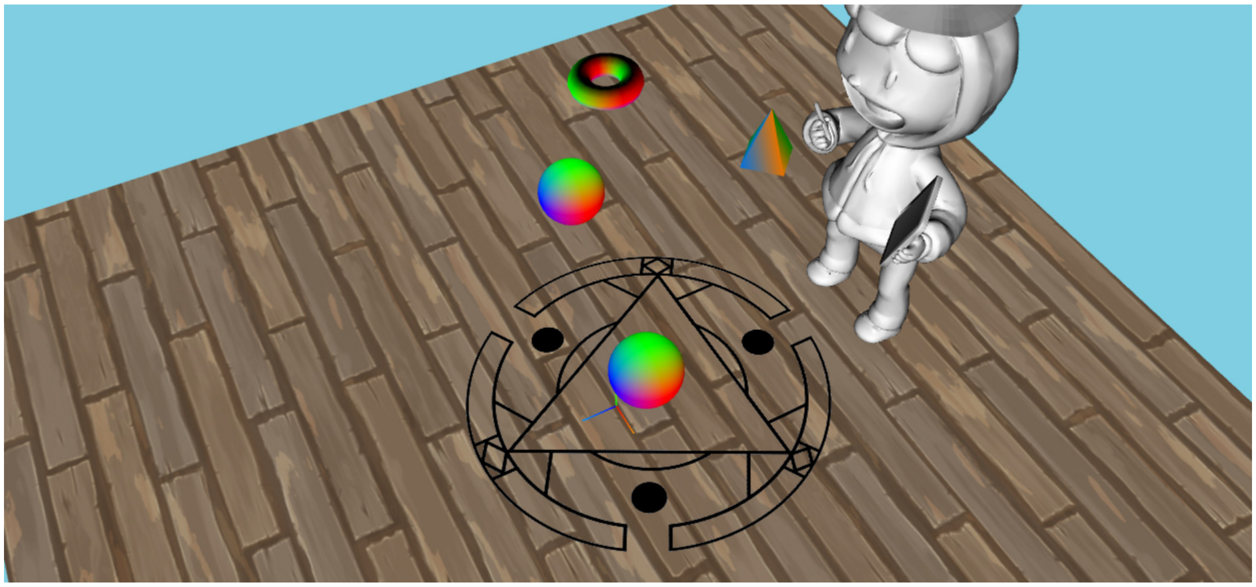
CPSC 314
Assignment 1: A Magician's Gambit.
Introduction to Three.js, WebGL 2.0, and Shaders

Due 11:59PM, Jan 24, 2020



1 Introduction

The main goals of this assignment are to setup your graphics development environment, including checking your browser compatibility, setting up a local server, and an initial exploration of the uses of vertex and fragment shaders. For this exploration you will be using a template provided by the instructor, including shader code (`.glsl` files in the `glsl/` folder). Your main work will be to develop a high level understanding of how the code works, to modify or write shaders, and to use rudimentary communication between the JavaScript program and the shaders. Some of the details of what is going on in the rest of the code will only become clear a bit later in the course. You are of course welcome to take a peek now, especially for the last part of the assignment. Some of the concepts are explained in Appendix A of your textbook, and in the web resources listed on the course web page.

To program a shader, you will use a programming language called GLSL (OpenGL ES Shading Language version 3.0). Note that there are several versions of GLSL, with more advanced features, available in regular OpenGL. Make sure that any code you find while trying to learn GLSL is the correct version.
This assignment uses a simple scene consisting of a "wizard" character, their "wizard hat", a "magic circle", and a group of basic THREE.js objects in the corner. Your task will be to help this wizard put on their wizard hat, move their magic circle, summon objects, and color the scene objects. You can move the camera around the scene by dragging with a mouse, pan by holding down the right mouse button while dragging, and zoom by scrolling the mouse wheel.

1.1 Template

- The file `A1.html` is the launcher of the assignment. Open it in your preferred browser to run the assigment, to get started.
- The file `A1.js` contains the JavaScript code used to set up the scene and the rendering environment. You will need to make small changes in it to answer the questions.
- The folder `glsl` contains the vertex and fragment shaders for the wizard, wizard hat, and magic circle geometry.
- The folder `js` contains the required JavaScript libraries. You do not need to change anything here.
- The folder `obj` contains the geometric models loaded in the scene.
- The folder `images` contains the texture images used.

1.2 Execution

As mentioned above, the assignment can be run by opening the file `A1.html` in any modern browser. However, most browsers will prevent pages from accessing local files on your computer. If you simply open `A1.html`, you may get a blank screen and an error message on the console similar to this:

```
XMLHttpRequest cannot load... Cross origin requests are
  only supported for protocol schemes: http, data, https.
```

Please see this web page for some option on how to run things locally:

`https://threejs.org/docs/#manual/en/introduction/How-to-run-things-locally`

We highly recommend that you run a local server, instead of changing browser security settings.

1. Follow the link `https://nodejs.org/en/` to download and install Node.js, which comes packaged with npm.

2. Open the link `https://www.npmjs.com/package/http-server` and follow the instructions to download and install a local command-line http server.

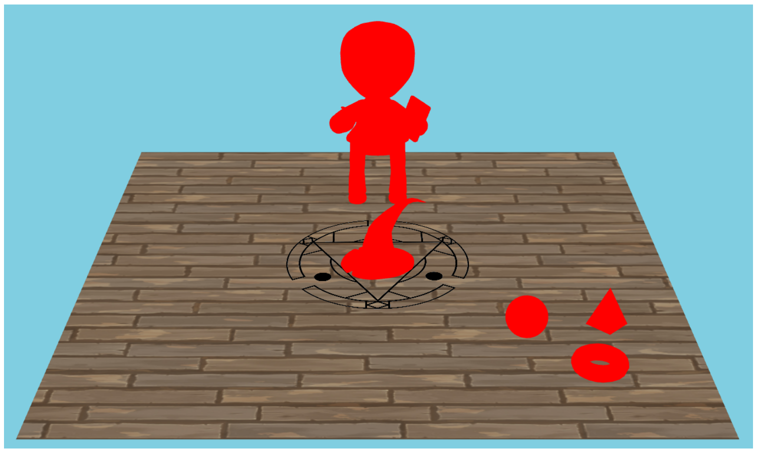
3. Go to the command-line or terminal and run `http-server [path]` where [path] is the path to the assignment folder.

4. Open your preferred browser and copy and paste the URL of the local server specified by the http-server on your command-line.

Please note that if you are using `Chrome`, you may need to do a "hard reload" or "empty cache and hard reload" in the browser to see any changes made to `.glsl` or `.js` files updated on your screen.

2 Work to be done (100 pts)

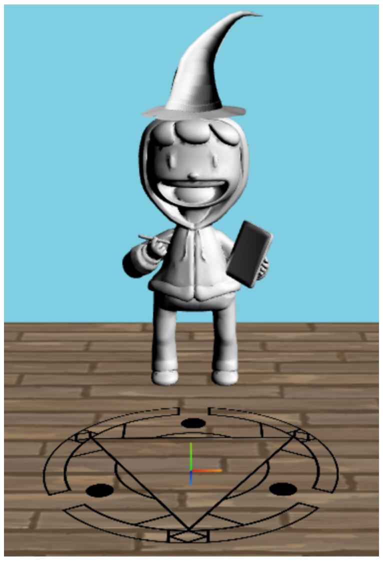
First, ensure that you can run the template code in your browser. See the instructions above, you should see the image below on your screen if the assignment is running properly. Study the template to get a sense of how it works.



1. (70 points)

   (a) **10 pts** Getting the Wizard Ready

   For this part you need to modify `A1.js` and the wizard shader `wizard.fs.glsl`. Carefully read through the provided .OBJ loading function in `A1.js`, and once you are familiar with how it works modify the parameters used to load the wizard hat `obj/hat.obj` so that the hat is an appriopriate size and placed on the wizards head. In `wizard.fs.glsl` we provide you with the shared variable `interpolatedNormal` which is passed from the wizard vertex shader to the fragment shader. Your task is to use this shared variable in the fragment shader to colour both the wizard and their hat. To do this, first read carefully through the provided notes in the shaders to get a sense of what they are doing, then

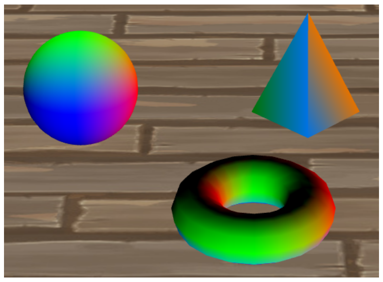


set the output color of the fragment shader to the dot product of the provided `lightDirection` and `interpolatedNormal` variables.

Hint 1: Do not simply set `out_FragColor` to the dot product, keep in mind that `out_FragColor` is a `vec4` consisting of RGBA values.
Hint 2: There is a **big difference** between writing 1.0 and 1 in GLSL, one is read a float while the other an int.

(b) **10 pts** Colourful Objects

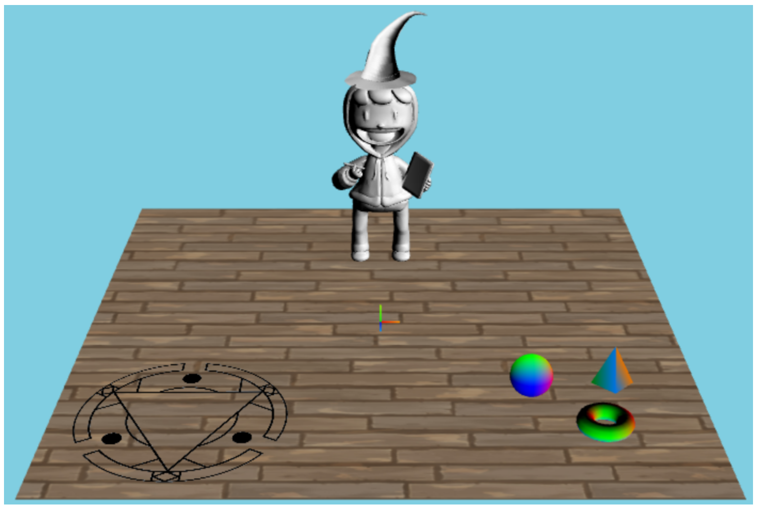
For this part you need to modify the item shaders `item.vs.glsl` and `item.fs.glsl` to color the items in the corner of the screen, and later on the items our wizard will be summoning. The items should be colored according to the normals across their surface, which can be achieved by following a similar approach in the wizard shaders `wizard.vs.glsl` and `wizard.fs.glsl`. You will need to create a new shared variable in the item vertex shader to pass interpolated normals to the fragment shader, and then use these interpolated normals for the fragment shaders output colors `out_FragColor` directly.



Hint 1: Don't forget about the alpha value of `out_FragColor`!

(c) **20 pts** Moving the Magic Circle

The variable `magicPosition` (the position of the magic circles center in world coordinates) declared in `A1.js` is changed using the keyboard, and passed to the magic vertex shader (in `magic.vs.glsl`) using `uniform` variables. Your task is to modify the magic vertex shader to move the magic circle in response to keyboard input. In order to do this, it is important to keep in mind which coordinate frame we are translating the magic circle in. If you find your magic circle is moving very slowly you are likely translating in the wrong coordinate frame!

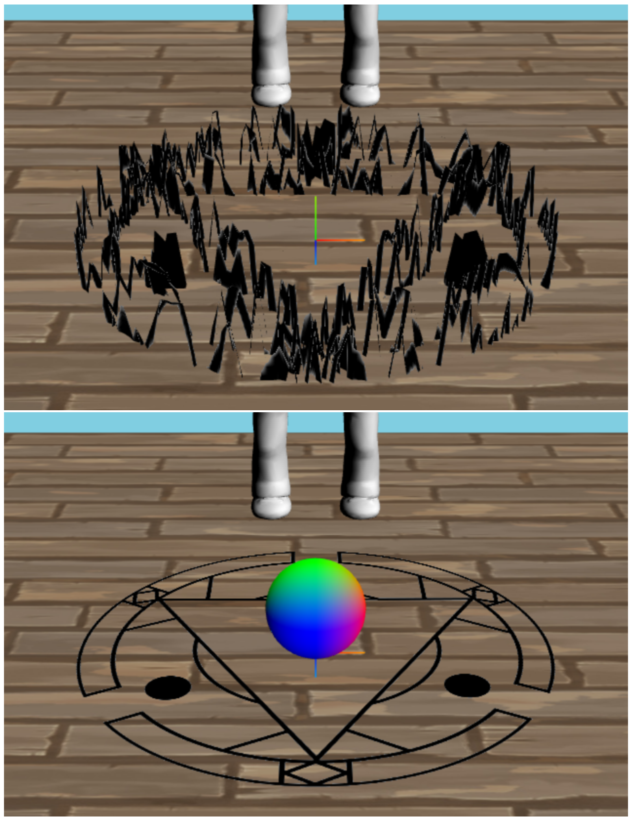


Hint 1: You may use one of the predefined transformation matrices, listed below.

```
uniform mat4 modelMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat3 normalMatrix;
```

(d) **30 pts** Magical Summoning!

Modify `A1.js` and the magic shader `magic.vs.glsl`, to make the magic circle "activate" when the `SPACEBAR` is pressed. The magic circle should stay activated for a duration of time, followed by the appearance of a random item at its center. You are free to use any built in THREE.js objects for the items, three are provided to you in the code as examples `sphere, pyramid, and torus`. The choice of which item appears **must** be randomly selected, and your implementation **must** support any number of summonings (this means that every time `SPACEBAR` is pressed one new item is created without any limitation on how many items may be eventually created). While it's activated, the magic circle should deform in an interesting way, and then stop once a set time has elapsed. It is up to you to decide how to make this deformation interesting, the only requirements are that it translates the vertices of the magic circle as a function of time.



Hint 1: You may find it useful to increment a variable on each update in `A1.js`, resetting it and/or capping it when appropriate.
Hint 2: Pressing `SPACEBAR` before a summoning has completed can simply reset its duration.
Hint 3: You should pass an appropriate uniform variable to the magic shader.
Hint 4: GLSL provides the trigonometry functions `sin(), cos(), and tan()`.
Hint 5: Distance to vertices can be used to enforce non-uniform deformations.

2. **Part 2:** (30 pts) Creative License

   For this part we want to see what you can do. Your ideas should use at least one new shader, and should be of a similar complexity to the previous tasks. If you have any doubts, make sure to OK it with a prof or TA. Some possible suggestions might be:

   - deform the vertices in more interesting ways.
   - explode models along face normals to view all the triangles that make it up.
   - change the wizard or items in some dynamic way.
   - dynamically change the colors of objects as a function of proximities or time.
   - create new shaders.

   Bonus marks may be given at the discretion of the marker for particularly noteworthy explorations.

2.1 Hand-in Instructions

You do not have to hand in any printed code. Create a README.txt file that includes your name, student number, and CWL username, and any information you would like to pass on to the marker. Create a folder called "A1" under your "cs314" directory. Within this directory have two subdirectories named "part1," and "part2", and put all the source files and your README.txt file for each part in their respective folder. Do not use further sub-directories. The assignment can be handed in on a department computer, which you can SSH into, with the exact command:

```
handin cs314 a1
```

You may also use `Web-Handin` by following this link `https://my.cs.ubc.ca/docs/hand-in`, logging in with your CWL credentials, and writing cs-314 for the course, A1 for the assignment name, and zipping your assignment folder for submission.

It is always in your best interest to make sure your assignment was successfully handed in. To do this, you may either use the `Check submissions` button in Web-Handin, or using the -c flag on the command line:

```
handin -c cs314 a1.
```