Universidade do Minho

Visualização e Iluminação I
Perfil de Computação Gráfica
Mestrado em Engenharia Informática
2019/2020

Screen-Space Ambient Occlusion

Authors:	***Student Number:***
Nuno Reis	A77310
Bernardo Silva	A77230

January 30, 2020

Contents

1 Introduction

During our lessons, Ambient Light has been characterized as a fixed light constant added to the overall lighting of a scene to simulate the scattering of light, that is, to simulate how light reflects from opaque surfaces and travels with lower intensity until it no longer becomes visible... Though, as we can see by that explanation, a fixed constant doesn't quite satisfy the latter. With this in mind, it becomes obvious that indirectly lit parts of a scene should have varying intensities. One type of indirect lighting approximation is called Ambient Occlusion(AO). AO attempts to satisfy the approximation to indirect lighting by darkening creases, holes and surfaces that are close to eachother. These areas are largely occluded by surrounding geometry and thus reflect minimal amounts of light, appearing darker to our eyes.

Ambient occlusion techniques are expensive as they have to take surrounding geometry into account. As we all know, ray-tracing gracefully provides a solution to this issue yet is computationally infeasible. Instead, in 2007, Crytek published screen-space ambient occlusion(SSAO), for use in their title: Crysis. The technique uses a scene's depth in screen-space to determine the amount of occlusion instead of real geometrical data. This approach is incredibly fast compared to real ambient occlusion and gives plausible results, making it the de-facto standard for approximating real-time ambient occlusion.

2 Screen-Space Ambient Occlusion

2.1 The Basics

The basics behind SSAO are simple: for each fragment on a screen-filled quad, we calculate an occlusion factor based on the fragment's depth values. The occlusion factor is then used to reduce or nullify the fragment's ambient lighting component. This factor is the result of taking multiple depth samples in a sphere sample kernel surrounding a fragment position and then comparing each of the samples with the current depth value. The number of samples with a higher depth value represents the occlusion factor.

It is therefore clear that both quality and precision are directly related to the number of samples taken. If the sample count is too low, then we have a drastic reduction in precision and the showing of an artifact named *banding*. In contrast, if the sample count is too high, performance drops drastically. It is possible, however, to reduce the amount of samples required by introducing some randomness into the sample kernel. By randomly rotating the sample kernel for each fragment we can get high quality results with a much smaller sampling rate. This does come at a price, however. Predictably, randomness introduces a noise pattern that requires fixing via blur.

2.2 Sampling Method

The first SSAO, developed by Crytek, had a defined visual style developed specifically for the Crysis title. By using a sample kernel in the shape of a sphere, walls took a gray tint as half of the kernel samples ended up in the surrounding geometry. In Crytek's case, such a thing added a characteristic style to their game and, above all, was a non-issue due to the game's location.

For that reason, outlined in one of the reference articles, it was decided that a hemisphere sample kernel, oriented along the surface's normal vector, was the correct choice. This allows us to sample the area accurately and stops internal geometry from contributing to the occlusion factor, producing more realistic results and removing this gray-feel previously described.

2.3 Sampling Buffers

SSAO requires geometrical info as we need some way to determine the occlusion factor of a fragment. For each fragment, we're going to need the following data:

- A per-fragment position vector.
- A per-fragment normal vector.
- A per-fragment albedo[1] vector.
- A sample kernel.
- A per-fragment random rotation vector used to rotate the kernel.

[1]Albedo, from the Latin word for 'whiteness', is the measure of the diffuse reflection of light. In other terms, it is the proportion of light that is reflected from a surface.

Using a per-fragment view-space position we can orient a sample hemisphere kernel around the fragment's view-space surface normal and use this kernel to sample the position buffer texture at varying offsets. For each per-fragment kernel sample we compare its depth with its depth in the position buffer to determine the amount of occlusion. The resulting occlusion factor is then used to limit the final ambient lighting component. By also including a per-fragment rotation vector we can significantly reduce the number of samples as stated previously.

As SSAO is a screen-space technique we calculate its effect on each fragment on a screen-filled 2D quad, but this means we have no geometrical information of the scene. To solve this, we refer back to deferred rendering as the technique stores the information needed in position and normal vectors within the G-buffer.

Since SSAO is a screen-space technique where occlusion is calculated based on the visible view it makes sense to implement the algorithm in view-space. Therefore the *FragPos* as supplied by the geometry stage's vertex shader is transformed to view space. All further calculations are in view-space as well so make sure the G-buffer's positions and normals are in view-space (multiplied by the view matrix as well). Using the tools given to us by NAU, it is possible to bind a position texture to obtain depth values for each of our kernel samples.

2.4 Normal-Oriented Hemisphere

We need to generate a number of samples oriented along the normal of a surface. as it is both difficult and implausible to generate a sample kernel for each surface we're going to generate a sample kernel in tangent space, with the normal vector pointing in the positive z direction. While normally the samples would be randomly distributed along the hemisphere, we want to place an emphasis on occlusions close to the fragment and, as such, make use of an accelerating interpolation wich is exponential in growth.

Each of the kernel samples will be used to offset the view-space fragment position to sample surrounding geometry. Quite a lot of samples are needed in view-space in order to get realistic results, which in many cases may be too heavy on performance. However, using the semi-random rotation technique described above, we can reduce the amount of samples required albeit at the cost of introducing noise.

2.5 The SSAO Shader

The SSAO shader runs on a 2D screen-filled quad that calculates the occlusion value for each of the generated fragments (for use in the final lighting shader). As such, we store the result of this stage in another framebuffer. The ambient occlusion result is, predictably, a grayscale value. The SSAO shader takes as input the relevant G-buffer textures, the noise texture and the normal-oriented hemisphere kernel samples and uses them to compute the result.

Using a process called the Gramm-Schmidt process we create an orthogonal basis, each time slightly tilted based on the value of our random vector. Note that because a random vector is used for constructing the tangent vector, there is no need to have the TBN matrix exactly aligned to the geometry's surface, thus no need for per-vertex tangent (and bitangent) vectors.

Following this, we iterate over each of the kernel samples, transform the samples from tangent to view-space, add them to the current fragment position and compare the fragment position's depth with the sample depth stored in the view-space position buffer.

Note that we add a small bias here to the original fragment's depth value (set to 0.025 in the example). A bias isn't always necessary, but it helps visually tweak the SSAO effect and solves acne effects that might occur based on the scene's complexity and, as such, we thought it'd be a nice even if small addition. Furthermore, we introduce a range check to stop our shader from considering depth values of surface far behind the test surface, something that happens whenever a fragment is aligned too close to the edge of geometry.

Between the SSAO pass and the lighting pass we first want to blur the SSAO texture as to mask the noise created by our randomly rotating kernels. This proved a very easy step as our random vector texture prodives consistent randomness.

Traversing the surrounding SSAO texels between -2.0 and 2.0, we sample the SSAO texture an amount identical to the noise texture's dimensions and then offset each texture coordinate by the exact size of a single texel. Finally, averaging the obtained results, we get a simple, but effective blur.

The last step in our path to ambient occlusion is its application. Applying it to the lighting equation is actually rather easy. All that's required is the multiplication of the per-fragment occlusion factor to the ambient component of our lighting.

3 Conclusion

Screen-space ambient occlusion is a highly customizable effect that relies heavily on tweaking its parameters based on the scene. While some may call this a manner of robustness, there is a certain obtuse component to it. In many cases, SSAO needs to be hand-crafted and tweaked with finesse to match the art-style and the scene composition it's being applied in. In any case, SSAO is a subtle effect yet adds a great deal of realism to properly lighted scenes. How it compares to other types of ambient occlusion, however, is another story.

3.1 Pros and Cons of AO Types

3.1.1 Screen-Space(SSAO):

Pros:

- Independent of scene complexitiy, works only with the pixels of the final image.
- Works with dynamic scenes.
- Can easily be integrated into any modern graphics pipeline.

Cons:

- Expensive, depending directly on image resolution.
- However, the higher the image resolution, the more expensive the effect will be and the bigger the necessity to simplify the algorithm.
- It produces noise and therefore requires blurring to mask said noise which may interfere with object edges.

3.1.2 Horizon-Based(HBAO+):

HBAO uses a physically-based algorithm that approximates an integral with depth buffer sampling. In other words, the upgrade enables HBAO to generate higher-quality SSAO whilst increasing the definition, quality, and visibility of the shadowing. For performance reasons, however, it is typically rendered at half-resolution. Unfortunately, rendering at reduced resolutions inevitably causes flickering that is challenging to hide in all situations. To overcome these issues, NVIDIA has completely redeveloped and revamped Screen Space Ambient Occlusion to create HBAO+, a paradigm shift in the field of SSAO rendering.

Pros:

- Higher image quality and reduced noise.
- NVIDIA hardware acceleration.
- Speedy implementation with a far richer, more detailed image.
- More detailed shading with deeper blacks and brighter whites.

Cons:

- Increased performance hit compared to SSAO.
- Reliant on temporal accumulation which causes it to misstep in dynamic scenes.

3.1.3 Voxel-Accelerated(VXAO):

VXAO (Voxel Accelerated Ambient Occlusion) is a method that works with voxels – the equivalent of a pixel in 3D. DirectX 12 allows doing retracing and voxelizing a scene in search structures.

By doing that, a slightly more accurate method that takes into account more complex scene geometry can be obtained. However, this method is even more expensive than other methods. VXAO is part of the VXGI (Voxel global illumination) surround lighting technology, which more correctly takes into account direct and reflected light. Principle of Operation The VXAO algorithm encompasses three passes: voxelization, voxel post-processing, and cone tracing. Voxelization means that triangle-based meshes are rendered into a 3D texture.

Its performance depends on the total number of triangles, their size, and the number of draw-calls needed to render them. Post-processing involves the processes of clearing, filtering, and downsampling voxels. Its performance depends on the total number of voxels produced during voxelization. Then cone tracing is conducted in screen space. Its performance depends on the shading rate, screen resolution, and the cone-tracing pass in 1080p resolution.

Pros:

- The most accurate shading.
- Capable of adapting to dynamic scenes with remarkable efficiency.

Cons:

- Only high-end GPUs will be able to take full advantage of it.
- Depending on the implementation, VXAO can be 3-4x slower than HBAO+.
- Relies heavily on hardware acceleration provided by the latest GPU models.