Project 3 Documentation

09/22/18

Andrew Eissen

CMSC 405 6381 (2188)

Section I: Deliverables

As per the Project 3 rubric requirements, the author's Project 3 submission includes all relevant documentation and files necessary to the running and interpretation of the program. More specifically, the package includes this documentation file in `.pdf` form, a set of four (4) HTML, CSS, and JavaScript source files (namely `index.html`, `styles.css`, `three.min.js` and `app.js`), and a `.zip` file of the relevant HD screenshot images for ease of external viewing. The author is aware that the inclusion of images in a Microsoft Word or `.pdf` file often results in grainy thumbnail images that are difficult to see, and has thus seen fit to include the relevant screenshots in the package for ease of viewing.

Section II: Design and test plan

As the author is a web developer by trade and has been for nearly four years at this point, he applied some of the elementary principles and conventional best practices of that facet of computer science as he has learned them from experience to this program. From the naming and organization of files into relevant folders in the package to the use of the conventional JavaScript Module Pattern paradigm, the author has provided what he believes to be an easily-read and readily-followed implementation of the project rubric-required design.

JavaScript is the author's "native language" of sorts, the first programming language he encountered when first learning to code after graduating from university the first time. Since first learning the language, he has come into contact with and made significant use of many different JavaScript frameworks and libraries familiar to most front-end web developers, from jQuery to Angular to KnockoutJS to RequireJS. Naturally, to help expedite some of the more tedious DOM interaction operations necessary to implement the rubric-required design in an optimized fashion, the author coded his first implementation of the Project 3 program using jQuery, a useful (though largely obsolete) JS library capable of simplifying the acquisition and manipulation of DOM elements by means of their ids or

class names and capable of handling the easy addition of event listeners as needed. A local copy of the latest minified jQuery release was thus included in the "`/js`" folder for a time, serving as a backup in the event of the external import failing for some reason (i.e. lack of Internet connection, etc.)

However, over the course of the program's development, the author came to realize that the use of jQuery was not really significantly simplifying matters to the extent required to justify its inclusion. Furthermore, the loading of an additional resource, however minified, was still considerably affecting load and render times enough for the author to notice after running some speed tests in all the major browsers. The author was initially wary of its removal, as he has not written in pure vanilla JavaScript since the earliest days of his web development career and as a result has since forgotten many of the standard DOM manipulation functions available by default. However, jQuery's removal from the contents of `app.js` was not as involved an affair as he had initially worried, and only took about an hour to replace with ES6-compliant alternatives. Its removal ultimately decreased load times as notated in the author's personal speed tests.

As far as the `app.js` file's organization was concerned, the author made use of many standard ES6 features that may not work for users of Internet Explorer 8 and below due to an inherit lack of compatibility and upstream compliance. As many modern dev shops are slowly dropping support for IE8 and other legacy browsers in favor of embracing ES6, the author made no effort to ensure backwards compatibility for legacy browsers in this project, especially given that it is not an exercise in proper web development principles but rather a study in `THREE`-mediated graphics. However, though many lovely ES6 features such as the `let` and `const` variable keywords (used to enable block scope), template literal strings, and the author's beloved spread operator were readily embraced herein, the author eschewed the use of the controversial ES6 `class` construct in favor of the tried-and-true ES5 immediately-invoked function expression (IIFE) Module Pattern paradigm.

Essentially, as JavaScript is a functional programming language, the function, rather than the class, is the essential means by which scope is created and access is handled. The Module Pattern paradigm, embraced by many dev shops, wraps the entirety of the module's contents inside an assigned IIFE that immediately calls itself and creates its own inner scope, ensuring that its contents are separated and rendered inaccessible to the outside global scope. From here, selected getter functions and Java-style `public` variables may be made publically accessible by returning an object literal containing the relevant function and variable properties, leaving the untouchable private members to handle all the inner application logic.

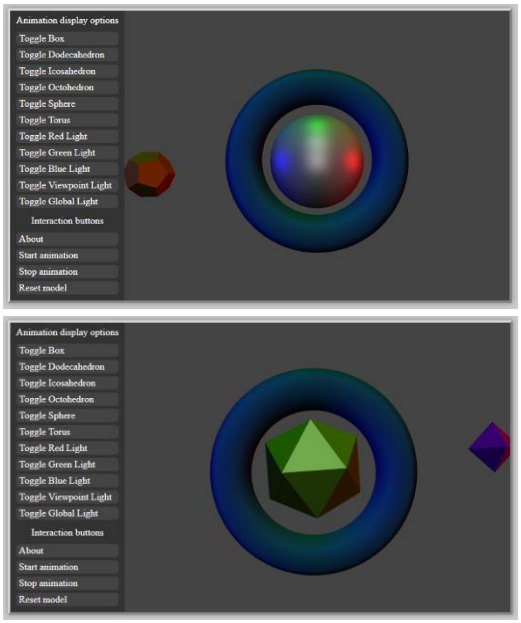
To simulate the `public` and `private` scope keywords present in Java, the author saw fit to create a pair of inner objects named `inaccessible` and `accessible` to which were added variables, arrays, objects, and functions as properties in the body of the program. As these "pseudo-access keyword" object names preceded the individual function name signatures, they took on the appearance of Java's `public` and `private` access keywords, allowing users to see at a glance what functions are globally available and which are locally restricted. As only the `accessible` object was returned from the module, only those functions (like `init`) included in that object were made externally accessible for use.

It is important to note that the author is aware of the latest ES6 module approach in which the use of the `export` and `import` keywords in an ES6 module is the means by which certain functions and variables are publically exposed; however, at the time of this program's writing, these constructs are still only beginning to see widespread browser support and usage by major organizations and shops. Under present circumstances, the author's chosen ES5-esque IIFE module approach is a safer and more common proven alternative that will work in all browsers that support the most basic of ES6 features.

As per the Project 3 design rubric, the author populated the model scene `canvas` with six total different objects included by default in the `THREE` library, along with five total different lights. As per the requirement to include a number of interactive elements in the form of checkboxes, buttons, or sliders to allow users to personally manipulate certain aspects of the scene in real time, the author included an on/off toggle checkbox for each of the objects and lights in the scene and a number of start/stop and reset animation buttons allowing users to play the default animation or reset the keystroke-mediated transformation operations to the defaults. The checkboxes were styled with CSS to appear similar to `JToggleButton`s, displaying a pressed appearance if unchecked and changing background colors accordingly to alert the user as to what scene objects are presently paused or disabled. The proper buttons are styled identically to these pseudo-toggle buttons to provide some semblance of aesthetic uniformity and consistency in the overall interface.

Additionally, the author augmented the type and number of default keystroke-mediated transformation operations available to the user. Originally, as per the Project 3 template HTML file "`modeling-starter.html`" accompanying this week's reading materials, the user could rotate the scene along its x, y, and z-axes and reset the scene to a default rotation coordinate position. However, following on from the keystroke conventions of his Project 2 submission, the author included some scale/zoom functionality used via the E and R keys and some "janky" pseudo-translation operations undertaken via the W, A, S, D, Z, and X keys as well. Though the scale and rotation operations work as intended, the translation operation was noted to warp the model a bit towards the left and right-hand sides of the canvas for reasons presently unknown to the author.

As far as the default, non-testing scene animation was concerned, the author expanded on the template file's `doFrame` and `updateForFrame` functions, expanding the types of rotation operations available to each scene object. When the main animation is playing, the six objects move and



rotate according to predetermined patterns denoted in their information objects, with some spinning on their y-axes, others moving opposite to one another on the z-axis, and still others moving on multiple axes and interacting with other objects in interesting ways. Specifically, the author designed several animated elements to move in relation to each other in a simple “basketball and hoop” design paradigm. The shiniest objects in the scene, the sphere and icosahedron objects, both move in relation to the multiple-axis-spanning torus ring in such a way that they spin right through the loop without clipping, as seen in the screenshots. This actually came about by accident during the author’s testing of the object rotations, and was further refined to remove any clipping due to size or rate of rotation.

As far as the author’s test plan was concerned, the design of the program, created and developed in accordance with the Project 3 HTML template file and made to respond primarily to user input, made testing difficult for the author. As such, he decided to go with a similar approach as that evidenced in his earlier Project 2 submission; that is, the creation of a function solely developed to run through a preset group of six (6) animations over a set interval (1600 milliseconds per transformation as with the aforementioned Project 2 method). This function, displaying the various transformation operations being performed in the browser console, was added as the primary button press handler of the “Start animation” button when the `DEBUG` `boolean` flag was set to true. Otherwise, the smoother preset animation mediated by way of the handler `inaccessible.handleFrame` was run instead.

As with the aforementioned Project 2 submission, this slower, message-accompanied function allowed the author to storyboard out the assigned transformations to be performed prior to the invocation of the method itself. This allowed him to ensure that his expected transformations matched

those of the method itself, thus displaying that everything was occurring as expected without any unexplained behavior being manifested in the model.

Section III: Test cases

Transformation	Method(s) tested	Testing process/method	Result of testing
Rotation: -0.45 on y-axis	THREE.Object3D.rotation	model.rotation.y += -0.45;	Scene swung on y-axis
Rotation: +0.30 on x-axis	THREE.Object3D.rotation	model.rotation.x += 0.30;	Scene swung on x-axis

Scale: 0.2 (zoom in)	THREE.Object3D.scale	model.scale.x += 0.2;
		model.scale.y += 0.2;
		model.scale.z += 0.2;
			Scene expands in viewport

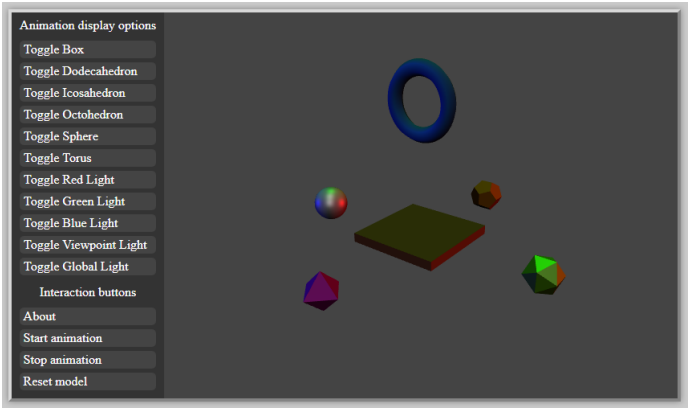
Rotation: +0.75 on y-axis	THREE.Object3D.rotation	model.rotation.y += 0.75;	Scene swung on y-axis
Translation: -2 on x-axis	THREE.Object3D.position	model.position.x += -2;	Scene shifts left on x-axis to user

Scale: -0.25 (zoom out)	THREE.Object3D.scale	model.scale.x += -0.25;
		model.scale.y += -0.25;
		model.scale.z += -0.25;
			Scene shrinks in viewport

Toggle: White lights off	ProjectThreeModule .inaccessible .handleLightSourceCheckboxChanges	Presses of "Toggle Viewpoint Light" and "Toggle Global Light"	Viewpoint and global lights are recolored to black
Toggle: Colored lights off	ProjectThreeModule .inaccessible .handleLightSourceCheckboxChanges	Presses of "Toggle Red Light," "Toggle Green Light," and "Toggle Blue Light"	Red, blue, and green lights are recolored to black

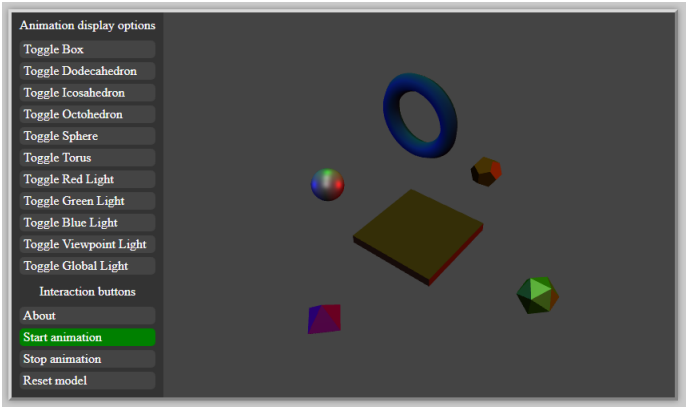
The test cases included herein were in large part based on those tested in the author's previous Project 2 submission, related to the transformation of the scene (in this specific case, the model itself) according to a set of predetermined rotation, translation, and scaling operations of set values. Included as well are tests related to presses of the toggle buttons included in the interface sidebar, specifically those concerned with the toggling of various groupings of light sources on and off. These tests handle the recoloring of the white lights and colored lights black to simulate their absence, as testing using the `Object3D.visible` property did not work as intended and bogged down the rendering times.

Test Case 1: Rotation -0.45 on y-axis



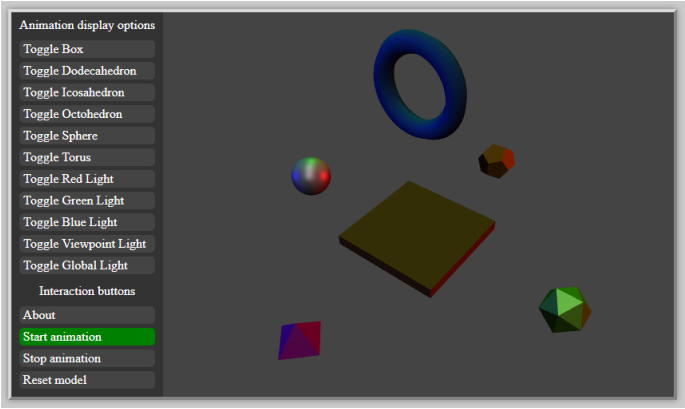
This first test case displays the result of the first of the six transformations of the testing method, namely a rotation of the model about the y-axis a value of -0.45. As expected, the model shifts to the left from the user's point of view.

Test Case 2: Rotation +0.30 on x-axis



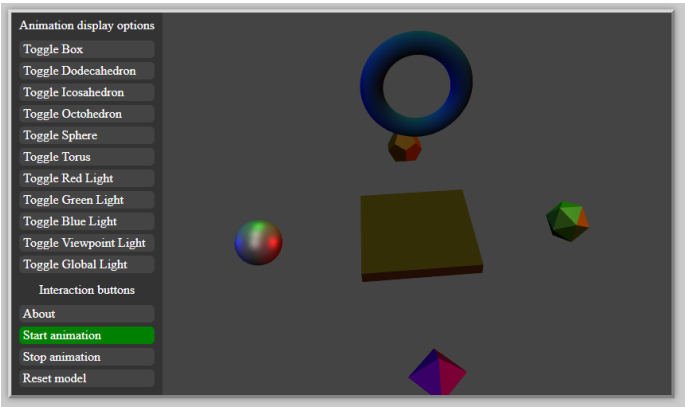
The second test case concerns another rotation transformation operation, specifically a rotation of the model on the appropriate x-axis a value of +0.30. As seen in the screenshot, the model is rotated in the positive "down" direction on the x-axis.

Test Case 3: Scale model +0.2



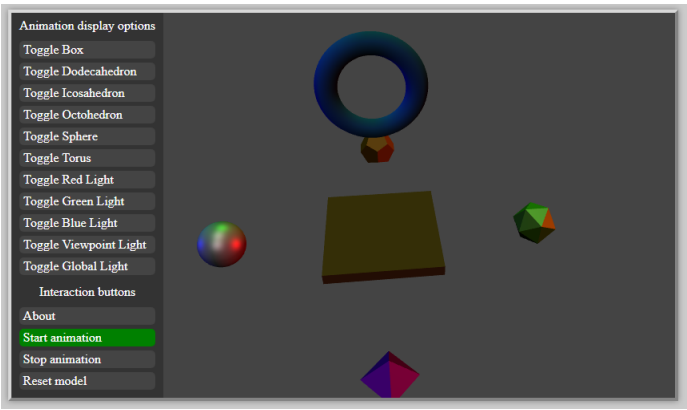
The third test case of the unit testing method tests the scaling/zooming transformation operation, in this case zooming in on the model by a factor of +0.2. As expected, the scene and model expand in the viewport and fill the screen more fully.

Test Case 4: Rotation +0.75 on y-axis



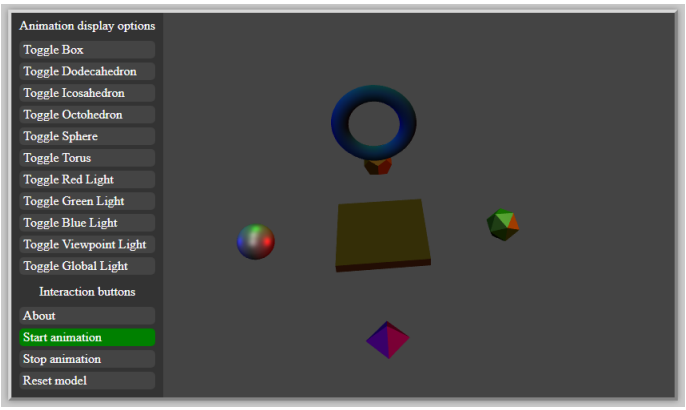
The fourth test case once again tests the rotation operation in a different context, rotating the adjusted scene on the y-axis once again, this time by a value of +0.75. As can be seen in the photo, the model is rotated on its y-axis to face the viewer.

Test Case 5: Translation -2 on x-axis



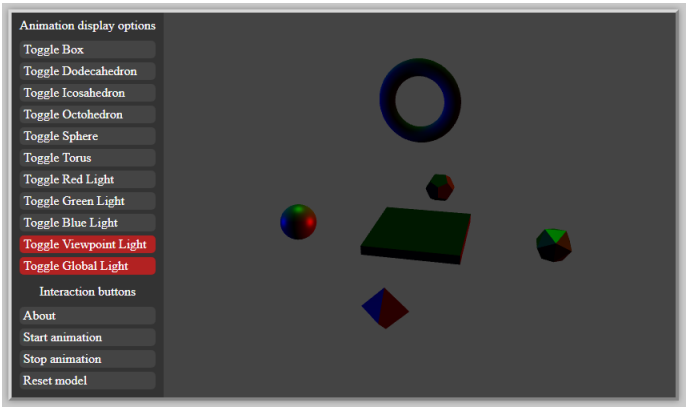
The fifth test case handles testing of the translation transformation operation, shifting the entire model along the x-axis by a factor of -2. As expected, the model appears to shift to the left from the viewer's perspective.

Test Case 6: Scale model -0.25



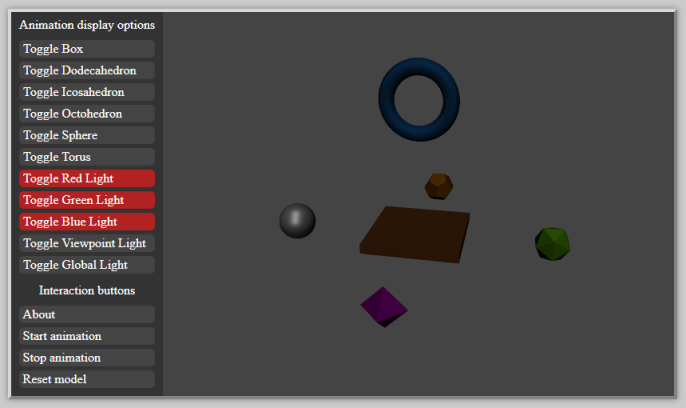
The sixth test case is the last frame of the testing function, used to once again demonstrate the scaling operation. This example zooms out of the model by a value of -0.25, giving the impression of the model receding from the view screen slightly.

Test Case 7: Toggle white lights off



The seventh test case displays the manner in which the toggle buttons may be used to affect scene elements, in this example turning off the white lights. On click of the appropriate elements, the viewpoint and global lights are recolored to black.

Test Case 8: Toggle colored lights off



The final test case once again displays the action of the toggle buttons on the scene elements, this time turning off the three primary color lights. On the press of the appropriate checkbox elements, the red, green, and blue lights are recolored to black.

Section IV: Lessons learned and potential improvements

Though the author has written a multiplicity of large web development projects and simple user scripts in JavaScript for several years at this point, he found this project to be a useful refresher in the beautiful simplicity of pure vanilla JavaScript and the specifics of its application. He has come to realize that he has perhaps become too reliant in large part on the specific syntactic sugar of the various frameworks and libraries with which he has worked in recent years and has since forgotten many of the basic JS aspects with which he used to be more familiar. This project has helped open his eyes to the unobserved gaps in his understanding of his native language and has inspired him to take another look over default JS at some point in the near future.

Further, the author was pleasantly surprised by the `THREE` library's sheer ease of use, readability, and lovingly-detailed documentation. Though there was some initial difficulty that often accompanies one's exposure to an unfamiliar library or framework, the `THREE` library was very logically organized, its functions and variables well-named, and its core well-documented, allowing the author to simply dive into coding without the OpenGL learning curve evidenced in the author's Project 2 and Week 5 discussion submissions. Right out of the box, the author was prepared and able to begin displaying images in the `canvas` element with ease. The author hopes to expand upon his knowledge of this library in the future. Despite his prior use of `canvas` elements and JavaScript-enabled animation to create in-browser flash games, he has not had much experience in the use of in-browser 3D graphics and hopes to expand upon this knowledge with further study of `THREE`.

As far as potential future improvements are concerned, the author would have liked to work out the issues related to the scene translation operation. As previously stated in the Design section of this document, the scene warps and distorts slightly when translated too far towards the edges of the canvas, resulting in a visible breakage of the scene objects themselves. As the translation function was optional and not required by the Project 3 rubric, this bug was rated significantly less important of an issue than some other bugs that occupied the author's mind during the unit testing period of the development period. All the same, he would have liked to implement some workaround but was unable to develop a fix given his current grasp of `THREE`.

Additionally, as far as the in-browser user interface is concerned, the author would have liked to expand upon the `styles.css` file to add some basic portability/responsive design support depending on device screen width and viewport scale. Though such basic support was initially planned, the implementation thereof was set on the backburner while other, more important bug fixes and issues took priority and thus never actually materialized. The author is aware that this project was primarily

concerned with the use of browser-based 3D graphics in a standard PC environment rather than the presentation thereof in devices smaller than a PC, but all the same, he has difficulty providing a less-than-stellar product when it comes to his chosen field of expertise.

Furthermore, towards the project's last days prior to submission and long after the solidification of the overall design and presentation, the author thought to add functionality allowing the user to adjust the individual light source's brightness and distance properties by hand in the form of a set of dynamic slider elements. However, as the GUI was already preassembled and the code written in such a way as to preclude such functionality without the restructuring of certain components, the author dismissed this idea and relegated it to the "could-have-done" mental archive. If he had more time to work on the program, the result would have likely been the addition of multiple light source sliders alongside the included checkboxes and buttons.

Related to this, the author similarly came up with the idea of having a slider which could be used to adjust the speed at which the non-`DEBUG` animation plays. As all the scene objects move at an `enum`-defined rate of speed or at specific fractions thereof, the author figured that the universal adjustment of this value via slider, if moved from the `enum`, could in effect speed up the entire scene depending on the user's chosen rate. However, as with the above light adjustment slider idea, the concept came too late in the development process for the author to successfully implement without refactoring certain parts of the GUI interface and the code base.

Additionally, the author tried multiple times to find room in the overall design for a status log like those evidenced in his vast user script repository and his previous two Computer Graphics project submissions. However, despite rearranging the design multiple times and even going so far as to tack on an extra `<div>` below the canvas for the log, the author could not find an acceptable design. However, given the use of colored toggle buttons to denote program functionality, the author determined the log

was not necessarily a critical element to include in the interface design, as its removal or lack of a presence would not impede the program's function as much as its absence would in a Java Swing-implemented program. Thus, the author ultimately elected to leave it out, though given more time would have likely found a place for it eventually.