HBAO

Gerard Martin Teixidor

May 11, 2021

Geometry pass

Since we are only computing ambient occlusion, the only information we need to store in the G Buffer are the face normals and the depth. There is no need to store the fragment positions or any material properties. Both, normals and depth, are going to be saved in view space.

To correctly implement SSAO, we need to retrieve the face normals instead of the vertex normals. To do so, we use the dFdx and dFdy GLSL instructions which compute the partial derivative of the fragment position. Given these two derivatives, we compute the normal vector using the cross product.

```
vec3 normal_view = normalize(cross(dFdx(pos_view),
                                   dFdy(pos_view)));
```

The depth of each fragment is computed the same way OpenGL computes the Z Buffer. This means that the depth, once in view space, is multiplied by the projection matrix and divided by the fourth component. Once the depth has been divided, we have to take into consideration that it is no longer lineal. In addition, because of this division now the depth value goes from -1 to 1. Finally, the depth is normalized to the range 0 to 1, although this step is not really necessary.

```
gl_Position = proj * view * model * vec4(vert, 1.0);
depth_view = (gl_Position.z / gl_Position.w) * 0.5 + 0.5;
```

All this data is being packed in a single RGBA texture, using 32 bits per channel. The normal is stored in the RGB channels while the depth is stored in the alpha channel. The use of 32 bits is needed in the alpha channel to avoid z-fighting problems. Since the normals does not require that much precision, a further improvement could be to use two textures, one RGB texture for the normals using 8 or 16 bits, and a single 32 bits channel texture for the depth. This would reduce the space required for the G Buffer reducing the memory bandwidth.

Retrieving position from depth

Once we are doing the light pass, we might want to retrieve a fragment position from the G Buffer. This can be done by unprojecting the depth previously stored in the alpha channel of the G Buffer. Considering the retrieved depth at texture coordinates s, t, the OpenGL projection matrix P, the field of view f and the aspect ratio a:

$$
\begin{aligned}
x_{view} =& s * tan(\frac{f}{2}) * a * -z_{view} \\
y_{view} =& t * tan(\frac{f}{2}) * -z_{view} \\
z_{view} =& \frac{P_{4,3}}{2d - 1 + P_{3,3}}
\end{aligned}
$$

As a reference, we used the post SSAO With Depth Reconstruction [1]. Take into consideration that the matrix of this post is different from the one used in this project. The projection matrix used in this project is the one provided by the function glm::perspective of the glm library.

To reduce the noise produced by some SSAO techniques, we can apply a post processing blur. This blur is a simple implementation of a Gaussian blur with the optimization of doing two passes, one for each axis, instead of a single one. This two passes implementation allows to reduce the time complexity from $O(n^2)$ to $O(n)$.

To blurrier image, we perform multiple passes of the same kernel of size $n = 5$. We could implement a better solution by using a single bigger kernel instead of multiple passes of a small kernel. This can be performed since a single blur radius is equivalent to the square root of the sum of the squares of the individual blur radius.

$$r = \sqrt{r_a^2 + r_b^2}$$

HBAO

The main idea of Horizon Based Ambient Occlusion is to scan the geometry around the shading point to find the horizon vector. Once this vector has been found, and knowing the point tangent vector, we can compute the ambient occlusion.

This implementation of the HBAO tries to be as accurate as possible to the original version. To do so, we used as a reference the published paper in conjunction with presentation slides [2]. These slides are interesting since many details are not present in the paper.

Implemented features

This section lists the original HBAO features which have been implemented.

Uniform directions distribution At each fragment, to compute the ambient occlusion sample, we sample 4 orthogonal directions. If we want more direction samples, we will use multiple orthogonal directions, making the sample directions alway a multiple of 4. Using these 4 orthogonal directions, in conjunction with per pixel randomization, allows to more uniformly distribute the depth samples.

Per pixel randomization. To avoid axis aligned artifacts, randomization is introduced during the sampling process:

- The 4 orthogonal directions are rotated randomly. The rotation is between 0 and 90 degrees since rotation more than a quarter of a circumference would produce the same result.
- Sampling along the direction ray is jittered by randomizing the step size. Between 0.1 and 1.0 of its original size.

To improve the uniformity of the noise, we used a blue noise texture from the blog post Free blue noise textures [3].

Face normals During the geometric pass, instead of computing the normal per vertex and interpolate them, we compute the normals per face. A more detailed explanation has been described in the previous Geometry pass section.

Tangent angle bias Due to the low tessellation of some geometry, to avoid false occlusions, we add a constant bias to the tangent angle. This will remove some correct ambient occlusion but will hide the aliasing effect of low polygonal geometry.

Per-sample attenuation Due to the limited radius, to avoid a hard contrast between the fragments with ambient occlusion and the ones without, a per-sample attenuation is applied. This attenuation weights the samples to be less important as we take more samples along the sampling direction.

Snapping samples Because the sampling along the sampling direction is done in view space, when we project the sampling view space fragment to texture space we can not guarantee the texture space fragment to be aligned with the texel of the G Buffer. Because of that, we have to snap the fragment to the nearest texel center and unproject the fragment to get the new view space fragment. This will guarantee that the depth from the G Buffer that we are using really corresponds to the depth of the new fragment in view space.

The only missing feature with respect to the original HBAO is the use of depth-dependent Gaussian blur. Instead, this implementation only uses a simple Gaussian blur.

References

[1] Etay Meiri. Ssao with depth reconstruction. `https://ogldev.org/www/tutorial46/tutorial46.html`. Accessed: 11/06/2021.

[2] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks*, SIGGRAPH '08, New York, NY, USA, 2008. Association for Computing Machinery.

[3] Christoph Peters. Free blue noise textures. `http://momentsingraphics.de/BlueNoise.html`. Accessed: 11/06/2021.