the ambient light. We treat the AO values at these vertices as missing data and we keep updating these values using the prediction of the current model during training.

Let f be the number of sample poses, where $\mathbb{M}^t = \{\mathbf{M}_j^t | j = 1..m\}$ is the set of m bone transformations at pose $1 \le t \le f$. Let $0 \le \mathbf{A}_{ti} \le 1$ be the AO value at vertex i of pose t. Our minimization problem is:

$$\min E = \sum_{t=1}^{f} \sum_{i=1}^{n} \left(\Theta_i^{\mathbb{M}^t} - \mathbf{A}_{ti} \right)^2 \tag{6}$$

We minimize the objective function (6) by the *Block coordinate descent* [Bertsekas 1999] as described in Algorithm 2, which alternatively updates: non-linear layer (spheres $\mathbf{o}$, r, and key points $\mathbf{c}$, $\mathbf{u}$, w), linear layer (α, β), and missing values.

Although a random initialization could work, we suggest initializing the solution as described in Appendix B, which utilizes the geometry properties of the skinning model for a good convergence.

The missing values are updated in the loop from line 8 to line 10, where ϕ is the set of all missing value positions in the matrix $\mathbf{A}$, i.e. vertices i in the self-intersections at pose t.

We stop the algorithm (line 11) when the number of iterations reaches $max\_iters = 500$, or the relative tolerance $\varepsilon = 1e-3$ is reached, i.e. each parameter change less than ε relatively to its current value.

ALGORITHM 2: Block Coordinate Descent Optimization

```
input  : $\mathbf{A} \in \mathbb{R}^{f \times n}$,
         $\{\mathbf{M}_j^t | t = 1..f, j = 1..m\}$,
         initialized values: $\mathbf{o}$, $r$, $\mathbf{c}$, $\mathbf{u}$, $w$, $\alpha$, $\beta$.
output : $\{\mathbf{o}_h, r_h | h = 1..s\}$,
         $\{\mathbf{c}_k, \mathbf{u}_k | k = 1..p\}$,
         $\{w_{kj} | k = 1..p, j = 1..m\}$,
         $\{\alpha_{ik}, \beta_i | i = 1..n, k = 1..p\}$.
1  repeat
2      Cache $\mathcal{A}$ and $\mathcal{B}$ ;                                // Eq. (7)
3      BFGS($max\_local\_iters$) ;                         // §4.1
4      Cache $\mathcal{C}$, $\mathcal{d}$, and $\mathcal{e}$ ;                           // Eq. (8)
5      foreach vertex i do
6          Constrained LS solve $\alpha_i$ and $\beta_i$ ;              // §4.2
7      end
8      foreach $(t, i) \in \phi$ do            // self-intersecting vertices
9          $\mathbf{A}_{ti} \longleftarrow \Theta_i^{\mathbb{M}^t}$ ;            // predict using current model
10     end
11 until $max\_iters$ iterations reached or relative tolerance $\varepsilon$ reached;
```

4.1 Non-linear Layer Update

We update parameters in the non-linear layer (line 2 and 3) by performing $max\_local\_iters = 20$ iterations of the *Broyden-Fletcher-Goldfarb-Shanno algorithm* (BFGS) [Fletcher 1987]. We chose BFGS over other gradient-based optimizers because of its quadratic convergence rate, compared to the sub-linear convergence rate of gradient descent or the linear convergence rate of conjugate gradient descent.

We can effectively compute the objective function (Eq. (6)) and its gradient by re-arranging its terms while taking advantage of the parameters in linear layer (α, β) being fixed:

$$\begin{aligned}
E &= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(\Theta_i^{\mathbb{M}^t} - \mathbf{A}_{ti}\right)^2 \\
&= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(1 - \left(\sum_{k=1}^{p}\alpha_{ik}\Gamma_k^{\mathbb{M}^t} + \beta_i\right) - \mathbf{A}_{ti}\right)^2 \\
&= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(\sum_{k=1}^{p}\sum_{k'=1}^{p}\alpha_{ik}\alpha_{ik'}\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t}\right. \\
&\qquad\qquad \left. +2\sum_{k=1}^{p}\alpha_{ik}\left(1 - \beta_i - \mathbf{A}_{ti}\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}\right) \\
&= \sum_{t=1}^{f}\sum_{k=1}^{p}\sum_{k'=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\alpha_{ik'}\right)\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} \\
&\quad +2\sum_{t=1}^{f}\sum_{k=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\left(1 - \beta_i - \mathbf{A}_{ti}\right)\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}
\end{aligned}$$

Then, we pre-compute and cache $\mathcal{A}_{kk'}$ and $\mathcal{B}_{tk}$ to quickly evaluate the objective function (Eq. (7a)). The gradient of E is computed by explicitly computing each term $\Gamma_k^{\mathbb{M}^t}$ using chain rule, product and quotient rule. Caching $\mathcal{A}_{kk'}$ and $\mathcal{B}_{tk}$ significantly reduces the complexity of value/gradient evaluation by an order of n, i.e. the mesh resolution. With our tested models (§5), we observed a speed up about two orders of magnitude with caching.

$$E = \sum_{t=1}^{f}\sum_{k=1}^{p}\sum_{k'=1}^{p}\mathcal{A}_{kk'}\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + 2\sum_{t=1}^{f}\sum_{k=1}^{p}\mathcal{B}_{tk}\Gamma_k^{\mathbb{M}^t} + \text{constant} \tag{7a}$$

$$\text{where: } \mathcal{A}_{kk'} = \sum_{i=1}^{n}\alpha_{ik}\alpha_{ik'} \tag{7b}$$

$$\mathcal{B}_{tk} = \sum_{i=1}^{n}\alpha_{ik}\left(1 - \beta_i - \mathbf{A}_{ti}\right) \tag{7c}$$

The unit-length constraints on normal vectors $\mathbf{u}_k, \forall k = 1..p$ are imposed by normalizing them before every objective function value evaluation in the BFGS optimization. The affinity constraints on blending weights w_k are imposed by projecting the corresponding gradient ∇_{w_k} to the hyper plane $\sum_{j=1}^{m}\nabla_{w_{kj}} = 0$. The sparseness constraints on w_k are imposed by not updating zero values, i.e. keeping the sparse matrix structure of w during the optimization.

For robustness, we start BFGS optimization with only updating normals $\mathbf{u}$ while keeping other parameters in the non-linear layer fixed. Then, until their relative values change less than $\epsilon = 1e{-}2$, we update normals $\mathbf{u}$, sphere positions $\mathbf{o}$ and radiuses r. Finally, until their relative values change less than ϵ, we update all parameters.

4.2 Linear Layer Update

We solve linear regression weights and biases for each vertex by *Constrained Linear Least-squares* (LS) (line 4 to line 7). Our solver is similar to the non-negative vertex skinning weights solver [James and Twigg 2005] where weights and bias are solved by a constrained LS $(\alpha_i, \beta_i) = \arg\min_{\mathbf{x}} \|\mathbf{A}_i\mathbf{x} - \mathbf{b}_i\|_2^2$. Similar to the caching for BFGS in our previous section, we re-arrange the terms in the objective function so that the cross product $\mathbf{A}_i^\top\mathbf{A}_i$ and $\mathbf{A}_i^\top\mathbf{b}_i$ are elements in the cache matrix C, vector d, and vectors $\{e_i | i = 1..n\}$:

$$E = \sum_{i=1}^{n}\left((\alpha_i, \beta_i)^\top \begin{bmatrix} \mathcal{C} & d \\ d^\top & f \end{bmatrix} (\alpha_i, \beta_i) - 2(\alpha_i, \beta_i)^\top e_i + \text{constant}\right) \tag{8a}$$

$$\text{where: } \mathcal{C}_{kk'} = \sum_{t=1}^{f}\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + \lambda \tag{8b}$$

$$d_k = \sum_{t=1}^{f}\Gamma_k^{\mathbb{M}^t} \tag{8c}$$

$$e_{ik} = \sum_{t=1}^{f}\left(1 - \beta_i - \mathbf{A}_{ti}\right)\Gamma_k^{\mathbb{M}^t} + \lambda \tag{8d}$$

We use the soft-affinity constraint with the scaling parameter $\lambda = f$ (Eq. (8b) and Eq. (8d)).

5 RESULTS AND DISCUSSIONS

We test our method on three rigged models and test animations acquired from Mixamo[2]. Our model training was implemented in C++ using CppNumericalSolvers [3] as the non-linear optimization framework, in which the objective function value and its gradient are computed by CUDA [4] and OpenMP [5]. The statistics of the models and training process are presented in Table 1. We use the same set of parameters for both generating training data and fitting the mode (as reported in §4, Appendix A, and Table 1).

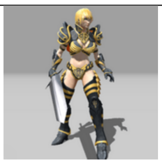
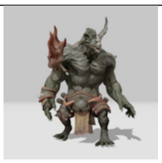
In Fig. 8, we visualize the proxy spheres and key points computed by our optimization algorithm. In this example, our optimization robustly converge to a plausible solution as the sizes of the proxy spheres are roughly at the same size of corresponding body parts. Notice that how our model can keep the mid-level surface details as the tested pose is quite different from the training poses, i.e. the neighbors of T-pose.

5.1 Comparisons

We evaluate our model by comparing it with previous approaches (Fig. 9). The static baked AO is generated by performing ray tracing at the rest pose with the same setting that we used to generate our training data (Appendix A). The screen-space AO (SSAO) is rendered by Maya 2018's viewport 2.0 with DirectX 11. The rendering resolution is 1920 × 1080 pixels and the radius of the sampling area is 64 pixels. Despite of the large radius, SSAO appears flat due to its local computation. We also compare our model with Kontkanen and Aila [2006] and Kirk and Arikan [2007]. For Kirk and Arikan's

[2]https://www.mixamo.com
[3]https://github.com/PatWie/CppNumericalSolvers
[4]https://developer.nvidia.com/cuda-zone
[5]https://www.openmp.org

Table 1: Test models in this paper: n denotes the number of vertices, m denotes the number of bones, f denotes the number of training poses, s denotes the number of proxy spheres, and p denotes the number of key points. For all models, we set the number of non-zero weights per vertex $n_{nz} = 8$ and the sparsifying norm $\gamma = 1.5$. The errors MSE_{init} and MSE_{min} are the mean-square errors, i.e. $\frac{1}{nf}E$, at the initialization and the end of the training, respectively. $time$ is the training time recorded on a computer with an 24-core 3.00 GHz CPU and GeForce GTX Titan V GPU. $iters$ is the number of global iterations, where each global iteration contains 20 (local) nonlinear BFGS iterations and one linear solver step. $size_{data}$ is the size of the training data and $size_{model}$ is the size of the trained model (all parameters). Scalar values in the training $data$ and AO $model$ were stored as 32 bit floats and indices are stored as 32 bit integers.

Vanguard	 Maria	 Warrok
$n = 5,890$	$n = 8,876$	$n = 6,557$
$m = 65$	$m = 66$	$m = 65$
$f = 21,438$	$f = 23,131$	$f = 21,438$
$size_{data} \approx 652$ MB	$size_{data} \approx 970$ MB	$size_{data} \approx 706$ MB
$s = 81$	$s = 130$	$s = 77$
$p = 500$	$p = 500$	$p = 500$
$MSE_{init} = 1.85\text{e}{-3}$	$MSE_{init} = 1.30\text{e}{-3}$	$MSE_{init} = 1.69\text{e}{-3}$
$MSE_{min} = 0.67\text{e}{-3}$	$MSE_{min} = 0.59\text{e}{-3}$	$MSE_{min} = 0.56\text{e}{-3}$
$time \approx 5.5$ h	$time \approx 14$ h	$time \approx 12$ h
$iters = 160$	$iters = 311$	$iters = 500$
$size_{model} = 563$ KB	$size_{model} = 820$ KB	$size_{model} = 598$ KB

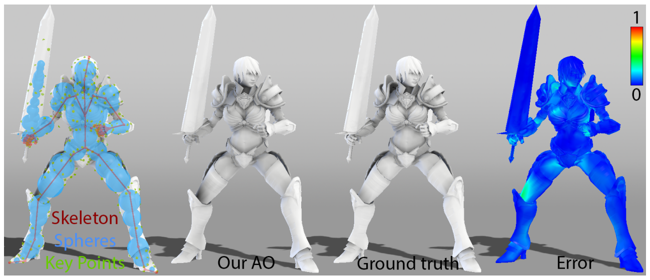


Figure 8: Leftmost: visualization of our calculated spheres (blue) and key points (green). Others: comparison of our AO model with the ground truth AO (NVIDIA's OptiX Prime ray tracer).

model, we set the number of pose clusters, the reduced dimensions, the number of vertex clusters, and the moving least squares blend parameter (α) to be 15, 1000, 15, and 10.0 respectively as these are a good set of parameters reported by the authors.

5.1.1 Model Fitting. We compare the effectiveness of our model fitting against previous works by testing these models on the training data (Fig. 9). In practice, this setup could be use to compress the animated AO map. The results show very low fitting errors with Kontkanen and Aila's model and our model. We believe the reduced models (clustering and principal component analysis) of Kirk and Arikan are not robust enough to capture the non-linear space for their pose interpolation. In contrast, our geometry-based clustering for model initialization (Appendix B) combined with the joint-optimization on two layers converges to a good local optimized solution.

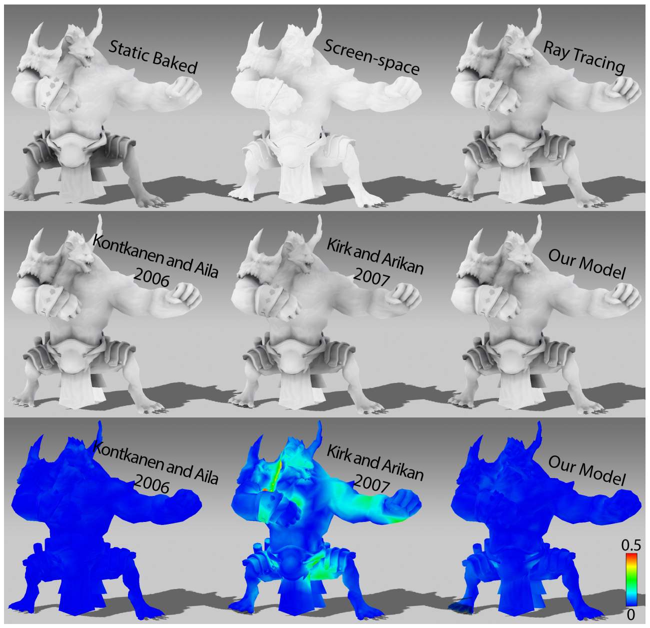


Figure 9: Comparison between different AO rendering methods. Top row: non data-driven methods. Middle row: data-driven methods used as compression, i.e. training and testing on the same data ("Fight" animation sequence). Bottom row: differences between the middle row and the ground truth AO (ray tracing).

5.1.2 Generalization. We performed two tests to compare the generalization of different models. In the perturbation test (Fig.10), we slightly move the character pose off the last frame of the training sequence. As shown from the results, the non-reduced linear model [Kontkanen and Aila 2006] suffers more from this perturbation than Kirk and Arikan model and our method.

In Fig. 11, we trained all models with our uniform joint rotation sampled data (Appendix A), then tested the model predictions on a novice animation. The training data does not contain complex poses (all training poses are the neighbors that differ from the T-pose by only one joint rotation) but it provides good coverage for the joint rotation ranges. With this setup, Kirk and Arikan model underfits the training data and produces a large testing error.

5.1.3 Size of Model. For real-time applications such as games, the size of the model is important not only for storage but also

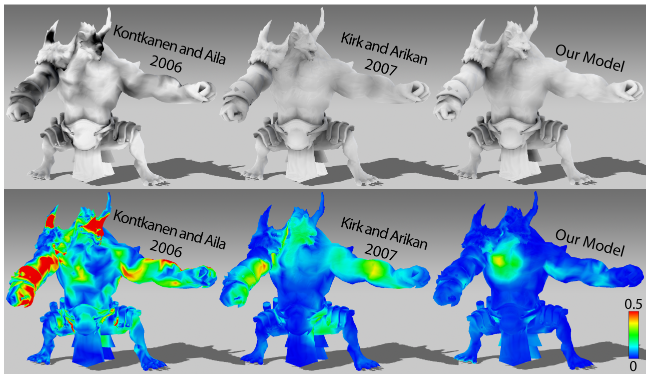


Figure 10: Perturbation test for model generalization: we trained different methods with "Fight" animation sequence and tested them on a slightly different pose with the poses in the training sequence. Top row: the AO outputs of different methods. Bottom row: differences from the ground truth AO (ray tracing).

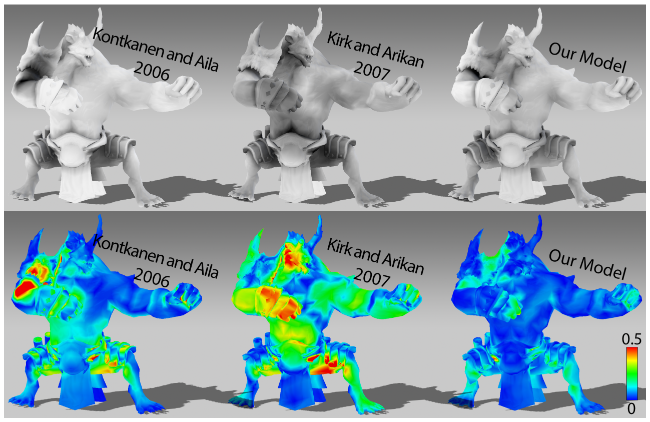


Figure 11: Model generalization test: we trained different methods with uniform joint rotation sampling (Appendix A) and tested them on the "Fight" animation. Top row: the AO outputs of different methods. Bottom row: differences with the ground truth AO (ray tracing).

for loading time. Despite using a large amount of training data with a naïve sampling ($size_{data}$ in Table 1), our model results in a very small run-time model ($size_{model}$ in Table 1). Even without compression, $size_{model}$ is approximately as small as a single low-resolution texture (less than a megabyte). In theory, the size of our AO model and the size of Kirk and Arikan's model [2007] are independent from the number of training, while the size of Kontkanen and Aila's model [2006] grows with the size of the training data. In Table 2, we compare the size of trained models from different methods. While offering higher AO quality, the sizes of our trained AO models are significantly smaller than trained models from previous methods.

Our method outperformed previous methods [Kirk and Arikan 2007; Kontkanen and Aila 2006] in all comparisons. Although a well designed training dataset could help improve problems of the previous methods, with our method, it is easier to automate the process by not having to tune the training animation or the training parameters.

Table 2: Sizes of trained models with different methods. Scalar values were stored as 32 bit floats and indices are stored as 32 bit integers. Training was done on the "Warrok" model with different test animation sequences. "Uniform" is our uniform sampling data as described in Appendix A. f is the number of samples.

Test sequence (f)	Uniform (21438)	Idle (180)	Fight (88)
[Kontkanen and Aila 2006]	5,179 KB	1,875 KB	1,875 KB
[Kirk and Arikan 2007]	1,625 KB	1,625 KB	1,625 KB
Our AO model	598 KB	583 KB	579 KB

5.2 Interaction

Intuitively, the proxy spheres and the key points are shadow blockers and light receivers. We can simply inter-connect proxy spheres/key points between different models to make interactions. This allows integration without retraining the models. In Fig. 12, we simulate the ground AO by simply adding a proxy sphere underneath the character, e.g. connecting x and z coordinates of the sphere to the root joint, assuming y is the up/down direction. In Fig. 1, we add the spheres set of one character to the other and vise versa. As the result, when two characters are close, they block the ambient light from each other. Fig. 13 shows the difference between having no interaction and having interaction between two characters. Fig. 13 also shows the benefit of our proposed γ-norm sparsification on reducing the double shadowing effect. Note that we do not need to retrain the models to setup the character-character interactions. This is a side benefit of the interpretability of our model.

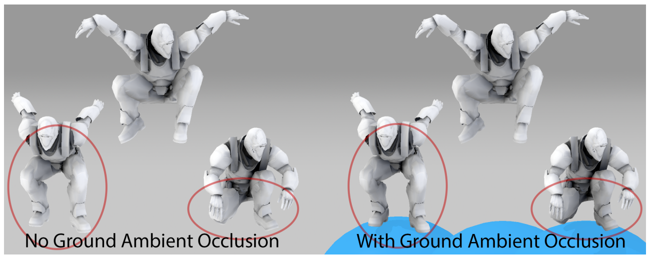


Figure 12: We can add a ground AO effect to a trained model by adding a large proxy sphere underneath the character. Notice the ground AO effect is less noticeable as the character jumps higher.

5.3 Run-time Performance

Fig. 14 shows a scene with 100 characters rendered in real time with our AO model. The AO of each character is computed from itself and four neighbors. The system is implemented by C++ with DirectX 12. For each frame in this 100 character scene, our CPU

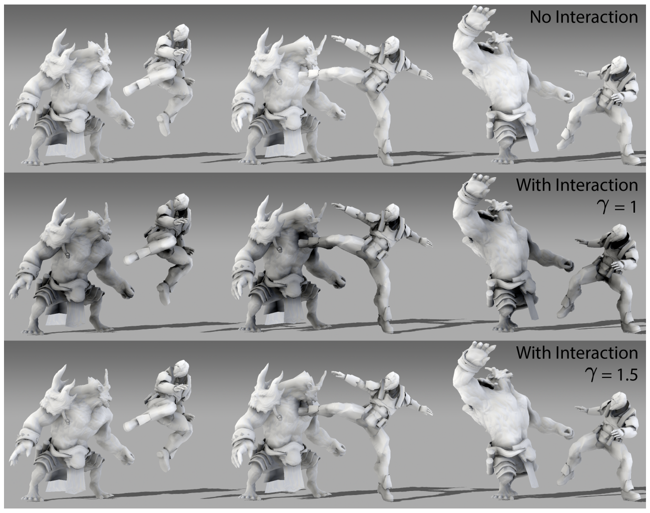


Figure 13: Top row: two individually trained models put together do not have a global illumination effect. Middle row: cross-illumination between models can be created by adding the set of proxy spheres of one character to the other character and vise versa, double shadowing makes AO appear very dark. Bottom row: our γ-norm sparsification ($\gamma = 1.5$) reduces double shadowing while still keeping the cross-illumination effect.

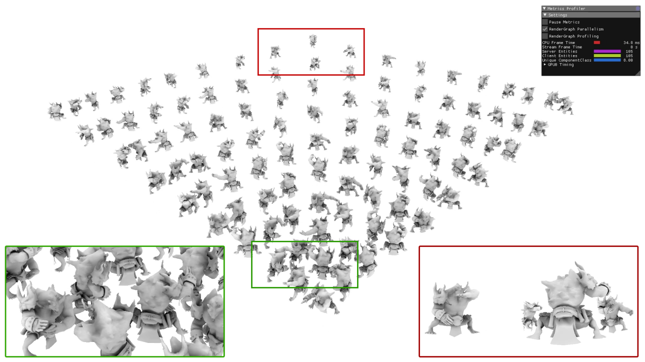


Figure 14: Screen capture of 100 animated characters rendered with our AO at nearly 30fps. Notice the AO interaction is stronger when characters are close (magnified view on the left), and AO interaction is less for distant characters (magnified view on the right).

(32 cores) takes about 20ms to compute the non-linear layer and all other CPU tasks. Our GPU (NVIDIA Titan V) takes about 10ms to compute the linear layer and all other rendering tasks. This performance can be greatly improved by moving the non-linear layer computation to the GPU.

6 CONCLUSION

This paper introduces a dynamic AO method for characters that excels in both quality (Figs. 1) and ease of use. It greatly outperforms state of the art methods [Kirk and Arikan 2007; Kontkanen and Aila 2006]. It also easily approximates shadowing with the ground (Fig. 12) and interaction with other characters (Fig. 13).

Our method has few tunable parameters and is free of stochastic computations (such as k-means initialization) that may need to be re-run multiple times.

One current limitation is the long training time. This can be improved by exploring batch optimization methods. An alternative solution is deploying parallel implementation at larger scale, e.g. on clusters, which could be more suitable for industrial production.

The current sampling strategy for poses is basic, and automatically obtaining a more effective set of sample poses may be possible. Designing a good pose sampling strategy is an interesting problem for future research.

REFERENCES

Okan Arikan, David A. Forsyth, and James F. O'Brien. 2005. Fast and Detailed Approximate Global Illumination by Irradiance Decomposition. *ACM Trans. Graph.* 24, 3 (July 2005), 1108–1114.

David Arthur and Sergei Vassilvitskii. 2007. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. 1027–1035.

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* 36, 4 (July 2017), 97:1–97:14.

D.P. Bertsekas. 1999. *Nonlinear Programming.* Athena Scientific.

Michael Bunnell. 2005. *Dynamic Ambient Occlusion And Indirect Lighting.* Vol. 2. 223–233. http://http.download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch14.pdf

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (July 2017), 98:1–98:12 pages.

Per H. Christensen. 2002. Note #35: Ambient Occlusion, Image-Based Illumination, and Global Illumination. PhotoRealistic RenderMan Application Notes. (2002).

Per H. Christensen and Wojciech Jarosz. 2016. The Path to Path-Traced Movies. *Found. Trends. Comput. Graph. Vis.* 10, 2 (Oct. 2016), 103–175.

Philip Dutre, Kavita Bala, Philippe Bekaert, and Peter Shirley. 2006. *Advanced Global Illumination.* AK Peters Ltd.

Roger Fletcher. 1987. *Practical Methods of Optimization; (2nd Ed.).* Wiley-Interscience.

Daniel Holden, Jun Saito, and Taku Komura. 2016. Neural Network Ambient Occlusion. In *SIGGRAPH ASIA 2016 Technical Briefs (SA '16).* Article 9, 9:1–9:4 pages.

Doug L. James and Christopher D. Twigg. 2005. Skinning Mesh Animations. *ACM Trans. Graph.* 24, 3 (July 2005), 399–407.

Ladislav Kavan, Adam W. Bargteil, and Peter-Pike Sloan. 2011. Least Squares Vertex Baking. In *Proceedings of the 22nd Eurographics Conference on Rendering (EGSR '11).* 1319–1326.

Adam G. Kirk and Okan Arikan. 2007. Real-time Ambient Occlusion for Dynamic Character Skins. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07).* 47–52.

Janne Kontkanen and Timo Aila. 2006. Ambient Occlusion for Animated Characters. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques (EGSR '06).* 343–348.

Dylan Lacewell. 2016. Baking With OptiX. (2016). https://developer.nvidia.com/optix-prime-baking-sample

H. Landis. 2002a. Global illumination in production. In *ACM SIGGRAPH Course Notes.*

H. Landis. 2002b. RenderMan in Production. ACM SIGGRAPH Course 16. (2002).

D. Larsson and H. Halen. 2009. The unique lighting of Mirror's Edge. In *Proceedings of the Game Developers Conference (GDC).*

Binh Huy Le and Jessica K. Hodgins. 2016. Real-time Skeletal Skinning with Optimized Centers of Rotation. *ACM Trans. Graph.* 35, 4, Article 37 (July 2016), 37:1–37:10 pages.

Nelson L. Max. 1988. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer* 4, 2 (01 Mar 1988), 109–117.

Oleg Mazonka. 2012. Solid Angle of Conical Surfaces, Polyhedral Cones, and Intersecting Spherical Caps. (2012).

Martin Mittring. 2007. Finding Next Gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses (SIGGRAPH '07).* 97–121.

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. 2017. Deep Shading: Convolutional Neural Networks for Screen-Space Shading. 36, 4 (2017).

Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. 2006. Real-time Soft Shadows in Dynamic Scenes Using Spherical Harmonic Exponentiation. *ACM Trans. Graph.* 25, 3 (July 2006), 977–986.

RTX 2018. NVIDIA Developer Blog: Introduction to NVIDIA RTX and DirectX Ray Tracing. (2018). https://devblogs.nvidia.com/introduction-nvidia-rtx-directx-ray-tracing

S. Schaefer and C. Yuksel. 2007. Example-based Skeleton Extraction. In *Proceedings of the 5th Eurographics Symposium on Geometry Processing (SGP '07)*. 153–162.

Ari Silvennoinen and Jaakko Lehtinen. 2017. Real-time Global Illumination by Precomputed Local Reconstruction from Sparse Radiance Probes. *ACM Trans. Graph.* 36, 6, Article 230 (Nov. 2017), 230:1–230:13 pages.

Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. 2007. Image-Based Proxy Accumulation for Real-Time Soft Global Illumination. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (PG '07)*. 97–105.

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments. *ACM Trans. Graph.* 21, 3 (July 2002), 527–536.

Lance Williams. 1978. Casting Curved Shadows on Curved Surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '78)*. 270–274.

S. Zhukov, A. Iones, and G. Kronin. 1998. An ambient light illumination model. In *Rendering Techniques '98*. 45–55.

A TRAINING DATA

We generate sample poses by wiggling each joint one by one (except for the root joint), i.e. rotating each joint to different angles depending on its range of motion and user-defined sampling steps. In particular:

- for joints on the body (from the hip to shoulders), we sample (Rx, Ry, Rz) in $\{-45^\circ, -15^\circ, 15^\circ, 45^\circ\}^3$ ($4 \times 4 \times 4 = 64$ poses per joint),
- for other joints (legs, arms, hands, fingers, head, etc.), we sample each pair (Rx, Ry), (Ry, Rz), and (Rz, Rx) in $\{-150^\circ, -120^\circ, -90^\circ, .., 90^\circ, 120^\circ, 150^\circ\}^2$ ($3 \times 11 \times 11 = 363$ poses per joint).

For each pose in the training data, we use linear blend skinning (LBS) to deform the geometry model and use ray tracing to compute the ground truth AO value at each vertex. Specifically, we use NVIDIA's OptiX Prime [Lacewell 2016] with 128 rays per sample point, 3 sample points per face, and without least squares filtering [Kavan et al. 2011].

B MODEL INITIALIZATION

We initialize proxy spheres by sampling on bones. For each bone j, we perform Eigen analysis on the geometry influenced by j, i.e. performing Eigen decomposition on the covariance matrix $\sum_{i=1}^{n} \mathbf{v}_{ij}^{(w)} \mathbf{v}_i^{(a)} \mathbf{v}_i \mathbf{v}_i^{\mathsf{T}}$, where $\mathbf{v}_{ij}^{(w)} \in \mathbb{R}$, $\mathbf{v}_i^{(a)} \in \mathbb{R}$, and $\mathbf{v}_i \in \mathbb{R}^3$ are skinning weight, area of the Voronoi cell, and position of vertex i, respectively. Let λ_1 and λ_3 be the smallest Eigen value and the largest Eigen value, respectively. We uniformly distribute $\sqrt{\lambda_3/\lambda_1}$ spheres with radius $\sqrt{\lambda_1}$ along the direction of the Eigen vector with respect to λ_3. Note that a more complicated sampling scheme can be used [Ren et al. 2006].

The initialized key points are generated by clustering n vertices to p clusters, where the users set the number of p. We generate the features for the clustering by concatenating vertex's positions, normals, and skinning weights together, in which each component can be scaled by the users. We employ the edge collapsing strategy proposed by Schaefer and Yuksel [2007] to keep vertices in the same cluster connected. Each cluster is then assigned to one key point

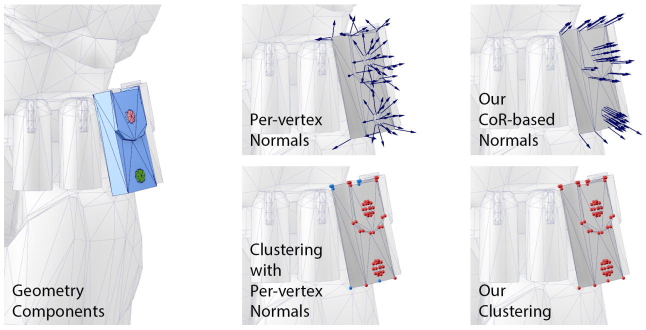


Figure 15: Left: the input model with many separated components, different components of the pouch are illustrated with different colors. Middle: using mesh normals lead to separated vertex clusters for the front (vertices in red) and the back of the pouch (vertices in blue). Right: our robust normal estimation based on optimized center of rotation (CoR) generates all normals pointing outward, thus produces one cluster for the whole pouch.

where positions, normals, and skinning weights are extracted from the feature of the cluster.

We handle models with separated geometry components by utilizing the skinning weights in addition to the geometry of the rest pose as follows:

In addition to the original mesh edges, we add pairs of vertices with closed positions and closed skinning weights as the extra edges for the clustering. Generally, vertices with closed positions but different skinning weights belong to different parts of the model. Using skinning weights, we can easily separate vertices that are closed in the rest pose, e.g. vertices at finger regions.

We compute the normal at each vertex as the normalized vector from the optimized center of rotation (CoR) [Le and Hodgins 2016] to the vertex. Instead of the proposed pairwise-based distance function, we use the Gaussian radial basis function $s(\mathbf{v}, \mathbf{v}') = \exp\left(-\frac{||\mathbf{v}^{(w)} - \mathbf{v}'^{(w)}||_2^2}{\sigma^2}\right)$ because this function is also valid for vertices with only one skinning weight, where the pairwise-based function is not. The effect of using CoR-based normals is shown in Fig. 15. Although CoR-based normals are generally less precise than the mesh normals, they are more robust for initialization and optimization. They never flip 180° from the desired direction, e.g. pointing inward to the medial axis of the model. If this happened, the optimization could suffer as the continuous descent search can hardly flip them back.

With the initialized proxy spheres and key points, we run the linear solver to initialize the linear regression weights and biases. This is also the solver in our linear layer update (§4.2).