Hybrid Ambient Occlusion

Christoph K. Reinbothe* Tamy Boubekeur+ Marc Alexa*

*TU Berlin
+Telecom ParisTech & *LTCI* CNRS

Abstract

Ambient occlusion captures a subset of global illumination effects, by computing for each point of the surface the amount of incoming light from all directions and considering potential occlusion by neighboring geometry. We introduce an approach to ambient occlusion combining object and image space techniques in a deferred shading context. It is composed of three key steps: an on-the-fly voxelization of the scene, an occlusion sampling based on this voxelization and a bilateral filtering of this sampling in screen space. The result are smoothly varying ambient terms in occluded areas at interactive frame rates without any precomputation. In particular, all computations are performed dynamically on the GPU while eliminating the problem of screen-space methods, namely ignoring geometry that is not rasterized into the Z-buffer.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Color, shading, shadowing, and texture;

1. Introduction

The human visual system is not sensitive to small errors in low frequency light variation. As a consequence, visually convincing approximations of global illumination can be achieved with a rather aggressive simplification of the light transport. In particular, indirect lighting can be approximated by sampling the geometry in the vicinity of a point, assuming that this geometry blocks the incoming light. This technique is called *ambient occlusion* and is by now a feature in any modern realtime 3D engine.

The basic idea of ambient occlusion goes back to *accessibility shading* [Mil94]: surfaces points are classified based on the radius of empty tangent spheres – larger empty spheres hint at more light reaching the surface at this point. Later, Zhukov et al. [ZIK98] introduced the notion of *obscurance* to equip local illumination models with a globally defined ambient term. Finally, Christensen [Chr03] introduced the technique of obscurance in a production environment (Pixar's RenderMan) under the (more common) name of *ambient occlusion*. If the geometry of the scene is static, ambient occlusion can be precomputed. We refer the reader to the recent survey by Méndez-Feliu and Sbert [MFS09] for a more complete overview of ambient occlusion history.

In this paper we consider the case of dynamic scene geometry, for which the computation has to be done in real time for each frame. So far, real time performance has been achieved by computing the ambient occlusion on the GPU in screen space (see Section 3 for more details and references). This limits the effects of ambient occlusion to the geometry that is actually rendered into the Z-buffer, ignoring parts that are either in the view frustum but occluded by other objects or outside the view frustum (e.g. behind the image plane).

We propose to solve this problem by performing a 3D rasterization of a region of interest (i.e. where ambient occlusion cannot be computed statically), independently of the current Z-buffer. This allows potential occluders located outside the field of view or behind a front object to be considered. As this approach combines both screen space and object space computation, we call our technique Hybrid Ambient Occlusion (HAO).

HAO is able to reproduce visually convincing ambient occlusion shadowing effects:

- in real time;
- for fully dynamic scenes, without restrictions on the animation and deformation undergone by the geometry and without any CPU precomputation;
- considering all nearby objects, even if they are located outside the (rasterized) field of view.

HAO is implemented via a multi-pass algorithm running on the GPU. It produces an AO map, which can be further combined and modulated within a deferred shading rendering engine. This AO map is generated in three steps:

1. a *3D rasterization* of the scene geometry is performed in the region of interest with the *single-pass voxelization* of Eisemann and Décoret [ED06] (see Section 5);
2. for each pixel of the screen, the ambient occlusion is estimated in **object space** using Monte-Carlo sampling where occlusion in a given direction is defined as a ray intersection with a non-empty cell of the 3D rasterization (see Section 6);
3. the resulting AO values are filtered in **screen space** using a separable approximation of the *bilateral filtering* on the AO map (see Section 7).

As a result, we obtain smoothly varying low frequency shadows at interactive rates for complex dynamic scenes on standard hardware (see Section 8).

2. Background

Ambient Occlusion (AO) captures the shadows created by occluded indirect light coming equally from all directions ("ambient" light) and is a special case of the *obscurance* technique [ZIK98]. It is usually approximated on a surface by only taking local geometry into account: the more geometry is within a certain distance to a point on the surface, the darker (less accessible) the shading at this point will be. More formally, considering a point $\mathbf{p}$ of the surface, equipped with a normal vector $\mathbf{n}$, the ambient occlusion $\mathscr{A}$ is defined as follows:

$$\mathscr{A}(\mathbf{p},\mathbf{n}) = \frac{1}{\pi}\int_{\omega\in\Omega} V(\mathbf{p},\omega)\langle\omega,\mathbf{n}\rangle d\omega \qquad (1)$$

where Ω is the hemisphere with center $\mathbf{p}$ and orientation $\mathbf{n}$, $V(\mathbf{p},\omega)$ is the *visibility* term defined to be one if geometry exists in direction ω and zero otherwise, and $\langle . \rangle$ is the dot product.

This basic definition is usually enhanced with a weighting kernel w to restrict the analysis to the geometry located within a maximum distance d_{max} and progressively decrease the contribution of occluders within this distance:

$$\mathscr{A}(\mathbf{p},\mathbf{n}) = \frac{1}{\pi}\int_{\omega\in\Omega} w(V(\mathbf{p},\omega),d_{max})\langle\omega,\mathbf{n}\rangle d\omega \qquad (2)$$

The kernel should reproduce its input at $\mathbf{p}$, and smoothly and monotonically decay to zero at d_{max}. Wendland [Wen95] has derived piecewise polynomial functions that have minimal degree and the desired compact support:

$$w(t,h) = \begin{cases} (1-\frac{t}{h})^4(\frac{4t}{h}+1) & \text{if } 0 \le t \le h \\ 0 & \text{if } t > h \end{cases} \qquad (3)$$

We use this kernel several times in our approach.

3. Related Work

Several techniques have been introduced over the last 5 years to approximate AO in real-time. These methods range from purely static to purely dynamic ones. Pharr and Green [PG04] focus on a static approach by precomputing the ambient occlusion value for a scene and storing the results in textures or as a vertex component on a per vertex base. This offline rendering pass limits this technique to static scenes.

Mendez-Feliu et al. [MFSC03] precalculate the ambient occlusion term on a per-patch basis and perform updates only for regions with moving objects. However a large number of patches are needed to produce high quality shadows.

In order to avoid the precomputation of ambient occlusion terms, Bunnell [Bun05] propose to transform meshes into surface discs (surfels) of different sizes, covering the original surfaces. Rather than computing visibility information between points on the mesh, they approximate the shadowing between these discs to determine ambient occlusion. However, highly tessellated objects are needed to get high quality shadows as visibility is estimated per-vertex only. Hoberock and Jia [HJ07] extend this algorithm to work on a per fragment basis.

Kontkanen and Laine [KL05] presented a technique for computing inter-object ambient occlusion. For each occluding object, they define an *ambient occlusion field* in the surrounding space which encodes an approximation of the occlusion caused by the object. This information is then used for defining shadow casting between objects in real time. This technique works particularly well for rigid transformation of objects, but is limited when arbitrary deformations are mandatory.

Kontkanen and Aila [KA06] address the special case of character animation, where the animation parameters can be used to control the AO values parametrically on the surface. This technique is very efficient at producing shadows on legs and arms, but cannot account for the neighboring geometry.

Shanmugam and Arikan [SA07] split up the computation of ambient occlusion into two parts: a high-frequency part, evaluated on near objects using an image space approach, and a low frequency part, approximated on distant objects with spherical occluders.

Finally, the Screen-Space Ambient Occlusion (SSAO) [Mit07, BS08] technique made full dynamic AO available to realtime applications by considering the Z-Buffer as a geometric guess of the scene and tracing rays on a per-pixel basis to evaluate the AO. This technique represents the state-of-the-art in realtime AO, but has a strong limitation: being performed entirely in screen space, it ignores any object located outside the field of view – yet these objects may have a significant influence on the ambient occlusion residing on visible objects. As a result,

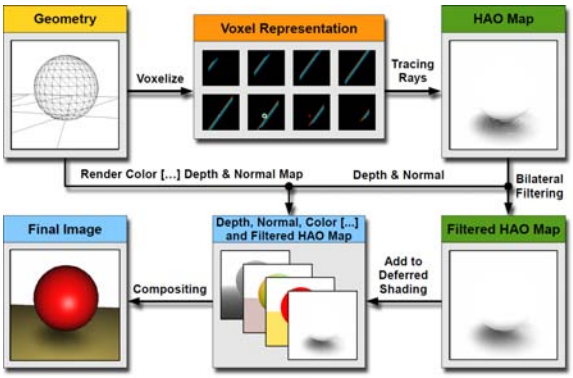


Figure 1: *Overview of the HAO work flow.*

the ambient occlusion term at a given visible point might change with the point of view.

4. Overview

Ambient occlusion is a low frequency effect and it is assumed that the imprecise shapes of shadow areas are visually insignificant. We exploit this property using two simplifications. First, the exact geometry of the scene is not important: we capture the scene in a 3D proxy (a regular grid) by performing a **3D rasterization**, acting as a *geometric* guess for space occupation and for fast ray intersection. Second, the occlusion sampling (on the Gauss sphere) can be quite sparse if we filter it to retain the necessary visual smoothness: we use a small number of rays for each pixel and smooth the result with a fast approximation of a **bilateral filter** (BL-filter) on the AO map. This filter is *geometry aware* and benefits from *deferred shading* by reusing the depth map computed prior to appearance maps, such as the normal map, the color map and, in our case, the AO map.

Basically, the rendering work flow is entirely supported by the GPU and can be summarized as follows (illustrated on Figure 1):

1. Generate a 3D proxy by rasterizing the region of interest into an occupancy grid.
2. Render color, normal, [...] and Z buffers.
3. Generate the AO map by tracing rays: sample the hemisphere for each pixel, using the 3D proxy to compute intersections.
4. Apply a BL-filter on the AO map using normal and depth information as additional range spaces.
5. Combine color, normal, [...] and HAO values on the output screen buffer.

Note that since static regions can embed precomputed AO maps, the HAO is performed only where *dynamic* ambient occlusion is required, within an arbitrary spatial domain specified by a simple bounding volume (which can be dynamic, too).

5. 3D Rasterization

In order to create a voxel representation of the geometry, we follow the idea of Eisemann and Décoret [ED06]: an orthogonal camera defines a bounding box (near/far plane) around an object and the bit set is used for coding the color of each pixel (x,y) as a binary table indicating the presence of geometry along the z direction.

In particular, we divide the bounding box of the geometry into slices; each slice represents a single layer within a 3D texture, holding one to four 32bits unsigned integer values per pixel. A voxel cell is then described by a single bit, stating the presence of geometry in its associated sub-volume. Using up to 8 output buffers simultaneously, leads to a maximum 3D resolution ranging from 256^3 to 1024^3 according to the pixel format.

The grid is created in a single pass by rendering the geometry located inside the bounding box from an orthographic *voxelizing* camera. At this stage, every unnecessary operation (e.g. texturing, depth-test) is disabled. Every newly rasterized fragment is logically combined with existing ones in the framebuffer by an OR-operator (see Figure 2).

One drawback here is the case of polygons which are aligned (up to numerical precision) with the viewing direction of the voxelizing camera. After rasterization, such polygons have a framebuffer footprint size of zero and thus are not stored into the 3D texture. Ideally, a second voxelizing camera should be set up to rasterize the volume of the scene from another local frame and correct the 3D texture information. However, this additional voxelization doubles a large part of the HAO cost (voxelization and sampling).

Actually, we observed that in typical real-time scenes, this artifact occurs only because geometry was designed "axis-aligned" (standard axis-aligned views in 3D modeling packages). We therefore rotate the voxelizing camera by a random angle prior to the voxelization process. In practice, we never observed any artifact with this simple setup.

Figure 2: *3D rasterization of the Stanford Dragon at three different resolutions.*

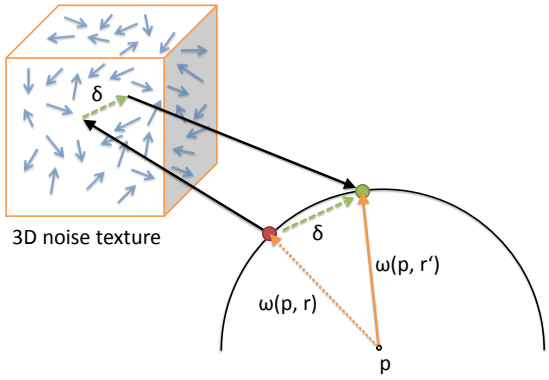


Figure 3: *By sampling jittering values δ from a 3D noise texture and applying them on a ray $\omega(p,r)$, the ray is shifted to its new direction $\omega(p,r')$.*

Figure 4: *Influence of the ray spread angle. From left to right:* 0.25*,* 0.5 *and* 0.75*.*

6. Ambient Occlusion Sampling

The AO map is generated by Monte-Carlo integration over the hemisphere of each pixel to evaluate Equation 2.

$$A(\mathbf{p},\mathbf{n}) = \frac{1}{n} \sum_{\omega_i \in \{\omega_1,\omega_n\}} w(V(\mathbf{p},\omega_i), d_{max}) \langle \omega_i, \mathbf{n} \rangle \quad (4)$$

We distribute rays $\{\omega_1,\omega_n\}$ over the hemisphere using a regular uniform sampling pattern, dynamically *jittered* with a space-dependent noise function (illustrated on Figure 3). This noise function is encoded as a small white noise 3D texture tilling the entire space and evaluated according to the position and normal at each pixel. The object space nature of this jittering procedure avoids flickering when the camera moves.

The number of samples (rays) and the *spread* (i.e. open angle within which the rays are distributed, see Figure 4) are user-defined parameters and greatly influence contrast, smoothness and frame rate.

As an optimization, we exploit the fact, that the AO map will be filtered later on, to reduce the number of steps along a ray. Instead of marching the 3D proxy and testing each cell along a ray to find an intersection, we use a fixed and typically small number of steps to test the grid occupancy uniformly along the ray. This provides two benefits: first less tests are performed (8 in most of our tests), and second, we do not need branching and perform exactly the same number of intersection tests for each ray, which is better with the current GPU programming model. Obviously, this simplification produces more noise in the AO map as some rays passing near the corner of non empty cells may miss them. Nevertheless, it greatly speeds up the ambient occlusion sampling and still produces visually smooth results once the filtering is performed.

If a filled voxel cell is found, the (actually binary) result is weighted by distance and angle between the pixel's normal and the traced ray. This produces progressive penumbra. Figure 5 shows the influence of the number of rays on visual quality, with and without filtering.

7. Bilateral AO Map Filtering

The AO sampling is the bottleneck of the entire process: it depends on the resolution of the AO map and the number of rays emitted for each pixel. As an optimization strategy, we generate a low resolution AO map (typically four times smaller than the actual screen resolution), and use a *joint bilateral upsampling* approach inspired by Kopf et al. [KCLU07] to both resample the HAO map at screen resolution as well as filtering the noise present in the initial sampling. This approach is based on bilateral filtering – for more background see the course by Paris et al. [PKTD08].

We extend the approach of Kopf et al. [KCLU07] by defining the bilateral kernel over:

- **domain**: a $m \times m$ window in image space, weighted by a piecewise polynomial function;
- **range**: a combination of kernels over distance in object space, angle between normals and AO values.

Note that m controls the smoothing effect of this filtering over the sparsely sampled AO, while defining the range space according to object space entities avoids AO *bleeding* between distant objects projected closely in image space.

Figure 5: *Influence of the number of rays. From left to right: 16, 32, 64 and 128 rays per pixel without (top) and with (bottom) filtering.*

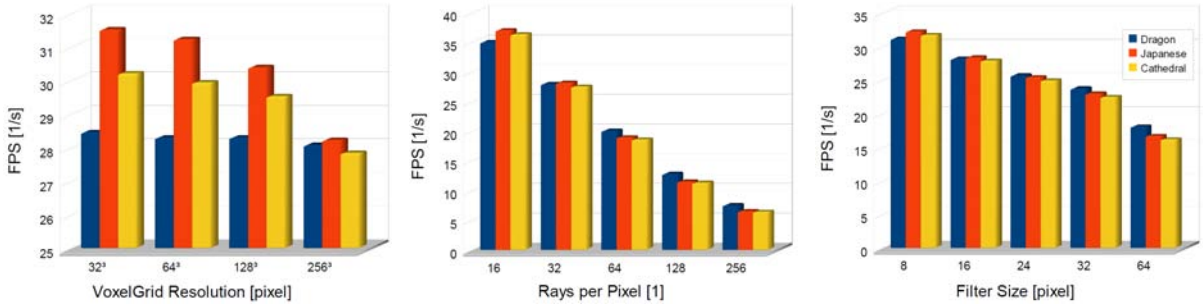


Figure 6: *Influence on performance by the three main parameters: voxelization resolution, number of rays per pixel and support size for BL filtering.*

More formally, let $\mathbf{x}$ be a pixel in the final full resolution buffer, then we define its AO as:

$$h^{m\times m}(\mathbf{x}) = \frac{\Sigma_{\mathbf{y}\in Q} w_d(\mathbf{x},\mathbf{y}) w_r(\mathbf{x},\mathbf{y}) h'(\mathbf{y}/k)}{\Sigma_{\mathbf{y}\in Q} w_d(\mathbf{x},\mathbf{y}) w_r(\mathbf{x},\mathbf{y})} \quad (5)$$

with Q denoting the set of pixels contained in the square of $m \times m$ pixels centered on $\mathbf{x}$, w_d the weighted kernel for screen space distances:

$$w_d(\mathbf{x},\mathbf{y}) := w(||\mathbf{x}-\mathbf{y}||, m) \quad (6)$$

w_r the weighted kernel defining the range influence:

$$w_r(\mathbf{x},\mathbf{y}) := w_{\mathbf{n}}(\mathbf{x},\mathbf{y}) \cdot w_{\mathbf{p}}(\mathbf{x},\mathbf{y}) \cdot w_h(\mathbf{x},\mathbf{y}) \quad (7)$$

with

$$\begin{aligned} w_n(\mathbf{x},\mathbf{y}) &:= w(1+\langle \mathbf{n}(\mathbf{x}), \mathbf{n}(\mathbf{y})\rangle, 2) \\ w_p(\mathbf{x},\mathbf{y}) &:= w(||\mathbf{p}(\mathbf{x}) - \mathbf{p}(\mathbf{y})||, e_{max}) \\ w_h(\mathbf{x},\mathbf{y}) &:= w(||h'(\mathbf{x}/k) - h'(\mathbf{y}/k)||, 1) \end{aligned}$$

$\mathbf{p}(\mathbf{x})$ the object space position corresponding to $\mathbf{x}$ (extracted from the Z map), $\mathbf{n}(\mathbf{x})$ the normal at this location (extracted from the normal map), h' the low resolution AO map generated at the previous step and k the scale ratio between h and h'.

Although not separable, the bilateral filter can be approximated in two passes [PV05], combining two one-dimensional filters (one horizontal and one vertical), while still obtaining visually convincing results:

$$\tilde{h}^{m\times m}(\mathbf{x}) := h^{1\times m}(h^{m\times 1}(\mathbf{x})) \quad (8)$$

This again improves the frame rate significantly. Note that all filtering and up-sampling steps are performed on the GPU.

8. Implementation and Results

We implemented HAO in OpenGL using NVIDIA's Cg language. We measured the run-time performances on a desktop PC equipped with an Intel Core2Quad Q6600 running at 2.4Ghz, 4GB RAM and a NVIDIA 9600GT under Windows Vista Ultimate x64. Three different test scenes (Figure 7) were used to examine different parts of our algorithm.

Parameter	Value
Screen resolution	800×600
HAO map resolution	400×300
Ray samples per pixel	32
Steps per ray	8
Voxel grid resolution	256^3
BL filter size	16

Table 1: *Settings of the test scenes*

The *Japanese* scene consists of about 21K triangles and is mostly made of axis aligned boxes.

The *Stanford Dragon* is made of about 871K triangles. Compared to the previous one, our performance measure shows that the HAO mapping is almost independent of the scene geometry complexity. Note that the even slightly better frame rate is obtained due to the fact that the region of interest in which the algorithm runs is defined as the bounding box of the model, and thus does not cover all the pixels of the screen.

The *Cathedral* is a standard test scene of AO methods and contains about 80K triangles with geometric features at different scales.

Additionally, we captured two frames of the *Toasters* animated scene (Figure 8), demonstrating the dynamic AO produced by our method.

Frame rate measurements are reported on Figure 6. The basic parameter settings are specified in Table 1. Then, we vary a single parameter and test its influence on the frame rate. We choose to inspect the influence of the voxelization resolution, the number of rays traced per pixel and the size of the bilateral filter support.

As expected, we observe that the number of rays is the most critical parameter. This confirms our choice on a rather sparse sampling of the ambient occlusion within a lower resolution AO map, followed by a bilateral upsampling. Sur-

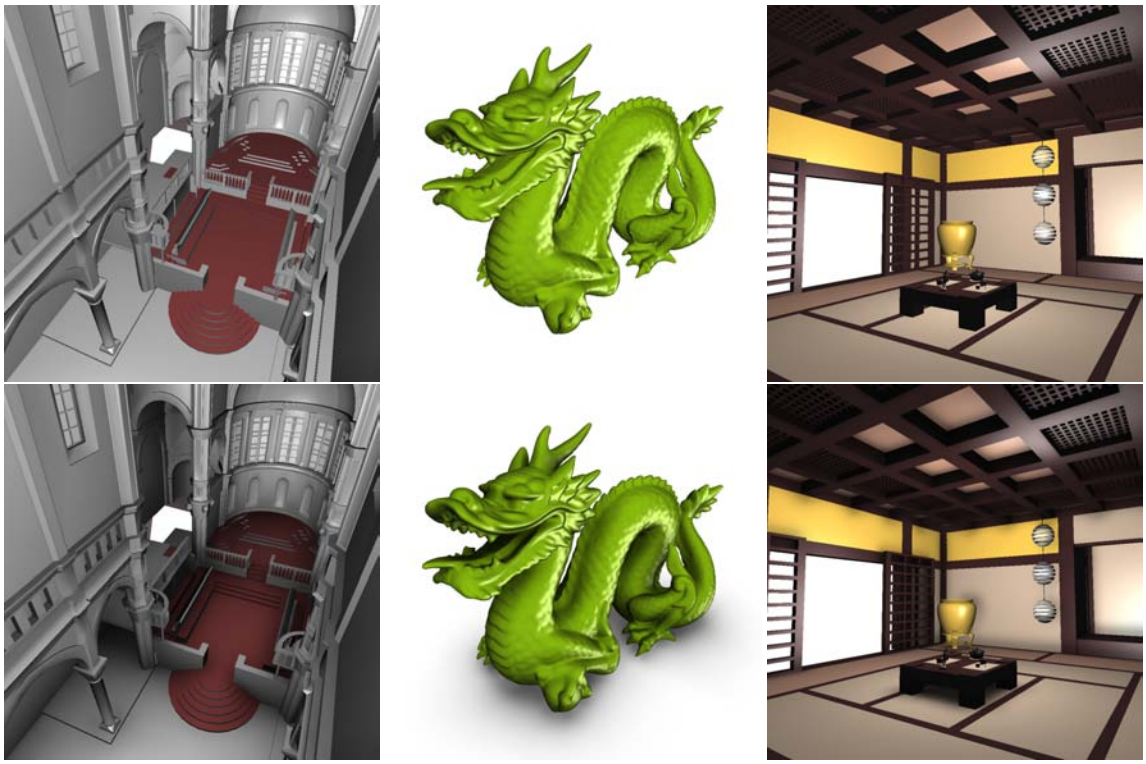


Figure 7: *Final rendering. Top: Phong shading alone. Bottom: with HAO mapping.*

prisingly, the voxelization cost is almost negligible, whatever the desired grid resolution is. We observed that higher resolution grids significantly improve the image quality only for narrow ray distribution (small *spread*), when a "sharper" AO is desired.

Last, in Figure 9, we compare HAO to SSAO. This illustrates the advantages of the hybrid nature of our approach: all objects around a location are taken into account, independently of their screen-space projection.

9. Conclusion

Hybrid Ambient Occlusion improves over Screen Space Ambient Occlusion by preserving the influence of geometry that is not rasterized into the Z-buffer but contributes to occlusions. As a consequence, shadow areas are consistently preserved during an animation, even for objects that become (un)occluded or enter the field of view.

This is achieved by a combination of 3D rasterization, local ray tracing and image space filtering. In fact, the algorithm is hybrid by two means: it combines rasterization and ray tracing as well as object space and image space computation.

A similar approach might be adopted for addressing other rendering techniques in dynamic realtime environments,

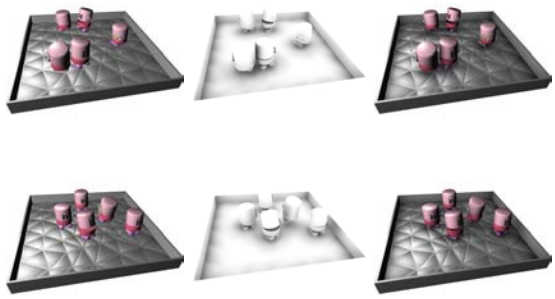
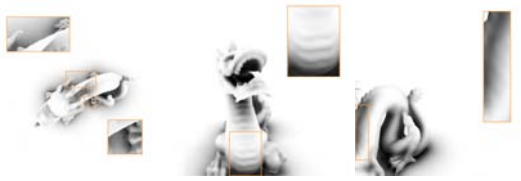


Figure 8: *Interactive animated scene. Left: Phong shading alone. Middle: dynamic HAO map. Right: final rendering.*

such as for instance *color bleeding*. More generally, 3D rasterization offers a simple and generic medium between forward and backward rendering: our future work will focus on introducing level of detail, such as *3D mipmaps*, for the generation of the 3D proxy and for computing intersections.

Acknowledgements We would like to thank Julien Hadim and Romain Pacanowski for early discussions on this work. The Dragon model is courtesy Stanford. The Toasters

(a) SSAO



(b) HAO

Figure 9: *Comparison of SSAO and HAO. Left: Showing the weakness of SSAO if geometry in front hides potential occluders in the back. Middle and Right: Occluders which are outside the viewport have no influence on visible geometry anymore.*

scene is courtesy Ingo Wald. The Sibenik cathedral scene is courtesy Marko Dabrovic.

References

[BS08] BAVOIL L., SAINZ M.: *Screen-Space Ambient Occlusion*. Tech. rep., Nvidia Corporation, August 2008.

[Bun05] BUNNELL M.: *GPU Gems 2 - Dynamic Ambient Occlusion and Indirect Lighting*. Addison-Wesley, 2005, ch. 14, pp. 223–233.

[Chr03] CHRISTENSEN P. H.: Global illumination and all that. In *Proceedings of SIGGRAPH 2003 course notes Nr. 9, RenderMan: Theory and Practice* (2003).

[ED06] EISEMANN E., DÉCORET X.: Fast scene voxelization and applications. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), ACM SIGGRAPH, pp. 71–78.

[HJ07] HOBEROCK J., JIA Y.: *GPU Gems 3 - High-Quality Ambient Occlusion*. Addison-Wesley, 2007, ch. 12, pp. 239–274.

[KA06] KONTKANEN J., AILA T.: Ambient occlusion for animated characters. In *Rendering Techniques 2006 (Eurographics Symposium on Rendering)* (jun 2006), Wolfgang Heidrich T. A.-M., (Ed.), Eurographics.

[KCLU07] KOPF J., COHEN M. F., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 96.

[KL05] KONTKANEN J., LAINE S.: Ambient occlusion fields. In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games* (2005), ACM Press, pp. 41–48.

[MFS09] MÉNDEZ-FELIU A., SBERT M.: From obscurances to ambient occlusion: A survey. *Vis. Comput. 25*, 2 (2009), 181–196.

[MFSC03] MÉNDEZ-FELIU A., SBERT M., CATÁ J.: Real-time obscurances with color bleeding. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics* (New York, NY, USA, 2003), ACM, pp. 171–176.

[Mil94] MILLER G.: Efficient algorithms for local and global accessibility shading. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 319–326.

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), ACM, pp. 97–121.

[PG04] PHARR M., GREEN S.: *GPU Gems - Ambient Occlusion*. Addison-Wesley, 2004, ch. 17, pp. 279–292.

[PKTD08] PARIS S., KORNPROBST P., TUMBLIN J., DURAND F.: A gentle introduction to bilateral filtering and its applications. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes* (New York, NY, USA, 2008), ACM, pp. 1–50.

[PV05] PHAM T. Q., VLIET L. J.: Separable bilateral filtering for fast video preprocessing. In *In IEEE Internat. Conf. on Multimedia and Expo, CD1Ű4* (2005), IEEE, pp. 1–4.

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), ACM, pp. 73–80.

[Wen95] WENDLAND H.: Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Adv. Comput. Math. 4*, 4 (1995), 389–396.

[ZIK98] ZHUKOV S., IONES A., KRONIN G.: An ambient light illumination model. In *Rendering Techniques* (1998), pp. 45–56.