Homework 4: Build Your Own HMD, Implement Stereo Rendering and Lens Distortion Correction

EE267 Virtual Reality 2025

Due: 05/01/2025, 11:59pm

Instruction

Students should use JavaScript for this assignment, building on top of the provided starter code found on the course webpage. We recommend using the Chrome browser for debugging purposes (using the console and built-in debugger). Make sure hardware acceleration is turned on in the advanced settings in Chrome. Other browsers might work too, but will not be supported by the teaching staff in labs, piazza, and office hours.

The theoretical part of this homework is to be done individually, while the programming part can be worked on in groups of up to two. If you work in a group, make sure to acknowledge your team member when submitting on Gradescope. You can change your teams from assignment to assignment. Teams will share handed-out hardware (starting this week).

Homeworks are to be submitted on Gradescope (sign-up code: **D3PYB3**). You will be asked to submit both a PDF containing all of your answers, plots, and insights in a **single PDF** as well as a zip of your code (more on this later). The code can be submitted as a group on Gradescope, but each student must submit their own PDF. Submit the PDF to the Gradescope submission titled *Homework 4: Build Your Own HMD, Implement Stereo Rendering and Lens Distortion Correction* and the zip of the code to *Homework 4: Code Submission*. For grading purposes, we include placeholders for the coding questions in the PDF submission; select any page of the submission for these.

When zipping your code, make sure to zip up the directory for the current homework. For example, if zipping up your code for Homework 1, find `render.html`, go up one directory level, and then zip up the entire `homework1` directory. Your submission should only have the files that were provided to you. Do not modify the names or locations of the files in the directory in any way.

Please complete this week's lab and watch the video before you start to work on the programming part of this homework.

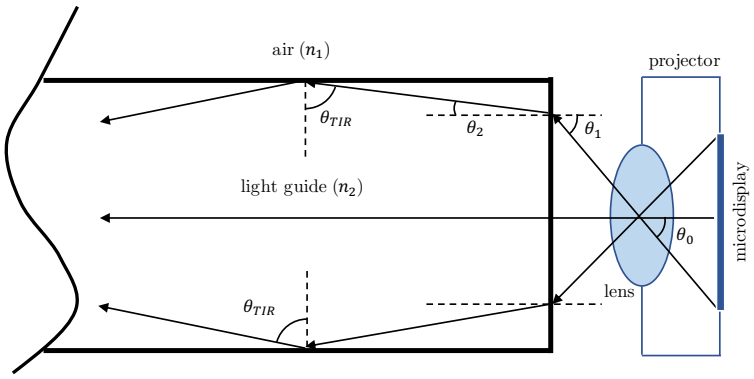

Figure 1: *Projected light coupled into a light guide.*

1 Theoretical Part

1.1 Consistent Occlusions in Optical See-through AR (*10pts*)

Optical see-through (OST) augmented reality (AR) displays overlay digital content onto the physical world using an optical combiner. Thus, the physical and digital objects follow an additive superposition principle with each component being mostly independent of the other. This is unlike the physical world where the lighting and shading of different objects influence each other, for example by casting shadows or by occluding each other. Occlusions in particular are one of the most important depth cues of human vision but, unfortunately, it is not straightforward to implement consistent occlusions between physical and digital objects in OST AR systems.

In general, we can distinguish between two cases of occlusions in AR: (a) a digitally displayed object should appear (at least partly) behind a physical object, i.e. the physical object occludes the digital object and (b) a physical object should appear (partly) behind a digital object, i.e. the digital object occludes the physical object.

(i) Briefly describe an engineering solution that solves problem (a). What information do you need of the physical object and the digital object so that your solution works well? Do you need any additional sensors, such as special cameras? Describe exactly what sensors or information you need and how your solution works. (*5pts*)

(ii) Briefly describe an engineering solution that solves problem (b). What information do you need of the physical object and the digital object so that your solution works well? If you cannot come up with a good technological solution for this case, describe in detail what would have to happen for this to work. (*5pts*)

1.2 Geometric Optics (*25pts*)

As illustrated in Figure 1, a projector emits light into one end of a light guide made of a special kind of glass. The light travels through the light guide and comes out at the other end (Fig. 2 left). The refractive index of the special glass n_2 is $\sqrt{2}$, and the refractive index of air n_1 is approximated as 1. In this question, the critical angle is defined with respect to the surface normal (dashed lines in figures). For simplicity, the thickness of the projector lens and possible refractions on the air–lens interface can be neglected.

(i) What is the critical angle for light entering air from the light guide? The critical angle is the largest angle of incidence for which refraction can still occur. For any angle of incidence greater than the critical angle, light will undergo total internal reflection (see Fig. 1). (*5pts*)

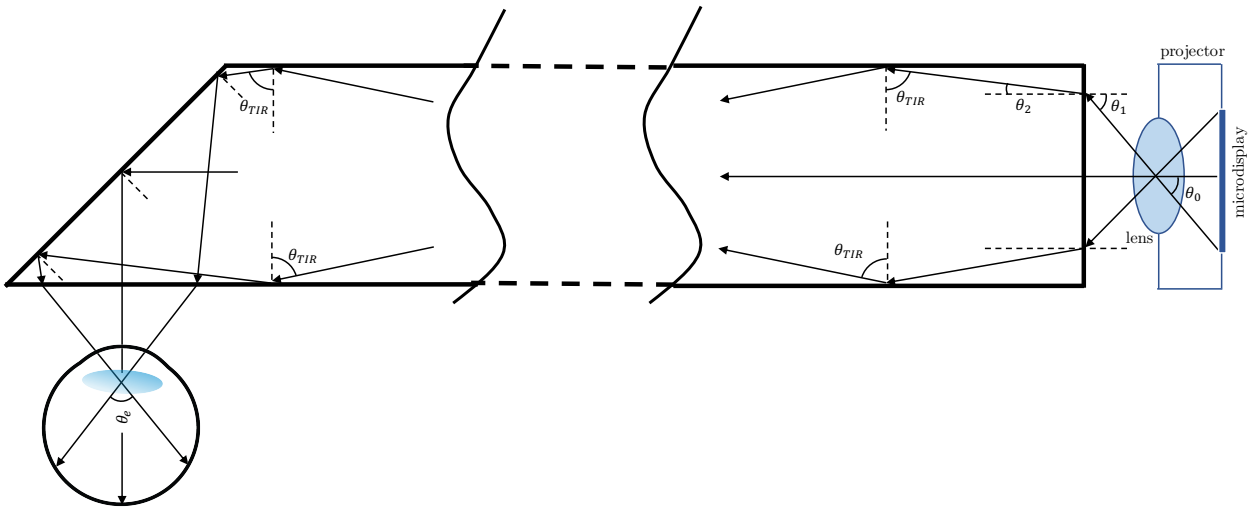

Figure 2: *Projected light coupled out of the light guide and into the viewer's eye.*

(ii) If light bounces in the light guide at $\theta_{TIR} = 60^\circ$, what is the angle at which light should be projected into the light guide (i.e., calculate θ_1)? (See Fig. 1.) (*5pts*)

(iii) Given projection angle θ_1 from the previous question, and focal length of lens $f = 60$ mm, what is the width of microdisplay such that all of the emitted light is steered into the light guide (assume the light guide is arbitrarily wide)? Assume that the microdisplay is 60 mm away from the lens. (*5pts*)

(iv) A perfect mirror is placed with slope of 45° at the other end of the light guide. What is the field of view θ_e (see Fig. 2)? (*5pts*)

(v) We hope to increase θ_e to 120° so that users have a more immersive experience. For this purpose, we decide to use the same light guide but made of a different material. What is the smallest refractive index, n_2', the material of the new light guide can have? Can you easily find a clear material with that refractive index? (Use Wikipedia to browse the refractive indices of different transparent materials.) (*5pts*)

Programming Part PDF Deliverables

The following questions in the programming part ask you to provide written responses:

- 2.1.3 Stereo Rendering Perceptual Experiments (*10pts*)
- 2.2.3 User-defined Distortion Coefficients (*10pts*)

Make sure to append your responses to the end of the PDF submission.

2 Programming Part

It's time to assemble your head mounted display! In the lab, you should have gotten all the hardware components for your HMD. **Make sure to assemble it following the instructions in this week's lab video.** In the programming part of this homework, you will extend the idea of anaglyph stereo rendering from last week to stereo rendering for your HMD, resulting in an even better 3D experience of your favorite teapot. For the HMD stereo rendering to work, we need a few parameters to set up our view frusta, pre-warp images to correct for optical distortions, etc.

View-Master Deluxe VR Viewer satisfies the Google Cardboard specs and the following specifications:

- Focal length of lenses: 40 mm
- Lens diameter: 34 mm
- Interpupillary distance (IPD): 64 mm
- Distance between lenses and screen: 39 mm
- LCD screen width: 132.5 mm, height: 74.5 mm
- Eye relief: 18 mm.

The distance between lenses and screen is measured when the lens is pulled to the end by the adjustment wheel. Even if you need to adjust the focus to see the display clearly, please stick to these values for your implementation.

After you have received your HMD, make sure to align the line rendered in the center of the screen with the line on the bottom holder in the View-Master. This mechanical centering is critical for achieving the correct stereo effect. Also, you need to move your browser window to the provided external display and set your browser to full-screen mode. Once you enter the full-screen mode, refresh the browser while opening `render.html` to initialize internal parameters. The task bar should not be visible on your screen!

2.1 HMD Stereo Rendering

The anaglyph stereo rendering you implemented in HW3 assumed that each eye saw the same screen and we used color filters to separate the different views. With the HMD, each eye actually sees a different portion of the screen, so the stereo rendering will be slightly different. The HMD retains full color information of the scene and also provides a wider field of view.

2.1.1 Calculating Parameters of the Magnified Virtual Screen Image (*5pts*)

Given the focal length of the lenses in your HMD, the distance between lenses and microdisplay, and the eye relief, implement `computeDistanceScreenViewer()` and `computeLensMagnification()` in `displayParameters.js` to compute the distance of the virtual image from the viewer's eyes as well as the magnification factor.

2.1.2 Stereo Rendering (*15pts*)

Using these values and the screen parameters reported above, we can implement stereo rendering. Fill in the `computeTopBottomLeftRight()` function in `transform.js` to define the appropriate view frustum for each eye. We use these values in `computePerspectiveTransform()` to create the projection matrices. Every parameter you need for the computations are found in the function arguments: `clipNear`, `clipFar`, and `dispParams`. The off-axis frustum for the HMD is different from that in HW3 (see lecture slides), so make sure to update your calculations! For stereo rendering with this method, you only need two rendering passes, one for each eye, as opposed to the three passes used for anaglyph rendering.

2.1.3 Stereo Rendering Perceptual Experiments (*10pts*)

Once the stereo rendering works, perform a few simple perceptual experiments on yourself. In the PDF with your solutions for the theoretical part, report your experience in a short paragraph for each of the following questions:

(i) Vary the interpupillary distance (IPD) in the code while looking at the same, static 3D scene. Describe the perceptual effect of the varying IPD. (*5pts*)

(ii) Experiment with the vergence–accommodation conflict (VAC). How far can you move objects out of and into the screen, i.e. away from the virtual image, while still being able to fuse the stereo images? Your visual system is quite resilient during continuous viewing, so errors might not be immediately apparent. Try closing your eyes for a bit, and then viewing the scene fresh. You should start seeing the effects then. Report min and max depth in which you can comfortably fuse the stereo image pair. You can move the teapots back and forth by pressing `w` and `s` on the keyboard and know the teapot's location from the z-component of model translation displayed on top of the browser window. (*5pts*)

2.2 Lens Distortion Correction

If you have implemented the stereo rendering correctly, you should see a 3D teapot floating in front of you. If you look carefully, however, you might notice that some things look a little distorted (lines you know are straight might not be straight anymore). However, if you look at the physical display, nothing looks distorted! These distortion comes from the lenses you see the screen through. Luckily, the distortion can be nicely modeled by the Brown–Conrady model you learned in class. Using this model, we can pre-distort the image we display on the screen so that when the light passes through the lenses and gets distorted by the lenses, lines are actually straight.

The distortion correction is applied to whatever image we want to ultimately display, and is applied as the final rendering passes. Because the correction may be different per eye, we have four rendering stages (Figure 3): the first two render the left and right views of the scene into an FBO with Phong lighting using the `fShaderMultiPhong.js` and `vShaderMultiPhong.js` shaders; and the last two apply the distortion correction you will implement in `fShaderUnwarp.js` to each eye's image. Keep in mind during your computations that your distortion correction shader will only be applied to one eye's image at a time, and that the image will only populate half of the screen.

2.2.1 Lens Distortion Center (*10pts*)

The first thing you'll have to do is to calculate the distortion center for each eye. As described in the lecture, the center of the distortion is the point on the screen which intersects the optical axis of the lens. Fill in `computeCenterCoord()` function in `stereoUnwarpRenderer.js` to compute the distortion center in texture (u,v) coordinates for each eye. The computed center coordinates are passed as uniforms (`centerCoordinate`) into `fShaderUnwarp.js`.

Hint: You'll end up using these values in the shader when computing your lens un-distortion and indexing into texture maps. Remember that texture are indexed with normalized texture coordinates (i.e. between 0 and 1).

2.2.2 Fragment Shader Implementation (*15pts*)

In `fShaderUnwarp.js` implement the distortion correction using the center of lens distortion parameters you've defined above. The lens distortion parameters are stored in the uniform K, where the first element corresponds to K_1 and the second corresponds to K_2 in the lens distortion equations in Lecture 7. Apply the lens distortion formula we discussed in class to compute the distorted texture lookup coordinates. Use these distorted coordinates to do the texture lookup.

Unfortunately, WebGL 1.0 doesn't support the boundary condition where the values outside of $[0, 1)$ is a user-defined

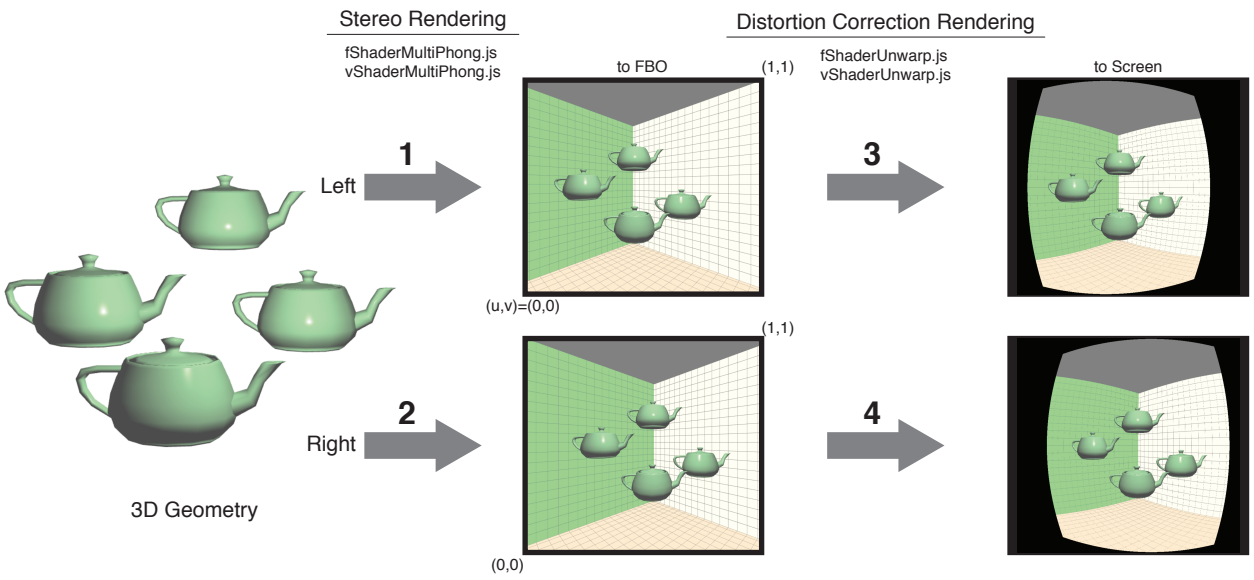

Figure 3: *The four rendering stages (as shown by the numbered arrows) for the multi-pass rendering pipeline including stereo and lens distortion correction rendering.*

constant value. To achieve this effect with your fragment shader, assign black color to the fragments if the lookup coordinate is outside of $[0, 1)$.

Keep in mind that the distortion is symmetric around the center of distortion, which corresponds to the center of the lens defined as the `centerCoordinate` uniform variable. Follow the lecture slide to normalize the distance between the lens center and the fragment.

You might find it useful to quickly toggle between the standard rendering mode and the unwarped mode by pressing the 1 button on the keyboard or click the button on the top.

2.2.3 User-defined Distortion Coefficients (*10pts*)

You should be able to decrease/increase K_1 with the 2/3 keys and decrease/increase K_2 with the 4/5 keys. This functionality is already implemented for you. You can see the updated values displayed at the top of your screen. By pressing the space bar, you can switch to a scene where a grid pattern is displayed to make it easier to see the distortion effects.

Manually adjust your distortion coefficients K_1 and K_2 until the lines are as straight as they get throughout your entire field of view. In the PDF submission, report the values (K_1, K_2) that worked best. Submit one screenshot as well as one photo (e.g., captured with your cellphone) through the lens that show the grid lines before and after distortion correction each. That's four images in total you should submit. Even though multiple sets of the distortion parameters may show equally good results, you can report one set of the best parameters. We will grade based on the photos and your code. After the adjustment, modify the initial distortion coefficients to your adjusted values in `state` of `stateController.js`.

Questions?

First, Google it! It is a good habit to use the Internet to answer your question. For 99% of all your questions, the answer is easier found online than asking us. If you cannot figure it out this way, post on piazza or come to office hours.