Neural Network Ambient Occlusion

Daniel Holden* University of Edinburgh
Jun Saito† Method Studios
Taku Komura‡ University of Edinburgh

Figure 1: *Comparison showing Neural Network Ambient Occlusion enabled and disabled inside a game engine.*

Abstract

We present Neural Network Ambient Occlusion (NNAO), a fast, accurate screen space ambient occlusion algorithm that uses a neural network to learn an optimal approximation of the ambient occlusion effect. Our network is carefully designed such that it can be computed in a single pass allowing it to be used as a drop-in replacement for existing screen space ambient occlusion techniques.

Keywords: neural networks, machine learning, screen space ambient occlusion, SSAO, HBAO

1 Introduction

Ambient Occlusion is a key component in the lighting of a scene but expensive to calculate. By far the most popular approximation used in real-time applications is Screen Space Ambient Occlusion (SSAO), a method which uses the depth buffer and other screen space information to calculate occlusions. Screen space techniques have seen wide adoption because they are independent of scene complexity, simple to implement, and fast.

Yet, calculating effects in screen space often creates artifacts, as complete information about the scene is unavaliable. The exact behaviour of these artifacts can be difficult to predict but by using machine learning we can learn an SSAO algorithm that minimises these errors with respect to some cost function.

Using a database of camera depths, normals, and *ground truth* ambient occlusion as calculated using an offline renderer, we train a neural network to learn a mapping from the depth and normals surrounding the pixel to the ambient occlusion of that pixel. Once trained we convert the neural network into an optimised shader which is more accurate than existing techniques, has better performance, no user parameters other than the occlusion radius, and can be computed in a single pass allowing it to be used as a drop-in replacement for existing techniques.

*email:contact@theorangeduck.com
†email:dukecyto@gmail.com
‡email:tkomura@ed.ac.uk

2 Related Work

Screen Space Ambient Occlusion Screen Space Ambient Occlusion (SSAO) was first introduced by Mittring [2007] for use in *Cryengine2*. The approach samples around the depth buffer in a view space sphere and counts the number of points which are inside the depth surface to estimate the occlusion. This method has seen wide adoption but often produces artifacts such as dark halos around object silhouettes or white highlights on object edges. Filion and McNaughton [2008] presented SSAO+, an extension which samples in a hemisphere oriented in the direction of the surface normal. This removes the white highlights around object edges and reduces the required sampling count but still sometimes produces dark halos. Bavoil et al. [2008] introduced Horizon Based Ambient Occlusion (HBAO). This technique predicts the occlusion by estimating the openness of the horizon around the sample point. Rays are marched along the depth buffer and the difference in depth is used to calculate the horizon estimate. This was extended by Mittring [2012] who improved the performance with paired samples. HBAO produces a more realistic effect but does not account for the fact that the camera depth map is an approximation of the true scene geometry. McGuire et al. [2011] introduced Alchemy Screen-Space Ambient Obscurance (ASSAO), an effect which substitutes a fixed falloff function into the general lighting equation to create a more physically accurate integration over the occlusion term. ASSAO produces a physically based result, but still does not deal directly with the errors introduced by the screen space approximation.

Machine Learning for Screen Space Effects So far machine learning has seen very limited application to rendering and screen space effects. In offline rendering Kalantari et al. [2015] used machine learning to filter the noise produced by Monte Carlo rendering at low sample rates. Ren et al. [2015] used neural networks to perform image space relighting of scenes, allowing users to virtually adjust the lighting of scenes even with complex materials. Finally Johnson et al. [2011] used machine learning alongside a large repository of photographs to improve the realism of renderings - adjusting patches of the output to be more similar to corresponding patches of photographs in the database.

3 Preprocessing

To produce the complex scenes required for training our network we make use of the geometry, props, and scenes of the Open Source first person shooter Black Mesa [Crowbar-Collective 2015]. We take several scenes from the game and add additional geometry and clutter to ensure a wide variety of objects and occlusions are present.

We produce five scenes and select 100-150 viewpoints using different perspectives and camera angles. From each viewpoint we use *Mental Ray* to render scene depth, camera space normals, and global ambient occlusion at a resolution of 1280×720. From each image we randomly pick 1024 pixels and perform the following process to extract the input features used in training. Experimentally we found these features to produce the best results.

Given a pixel's depth we use the inverse camera projection matrix to calculate the position of the pixel as viewed from the camera (the *view space position*). We then take $w \times w$ samples in a view space regular grid centered around this position and scaled by the user given AO radius r. We reproject each sample into the screen space using the camera projection matrix and sample the GBuffer to find the corresponding pixel normal and depth. For each sample we take the difference between its normal and that of the center pixel. Additionally we take the difference between its view space depth and that of the center pixel. These values we put into a four dimension vector. We then calculate the view space distance of the sample to the center pixel, divide it by the AO radius r, subtract one, and clamp to the range $[0, 1]$. Using this value we scale the four dimensional input vector. This ensures that all samples outside of the occlusion radius have a value of zero and cannot influence the output. We concatenate the values from each of these samples into one large vector. This represents a single input data point $\mathbf{x} \in \mathbb{R}^{w^2 4}$ where in this work $w = 31$. We then take the ambient occlusion value of the center pixel as the corresponding output data $\mathbf{y} \in \mathbb{R}^1$.

Once complete we have a final dataset of around 500000 data points. We then normalise the data by subtracting the mean and dividing by the standard deviation.

4 Training

Our network is designed such that it can be computed by a shader in a single pass. We therefore use a simple four layer neural network where the operation of a single layer $\Phi_n(\mathbf{x})$ is given by the following

$$\Phi_n(\mathbf{x}) = PReLU(\mathbf{W}_n\,\mathbf{x} + \mathbf{b}_n, \alpha_n, \beta_n) \tag{1}$$

where $PReLU(\mathbf{x}, \alpha, \beta) = \beta\, max(\mathbf{x}, 0) + \alpha\, min(\mathbf{x}, 0)$ is a variation of the *Parametric Rectified Linear Unit* first proposed by [He et al. 2015] with an additional scaling term β for the positive activation. The parameters of our network are as follows $\theta = \{\mathbf{W_0} \in \mathbb{R}^{w^2 4 \times 4}, \mathbf{W_1} \in \mathbb{R}^{4\times4}, \mathbf{W_2} \in \mathbb{R}^{4\times4}, \mathbf{W_3} \in \mathbb{R}^{4\times1}, \mathbf{b_0} \in \mathbb{R}^4, \mathbf{b_1} \in \mathbb{R}^4, \mathbf{b_2} \in \mathbb{R}^4, \mathbf{b_3} \in \mathbb{R}^1, \alpha_\mathbf{0} \in \mathbb{R}^4, \alpha_\mathbf{1} \in \mathbb{R}^4, \alpha_\mathbf{2} \in \mathbb{R}^4, \alpha_\mathbf{3} \in \mathbb{R}^1, \beta_\mathbf{0} \in \mathbb{R}^4, \beta_\mathbf{1} \in \mathbb{R}^4, \beta_\mathbf{2} \in \mathbb{R}^4, \beta_\mathbf{3} \in \mathbb{R}^1\}$.

The cost function of our network consists of the mean squared regression error and a small regularisation term scaled by γ which in this work we set to 0.01. Here $\mathbf{x}$ is the input features extracted in the preprocessing stage and $\mathbf{y}$ is the corresponding ambient occlusion of the pixel.

$$Cost(\mathbf{x}, \mathbf{y}, \theta) = \|\mathbf{y} - \Phi_3(\Phi_2(\Phi_1(\Phi_0(\mathbf{x}))))\|^2 + \gamma\,|\theta| \tag{2}$$

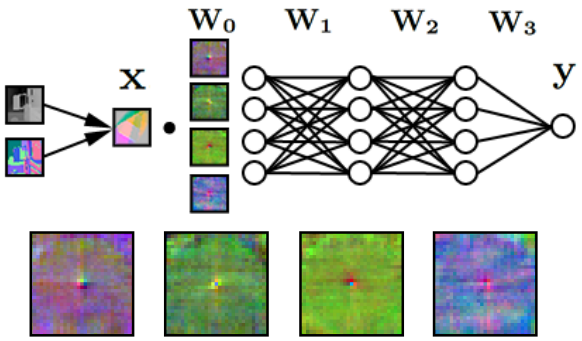

Figure 2: *Top: overview of our neural network. On the first layer four independent dot products are performed between the input and* $\mathbf{W}_0$ *represented as four 2D* filters. *The rest of the layers are standard neural network layers. Bottom: the four filter images extracted from* $\mathbf{W}_0$.

We pick random elements from our dataset in mini-batches of 16 and evaluate the above cost function. The parameters of the network are then updated using derivatives calculated by Theano [Bergstra et al. 2010], and the adaptive gradient descent algorithm *Adam* [Kingma and Ba 2014]. To avoid over-fitting, we use a *Dropout* [Srivastava et al. 2014] of 0.5 on the input layer. Training is performed for 100 epochs and takes around 10 hours on a NVIDIA GeForce GTX 660 GPU.

5 Runtime

After training the data preprocessing and neural network operation need to be reproduced in a shader for use at runtime. The shader is mostly a straight forward translation but has a few exceptions which are detailed below. The full shader and network weights are provided in the supplementary material.

As the total memory required to store the network weight $\mathbf{W_0}$ exceeds the maximum memory reserved for local shader variables it cannot be stored in the shader code. Instead we observe that multiplication by $\mathbf{W_0}$ can be described as four independent dot products between columns of the matrix and the input $\mathbf{x}$. As the input $\mathbf{x}$ is produced by sampling in a 2D grid, we can compute these dot products in 2D, and store the weights matrix $\mathbf{W_0}$ as four 2D textures we call *filters*. These 2D textures are then sampled and multiplied by the corresponding components of the input vector (see Figure. 2).

Performing the dot product in 2D allows us to approximate the multiplication by $\mathbf{W_0}$. We can compute the dot product between the input and the filters in just a few sample locations and rescale the result using the ratio between the number of samples taken and the full number of elements in $\mathbf{W_0}$. We use stratified sampling - regularly picking every nth pixel of the filters and taking the dot product with the corresponding input sample, finally multiplying the total by n. We also introduce a small amount of 2D jitter to the sampling locations to spread the approximation in the screen space. This allows us to accurately approximate the multiplication of $\mathbf{W_0}$ at the cost of some noise in the output. As with other SSAO algorithms, the output is therefore post processed using a bilateral blur.

6 Results

In Figure. 3 we visually compare the results of our method to SSAO+ (16 samples), HBAO (64 samples), and the ground truth. HBAO produces good results in general but requires almost twice

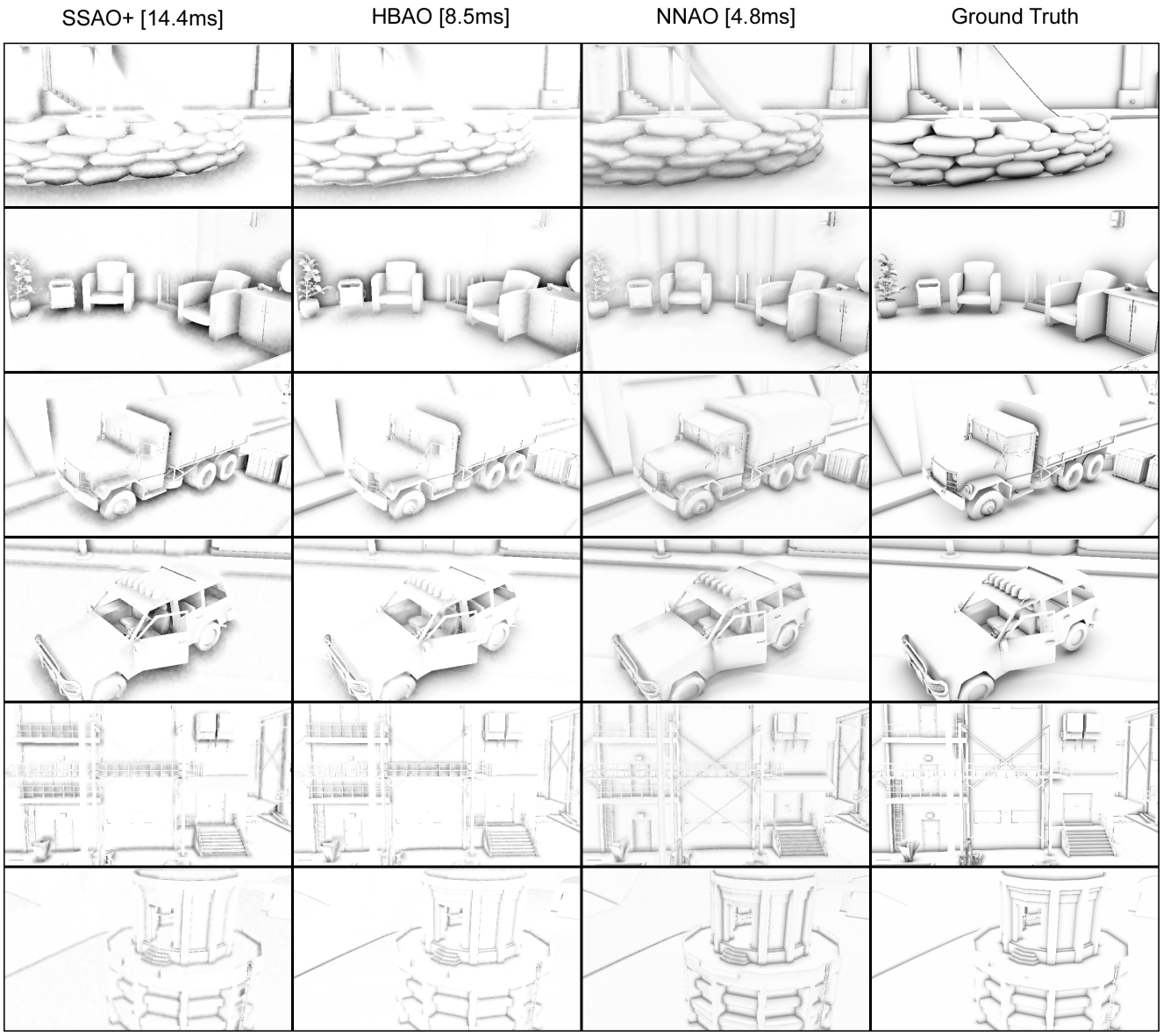

Figure 3: *Comparison to other techniques. From left to right: SSAO+, HBAO, NNAO (Our Method), Ground Truth.*

the runtime of our method (See Figure. 4) and in many places creates areas which are too dark. See: under the sandbags, behind the furniture, inside the car, between the railings, on the stairs. In these cases our method is more conservative as it tries to minimise the mean squared error. This may also be why our method appears less contrasted than other methods.

In Figure. 1 we implement our method in a game engine. This shows our network generalises beyond the training data and additionally that it works in interactive applications. Please see the supplementary video for a longer demonstration.

In Table. 1 we perform a numerical comparison between our method and previous techniques. Our method has a lower mean squared error on the test set with comparable or better performance. All measurements are taken at half resolution (640×360) on a NVIDIA GeForce GTX 660 GPU. Due to the unpredictability in measuring GPU performance runtimes may vary in practice.

7 Discussion

In Figure. 5 we visualise what is being learned by the neural network. We show the activations of the first three filters using the cyan, yellow, and magenta channels of the image. Each filter learns a separate component of the occlusion with cyan learning unoccluded areas, magenta learning the occlusion of horizontal surfaces and yellow learning the occlusion of vertical surfaces.

Our method is capable of performing many more samples than other methods in a shorter amount of time because it samples in a regular grid. This gives it very good cache behaviour. There is also no data dependency between samples which gives our method a greater level of parallelism. Each sample is re-used by each of the four filters, resulting in less noise. On the other hand, while our method may have better IO performance than other methods, it does require significantly more computational and so this can become the new bottleneck.

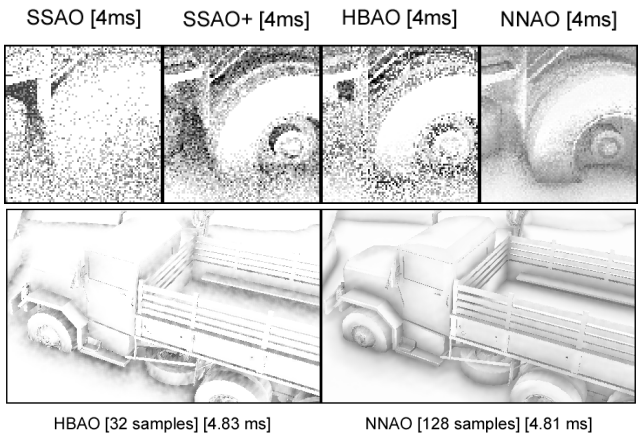

Figure 4: *Given similar runtimes, our algorithm performs more samples in a shorter time, producing less noise and a better quality output in comparison to HBAO, which appears blotchy at low sampling rates.*

Algorithm	Sample Count	Runtime (ms)	Error (mse)
SSAO	4	1.20	1.765
SSAO	8	1.43	1.558
SSAO	16	14.71	1.539
SSAO+	4	1.16	0.974
SSAO+	8	1.29	0.818
SSAO+	16	14.46	0.811
HBAO	16	3.53	0.965
HBAO	32	4.83	0.709
HBAO	64	8.50	0.666
NNAO	64	4.17	0.510
NNAO	128	4.81	0.486
NNAO	256	6.87	0.477

Table 1: *Numerical comparison between our method and others.*

7.1 Limitations & Future Work

Our method is not trained on data that includes high detail normal maps in the GBuffer. Although our method can be used on GBuffers with detailed normals (see Figure. 1) it is likely our method would perform better if trained on this kind of data. Reducing the sampling count of our method below 64 does not reduce the runtime very much. In this case further control over the performance would be desirable. Our technique produces ambient occlusion but we believe it could also be applied to other screen space effects such as Screen Space Radiosity, Screen Space Reflections and more.

7.2 Conclusion

We present a technique for performing screen space ambient occlusion using neural networks. After training we create an optimised shader that reproduces the network forward pass efficiently and controllably. Our method produces fast, accurate results and can be used as a drop-in replacement to existing screen space ambient occlusion techniques.

References

BAVOIL, L., SAINZ, M., AND DIMITROV, R. 2008. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks*, ACM, New York, NY, USA, SIGGRAPH '08, 22:1–22:1.

BERGSTRA, J., BREULEUX, O., BASTIEN, F., LAMBLIN, P., PASCANU, R., DESJARDINS, G., TURIAN, J., WARDE-FARLEY, D., AND BENGIO, Y. 2010. Theano: a CPU and GPU math expression compiler. In *Proc. of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

CROWBAR-COLLECTIVE, 2015. Black mesa. `http://www.blackmesasource.com/`.

FILION, D., AND MCNAUGHTON, R. 2008. Effects & techniques. In *ACM SIGGRAPH 2008 Games*, ACM, New York, NY, USA, SIGGRAPH '08, 133–164.

HE, K., ZHANG, X., REN, S., AND SUN, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR abs/1502.01852*.

JOHNSON, M. K., DALE, K., AVIDAN, S., PFISTER, H., FREEMAN, W. T., AND MATUSIK, W. 2011. Cg2real: Improving the realism of computer generated images using a large collection of photographs. *IEEE Transactions on Visualization and Computer Graphics 17*, 9 (Sept), 1273–1285.

KALANTARI, N. K., BAKO, S., AND SEN, P. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2015) 34*, 4.

KINGMA, D. P., AND BA, J. 2014. Adam: A method for stochastic optimization. *CoRR abs/1412.6980*.

MCGUIRE, M., OSMAN, B., BUKOWSKI, M., AND HENNESSY, P. 2011. The alchemy screen-space ambient obscurance algorithm. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, ACM, New York, NY, USA, HPG '11, 25–32.

MITTRING, M. 2007. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 Courses*, ACM, New York, NY, USA, SIGGRAPH '07, 97–121.

MITTRING. 2012. The technology behind the "unreal engine 4 elemental demo". In *ACM SIGGRAPH 2012 Talks*, ACM, New York, NY, USA, SIGGRAPH '12.

REN, P., DONG, Y., LIN, S., TONG, X., AND GUO, B. 2015. Image based relighting using neural networks. *ACM Trans. Graph. 34*, 4 (July), 111:1–111:12.

SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res. 15*, 1 (Jan.), 1929–1958.

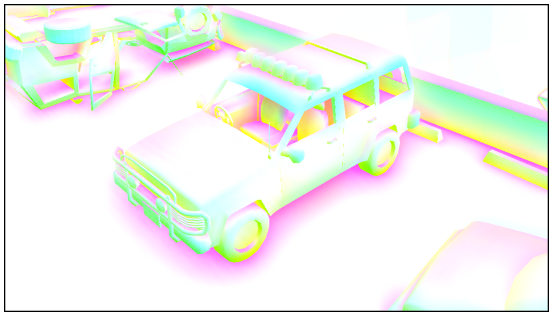

Figure 5: *The activations of the first three filters represented by the cyan, yellow, and magenta channels of the image.*