Ambient Obsurance Baking on the GPU

Peter-Pike Sloan

Jason Tranchida

Hao Chen

Ladislav Kavan
University of Pennsylvania, ladislav.kavan@gmail.com

Ambient Obsurance Baking on the GPU

Abstract
Ambient Occlusion and Ambient Obscurance are coarse approximations to global illumination from ambient lighting, commonly used in film and games. This paper describes a system that computes Ambient Obscurance over the vertices of complex polygon meshes. Novel contributions include pre-processing necessary for "triangle soup" scene representations to minimize artifacts, a compact model for different classes of instanced decorator objects such as trees and shrubs, a compact model for pre-computed visibility to be used on dynamically placed objects, and an approximation to model the occlusion of small decorator objects when ray tracing.

Keywords
ambient obsurance, global illumination, GPU

Disciplines
Computer Sciences | Engineering | Graphics and Human Computer Interfaces

Comments
SIGGRAPH Asia 2013 was held November 19-22, 2013, in Hong Kong.

Ambient Obscurance Baking on the GPU

Peter-Pike Sloan
NVIDIA/Activision

Jason Tranchida
Bungie

Hao Chen
Bungie

Ladislav Kavan
University of Pennsylvania

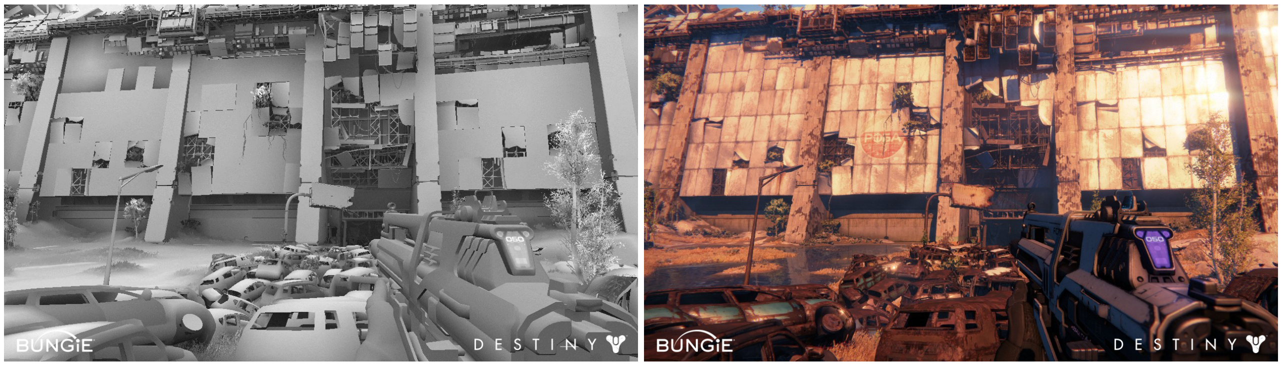

Figure 1: *Ambient Obscurance (left) and a final frame in a game (right). Our method supports instanced trees and shrubs and the gun lighting changes as the character moves thanks to our new dynamic Ambient Obscurance model. ©Bungie*

Abstract

Ambient Occlusion and Ambient Obscurance are coarse approximations to global illumination from ambient lighting, commonly used in film and games. This paper describes a system that computes Ambient Obscurance over the vertices of complex polygon meshes. Novel contributions include pre-processing necessary for "triangle soup" scene representations to minimize artifacts, a compact model for different classes of instanced decorator objects such as trees and shrubs, a compact model for pre-computed visibility to be used on dynamically placed objects, and an approximation to model the occlusion of small decorator objects when ray tracing.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Display Algorithms

Keywords: Ambient Obscurance, global illumination, GPU

1 Introduction and Motivation

Generating realistic imagery in video games is a challenging problem. While real-time global illumination has received much attention [Ritschel et al. 2012], most of the techniques are still too costly, particularly on game consoles. One common alternative is to pre-compute, or "bake", some form of lighting or light transport [Lehtinen 2007]. This work focuses on baking Ambient Obscurance (AO) [Zhukov et al. 1998] to the vertices of large game levels, including support for instanced decorator objects and spatial sampling necessary for dynamic objects. While visibility has been pre-computed for shots in film using GPUs [Pantaleoni et al. 2010], we are focused on baking for games with strict memory constraints. The game the system was developed for has the following constraints/concerns that led to our solution:

- Ease of authoring levels is paramount, the geometry is very unstructured, heavily instanced and often interpenetrating. Prior experience with unique parameterizations led us away from them due to more complicated workflows.
- Since outdoor lighting uses a dynamic time of day, we were looking for a way to attenuate lighting particularly in regions that transition from outside to interior spaces.
- Given the rich geometry of game levels, memory budget was a primary concern. Storing an extra set of UV coordinates and pre-computed textures was impractical. For instanced objects such as trees even per-vertex values would be too expensive.
- The rendering system in the game uses deferred shading and only a small amount of space in the G-buffer is available.

This paper describes a system that extends prior work [Kavan et al. 2011] on baking signals to mesh vertices. While real time solutions for AO exist [Ritschel et al. 2012; Loos and Sloan 2010] it is well known that even on high end platforms the screen space algorithms do not adequately handle large distances due to the limited scene model that they employ, causing temporal artifacts as geometry gets added into the frame buffer. Our emphasis is particularly focused on distant occlusion and is useful even on high end hardware. Our proposed method handles complex triangle soup geometry including inter-penetrations in a robust way and uses a compact operator to model the variation in AO for large instanced objects like trees. To handle dynamic objects freely moving through the environment, we provide a model for pre-computing spatial visibility using compact models, introducing a novel form of regularization. Finally, we propose an approximation to efficiently model the occlusion of small decorator objects such as bushes and shrubs.

2 Ambient Obscurance

Ambient Occlusion [Landis 2002] is a shadowing term for infinitely distant constant lighting. A related concept of Ambient Obscurance [Zhukov et al. 1998] is an approximation that uses the distance to the first occluder to attenuate visibility. While we initially experimented with computing Ambient Occlusion at multiple scales and blending, Ambient Obscurance generated more satisfying results, particularly for interior scenes. Mathematically, Ambient Obscurance at a point $\mathbf{p}$ is expressed using the following integral:

$$AO(\mathbf{p}) = \frac{1}{\pi} \int_{\Omega} \rho(d(\mathbf{p}, \omega)) \cos\theta d\omega \tag{1}$$

where $d(\mathbf{p}, \omega)$ computes the distance between the point $\mathbf{p}$ and the closest blocker in the direction ω and θ is the angle between the ray direction ω and the surface normal at $\mathbf{p}$. We use an exponential kernel: $\rho(x) = 1 - \exp(ax)$, where the parameter a is defined so that at distance d (in meters) the visibility will be attenuated by one half. This leads to $a = \log(0.5)/d$. In our example we always compute AO for two distinct distance targets, $2m$ and $300m$, and allow the artists to blend them per level. While screen space approximations to obscurance exist [Loos and Sloan 2010], their performance and quality deteriorates for large radii and distant geometry. If desired, screen space techniques can be used to complement our method since they handle small scale detail well.

2.1 Least Squares Vertex Baking

We build off the least squares vertex baking technique [Kavan et al. 2011], which is based on minimizing the error function:

$$E(\mathbf{x}) = \int_S (f(\mathbf{p}) - g_{\mathbf{x}}(\mathbf{p}))^2 d\mathbf{p} + \alpha \int_S (R_{\text{edge}}(\mathbf{p}))^2 d\mathbf{p} \tag{2}$$

where f is the input Ambient Obscurance signal, $g_{\mathbf{x}}$ is its piecewise linear approximation, $\mathbf{x}$ is the vector of coefficients, i.e., the per-vertex values, and S is the surface of our scene. The first integral in Equation 2 models squared approximation error integrated over the surface and the second integral is an edge based regularizer, see Kavan et al. [2011] for details. We use a sparse direct solver to solve the resulting linear system. We experimentally found that $\alpha = 1$ seems to work well for our scenes.

There are two modifications that we made to this technique to work on our large game levels. First, there is large coarse distant geometry that needs to compute AO and affect the AO of the playable area. If we sampled per unit area the playable area would be undersampled and the distant geometry would be oversampled. Instead we have a maximum number of samples per face; the finite element framework [Kavan et al. 2011] takes care of their appropriate weighting. This causes more samples to be focused in the playable area. Second, when tracing rays, we compute the closest hit (which is necessary for obscurance) and track the number of rays that were back facing at each surface location. When the number of rays hitting back facing geometry is above a threshold (25%) we invalidate this sample. We found this to be more robust then using an AO threshold as in [Kavan et al. 2011].

For ground cover like trees and shrubs, we initially tried to directly instantiate (replicate) the geometry, but this significantly increased the time and memory needed to represent the scene bounding volume hiearchy (BVH). On a medium sized scene this resulted in adding about 30 seconds to the BVH build in OptiX [Parker et al. 2010]. It can also lead to aliasing with low sampling densities. Instead we implemented the following approach: shoot the opaque ray first, then shoot a ray from the origin to the intersection location where we intersect a second BVH of just the bounding boxes of these objects. Each object has a density and exponential attenuation is used to modify the obscurance multiplicatively, computing the length of the intersection of a ray and a given bounding box. In a pre-processing step we compute the amount of overlap between a bounding box and the rest of the shrubbery bounding boxes, reducing the density so that visibility does not darken too much where bounding boxes overlap. This generates a smoother and more pleasing AO result on the terrain. While related ideas have been used to aggregate foliage [Lacewell et al. 2008], using OptiX we do not have fine grained access to the BVH. Since we will be integrating the results over surface meshes a highly accurate solution is not necessary.

The trees are treated as semi-transparent objects, where each intersection attenuates visibility by 50%, and are in the same BVH as the shrubs. Both simply attenuate the current AO value for a ray. It is interesting to relate this form of attenuation to the similar exponential attenuation involved in Ambient Obscurance. While AO increases the visibility the longer a ray moves through free space, with shrubs we are modeling occlusion as a participating media, decreasing visibility the longer the intersection interval.

2.2 Decorator Objects

The above described pipeline works well for buildings and terrain. Unfortunately, storing per-vertex AO for all instances of the decorator objects would require too much memory. Therefore, in this section we propose a more compact representation especially designed for instanced geometry.

There are two classes of decorator objects: large ones (trees, instanced objects outside of the playable area, ...), and small ones (grass clumps, bushes, ...). The geometry of the trees often have poor normals, complex interpenetrations, and issues with back faces since they are rendered using billboard textures. We sampled the regions around all of the trees volumetrically with a fixed target sampling density and used tri-linear interpolation to generate AO at the vertices. A two level sparse volume texture was created which guarantees that any sample inside any tree instance bounding box would have all 8 samples used in tri-linear interpolation present. Each valid voxel has a spherical visibility function that reprents AO at two distances using quadratic spherical harmonics (SH). Spherical samples are baked in a similar way as surface samples. This volume data is only used by the tool to reconstruct values at vertices and very compact models (4 floats) that are used in the game.

For every individual tree model, we compute a "local AO" signal by simply baking the tree in isolation, i.e., the local AO only takes into account the object itself. We then complement this per-model signal with per-instance AO representation, the "global AO", which takes into account the AO contribution due to the placement of this instance in the scene. The global AO component needs to be compact; we use a 3D linear function (four scalar coefficients). These per-instance coefficients are determined by a least squares fit of the residual between the actual AO signal and local AO times the function. This composition works well, because local AO captures details due to the object's geometry and global AO captures the gross effect of the surrounding scene geometry, see Figure 2. Specifically, the parameters $\mathbf{y}$ of the global AO function $G_{\mathbf{y}}$ are computed by solving the following optimization problem for each object instance:

$$\min_{\mathbf{y}} \sum_i (f(\mathbf{s}_i) - G_{\mathbf{y}}(\mathbf{s}_i) L(\mathbf{T}_j \mathbf{s}_i))^2 \tag{3}$$

where $\mathbf{s}_i$ are the world-space surface samples on the specific object instance, $f(\mathbf{s}_i)$ is the gold standard AO signal, $G_{\mathbf{y}}$ is the linear function of global AO we are solving for, L is the previously baked per-object local AO, and $\mathbf{T}_j$ is a matrix which converts the coordinates of sample $\mathbf{s}_i$ from the world space to the object space. This is a linear least squares problem with four unknowns.

2.2.1 Resampling

For very small decorators, like clumps of glass, a single scalar is sufficient. In that case, we just want to match the terrain near the

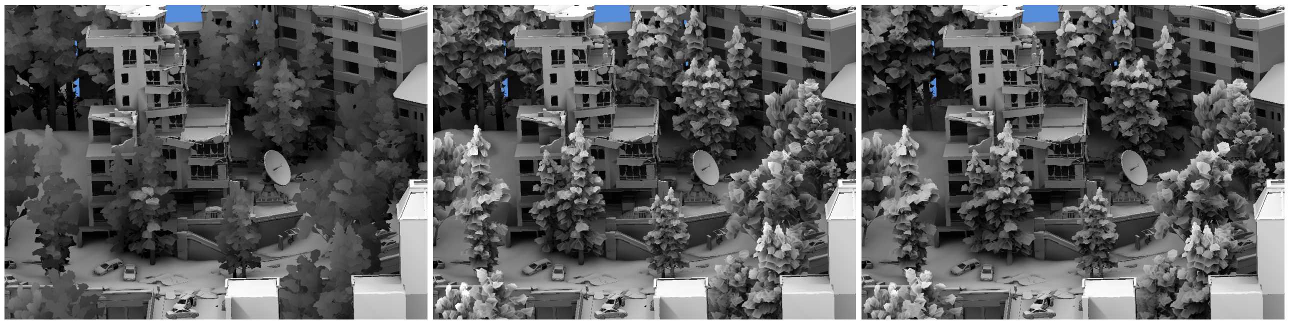

Figure 2: *An example of AO with instanced trees. Linear global AO provides a gross approximation (left); our model provides higher quality by incorporating detailed per-object local AO signal (middle). Compare with gold standard AO (right). ©Bungie*

decorator. This resampling is done by sampling the AO on the geometry within a small sphere around the point by shooting rays from the boundary of the sphere toward its center. These values are weighted by the distance to the boundary. The same machinery is used to resample the AO onto lower LOD models of the scene where resampling over the surfaces and doing a least squares fit produces better results compared to just resampling at the vertices.

2.3 Dynamic Objects

Our game utilizes a visibility system [Umbra 2013] which provides us with a set of coarse visibility regions, represented as axis-aligned boxes and a graph representing their connectivity. The game engine uses this visibility system for run-time culling e.g. during rendering, path planning, and AI processing. We take advantage of the visibility regions and store extra information in them to model AO for dynamic objects. These regions are conservative and often overlap; naive sampling inside of them causes artifacts due to visibility bleeding. However the system is capable of determining if a point inside the region is actually considered visible (i.e., not on the other side of a diagonal wall, or buried inside a wall.) Before the bake is launched we compute a one bit flag at $8 \times 8 \times 8$ samples in the region where the bit is set if the point is valid. We sample spherical obscurance functions at each of the valid points, convolving with a clamped cosine function so that we can generate AO for any input normal. This would be an impractically large amount of data to use at run-time and therefore, we fit a more compact model per region. We start be eliminating any points that see too many back faces (they are typically inside of an object and would bias the fit) and then solve for a simple model per-region consisting of a center point $\mathbf{c}$ and 6 directional derivatives, two in each of the x/y/z axis relative to the center point.

Mathematically, the model can be expressed as:

$$AO_{\text{dynamic}}(\mathbf{p}) = \mathbf{v}_{\text{mean}} + \sum_{i=0}^{2}(\mathbf{p}-\mathbf{c})_i \mathbf{v}_{2i+s_i} \tag{4}$$

Where $\mathbf{p} \in \mathbf{R}^3$ is a query point in world space, $\mathbf{v}_{\text{mean}}$ is the constant term, $\mathbf{c}$ is the center point of the model, $\mathbf{v}_j$ are the 6 directional derivatives (ordered $+x, -x, +y, -y, +z, -z$) and $\mathbf{s}_i$ is 0 or 1 depending on the sign of $(\mathbf{p}-\mathbf{c})_i$, i.e., the i-th component of the 3D vector $\mathbf{p}-\mathbf{c}$. The model is non-linear because of the center point, but for a fixed center the 7 remaining parameters can be solved for using linear least squares. While we have experimented with optimizing the whole model using a non-linear solver, in practice we found it sufficient to try two candidate centers: the center of mass of the valid samples in the region and the center of the region. The system is solved once and applied for each of the 18 channels of quadratic SH at two distances. We settled on this model because we wanted a room with 6 walls to be well represented; here the center would have high visibility and AO would darken as we move towards any of the 6 walls. A model such as a single gradient does not have enough degrees of freedom to resolve this common case. The final coefficients are stored for each region. In the engine we currently use just linear SH and a single function, which trades off computation and storage in the vertex shader vs. accuracy. This turns out to work surprisingly well. For a single occluding plane, AO convolved with a cosine can be exactly represented in linear SH, which is a common case (a ground plane). Evaluating the model is a vector difference and 3 MAD instructions that are easily vectorized.

The models are sampled based on the current position of the character or other dynamic object. Without further precautions this would lead to popping artifacts when switching between models and we implement two techniques to address this. First, the models in each cluster are regularized based on how well they predict neighboring clusters using the connectivity graph. Mathematically, we seek to minimize this error function:

$$E = \sum_{i \in \text{regions}} (\mathbf{M}_i - \mathbf{M}_{\text{orig},i})^2 + \beta \sum_{j \in N(i)} w_{ij}(\mathbf{M}_i(\mathbf{b}_{ij}) - \mathbf{M}_j(\mathbf{b}_{ij}))^2$$

where $\mathbf{M}_{\text{orig}}$ is the model fit to the region alone and $\mathbf{M}$ is the refined model, β is a weight for regularization, w_{ij} is the area of the boundary between regions i and j and $\mathbf{b}$ is a point on the center of this boundary. This is related to the regularized least squares fit used in previous work [Kavan et al. 2011]. The first sum is over all the regions and the second sum is over the neighbors of region i, denoted as $N(i)$. Instead of solving a global linear system, we simply use a variant of block coordinate descent to move towards a more balanced model of each region independently. We apply several iterations of this process. While smoothing the differences between neighboring regions, this process cannot guarantee a continuous function at the end and therefore, the second technique we use at run time is temporal blending. Specifically, the visibility model is blended with the model used in the previous frame. This smoothes any popping artifacts using exponential decay.

3 Pre-Processing and Runtime

The input vertex coefficients consists of blended long and short AO values, currently 97% using the long radius and 3% using the short radius. The values are also remapped to give the artist more control over the result, although investigating this in more detail is an area of future work. At runtime the AO results are evaluated in the vertex shader and interpolated and stored using 5 bits of the G-buffer. Dithering has to be used to make banding not be a significant problem. AO is used to modulate artistically placed ambient

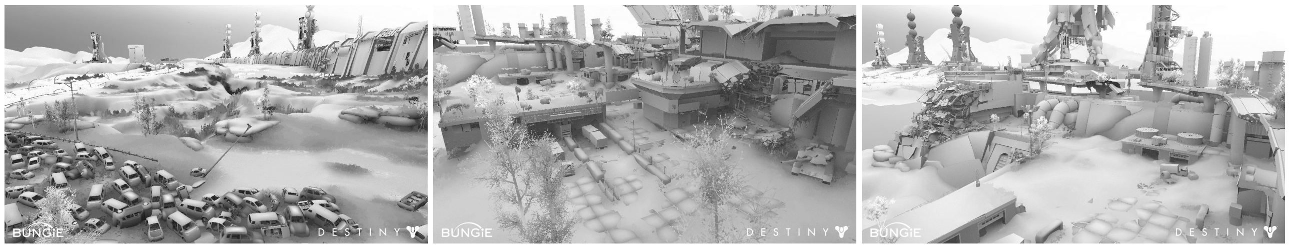

Figure 3: *Examples of Ambient Obscurance in a large game world.* ©*Bungie*

illumination. For dynamic objects the visibility system tags every object with the cluster it is in, and an index is stored that references the world space model for each cluster. The AO is computed and blended with the value used at previous frame. During the preprocess very small regions are absorbed into the most compatible neighbor, to minimize the required storage.

4 Results

The system has been deployed in a large game studio. Baking jobs are scheduled on a farm with tens of servers that each have a NVIDIA GTX680 graphics card. Backup servers with lower end GPUs can also be used. The target is for a bake to take less than 5 minutes, which is reasonable for our current scenes. The results of our technique are shown in Figure 3.

Table 1 summarizes timing breakdown for a sample level. The computations are done using OptiX [Parker et al. 2010]; we found that several optimizations to naive gathering at each sample point can significantly increase performance. Re-grouping rays based on compact cones of 32 rays and gathering across a warp followed by a reduction pass instead of at each gather point significantly increases the GPU tracing performance (taking 46-73% of the time when not doing these optimizations depending on the stage). We trace 256 directions, importance sampled over the hemisphere for surface points, and 1024 for volume points. Just under 65% of the time is spent ray tracing.

Description	Time (s)
Process geometry, 1.5M V 1.8M F	2
Samples Mesh, 6.7 M samples	< 1
Flatten 46 trees, 7856 "fog" proxies for shrubs	< 1
OptiX BVH build	10.5
Ray trace samples	35.1
Build sparse volume for "trees", ray trace 11829 pts	0.53
Ray trace region for dynamic AO 332741 pts	9.2
Least squares solve	8.6
Resample LOD geometry 5.7 M pts	5.7
Least squares solve LOD 0.65M V 0.8M F	4.2
Total	78

Table 1: *Timings for scene geometry with 1.5 million vertices and 1.8 million faces, running in an NVIDIA K20 GPU.*

5 Conclusion

We presented a system for baking Ambient Obscurance on large game levels. A previous solution for static geometry has been extended to handle large scale scenes; a compact approximation has been derived for heavily instanced objects such as trees, and an approximation of visibility in volumetric regions has been presented in order to address dynamic objects and characters. The system leverages GPUs for ray tracing and has been deployed at a large game studio. Several variations of AO were initially tried, along with alternative models for instanced objects and dynamic characters. The runtime performance is essentially free and the results are complemented well with traditional SSAO to model finer scale details if one can afford the runtime overhead of those techniques, typically a couple of milliseconds. Using OptiX, a ray tracing domain specific language, allowed for rapid exploration of possible solutions and proved to be extremely valuable.

For future work, the instancing model could be improved by optimizing for the per-object local AO using alternating least squares (instead of just computed local AO). It would also be possible to explore a more refined representation of the volumetric model used for dynamic AO, including optimization of the center point.

Acknowledgements

Keith Morely and Dave McAllister on the OptiX team at NVIDIA helped considerably. We thank the anonymous reviewers for their feedback.

References

KAVAN, L., BARGTEIL, A. W., AND SLOAN, P.-P. 2011. Least squares vertex baking. *Comput. Graph. Forum 30*, 4.

LACEWELL, D., BURLEY, B., BOULOS, S., AND SHIRLEY, P. 2008. Raytracing prefiltered occlusion for aggregate geometry. In *IEEE Symposium on Interactive Raytracing 2008*.

LANDIS, H., 2002. Global illumination in production. ACM SIGGRAPH 2002 Course #16 Notes, July.

LEHTINEN, J. 2007. A framework for precomputed and captured light transport. *ACM Trans. Graph. 26*, 4.

LOOS, B. J., AND SLOAN, P.-P. 2010. Volumetric obscurance. In *Symposium on Interactive 3D Graphics and Games*.

PANTALEONI, J., FASCIONE, L., HILL, M., AND AILA, T. 2010. Pantaray: fast ray-traced occlusion caching of massive scenes. *ACM Trans. Graph. 29*, 4 (July), 37:1–37:10.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. Optix: A general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4.

RITSCHEL, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The state of the art in interactive global illumination. *Comput. Graph. Forum 31*, 1 (Feb.), 160–188.

UMBRA, 2013. http://www.umbrasoftware.com/.

ZHUKOV, S., INOES, A., AND KRONIN, G. 1998. An ambient light illumination model. In *Rendering Techniques*.