FRR Lab2 Explanation

Rafael Ávila
29/05/2020
2019-2020/Q2
MIRI-FRR

Geometry Render Pass

To see how the first pass is done, take a look at the package *entityRenderer* and the files *entityVertexShader.txt* and *entityFragmentShader.txt*.

You can also see the results of both normals and depth in the pictures attached.

SSAO Render Pass

This second pass is done in the *ssaoRenderer* package. The vertex shader is quite simple as you only need to render a quad, but the most interesting part is on the fragment shader (*ssaoFragmentShader.txt*). There are 3 different strategies available for this pass.

- Strategy 1: Improving Depth-only SSAO
- Strategy 2: Using Normals
- Strategy 3: Separable SSAO (Strategy 5 on the slides). I will talk more about this one on the next part.

You can see the up to 100 random vectors generated before the execution on the class *ssaoShader.java*, on the second to last method.

The randomized texture consists on a texture where each pixel will have a different RGB color. This will be translated into a 3D vector. This texture was created to match the dimensions of the display (1 color per pixel) using a simple Matlab script *colrandomizer.m*.

Just one more detail. If no blurring is applied, the result obtained here will be rendered directly onto the screen instead of in an FBO.

Separable SSAO

To see the details regarding separable SSAO see the fragment shader mentionned above and look a the second part of the main if condition.

As before. you can see the up to 20 random vectors generated before the execution on the class *ssaoShader.java*, on the last method. This ones will be generated in two perpendicular directions, and then they will be rotated at each pixel.

Note: I know that they don't have to be entirely random, they have to follow a pattern depending on the size of the kernel used to blur the result. But, I wasn't capable of doing it, so I approximated the result by randomizing the samples at each pixel (using the random texture).

Blurring Render Pass(es)

The blurring pass was implemented in two steps: horizontal and vertical, in order to improve the performance. You can see how it is implemented on the *blurEffect* package and more precisely on *horizontalBlurVertex.txt*, *verticalBlurVertex.txt* and *blurFragment.txt*.

Notice how there is only one fragment shader shared by both steps. This is due to the fact that the fragment shader only averages the color of the pixels selected by the vertex shader, so it does exactly the same for both passes.