CS 135: Assignment 3: Fun with WebXR

In this assignment, you will gain experience in basic WebXR development in a web browser. The lab is due in 2 weeks. This is the final lab; for the remaining lab sessions this quarter, please use the time to work on your final projects.

Due date

Tuesday Nov. 8th (Section 21)
Thursday, Nov 10th (Section 22)

Important Notes:

1. Read all submission instructions carefully, and submit all required documents.
2. The WebXR Documentation (https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API) and samples (https://immersive-web.github.io/webxr-samples/) are good sources of information/examples.
3. Check Slack for any bugs, updates, or important information for this programming assignment.

Part 1: Setup

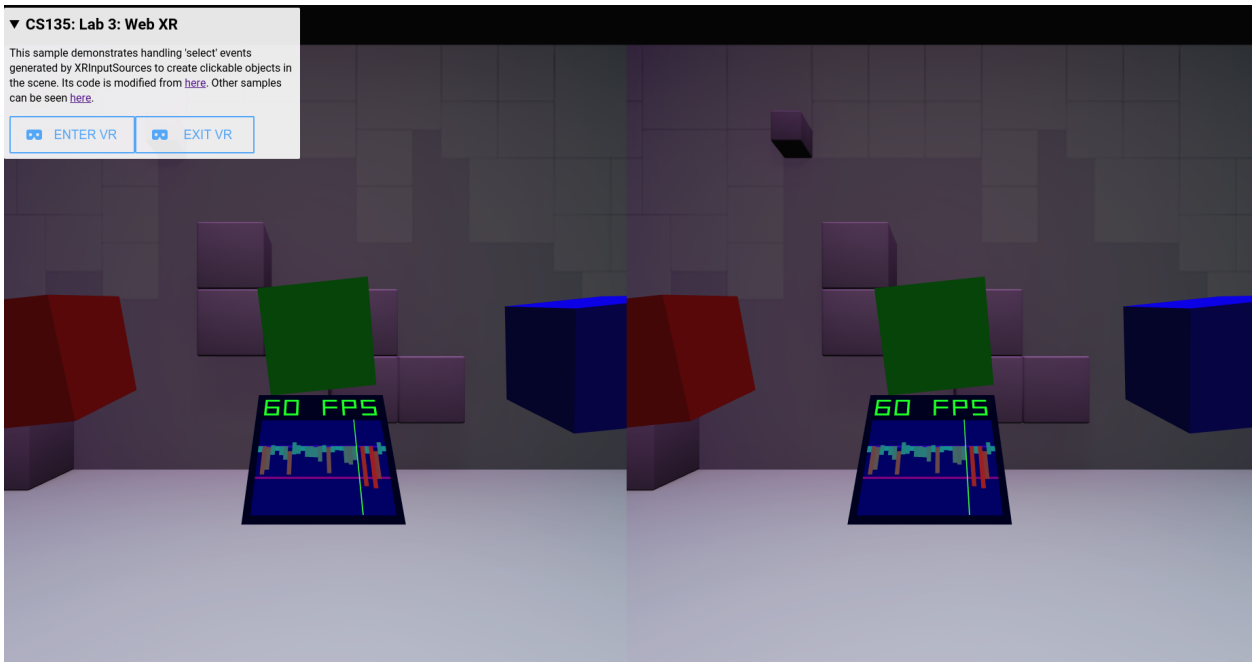
In this lab, you will be experimenting with WebXR. WebXR provides APIs to render VR/AR content from a browser, using WebGL under the hood. WebXR allows access to VR/AR input controllers, if you have them, but for this lab, we will be using a hardware emulator and work entirely in the browser.

To do this, you will need to install a few things:

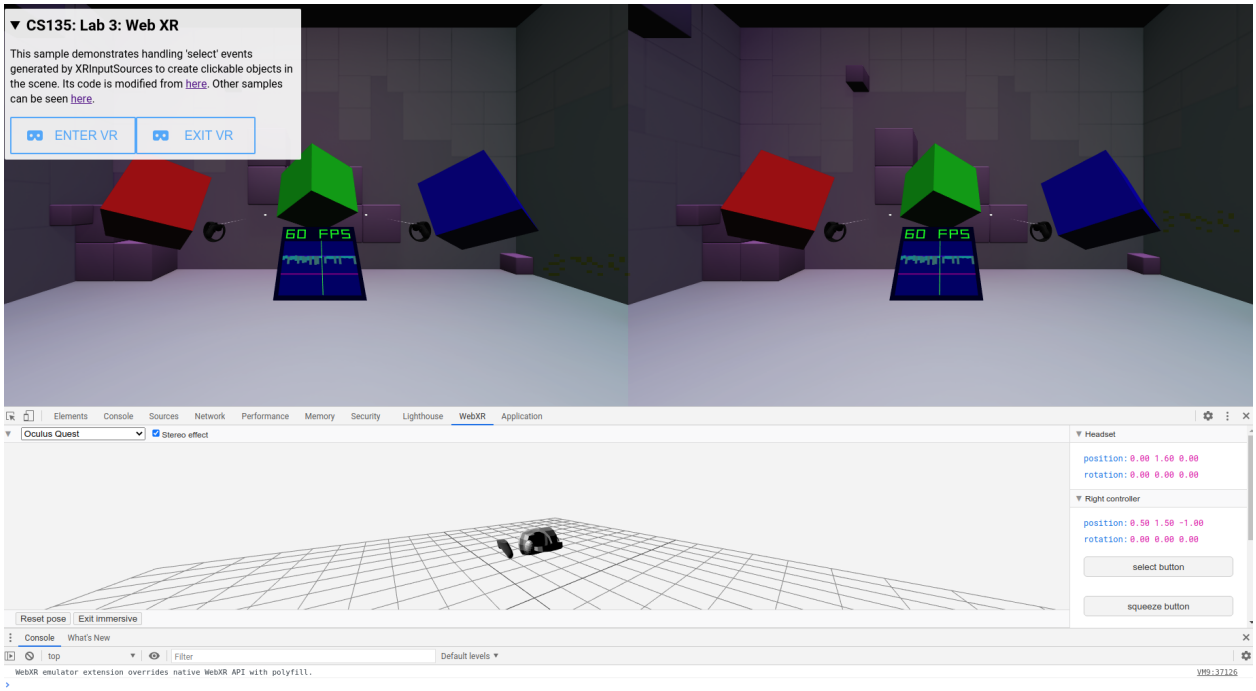
- A recent version of Google Chrome (79+ should be enough, tested on 86). Once you have done this, you can visit the WebXR sample page <https://immersive-web.github.io/webxr-samples/>, and you should see a green checkmark at the top showing that your browser supports WebXR.
- WebXR API emulator to emulate VR hardware: <https://chrome.google.com/webstore/detail/webxr-api-emulator/mjddjgeghkdijejnciaefnkjmkafnnje/>
- Your favorite web server. The easiest way is probably to run a local web server in your browser using the “Web Server for Chrome” browser extension: <https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb/>

Now save and extract the lab files (http://www.cs.ucr.edu/~jiasi/webxr_lab.zip) onto your local web server. If you're using "Web Server for Chrome", configure the extension to host the lab files folder.

When you open "index.html" (or whatever localhost Web Server for Chrome is set to), you should see a scene with 3 cubes. Click "Enter VR" on the right button to see the view from each eye.



Next open up the WebXR emulator from Chrome menu > More Tools > Developer Tools > WebXR tab. You should have a display similar to that shown below (you may wish to re-arrange the layout):



There are several important parts of the screen:

- **Top third of the screen**: The WebXR application.
- **Middle third of the screen:** The emulated VR hardware. Please select “Oculus Quest” in the drop-down menu for this lab. Play around with the headset and controllers to rotate and translate them (first click = translation controls, second click = rotation controls). On the right, you can see the position and rotation of the headset and controllers. There are “Select button” and “Squeeze button” controls for each of the controllers.
- **Bottom third of the screen**: The console where you can view debug messages.

Play around with the default app to get familiar with it. The default behavior is:

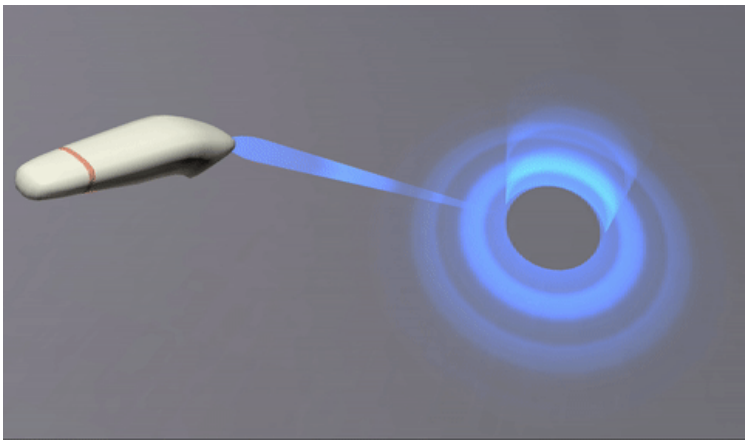
- **Select**: When you press and de-press the “Select button” while interacting with a cube , its size and color should change.
- **Squeeze**: When you press, move the controller, and de-press the “Squeeze button” while interacting with a cube, the cube should follow the controller’s position and also change color and size. There may be a slight change in the cube’s position when you first press the button; that is normal.

Great, now you’re all set up! Let’s move on to the fun part!

Part 2

You will mainly modify the index.html file for this lab. Open it up and get familiar with the Javascript content. There are comments sprinkled throughout to help you understand. The WebXR documentation linked in “Important Notes” above can also help.

One thing you may wish to know is that each controller has a ray emanating from it, pointing in a straight line. This ray has both an origin and a direction in world space, and is already visualized for you in the default app. For more information, see the documentation.



Example of a ray emanating from a controller

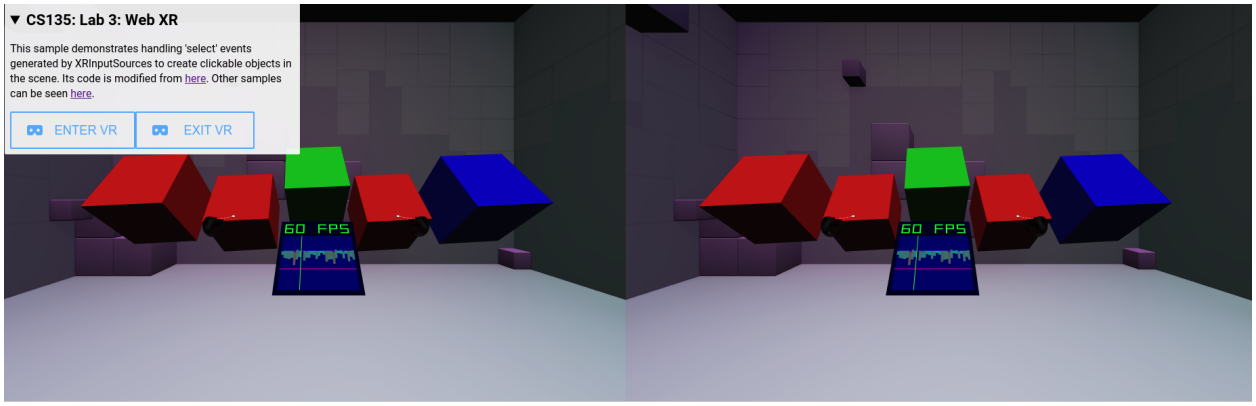
For debugging, you can use commands like “`console.log("variable name " + variable);`” to print things into the WebXR console.

A) The default behavior of the app is that once the “Squeeze button” is de-pressed, the cube goes back to its original position. Modify the code so that the cube stays at its current position when the button is de-pressed. In other words, you can now drag and drop the cubes anywhere you want in the scene.

Hint: Modify the `onSqueezeEnd` function and comment out the unnecessary lines.

B) Next, modify the code so that if the “select button” did not result in a cube changing color/size (*i.e.*, the hit test failed), then a new cube is created somewhere along the raycast from the controller, in a color of your choosing. In other words, you can create new cubes at arbitrary locations in the scene.

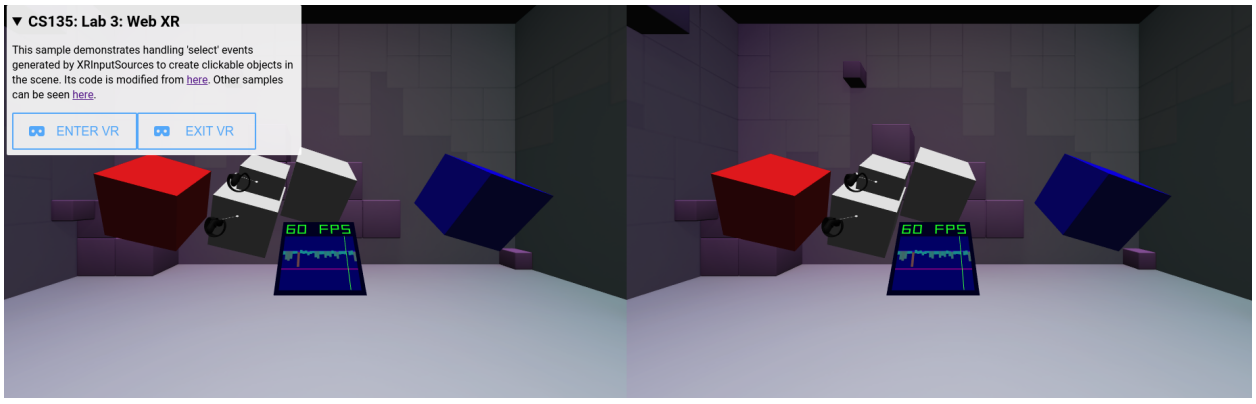
Hints: Add the necessary code to the `onSelectStart` function to create the new box. There is an `addBox` helper function that may be handy. You can get an idea from `onXRFrame` on how to get the origin and direction of the `targetRayPose` emanating from the controller.



Example: new red cubes created surrounding the center green cube.

C) Finally, modify the code so that when any two cubes are within 0.25 m[1] of each other, they both change color to white. For example, if I grab one cube using the “squeeze button” and drag it on top of another cube, they should both turn white instantly. A cube should keep its white color even if it moves away. If the “select” button is pushed on a white cube, a new random color is selected. This should work for both the original cubes and the new cubes you created.

Hints: Add the necessary code to the `onXRFrame` function to detect the intersection and change the color. There are some useful vector manipulation helper functions (e.g., `length`) in `js/third-party/gl-matrix/src/gl-matrix/vec3.js`. The `boxes` array contains a list of the current cubes in the scene. `onSelect` and `onSqueeze` have examples of how to change the box color.



Example: The three cubes in the middle intersected with each other, so they turned white.

Grading Rubric

[1] As long as the cubes’ “position” attributes are within 0.25 m, you can trigger the color change. You do NOT need to do anything fancier, for example determining if the cube faces are within 0.25 m of each other.

Name	Points	Description
Part 1		
Setup	0	Download and install necessary browser and browser extensions, test basic WebXR app. No points.
Part 2		
Cube positions	10	Cubes remain where you put them after “squeeze button” is depressed
New cube	10	A new cube is created along the controller’s raycast if “select button” does not interact with an existing cube
Cube intersection	10	Intersection of two cubes within 0.25 m detected
Color change	5	Cubes turn white upon intersection
Total	**35**	

Submission Instructions

1. Create a zip file containing the following items:
    a. The index.html file you modified for this lab.
    b. A README.txt file containing any special instructions or notes you think are relevant for evaluating your assignment.
2. Name the file by separating NetIDs with underscores- _cs135vr_HW3.zip. EXAMPLE: If steve1 and anna2 worked together, the file should be called steve1_anna2_cs135vr_HW3.zip
3. Only one team member should submit the zip.

Materials from The Immersive Web Community Group <https://github.com/immersive-web/webxr-samples> is gratefully acknowledged.