

CPSC 314

Assignment 1 Due Fri Sept 23, 2022

Introduction to Three.js, and GLSL shaders (6% of final grade)

In this assignment you will gain basic experience with Three.js and GLSL shaders. Outcomes include the ability to:

- run WebGL code based on Three.js;
- debug basic javascript errors and shader errors;
- modify basic 3D scenes in Three.js; defining a custom 3D object;
- make simple modifications to a fragment shader.

Copy and expand the zip file in a local “cs314” directory for this assignment:

```
cp a1.zip ~/cs314/a1.zip
cd ~/cs314
unzip a1.zip
```

Files include:

- **a1.html**: vertex-shader and fragment-shader code; runs **a1.js**
- **a1.js**: main javascript code
- **obj/teapot.obj**: 3D model vertices (**v**) and triangle (faces) (**f**)
- **images/floor.jpg**: image that appears on the rendered floor
- **js/*.js**: three.js libraries and other supporting code

Enabling local file access

View your local version of **a1.html** to ensure that your web browser is properly enabled to run Javascript and WebGL. Chrome, Firefox, and Safari all support WebGL. If you get a blank image, black floor, or you are missing a rendered teapot when launching **a1.html**, this likely means that your browser has not been enabled to access local files. For your own machine, the fix depends on your OS and your browser.

Firefox does not require any special steps on the lab machines in ICCS 005. A recommended solution for Chrome is to install a local http server, i.e., see: <https://www.npmjs.com/package/http-server>. This can be launched from your assignment 1 directory using

```
http-server . -p 8000
```

Then use <http://localhost:8000/a1.html> as your URL.

Alternatively, Chrome can be opened from the command line to allow local file access, i.e., execute `open -a "Google Chrome" a1.html --args --allow-file-access-from-files` from a terminal window while in your assignment 1 directory. **Strong caveat:** Be aware that this opens a security loophole when browsing the web with Chrome, until you quit Chrome.

See also threejs.org -> documentation -> How to run things locally.

Assignment steps

Now walk through the following steps, and implement the requested changes. After each edit to `a1.html`, `a1.js`, or to a GLSL shader file, you will want to do a "forced reload" of `a1.html` in order to see the changed result. When introducing errors, fix the error before moving on to the next step. All other changes can be made in a cumulative fashion.

1. Introduction to Three.js and Shaders (12 points total)

Parts (a)-(h), (m-o), are worth 3 points taken altogether. There is nothing to hand-in here, but you should be prepared to discuss these with your TA.

- (a) Run the code by launching `a1.html`. Open the console window (Ctrl+Shift+J Windows / Linux, or Cmd+Opt+J Mac). Look for the "hello world" message. Change the console message to "CPSC 314 Assignment 1 by FOO", where FOO is your name.
- (b) Attempt to print the result of a division by zero. What happens?
- (c) Attempt to use a variable name that does not exist yet. What happens?
- (d) Add a new variable using "var foo;" and then print it's value without first initializing it. What happens?
- (e) Remove the terminating semicolon from one of the early lines in `a1.js`. What happens? Why? Do a web search on "use of semicolon in javascript" to understand why you should still use terminating semicolons.
- (f) Insert the following code at the beginning of `a1.js`. What are the resulting values of `a` and `b`? Why?

```
a=10; b=3;
function go() {
    var a = 14;    b = 15; }
go();    console.log('a=',a,'b=',b);
```
- (g) Change the background colour to be light blue.
- (h) Know how to orbit, pan, and zoom in the scene. This is done via left-mouse-drag, right-mouse-drag, and mouse-wheel. On a Mac trackpad: click-drag, two-finger click-drag, two-finger drag, respectively.
- (i) (1 point) Change the custom object, which is currently a white square, to be a set of purple vertical walls that form a triangular room when seen from above. Modify the properties of the material as needed.
- (j) (1 point) Create an instance of a second torus. Change the orientation so that it is parallel to the ground. Change the position so that the first torus and second torus are linked together, like links in a chain.
- (k) (1 point) Build a twisting stack of three cubes, coloured yellow, red, and blue (from bottom to top).

- (l) (1 point) The light source, shown as a red sphere, can currently be moved using the W,A,S,D keys. Change the code so that the movement of the light source is bounded to $x, y \in [-5, 5]$. Make the light source yellow.
- (m) Edit the teapot geometry in `obj/teapot.obj`. First, make a second copy of the first vertex, and comment out one of these copies by adding a leading '#' sign) to the line. Then change the coordinates of the first vertex to `(-4,1.8,0.0)`. View the result to see where the first vertex is located. Then change the first vertex back to its original value.
- (n) The teapot has its own vertex shader and fragment shader programs, given in `a1.html`, called `myFragmentShader`. We will be experiment with making a few small changes to the fragment shader. First, remove a semicolon from one of the fragment shader statements. What happens? Look at the console log. Add the semicolon back in.
- (o) Save a commented copy of the key line in the current fragment shader, i.e., `gl_FragColor =`. Now change the fragment shader to make all pixels of the teapot green. Preserve this change as a commented line, and revert to using the original shader.
- (p) (2 points) Change the fragment shader to render the intensity according to light coming from a fixed direction. In the shader, define a lighting direction, $L=(0.0,0.0,-1.0)$, which is a vector that points towards the light. The decimal values are important here, because otherwise the numbers will be interpreted as integers. Now define a float, i , computed to give a diffuse lighting model, $N \cdot L$. GLSL provides a function `dot(a,b)` that computes $a \cdot b$. Assign the computed intensity, i , to each of the red, green, and blue components of the final fragment, to produce a grey-shaded teapot.

(q) (3 points) Creative Component

Make a copy of `a1.html` and `a1.js` to create `creative.html` and `creative.js`. Edit `creative.html` to read in `creative.js` instead of `a1.js`. Then use this as a starting point to build a more interesting scene. The most compelling scenes will be shown in class. You are free to experiment with any-and-all features. Provide a very brief summary of your creative component in `README.txt` that summarizes your work for this part, what you learned while doing it, and, attributions for any code or objects that you may have used from elsewhere.

Possible ideas include: building a highly simplified model of a forest, a room, or other scene; building a scene that demonstrates experimentation and exploration of three.js capabilities; generation of your own custom geometry directly in javascript or via a generated OBJ file; any combination of the above.

Use this as an opportunity to do something fun, and at the same time to do a bit of self-guided learning. We are all learning together, so any and all questions can be asked on Piazza. Extra credit (and instructor kudos!) go to those who help others on Piazza. But the real reason to help others is that you will also learn more yourself – “when one teaches, two learn.”

Submit your assignment by the deadline using: `handin cs314 a1` on the undergrad Linux machines, or you also use web handin:

`https://www.students.cs.ubc.ca/cgi-bin/protected/handin`.

Show your demo to a TA in your lab section, or during the extra face-to-face hours (to be arranged). You'll get the chance to meet your TA, to describe your efforts for the creative component, and to explain the results for the various experiments you performed above.