1 Stereo

Points: 7 (+ max 5 bonus)

In this exercise a multi pass stereo rendering approach is used. This means that the view of each eye is rendered into a separate framebuffer. Each framebuffer has its own color and depth texture attached which allows pixel-wise access to the color and depth information. These two views are combined in a final render pass that draws a screen filling rectangle and combines the color information in various ways in the fragment shader.

The different stereo modes can be chosen via the mode parameter in the UI section *finalization pass*. There are the following modes:

- input - showing either the color or depth texture (determined by *image_type*) of one eye (determined by *image_idx* with 0 for left and 1 for right)
- random_dots - showing a random texture
- remapped - need to be implemented by you: should show occlusions in the selected view (same parameter as input-mode)
- parallax - showing the parallax values which you can be used if task 3 is solved
- anaglyph - need to be implemented by you except for *color anaglyph*) (different anaglyph modes can be selected through *anaglyph_mode*)
- autostereogram - need to be implemented by you: should show random dot auto-stereogram of the scene

In order to be able to use these stereo rendering modes you need to implement task 1.1a.

1.1 Transformation Matrices (2 points)

The user can manipulate the view of the scene by holding the left mouse button and moving the mouse. The scene can be shown in stereo if the *enable stereo* toggle button is clicked. This enables the `indirect_two_pass_stereo()` method. In the stereo case the view point that is manipulated by the mouse movement lies in the middle between the two eyes. In order to render the scene in stereo, the correct projection and model-view matrices have to be computed for each eye. Otherwise both views look the same (as in the source code you got).

a) Calculate the projection and model-view matrix for each eye. (**1.5 point**)

b) Make use of the cyclopic lighting toggle in the UI and change the projection and model-view matrices to enable and disable lighting in cyclopic eye. Have a look at the lighting results for each eye and explain the difference. (**0.5 point**)

1.2 Anaglyph (1.5 Points)

For this task you can use your own created anaglyph glasses (red for the left eye and cyan for the right eye). :) There are multiple ways of computing anaglyph stereo images. This is done by combining the color channels of both views in the fragment shader. The `finalize.glfs` is the fragment shader used for the screen filling rectangle.

Set the mode parameter in the UI section *finalization pass* to anaglyph and display the randomly generated boxes by using the *show_boxes* toggle button. Since the boxes are colored in all colors you will be able to see how different the anaglyph modes look and which colors cannot be fully reproduced using anaglyph.

(The different anaglyph types are explained in the introductory slides for this exercise.)

a) Implement true anaglyph. (**0.5 point**)

b) Implement gray anaglyph. (**0.5 point**)

c) Implement half color anaglyph. (**0.5 point**)

1.3 Eye Separation (1 Point)

The resulting stereo image quality depends on the chosen eye separation value which can be changed in the UI in the section *stereo parameters*. The value is the distance between the eyes given in percentage of the whole viewport. You can visualize the current eye separation value by using the *show eye separation* toggle button if stereo is enabled.

Make a user study to determine a range of values for the eye separation where no stereo frame violation happens and a clear 3D impression can be perceived. For this, try different eye separation values and make a note regarding the visual quality. Measure the eye separation on your screen in cm (using the *show eye separation*) and make a note how you measured (sphere center or inner/outer distance). Submit your results as a text file. (**1 point**)

1.4 Remapping (2.5 Points)

Using color and depth textures has the advantage that we can access these information pixel-wise in the fragment shader. This can be used to show occluded regions that only one eye can see.

If you finished this task you will also be able to use the parallax mode.

ref to slides

a) Implement the `vec3 remap_window_coords(...)` in `finalize.glfs` that maps the pixel coordinate of one view to the corresponding coordinate in the other view. (**1.5 point**)

b) Implement the visibility check in `bool check_visibility(...)` in `finalize.glfs` that uses the remap function to compare the depth values between a pixel in one view and the corresponding pixel in the other view to decide if the pixel is visible for the other eye. (**1 point**)

1.5 Bonus tasks (max. 5 Points)

a) Implement your own optimized anaglyph in the `finalize.glfs` fragment shader. Write a comment about the reason why your solution is an optimized version of anaglyph (what problem did you address with your approach?). (**0.5-2 point**)

b) Implement random dot auto-stereogram in the `finalize.glfs` fragment shader. (**3 point**)

c) Exchange the multi-pass rendering by using one pass rendering. (**5 point**)

Submission

Put all files in the exercise folder that you modified or added in a Zip archive (named according to the pattern `[YourName]_ex1.zip`) and upload that archive to Opal. Make sure that your solution runs on the type of system used for evaluation (Windows, Visual Studio $\geq$ 2017). If you do not have access to such a system and are in doubt whether your solution will work in this configuration, contact your tutor or supervisor.