Project in EDAN35 High Performance Computer Graphics

Cem Daniel Marcus

Lund University
Sweden

Abstract

In this paper, we describe the implementation of a few different shaders: Fur, depth of field, screen space ambient occlusion (from here on SSAO) and bloom. The algorithms of our implementations will be described. We will also present the results of our solutions and discuss around how they can be improved.

1 Introduction

We got the inspiration for the project from the lectures. Then we dug a little deeper by watching tech-demos and browsing through nVidias GPU Gems. We found so many interesting shaders that we wanted to do that we realised that there would be no time to implement a full game. Besides, we already made a game in the previous course. We worked with the Sponza scene from lab 2 because we felt that it was an interesting enough scene in itself. It is also good for displaying the most of the effects we wanted to implement.

2 Algorithms

2.1 Bloom

Bloom is an effect that appears in real cameras where light from very bright areas bleed over to other areas that are not lit up as much. We have recreated this effect by giving extra light to pixels neighboring areas with high light intensity. The algorithm starts by sampling pixels in an area around the current pixel. For each of those samples with intensity higher than a threshold we add extra intensity to the current pixel. We weight the light from the outermost samples higher because it gave the best result. [Policarpo et al. 2005]

2.2 Depth of field

Depth of field is an effect that is created both in our eyes and in a camera. Its an effect that appears naturally to us as it helps give us a better sense of depth and makes it easier to focus on certain areas. In movies and photography this effect is also used to point the viewer's attention at a certain place. This also occurs naturally as the camera in this sense works in the same way as the eye with a lens focusing the light. For computer graphics, this sense of depth does not exist as we have a flat screen and the images come from a virtual world. The implementation of this effect is made in our post process shader. The implementation is an approximation on how the real eye works. When a point is out of focus, instead of being projected onto one single point, it will instead be projected to several points on the screen, creating a blur at areas that are out of focus. To decide how big an area that is to be blurred, we calculate the distance between our adjustable focal plane and the depth in our depth buffer as

$$abs(FocusPoint - depth_{cp})$$

where cp is the current pixel in the fragment shader. This value determines how big area we will blur the point with. Then the pixel is blurred by looking at nearby pixels quadratically. For each of those pixels that have less depth than the current pixel we mix it with the current pixel as

$$mix(colorBuffer_{cp}, colorBuffer_{np}, abs(FocusPoint - depth_{np}))$$

where np is the nearby pixel that we are currently blending with, and colorBuffer is the already rendered scene. Here we use the surrounding pixels when we mix to better approximate that those pixels blur out over the current pixel. [Derners 2010]

2.3 SSAO

In real time computer graphics light must be approximately calculated. One of the terms often used for this approximation is the ambient term, this term is just a small constant of light that is added to represent the small amount of light scattered to areas that are not directly lit by light sources. This is a very flat approximation of those areas. The screen space ambient occlusion is a better algorithm that takes into account how the close surrounding of the point looks. This is done by taking random sampling points in a sphere around the current pixel in the fragment shader. For each of those sampling points we compare the depth of the random point with the corresponding depth from the depth buffer. If the value in the depth buffer is lower than the depth in the sampling point, it means that the point is hidden behind an object. In this case we make the ambient factor lower to represent that this area is more hidden and less light scatters to it. [Doggett and Akenine-Mller 2010]

2.4 Fur

The fur effect as originally described by Lengyel et al uses both shells and fins to display good looking fur in real time over arbitrary surfaces, but only the shells were implemented here. The method of shells works by discretizing the hair at different lengths of the strands, and then drawing all of these shells on top of each other. This gives a great effect on surfaces that are parallel to the image plane, but surfaces that are facing away usually look very bad due to most of the shells being invisible here. This problem could be alleviated by rendering fins that are extruding perpendicularly from the surface they are drawn on, but since all of the work in this implementation of fur is being done in the vertex and fragment shaders, this was not suitable. This makes it impossible to render good fur on simple geometric objects with sharp angles such as tetras or even cubes, but it looks good enough on rounder objects such as balls or torii.

The main loop of the application only needs to make sure that the object is being rendered several times, once for each shell layer, and send information to the shaders about the current layer. The vertex shader then continues by moving the vertices in the normal direction so that the object is enlarged, and then the fragment shader does the rest of the work.

The fragment shader first uses the texture coordinates to produce pseudo-random numbers that are constant on small areas of the object. Rendering this in greyscale gives the object a checkered pattern on the surface. Each small square is then used to approximate a strand of hair. The good thing about this approach is that the

Figure 1: Scene with 8 light sources without bloom effect

Figure 2: Same scene as in figure 1 with bloom effect

thickness of the hair can be altered in real time by modifying the random number generator, but the ugly backside is that the strands are rendered with a square cross section, and that the distance between the hairs is fixed. In a more advanced application one should instead render the hairs realistically in a preproceesing stage and then create these shell textures to look more natural.

Since each strand of hair gets a unique number associated to it, they can all have a bit different characteristics. For example they can have different length, or not be drawn at all if the number lies outside a given interval to create more sparse fur. The strands that won't be rendered are given an alpha value of 0.0 so they are see-through. Each strand gets its color from the skin, which is the geometric shape rendered with a texture as usual. The brightness of this color is modulated a little for each strand so that strands on a skin with a constant color can be separated, and ambient occlusion is approximated by making the innermost layers darker than the tip layers.

The hair strands should realistically get a smaller cross-section when nearing the tips, but since the shells are created in real time this is not possible using the pseudo-random number like this. Instead this is modeled by decreasing the alpha value of the shells the farther away from the skin they are rendered. [Lengyel et al. 2001]

3 Results

The results of the bloom, depth of field and SSAO shaders will be shown in the Sponza scene with spotlights as light sources and a base of shaders consisting of Phong shading, angular falloff, distance falloff and shadow mapping. As many different shaders have been implemented, displaying the results will need many images, and therefore different choices of parameter settings or different models of the same shader will not be shown. Those elements will instead be discussed in the discussion part of this paper. The fur shader was not implemented in the Sponza scene, but only on a torus shape.

3.1 Bloom

As we have angular falloff and distance falloff, we will not get any bright lights in our scene that are contrasting with areas with no lighting, instead we will have more smooth light transitions. This will result in a high intensity threshold not giving very interesting results near the edges of the light cone. We therefore display the bloom effect with a very low threshold to make the bloom effect appear over the whole light cone, and also it looks cool.

Figure 3: Scene with 8 lights and no depth of field

3.2 Depth of field

For our depth of field effect we have chosen to show the result with a foreground object that is much closer to the camera then the rest of the scene. If the scene would have been depicted by a real camera or with our eyes the background would be blurred if the focal plane was close to the lins. In the image its easy to see that our method recreates this effect.

3.3 SSAO

The SSAO effect replaces the normal ambient factor of approximation of the rendering equation. There for we compare our SSAO with the normal ambient term. The results clearly show that the SSAO gives a much more dynamic ambient term. The SSAO makes it much easier to actually see the shapes and depths of objects in areas not directly lit by any light.

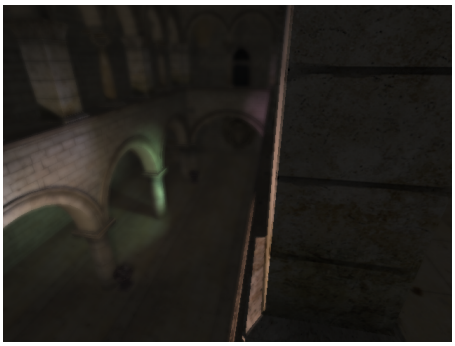

Figure 4: Same scene as in figure 3 with depth of field

Figure 5: Scene with no lights

Figure 6: Same scene as in figure 5 with SSAO

3.4 Fur

The fur shader gives a good appearance of fur on objects in real time. With the right parameters the effect looks quite good despite its drawbacks.

4 Discussion

4.1 Bloom

In the bloom shader we used a model that gives a big effect on our lightsources. In other cases you may want only very intense light sources to bleed over. Like sunlight coming in from a window bleeding out on the window frame. This can be done with a higher threshold value for how high intensity that is needed for an area to blend over. Another thing that could have been done differently is the way we blend. With our current algorithm we blend more light into the area from the pixel far away in the quadratic area we are blending in. We made it this way as it gave us a better final result. But it would probably in other situations give a better effect the other way around, especially with higher threshold (we have not tested those situations).

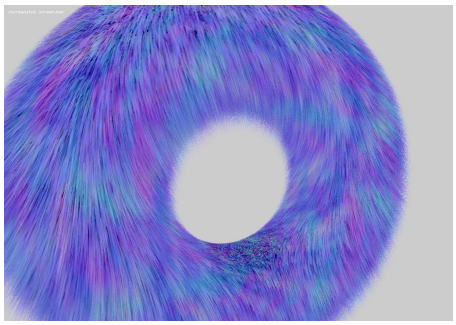

Figure 7: Fur effect with long, uncombed hair

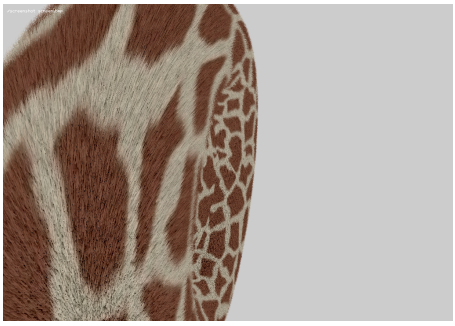

Figure 8: Fur effect with short, combed strands of hair

4.2 Depth of field

The depth of field effect works pretty good, especially when the focus plane is close to the camera. Due to the non-linear depth buffer, the in focus depth becomes larger far away from the camera and the effect does not work too good. This could be fixed by spending more time tweaking parameters and calculating more on the depth. Our approximation of the effect is done by setting the focus to a fix value and adjusting the blur only from this parameter. The effect could have been more physically correct by implementing a model of a physical camera with physical parameters.

4.3 SSAO

Our SSAO is implemented with random points in a sphere. A half sphere with the z-axis in the direction of the surface normal would probably give a better effect since planar areas such as walls should have no occlusion. The noise that is unevitable with the random generation algorithm could also be blurred for a more smooth effect.

4.4 Fur

The fur shader would look better if the fins from the original paper were actually implemented. Different lighting effects could have been used to make it look more realistic, and the fur could also be animated. Growing the hair in preprocessing and saving it in textures could have made the effect more realistic as well as faster, but we wouldn't be able to tweak the thickness and density in real time.

References

DERNERS, J., 2010. Chapter 23. Depth of Field: A Survey of Techniques. `http://http.developer.nvidia.com/GPUGems/gpugems_ch23.html`, Dec.

DOGGETT, M., AND AKENINE-MLLER, T., 2010. Procedural Shading. `http://fileadmin.cs.lth.se/cs/Education/EDAN35/lectures/L4-proc-shaders.pdf`, Dec.

LENGYEL, J., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2001. Real-time fur over arbitrary surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, 227–232.

POLICARPO, F., FONSECA, F., AND GAMES, C., 2005. Deferred Shading Tutorial.