INF584 - Image Synthesis - Final Project

Marie Audouard

***Abstract*—In this project, I implemented the paper [1] about Horizon-Based Ambient Occlusion.**

Introduction

The goal of this project is to implement a research paper on rendering techniques. I have chosen to work on the Horizon-Based Ambient Occlusion (HBAO) described in [1]. This method works in real time, and uses only image-space information obtained after rasterization. The result of this algorithm is an approximation of the amount of light reaching each point of the image, which gives a more realistic look to the shadows of an image (in particular, at the junction of elements or in the folds of cloth).

I. Overview of the paper

The objective of this paper is to get an ambient occlusion comparable to one obtained with ray marching, but that would be much faster. The calculations of the ambient occlusion are based on the Rendering Equation of Kajiya for the color response at point P:

$$L_0(\omega_0) = \int_0^{2\pi} \int_0^{\pi/2} L(\omega) f(\omega, \omega_o) \cos(\theta)\, d\theta d\phi$$

For ambient occlusion, we do not consider the different possible lights, but rather how much a point is hidden by its neighbours, which is what will reduce the light intensity at that point. Thus, the equation that the paper solves is the following:

$$A = 1 - \frac{1}{2\pi} \int_0^{2\pi} \int_0^{\pi/2} V(\omega) W(\omega) \cos(\theta)\, d\theta d\phi$$

with $V(\omega) = 1$ when ω intersects an occluder when starting from the point P and 0 else, and $W(\omega)$ is a linear attenuation function that represents the fact that the farther an occluder is, the less it hides P. What we do here is that we subtract from 1 (perfectly illuminated P) the integral over the occluded parts, which determines how much the point is hidden (and thus how much it is in the shadows). The key idea of the paper is that since we are working in image space, with a height field, the occluded part calculated by the algorithm is continuous. Which means that the perceived occluded part looks like this[1] :

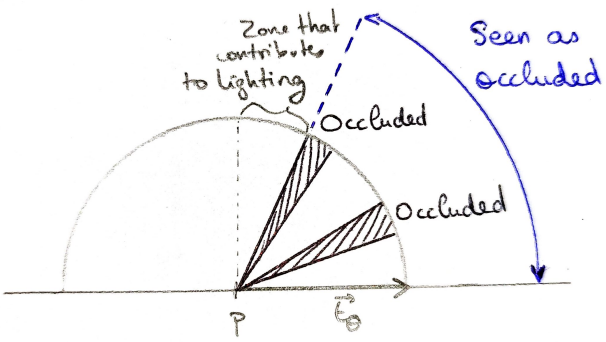

Fig. 1. Zone perceived as occluded using the height field

By defining $t(\theta)$ and $h(\theta)$ as follows, our equation

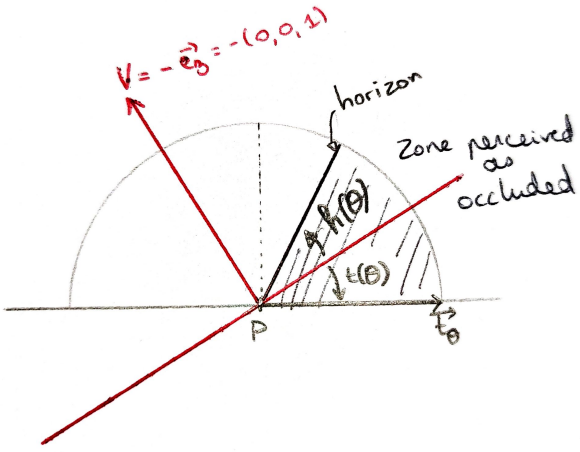

Fig. 2. Definition of $t(\theta)$ and $h(\theta)$

[1]To understand better the paper, I used the slides [2], my diagrams are inspired by them and the original paper.

for the ambient occlusion becomes:

$$A = 1 - \frac{1}{2\pi} \int_0^{2\pi} \int_{t(\theta)}^{h(\theta)} W(\omega) \cos(\theta) \, d\theta d\phi$$

The authors of the original paper have chosen for the attenuation function: $W(\omega) = W(\theta) = \max(0, 1 - r(\theta)/R)$ with $r(\theta) = \|P - HorizonPoint(\theta)\|$, and R is the radius of influence (we only consider the points that are inside this radius for occluding candidates).
Since W depends only on θ, we can integrate the equation of A over ϕ, which yields:

$$A = 1 - \frac{1}{2\pi} \int_0^{2\pi} W(\theta)(\sin(h(\theta)) - \sin(t(\theta)) \, d\theta$$

The idea of the algorithm is to approximate A using a Monte-Carlo approach, by sampling directions θ around P, and for each of these directions, sampling the depth image along the ray in direction θ between P and the influence radius R in order to find the horizon angle. For a given θ, the horizon angle is calculated by taking the maximum of the candidate angles for the horizon and the tangent, as can be seen on this diagram:

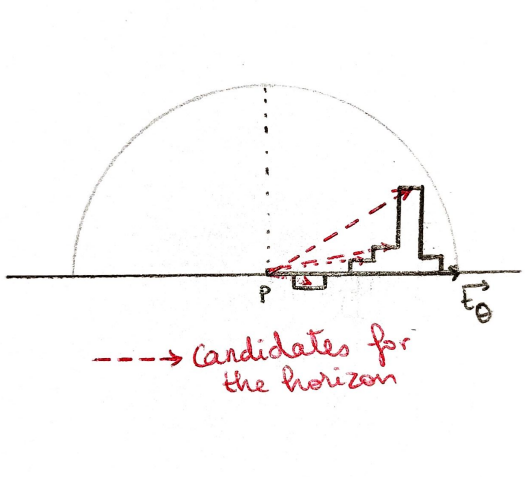

Fig. 3. Calculation of $t(\theta)$ and $h(\theta)$

I will be giving more details about the actual calculations in the implementation section.
To avoid the artifacts that can be caused by this sampling (always choosing the same directions and the same step sizes), the authors introduce some jitter. They also mention the importance of snapping the texture coordinates (of the depth and normal images) to the closest texel center to have the exact attributes corresponding to our coordinates.

II. Description of the implementation

For my implementation, I started by outlining the key steps needed:

1) Get access to the depth and normal images in my shader
   - Create a first pass that fills a frame buffer
   - Create the textures that will be filled by the first pass
   - Bind these textures to my shader
2) Implement the algorithm in a second dedicated pass of shaders. The use of a shader is of course necessary to make the calculations much faster, since they can be done in parallel because they are independent at each point.

First, I had to understand the way frame buffer works in OpenGL, and in particular how to use several draw buffers bound to different color attachments to be able to store my different variables of interest (depth and normal at first, and then color when displaying the ambient occlusion combined with the result of the rasterization). To ensure that I was properly calculating the depth and normal images, I decided to add the possibility to display them in my code.

Then, I had to implement the different operations required to calculate $t(\theta)$ and $h(\theta)$.

The first one was to get the camera space coordinates from the image space coordinates, which is calculated using the field of view and the width and height of the image and with the Z field being the depth value passed as a texture. I also had to be able to snap the coordinates to the closest texel center to ensure the matching of the attributes, as mentioned previously: the operation is just to multiply by the width (resp. height) the X (resp. Y) and round to the closest integer, then add 0.5 and divide by width (resp. height).
The most important calculation was how to measure the angles. A key insight was that the paper calls $\vec{V}$ the view direction at the start, but they actually use $\vec{V} = -\vec{e_z}$ (They mention it in the second half of the paper, and it is what makes their angle calculations correct). We are thus trying to find the angle between our vectors (tangent vectors or horizon vectors) and the plane of the image. The angle is given by the following formula (also given in the original paper): $\alpha = \arctan(-D.z/\|D.xy\|)$ with D the vector that we consider.

Then, I had to use this information to implement the calculations for one given θ. I created pseudo-random functions using hash functions to add the jitter using code from [3]. To ensure that the tangent would be along the same direction as the steps, I calculated it as follows: $\vec{t_\theta} = \vec{n} \times \begin{pmatrix} \sin(\theta) \\ -\cos(\theta) \\ 0 \end{pmatrix}$.
Then, I took the steps in image space with: $texCoords_{next} = texCoords_P + step \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}$.
An important point was to convert the radius of influence from camera space to image space, to get a correct value for the step.
Finally, I just had to assemble all of these functions to form my implementation of the paper. I decided to then add a visualization showing the result of this ambient occlusion multiplied by the rasterized colors, to see how much better the final rendering is using this algorithm. One can note that the ambient occlusion could also be added on top of the result from the ray tracer, but it would be less relevant seeing as ambient occlusion seeks to approximate results that can be directly obtained in a ray tracer at the cost of needing more calculation time and power.

The final code is available here.

III. Results

At first, I had used noise that varied in time, but it worsened the noise since it was constantly changing. I decided then to use a random function that varied only in space, which made the final result look more crisp, and the noise is a lot less noticeable when it is not constantly moving. I also think it might be better for denoising algorithms that could be applied in a later pass. Displaying the ambient occlusion also makes clear its use: it improves greatly the rendering of the shadows that are found at folds in cloth, or at intersections between objects. This is even more visible when combining the results of the ambient occlusion with the rasterization: the shadows are much deeper, and also a lot more realistic. The details are also much more defined.

Fig. 4. Rendering of the file denis.off with rasterization

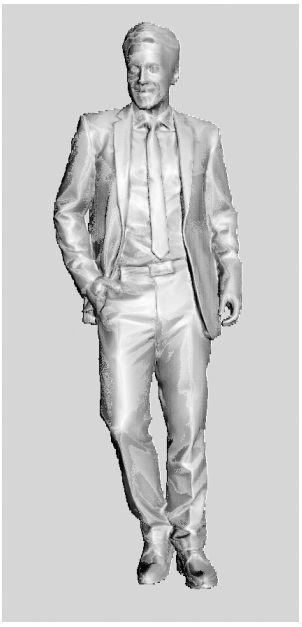

Fig. 5. Horizon-based Ambient Occlusion result for the file denis.off

The program is very fast: the ambient occlusion is calculated in less than 1 millisecond, around 750 micro seconds on average (between 250 and 1050 microseconds) for an image of 1024x768 pixels. For an image of size 1920x991 pixels, the speed is mostly the same. I note that there is a lot of variation between two calls. To give an order of comparison, the rasterization process takes a very variable time, but that is most of the time more than 1 millisecond, and usually around 15 or 30 milliseconds on my computer.

Fig. 6. Rendering of the file denis.off with rasterization and HBAO

CONCLUSION

This project enabled me to discover a lot of details about OpenGL (especially the frame buffers). I managed to implement the functionalities of the paper, and the final result is very convincing. The positive effect of the algorithm on the realism of an image is very clear, and the fact that it runs in real time is what makes the use of this algorithm so useful. A logical improvement would be to add a next pass with a denoising filter.

REFERENCES

[1] Louis Bavoil, Miguel Sainz and Rouslan Dimitrov. *Image-Space Horizon-Based Ambient Occlusion*, for *NVIDIA Corporation*, 2008, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=13bc73f19c136873cda61696aee8e90e2ce0f2d8

[2] Jorge Jiménez. *Practical Real-Time Strategies for Accurate Indirect Occlusion*, in *SIGGRAPH 2016 Course: Physically Based Shading in Theory and Practice*, 2016, https://blog.selfshadow.com/publications/s2016-shading-course/activision/s2016_pbs_activision_occlusion.pdf

[3] Mark Jarzynski and Marc Olano, *Hash Functions for GPU Rendering* in *Journal of Computer Graphics Techniques (JCGT)*, vol. 9, no. 3, 21-38, 2020. Available online http://jcgt.org/published/0009/03/02/, and an implementation is given at https://www.shadertoy.com/view/XlGcRh, accessed 24 March 2024.