# RenderBender - Screen Space Shadowing

*Advantages and Disadvantages*

**Stefan Seibert & Stefan Radicke**
HdM Stuttgart - Game Development
mail@stefanseibert.com – www.stefanseibert.com

**Abstract**

Shadowing is a time consuming part of a real time rendering pipeline. Over the past years screen space shadowing became an interesting and more adopted approach in real time rendering engines since this can be done at a very performant stage of the image creating pipeline. Goal of this project was to implement screen space shadowing within an AAA game engine and exploring advantages and disadvantages of this technique. This project was conducted within the scope of a so called ”Innovation Project” as part of the ”computer science and media” master programme at the Stuttgart Media University.

## Introduction

The most common solution to provide shadows in real time rendering image generation is through shadow mapping. More advanced software uses so called cascaded shadow maps where depending on the position of the lights and the camera in the 3d world, the resolution of the shadow maps are adapted in several levels, since objects that are far away also only need to be represented with lower detailed shadows. This is a well researched area that can generate a high quality shadowing system but also is costly. Especially close up areas with a lot of details are typically on a scale where shadow maps cannot provide enough resolution which leads to artifacts and problems. Lately screen space shadows are seen as interesting alternative since they are mostly solved within the pixel shader and are therefore faster to compute then shadow mapping techniques. The technique is further on referenced as SSS [Screen Space Shadows]. The basic idea for generating these SSS is to ray-march along an imaginary line between the questionable image position and the light source. For every position along the line that is tested, it is checked if the depth buffer value of the scene at that position is smaller (which means closer to the camera) as the questionable ray position. This means that the original position must be in shadow since there is an occluding object between the light source and the position.

## Content of the project

1. Gaining knowledge about the different approaches for SSS
2. Reimplementing these approaches within an existing software base
3. Gaining knowledge of advantages of this algorithm
4. Gaining knowledge of dsadvantages of this algorithm
5. Getting to know the different parameters that influence the algorithms behavior
6. Gaining knowledge about different ways to improve the results

## Used technology for the project

The algorithm research was performed within the software ”Autodesk Stingray” which is an real time rendering application that supports the most modern computer graphic interfaces like DirectX 12. The application has a interface to provide it with custom C++ or shader code which is written in a HLSL based language set, enriched by some custom overloads that ease the development. The algorithm is purely shader based and can be hot reloaded and unloaded which is perfect for a testbed and to see direct results of changing parameters.

### Pseudocode Implementation

The simplified, basic idea of the SSS algorithm is as following:

```
var ray = lightViewPos - pixelViewPos
var oneStep = ray / amountOfIterations
var viewPosition = pixelViewPos
for(int i = 0; i < amountOfIterations; i++) {
        viewPosition = viewPosition + oneStep
    ssPosition = view_to_ss(viewPosition)
    depthValue = depth_buffer(ssPosition.xy)

    if (depthValue < ssPosition.z)
        return true // hit
}
return false // no hit
```

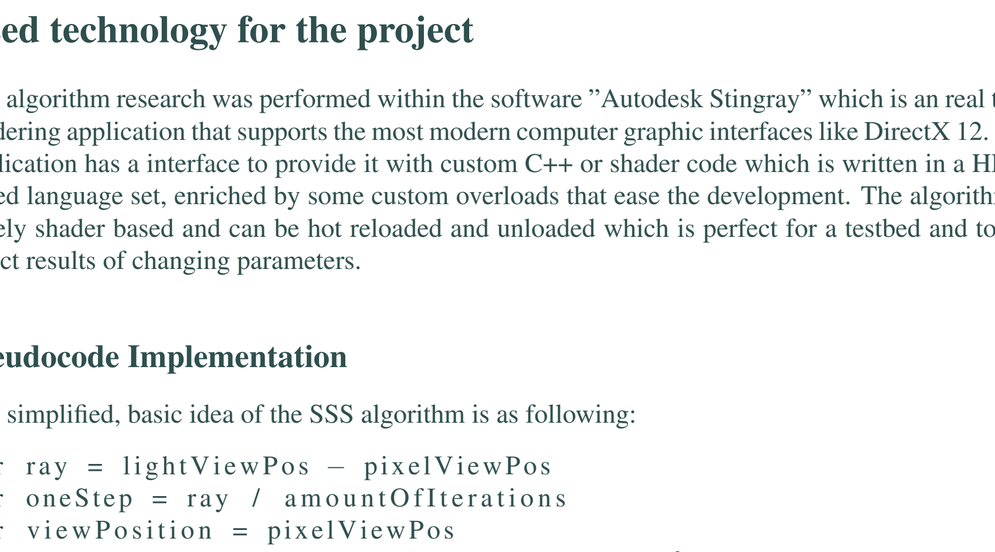

**Figure 1:** Screen Space Shadows

**Figure 2:** Original Picture

**Figure 3:** Picture with added SSS

**Figure 4:** Added depth visualization in a scene through SSS

## Conclusions

- SSS generates a lot of errors in its default algorithm idea
- A lot of this errors can be fought by iterating multiple times, tweaking parameters etc.
- Some problems are implied by the nature of the algorithm but there are proposed solutions
- A final good looking implementation is very hard but can also be very rewarding

## Forthcoming Research

Several published research articles can be found how to improve the algorithm, which could be used for further studies. An finishing report will point out these ideas next to the other findings of this project.

## References

[1] Kapralos Hogue Cowan, Khattak. Screen space point sampled shadows. 2015.

[2] Nikolas Kasyan. Playing with realtime shadows. 2013.

[3] Seidel Ritschel, Grosch. Approximating dynamic global illumination in image space. 2009.

## Acknowledgements

Thanks to Prof. Stefan Radicke for supervising the project and Tobias Persson, Jim Sagevid for giving advice on the render pipeline implementation of the Stingray Engine. Also thanks to the whole Team of Autodesk Stingray for providing learning material and tips.