SEED // Electronic Arts

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

Binh Huy Le Henrik Halen(*) Carlos Gonzalez-Ochoa JP Lewis

(*) presenter

https://www.ea.com/seed/news/i3d2019-dynamic-ao

I am Henrik Halen, I am a rendering engineer working at a research division within Electronic Arts called SEED. While our group is geographically diverse, the authors of this paper are based in California

Ambient Occlusion (AO)

2

This paper presents a method computing an approximation of ambient occlusion in object space, using a custom machine learning regression approach.

Ambient Occlusion (AO)

- Exposure to ambient lighting
- AO value at a point on 3D scene:
  is its exposure to the ambient lighting,
  or (1 – shadow) cast by the scene.

3

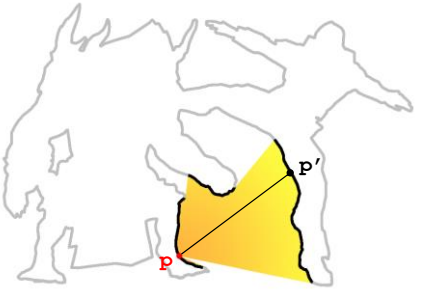
The ambient occlusion for any given point in a 3D scene can be defined as it's exposure to ambient lighting, or in other words the opposite of shadow cast from all other points in the scene.

Note: fig on the right visualizes points (black) cast shadow on red point

Ambient Occlusion (AO)

- Exposure to ambient lighting
- AO value at a point on 3D scene:
  is its exposure to the ambient lighting,
  or (1 – shadow) cast by the scene.
- Expensive to compute:
  $O(n^2 \times \text{visibility\_test})$

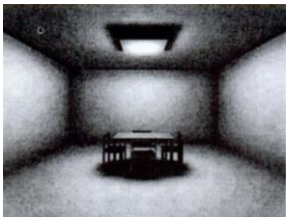
```
for every point p do
    for every point p' do
        test(p see p'?)
    end for
end for
```



4

If done correctly this is an expensive problem to solve. Essentially for every given point in the scene, you need to know how occluded it is by all the other points.

Previous Work: Baked AO



[Zhukov et al. 1998]

[Kavan et al. 2011], [Landis 2002]

[Sloan et al. 2013]

- Precomputed
- × Only for static scenes

5

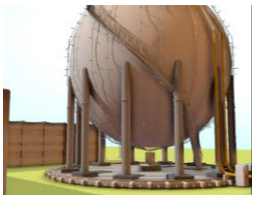
Precomputed ambient occlusion has existed for quite some time. This can provide a high quality occlusion term, stored in for example textures, or per vertex of polygonal objects. However, since it's baked, it can't change in runtime, and as such does not work for dynamic objects or animated characters, and does not change depending on changes in the scene.

S. Zhukov, A. Iones, and G. Kronin. 1998. An ambient light illumination model. In Rendering Techniques '98. 45–55.
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.
Peter-Pike Sloan, Jason Tranchida, Hao Chen, and Ladislav Kavan. 2013. Ambient obscurance baking on the GPU. In *SIGGRAPH Asia 2013 Technical Briefs* (SA '13).

Previous Work: Screen-space AO

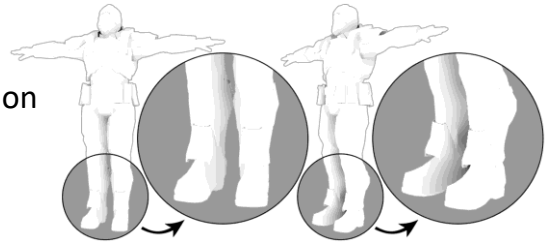
[Mittring 2007]

[Bavoil et al. 2008]



[Ritschel et al. 2009]

- ✓Fast, good for dynamic scenes
- × No global (long range) interaction
- × Artifacts with geometry hidden from camera



6

In contrast, a commonly used approach for games is real time screen space ambient occlusion. There are many different approaches to this, and work continues to improve quality. However, screen space ambient occlusion has a handicap in that has a lack of information to work with, it can only infer occlusion from what is visible on screen. This means that even for the state of the art methods, assumptions have to be made, and often those assumptions are incorrect.

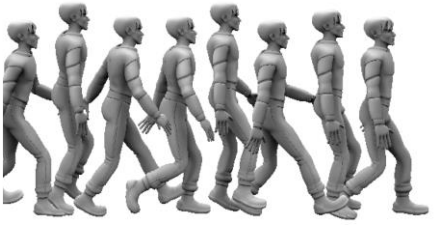
Martin Mittring. 2007. Finding Next Gen: CryEngine 2. In ACM SIGGRAPH 2007 Courses (SIGGRAPH ’07). 97–121.
Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. 2009. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (I3D '09).
Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. 2008. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks* (SIGGRAPH '08). ACM, New York, NY, USA, Article 22

NOTE: bottom right corner was rendered with Maya’s SSAO.
In contrast, a commonly used approach for games is real time screen space ambient occlusion. There are many different approaches to this, and continued work has

continued to improve the quality of the results. However, screen space AO inherently has a lack of information to work with. It can only know what's on screen. This not only presents problems where geometry that is outside of the view frustum should cast occlusion, but the algorithms also has to make guesses when geometry is within the frustum, but occluded from the camera. Depending on the technique the screen space search radius can present a situation where a compromise needs to be reached between performance and quality.

Previous Work: Object-space AO

[Kontkanen and Aila 2006]



[Kirk and Arikan 2007]

- Data driven approach: train regression model from ray-traced AO

✓For animated character (but not general dynamic scenes)

×Not good in generalization, mostly animation compression

(more details will be presented later in our comparisons)

7

Alternative solutions suitable for dynamic geometry, and characters in particular, are object space solutions. The previous work in this area generally focuses on using regression for an accurate object space solution for Ambient Occlusion. These models use regression to train parameters using some form of high quality Ambient Occlusion as ground truth. This allows for a fast and accurate runtime model. The previous work in this area works well for matching ground truth for poses that are in the training set, and as such can be seen as a form of data compression, but they do not generalize well. Our model is an approach in this category.

Note: these works only focused on character but not general dynamic scenes.
However, characters make up most of dynamic scenes so the solutions are still useful.
Our work also focus on the same thing.
Generalization is the key limitation of these works.

Our Proposed Method

- Data driven approach: fully automatic model training
- Bi-level object space: high quality, global effects
- Skinning-like model: fast, simple, GPU friendly, good generalization
- Interpretability: adding character-character interaction, ground-character interaction after model trained

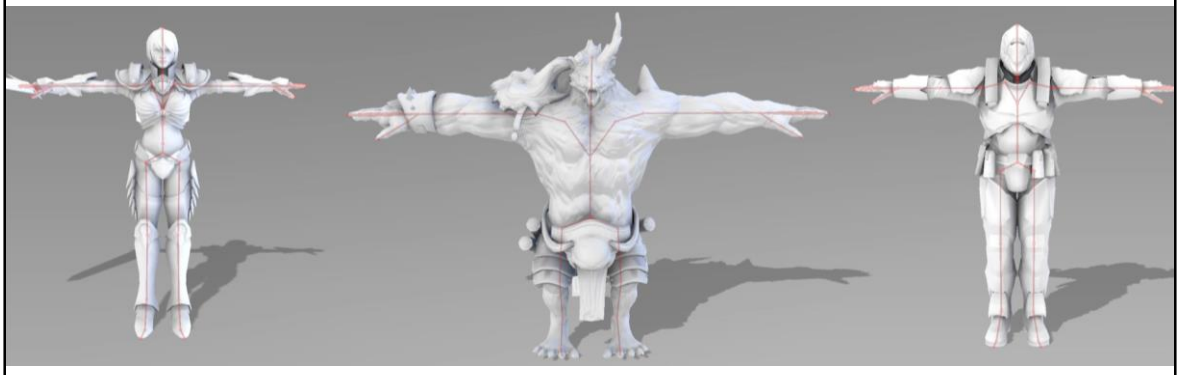
Skeleton animation
(bone transformations)

Per-vertex AO

8

Our approach is a data driven approach, and includes a fully automated training setup.
It is a Bi-level object space approach which produces high quality global effects.
It's a skinning-like model, which lends itself to fast, simple and GPU friendly implementation.
It handles poses well that are not in the training set, and it's interpretability allow for modes of operation that have not been seen during training, such as character to character interaction and ground-character interaction.

Training Data



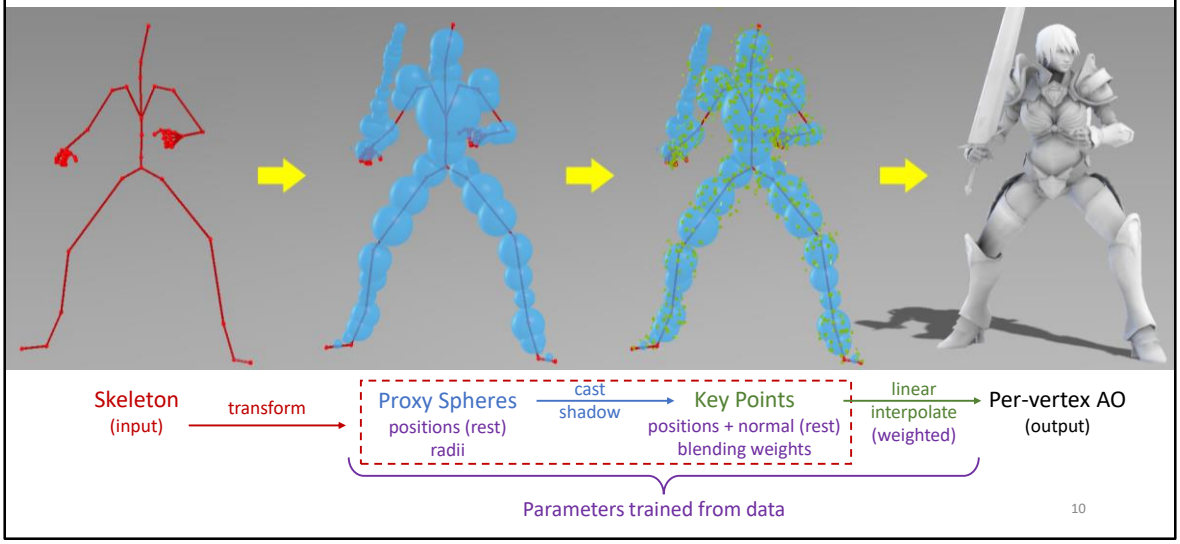
Generate training data for each character model:

- Compute ground truth, per-vertex AO with ray tracing (Nvidia OptiX)
- Sample joint rotations (automatically): {Rx, Ry, Rz}^3
- Allow user-defined (artist-made) poses

9

For training, we automatically generate training data by individually rotating each joint to a set number of positions.
The ground truth can be generated with whichever method you prefer, we use Nvidia optiX to provide high quality ray traced ambient occlusion for training.
The way we generate poses is purely for automation, but any set of poses can be used for the training, and as such artist generated poses can also be used.

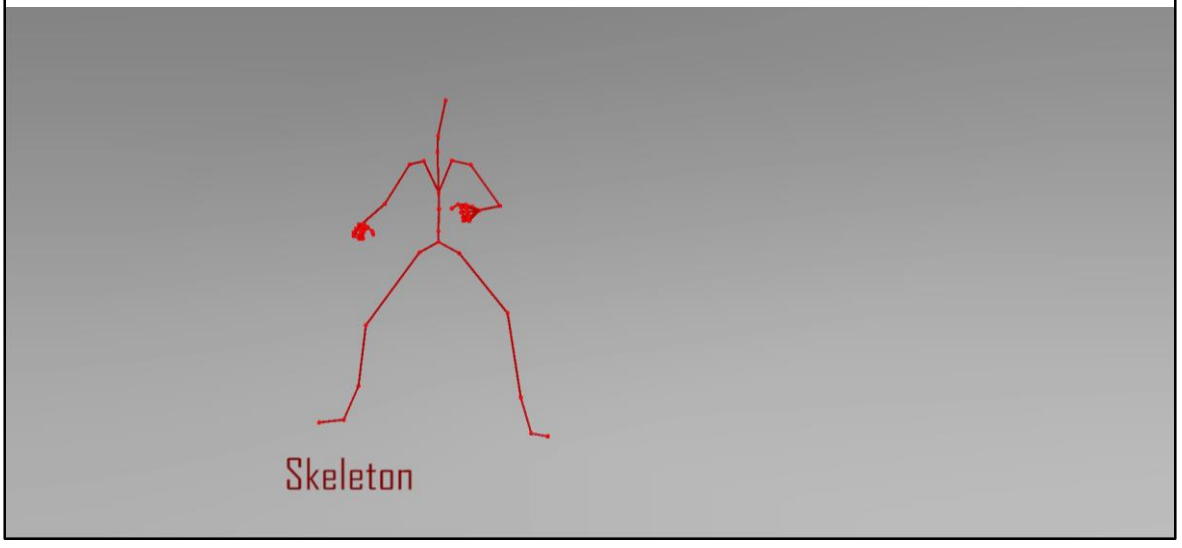
Model



This is a high level description of the ambient occlusion model we are using.
On a very high level you can see our model as one that takes the state of the skeleton as input, and outputs per vertex ambient occlusion.
Essentially we have a skeleton, on that skeleton a number of proxy spheres are placed.
We also place what we call key points, which are represented as points with a position and a normal.
The proxy spheres cast occlusion onto the key points.
Each vertex receives occlusion from a number of key points.
So what we train here is the positions and radiuses of the proxy spheres. They positions and normal of the key points, and the weights per vertex for how each key point effects that vertex.

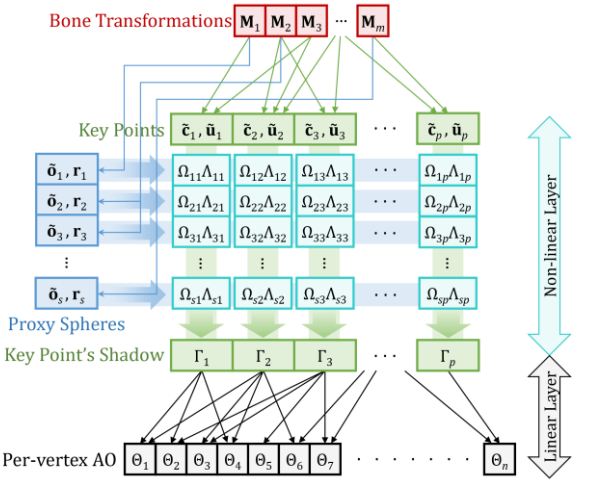
- May hide this slide if presentation time is tight.
- Do not need to talk about details (see in our paper).
- Key message is: training is fully automatic.

Model: with Animation



The proxy spheres are rigidly skinned, while the key points don’t need to be rigid.

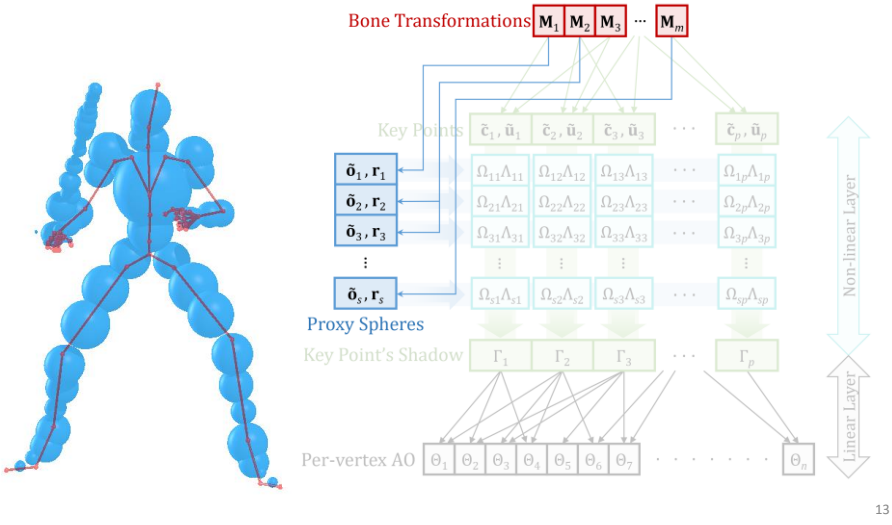
Model: Computing Diagram



Our model contains two layers:

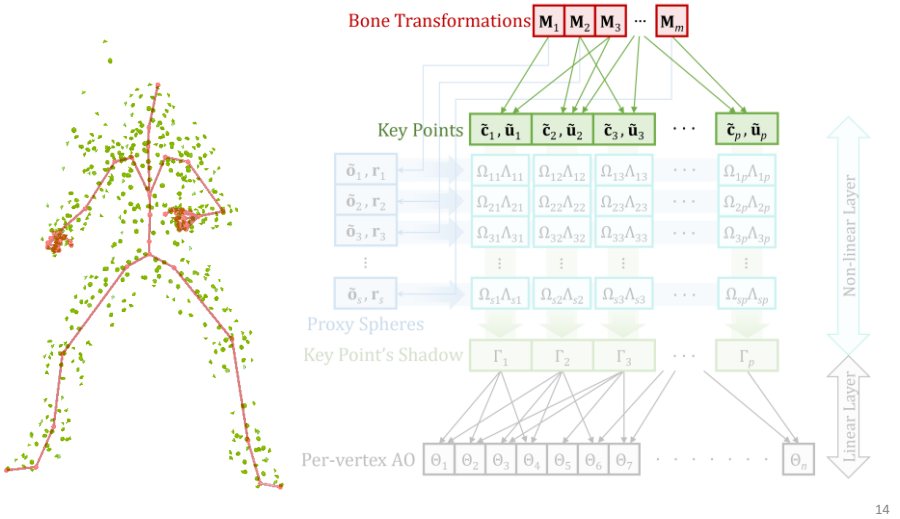
- A dense, non-linear layer on top.
- A sparse, linear layer at the bottom.

Model: Computing Diagram



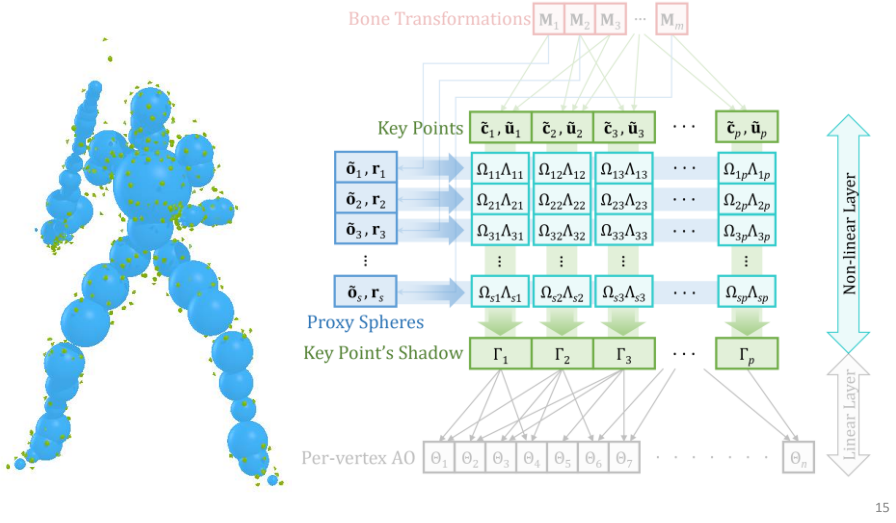
The input of the model is bone transformations (M).
They drive transformations of proxy spheres, resulting the changes in their centers (o tilde).
Notice that bone transformations are rigid so the radii of spheres (r) do not change

Model: Computing Diagram



And bone transformations also drive transformations of key points, resulting changes of their positions (c tilde)and normals (u tilde)

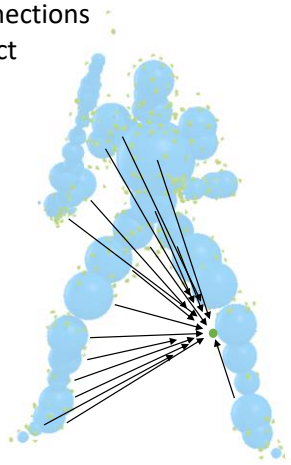
Model: Computing Diagram

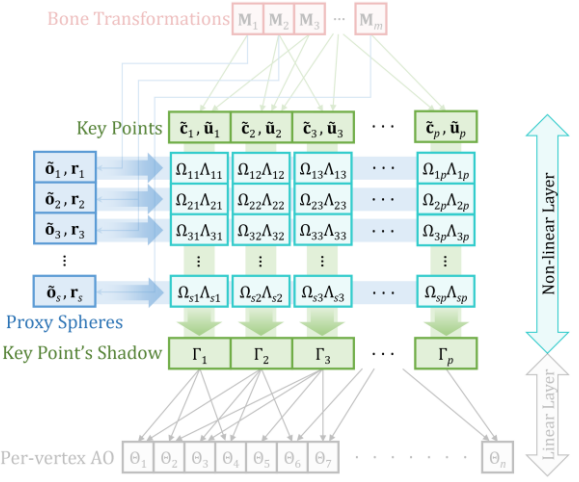


So on the non-linear layer at runtime, we have transformed proxy spheres and transformed key points.

Model: Computing Diagram

Dense connections
Global effect

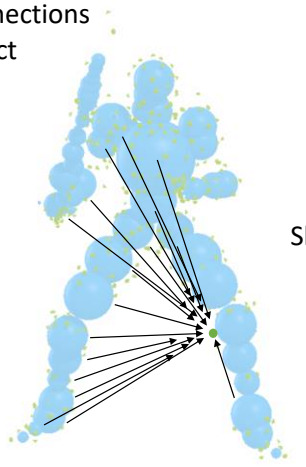


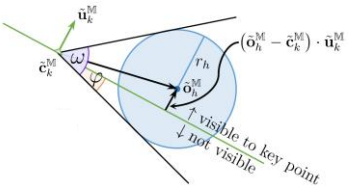


Each key point receives contributions from all spheres.
These Nonlinear dense connections carry global effects but it is costly.
Therefore, we only use small numbers of spheres and key points, typically about 50 spheres and 500 key points.

Model: Computing Diagram

Dense connections
Global effect





Shadow from spheres on key point k: $\Gamma_k^{\mathbb{M}} = \sum_{h=1}^{s} \left(\Omega_{hk}^{\mathbb{M}} \Lambda_{hk}^{\mathbb{M}} \right)$

$\Omega_{hk}^{\mathbb{M}}$ Normalized solid angle

$\Lambda_{hk}^{\mathbb{M}}$ Visibility ratio

Both can be computed in closed-form
[Sloan et al. 2007], [Mazonka 2012]

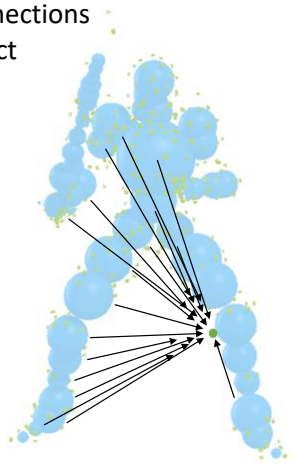
17

The shadow cast on each key point is the sum of the shadows cast by all spheres onto that key point.
The shadow from any given sphere onto any key point is the product of the solid angle of there sphere, and a front/back visibility function.
For a point with a normal, the visibility ratio is defined as the ration of the sphere that is visible inside the hemisphere of the key point as defined by it's normal.

Shadow cast on key point k is summed from all spheres h=1..s
Each shadow cast from a sphere to a key point is the product of solid angle and a front/back visibility function.
Notice these calculations have closed-form solutions. However, they are not differentiable in the whole range. Thus, difficult to train the model.
Therefore, we made some approximation.
Our approximated functions are shown in red curves.
See more details in our paper.

Model: Computing Diagram

Dense connections
Global effect



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

Our Approximated Solid Angle

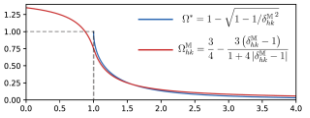
$$\Omega_{hk}^{\mathbb{M}} = \frac{3}{4} - \frac{3\left(\delta_{hk}^{\mathbb{M}} - 1\right)}{1 + 4\left|\delta_{hk}^{\mathbb{M}} - 1\right|}$$

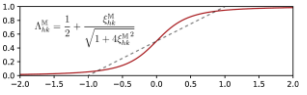
where: $\delta_{hk}^{\mathbb{M}} = \frac{\left\| \tilde{\mathbf{o}}_h^{\mathbb{M}} - \tilde{\mathbf{c}}_k^{\mathbb{M}} \right\|_2}{r_h}$

Our Approximated Visibility Function

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

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





Differentiable in the whole range

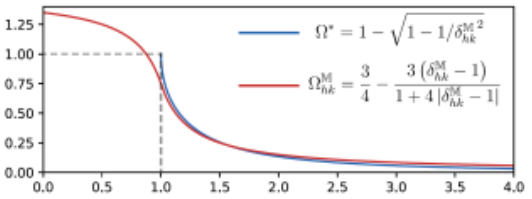
18

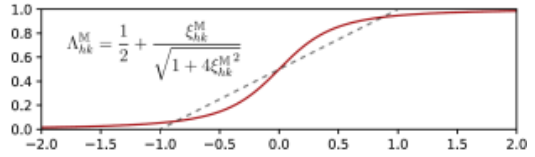
Now, both the solid angle function and the visibility functions have closed form solutions. However, these solutions are not differentiable in the whole range. We want to compute gradients using these functions during regression training, so these functions are unsuitable for our needs.
which makes them unsuitable for the regression training we use.
For that reason we created two approximate functions that closely match the closed for solutions, but are differentiable through the full range.

Shadow cast on key point k is summed from all spheres h=1..s
Each shadow cast from a sphere to a key point is the product of solid angle and a front/back visibility function.
Notice these calculations have closed-form solutions. However, they are not differentiable in the whole range. Thus, difficult to train the model.
Therefore, we made some approximation.
Our approximated functions are shown in red curves.
See more details in our paper.

Model: Computing Diagram

Dense connections
Global effect



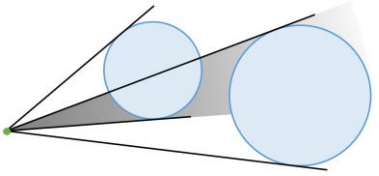


Differentiable in the whole range

19

The Red lines here represent our approximate functions

Model: Handling Double Shadowing



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

γ-norm function
$\gamma > 1$

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

Double Shadowing

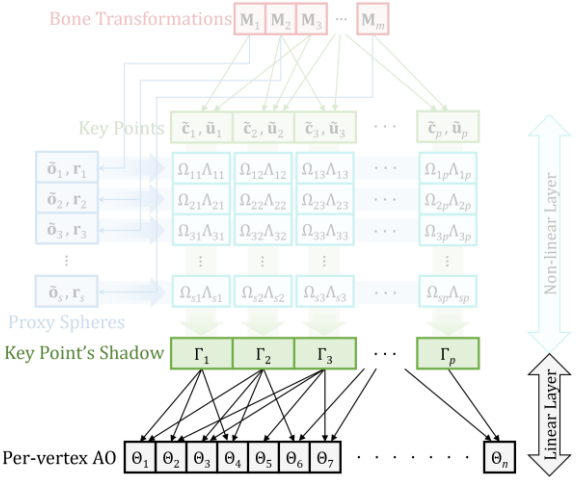
Our Reduced Double Shadowing

20

Now, as the observant listener would have noticed, the solid angle projection onto the key points can produce over-shadowing when the solid angle of two or more spheres overlap from the point of view of a key point. To alleviate this we compute the output of the non-linear layer using a gamma-norm function. This works by emphasizing the contribution from the spheres with the largest shadow contribution, or in other words we sparsify the contribution vector similar to the maximum norm. While this is not an analytically correct solution, it saves us from the computationally heavy approach of doing multiple passes.


Double Shadowing
Ours
[0.5x replay]

Model (Computing Diagram)



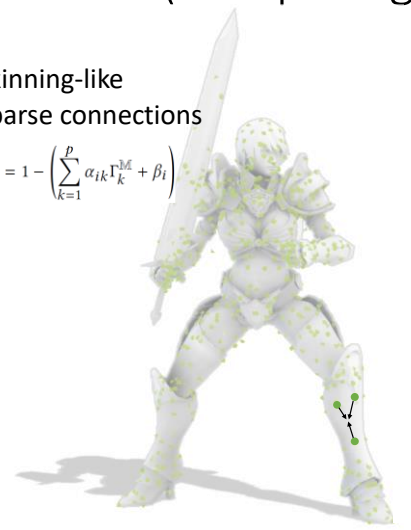
22

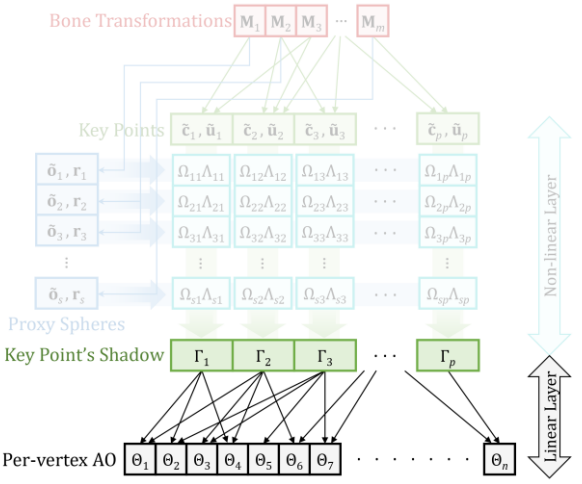
Once we have the AO calculated for each key point, in the linear layer we compute the final per-vertex AO as a linear combination of the shadow received by a number of key points.

Model (Computing Diagram)

Skinning-like
Sparse connections

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





23

So each vertex only receives contribution from some local key points.
This sparse linear connect helps to reduce computing cost both during training and in runtime.
As can be seen in the formula we include both a weight, alpha, as well as a bias Beta in order to encode local detail at the vertex.
It is similar to skinning so it could be effectively implemented in GPU's shaders.
We implement this as a sparse operation on the GPU as well.

Model Fitting

- Minimize sum of squared errors: $\min E = \sum_{t=1}^{f} \sum_{i=1}^{n} \left(\Theta_i^{\mathbb{M}^t} - \mathrm{A}_{ti} \right)^2$
- Block coordinate descent, alternatively update:
  - Non-linear layer: using Broyden–Fletcher–Goldfarb–Shanno (BFGS)
  - Linear layer: using constrained linear least squares

- Parameters to fit
  - Rest positions and radii of spheres
  - Rest positions and normals of key points } Non-Linear Layer
  - Blending weights of key points
  - Weights and biases for key point to vertex AO } Linear Layer

25

The objective of our training is to minimize the error for all training poses with regard to a ground truth ambient occlusion.
The parameters we modify during fitting are broken down into the layer they are associated with.

We perform block coordinate descent to minimize the objective function:
For a number of iterations, we alternate between updating the parameters for the non-linear and linear layers separately
For the non-linear layer, we update the associated parameters using Broyden-Fletcher-Goldfarb-Shanno method. We choose BFGS because of it's quadratic convergence rate and because it's requirements for convergance fits our model nicely.
For the linear layer we use constrained least squares to update the weights and biases

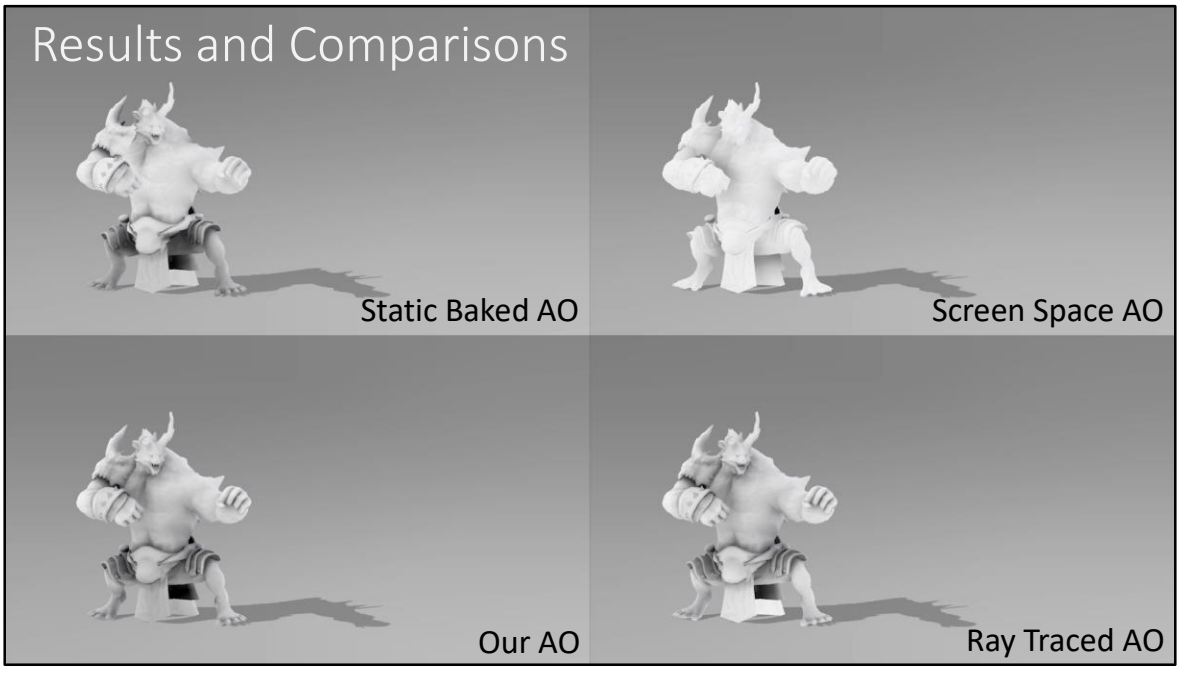
The parameters we optimize for are as follows
For the Non-Linear layer, we update rest positions and radii of spheres, rest positions and normals of key points, as well as their skinning weights
For the linear layer we optimize the weights and biases for the linear combination of key point shadow onto final vertex ambient occlusion

Model Fitting

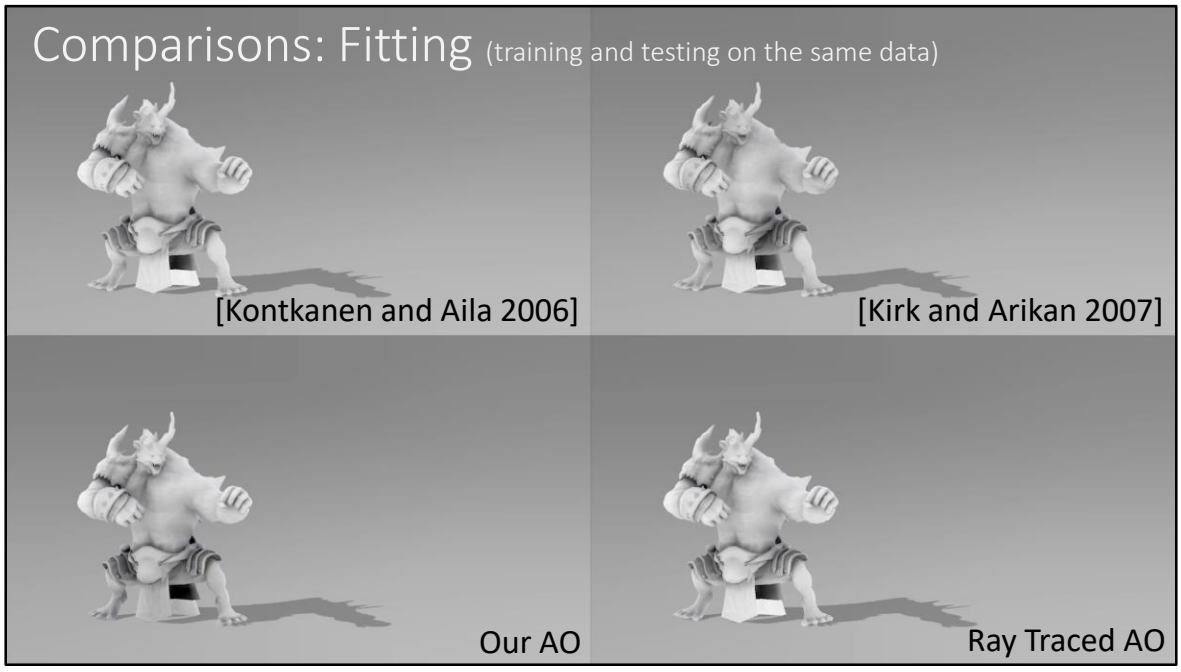
- Block coordinate descent allows for treating parameters from the *other* layer as fixed:
  - Linear layer parameters fixed during Non-Linear BFGS
  - Non-linear parameters fixed during Linear Least Squares solve
- Allows for optimization by pre-computing the other layer's parameters
  - Observe 2 orders of magnitude speedup with caching
- Implemented in parallel on CPUs and GPU

33

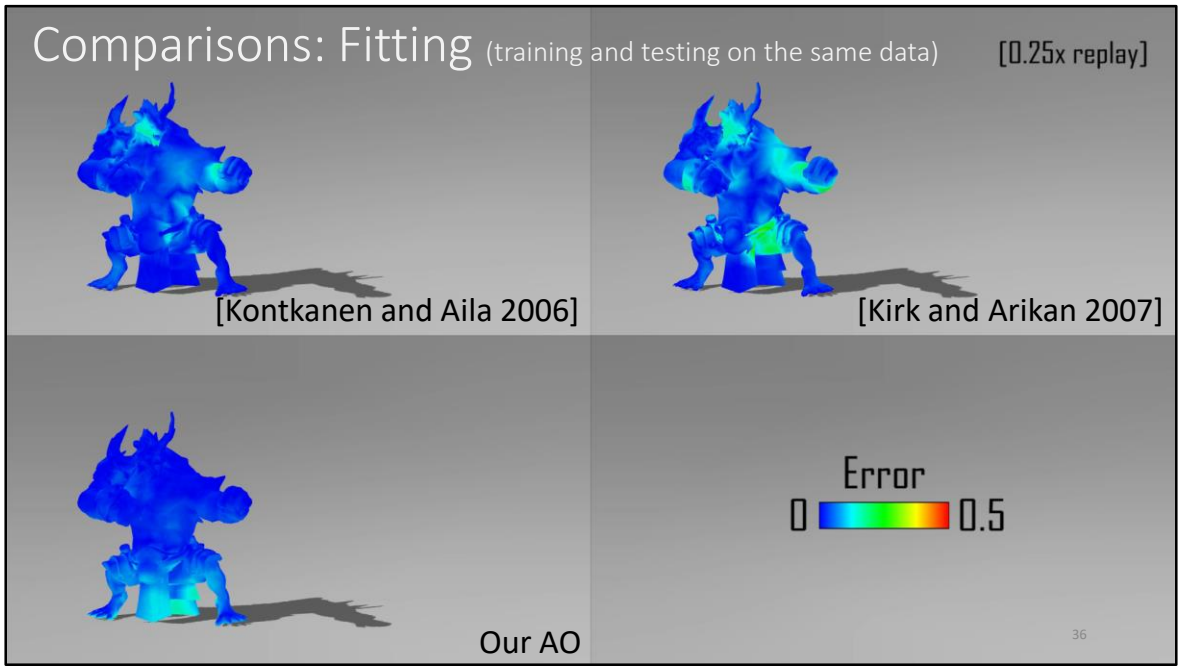
In addition to the quadratic convergence rate of BFGS, our use of block coordinate descent allows us to perform further optimization of our training process.
When performing the update for the relevant layer, block coordinate descent allows us to fix the parameters of the other layer, which means we can precompute these parameters.
In our case, this precomputation allowed for a speedup of 2 orders of magnitude



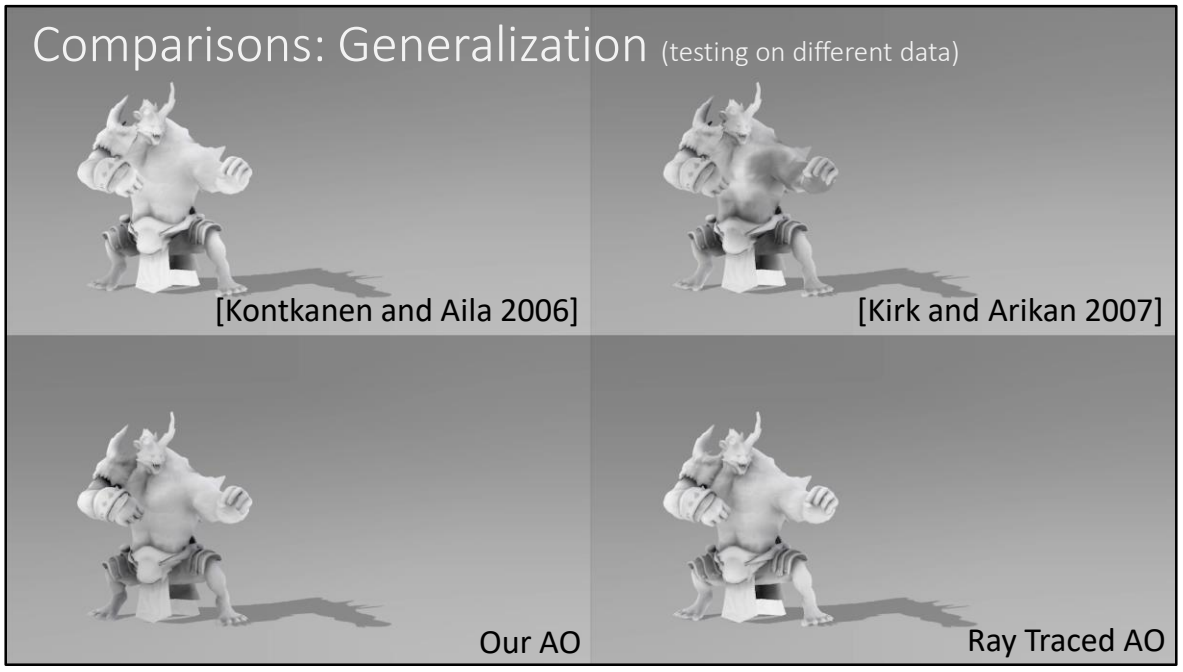
These first comparisons are related to different types of techniques. On top you can see the static baked AO, which cannot update when the pose changes. This is opposed to the most commonly used real time technique in screen space ambient occlusion. On the bottom we see our method as well as the ground truth ray traced ambient occlusion.



These are some comparisons with other object space techniques. In these examples the runtime poses are the same as the poses in the training data.

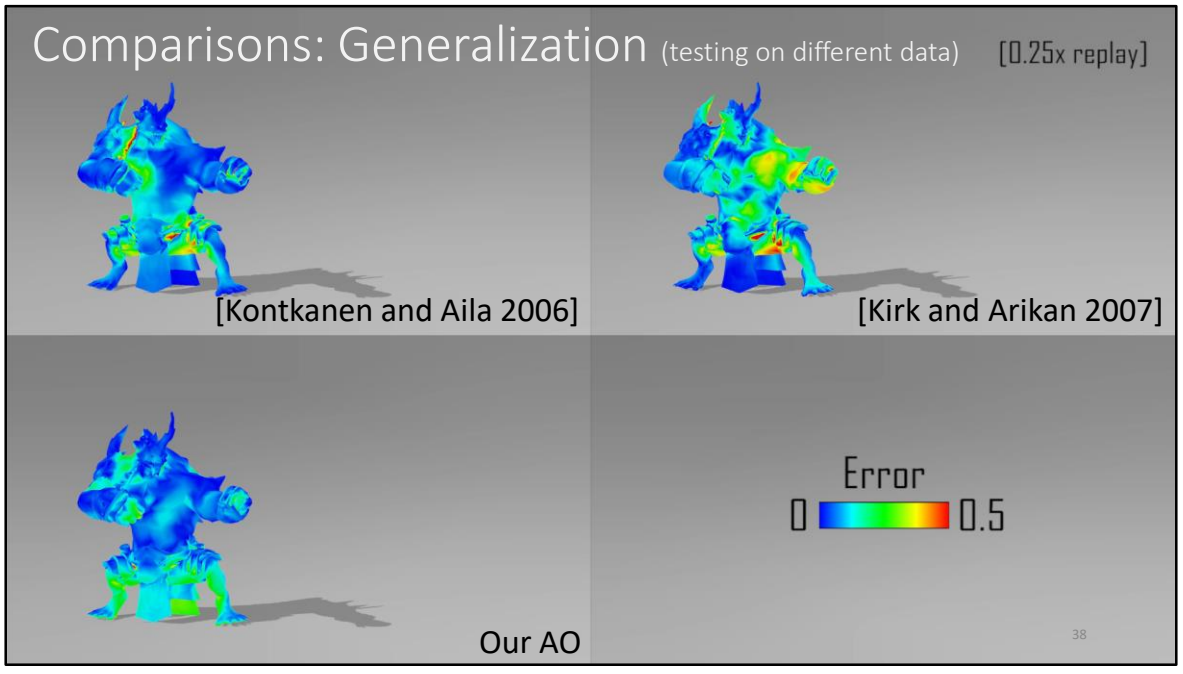


And here we can see the error visualized compared to the ground truth data.

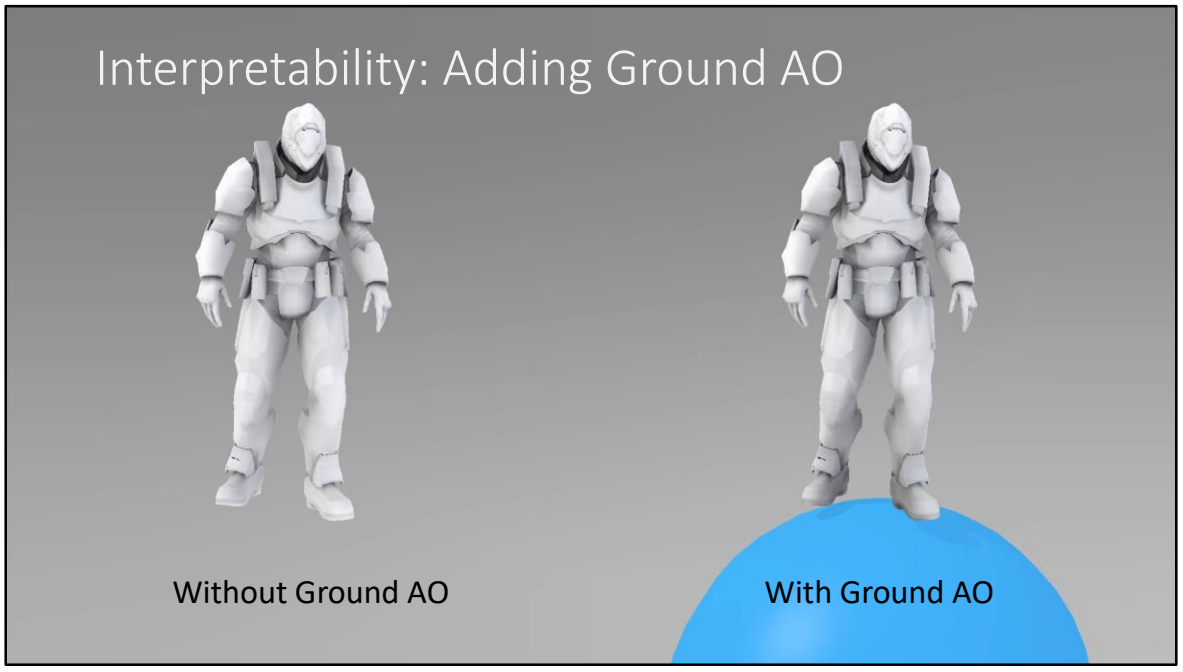


Can talk more about the differences here.

In these examples we show how well the method handles generalization, that is to say the poses are different from the training data. We can see in this case our model can still closely match the ground truth data.

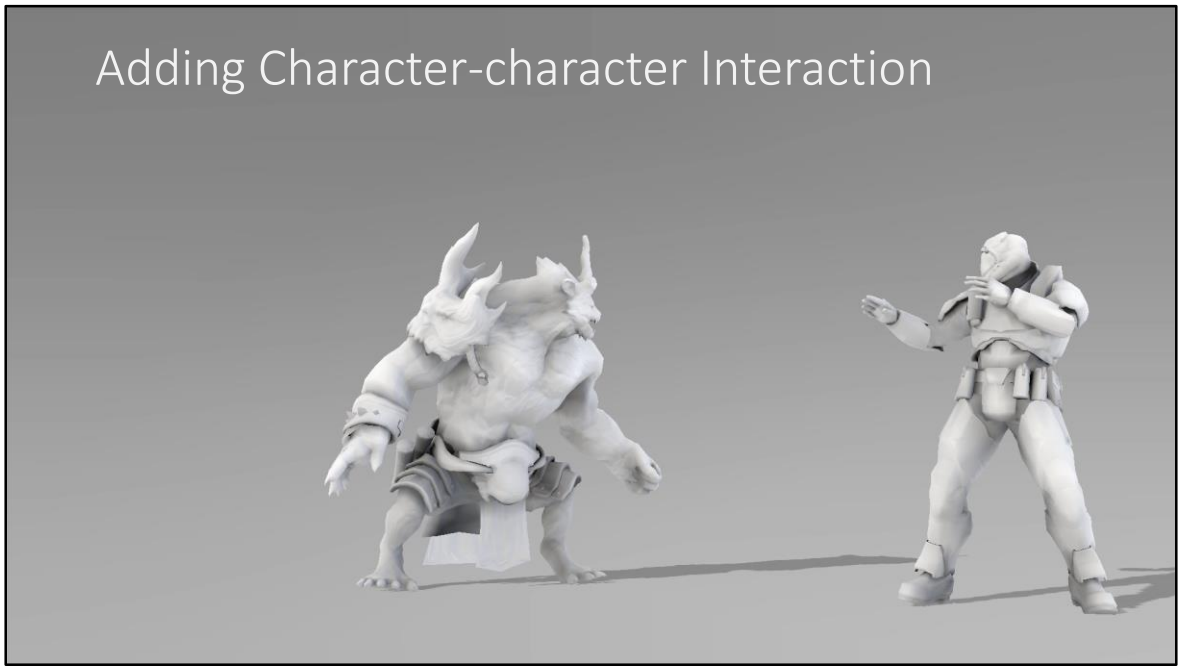


We can see that the error compared is higher on all models for this generalized case



Interpretability also allows us to case occlusion onto the character, with high quality, from any proxy sphere. This allows us to for example very easily add an occlusion term cast by the ground by attaching a proxy sphere that is always below the ground.

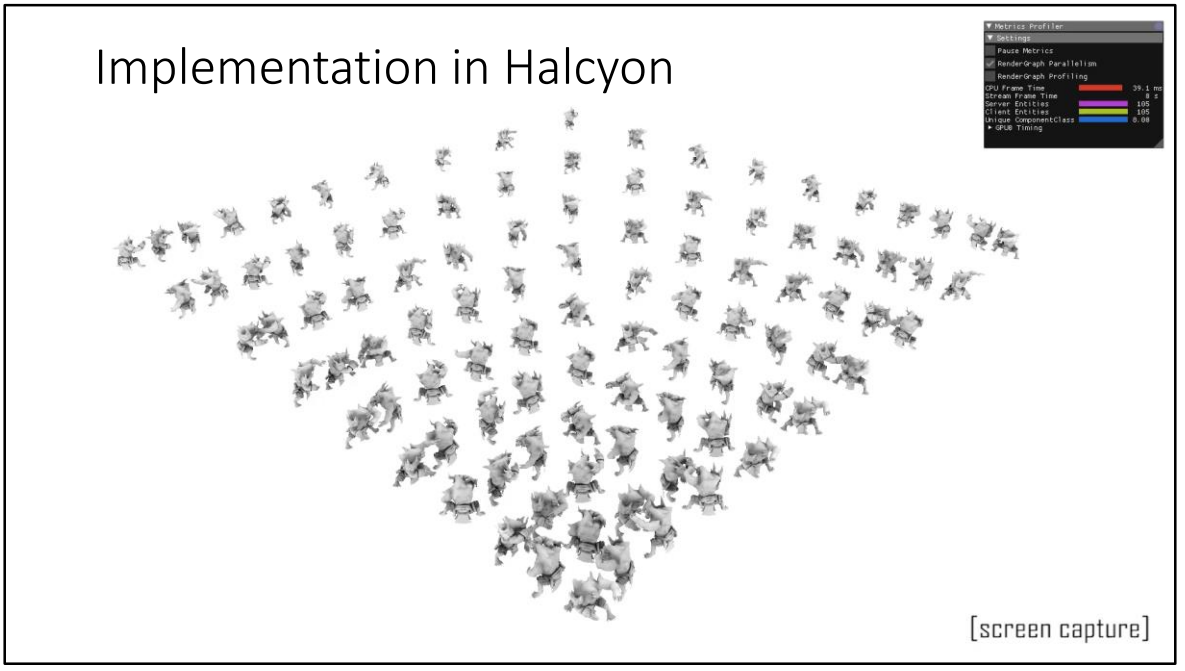
IMPORTANT NOTE, SHOULD MENTION: Can be done very quick, thanks to the Interpretability of our model.
Manually add a proxy sphere and connect x-y translation to root bone of the character.



This is an example of good interpretability of our model. We only train the model on individual characters, with no knowledge of interaction with other characters or objects. By connecting the proxy spheres of one character to the other, we can allow for ambient occlusion interaction. The system is robust enough to handle this in an agreeable way even though this interaction is not part of the training set.

Another example of Interpretability:
Train independent models for each character, connect proxy spheres to create shadow casting between characters.

Implementation in Halcyon



As an example of the real-time aspects of this model, we have a scene here with one hundred characters, where the nearest characters all cast occlusion onto each other. We can see that the characters that are more closely bunched up receive more general occlusion than the ones that are spaced out further. In this example the linear layer is computed on the GPU, however the non-linear layer is calculated on the CPU. Most of the frame time is taken up by this CPU calculation of the non-linear layer, so if this was moved to the GPU we predict things would be quite a bit faster.

- 100 characters
- AO for each character casted from itself and 4 neighbors
- AO effect on near characters v.s. distant characters

Conclusion

Data-driven, Object-space AO Model for Animated Characters

- ✓Fast and simple runtime model: no intersection test
- ✓Robust: good generalization, high quality
- ✓Interpretability: approximates shadowing with the ground and interaction with other characters

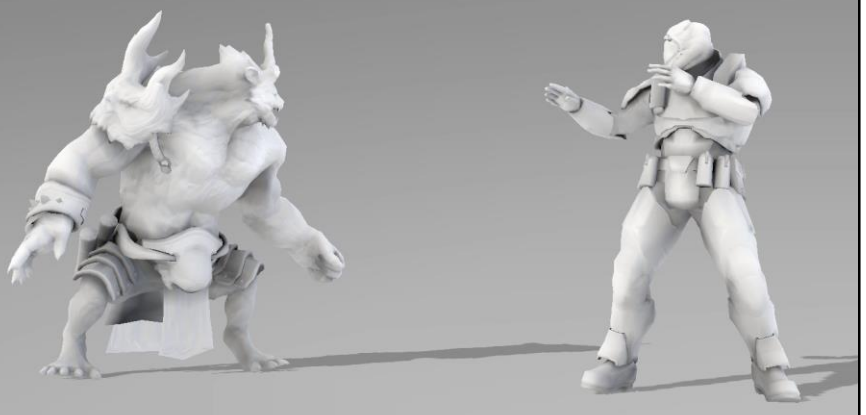
Future work: faster training, better automatic data sampling

SEED.EA.COM – We are hiring

We are a cross-disciplinary team within EA Worldwide Studios.
Our mission is to explore, build and help define the future of interactive entertainment.

No Character-character Interaction

(proxy spheres of each character only cast shadows
to key points of the same character)



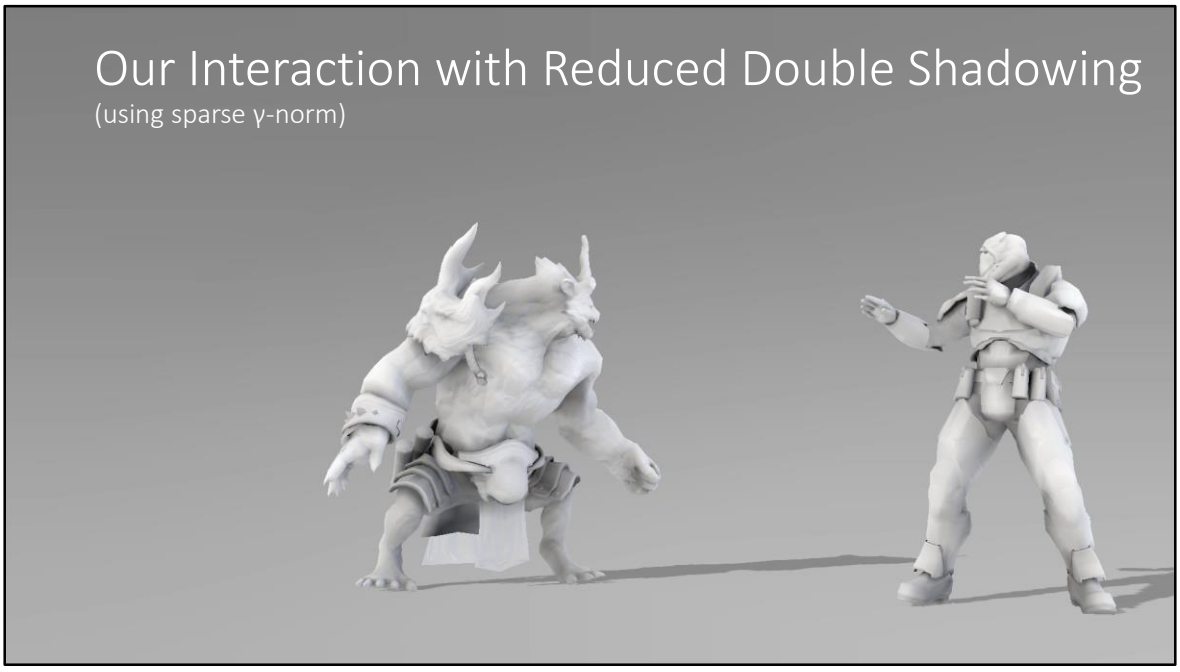
Interaction with Double Shadowing

(adding shadows from proxy spheres of character 1
to key points of character 2 and vice versa)

[0.5x replay]

Our Interaction with Reduced Double Shadowing

(using sparse γ-norm)




Double Shadowing
Ours
[0.5x replay]

Data and Training

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

n : number of vertices
m : number of bones
f : number of training poses
$size_{data}$: size of training data

s : number of proxy spheres
p : number of key points

$size_{model}$: size of trained model

50

In addition to the quadratic convergence rate of BFGS, our use of block coordinate descent allows us to perform further optimization of our training process.
When performing the update for the relevant layer, block coordinate descent allows us to fix the parameters of the other layer, which means we can precompute these parameters.
In our case, this precomputation allowed for a speedup of 2 orders of magnitude

ALGORITHM 1: Compute Per-vertex Ambient Occlusion

input : $\mathbb{M} = \{\mathbf{M}_j | j = 1..m\}$, where $\mathbf{M}_j = \left[\mathbf{R}_j | \mathbf{t}_j\right]$. // Bone transformations

parameters: $\{\mathbf{o}_h, r_h | h = 1..s\}$, $\{\mathbf{c}_k, \mathbf{u}_k | k = 1..p\}$, // Spheres and key points

$\{w_{kj} | k = 1..p, j = 1..m\}$, // Blending weights of key points

$\{\alpha_{ik}, \beta_i | i = 1..n, k = 1..p\}$. // Weights and biases of key points to vertex

output : $\{\Theta_i^{\mathbb{M}} | i = 1..n\}$. // Per-vertex AO

1 **foreach** *proxy sphere h* **do**
2 | Compute center $\tilde{\mathbf{o}}_h^{\mathbb{M}}$;
3 **end**
4 **foreach** *key point k* **do**
5 | Compute position $\tilde{\mathbf{c}}_k^{\mathbb{M}}$ and normal $\tilde{\mathbf{u}}_k^{\mathbb{M}}$;
6 | Compute shadow $\Gamma_k^{\mathbb{M}}$;
7 **end**
8 **foreach** *vertex i* **do**
9 | Compute AO value $\Theta_i^{\mathbb{M}}$;
10 **end**

51

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 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.

BFGS v.s. Conjugate Gradient

- BFGS allows for quadratic convergence rate
- Conjugate gradient (typical for NN) has linear convergence rate
- However, BFGS methods are not guaranteed to converge unless the function has a quadratic Taylor expansion near an optimum
- Our objective function is sum of squared errors, which is similar to quadratic, so BFGS is suitable

53

Model Fitting

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

54

Model Fitting

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

$$= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(1 - \left(\sum_{k=1}^{p}\alpha_{ik}\Gamma_k^{\mathbb{M}^t} + \beta_i\right) - \mathbf{A}_{ti}\right)^2$$

$$= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(\sum_{k=1}^{p}\sum_{k'=1}^{p}\alpha_{ik}\alpha_{ik'}\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + 2\sum_{k=1}^{p}\alpha_{ik}\left(1 - \beta_i - \mathbf{A}_{ti}\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}\right)$$

$$= \sum_{t=1}^{f}\sum_{k=1}^{p}\sum_{k'=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\alpha_{ik'}\right)\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + 2\sum_{t=1}^{f}\sum_{k=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\left(1 - \beta_i - \mathbf{A}_{ti}\right)\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}$$

Model Fitting

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

$$= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(1 - \left(\sum_{k=1}^{p}\alpha_{ik}\Gamma_k^{\mathbb{M}^t} + \beta_i\right) - \mathrm{A}_{ti}\right)^2$$

$$= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(\sum_{k=1}^{p}\sum_{k'=1}^{p}\alpha_{ik}\alpha_{ik'}\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + 2\sum_{k=1}^{p}\alpha_{ik}\left(1 - \beta_i - \mathrm{A}_{ti}\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}\right)$$

$$= \sum_{t=1}^{f}\sum_{k=1}^{p}\sum_{k'=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\alpha_{ik'}\right)\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + 2\sum_{t=1}^{f}\sum_{k=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\left(1 - \beta_i - \mathrm{A}_{ti}\right)\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}$$

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

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

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

56

Model Fitting

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

$$= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(1 - \left(\sum_{k=1}^{p}\alpha_{ik}\Gamma_k^{\mathbb{M}^t} + \beta_i\right) - \mathbf{A}_{ti}\right)^2$$

$$= \sum_{t=1}^{f}\sum_{i=1}^{n}\left(\sum_{k=1}^{p}\sum_{k'=1}^{p}\alpha_{ik}\alpha_{ik'}\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + 2\sum_{k=1}^{p}\alpha_{ik}\left(1 - \beta_i - \mathbf{A}_{ti}\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}\right)$$

$$= \sum_{t=1}^{f}\sum_{k=1}^{p}\sum_{k'=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\alpha_{ik'}\right)\Gamma_k^{\mathbb{M}^t}\Gamma_{k'}^{\mathbb{M}^t} + 2\sum_{t=1}^{f}\sum_{k=1}^{p}\left(\sum_{i=1}^{n}\alpha_{ik}\left(1 - \beta_i - \mathbf{A}_{ti}\right)\right)\Gamma_k^{\mathbb{M}^t} + \text{constant}$$

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

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

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

Precompute

- Great speedup of training
- Similar for Linear Layer

57