Project 4 Documentation

10/05/18

Andrew Eissen

CMSC 405 6381 (2188)

Section I: Deliverables

As per the Project 4 rubric requirements, the author's Project 4 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 six (6) HTML, CSS, and JavaScript source files (namely `index.html`, `styles.css`, `gl-matrix-min.js`, `trackball-rotator.js`, `data.js` and `app.js`), two (2) `.jpg` texture images, 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

This project took significantly longer and evidenced significantly more problems than any of the author's projects to date, largely due to one main factor. The biggest lesson learned by the author over the course of the project was the inherent danger of preemptive optimization. He blames many of the project's ills on his failed attempt to optimize too soon prior to bringing the project canvas-mediated scene up to an appropriate level of completion first.

Though he started the project nearly nine days prior to the submission deadline, he spent the first few days trying to optimize the inner workings of the JavaScript portion of the code in such a way as to allow him to build upon the precedent established in his Project 3 Three.js submission, namely, defining scene objects and light sources in a set of arrays composed of configuration objects for ease of access and adjustment. However, sometime around Monday or Tuesday afternoon, he had so successfully confused himself in this attempt that he could no longer make sense of the program's functions nor even manage to render the scene properly anymore. Though it still appeared on the canvas, several hand-assembled scene objects, such as the Sierpinski triangular structure, refused to

display at all within the confines of the author's initial optimization strategy, producing a multiplicity of errors in the browser console and bugging out in unexplainable ways.

As he could no longer continue the project in its present form while still making the deadline, he was forced to trash the attempt and start anew, having bitten off more than he could chew. He returned to the starting place of his inspiration, the Project 4 template files and resources packet. Therein, he took another look at the various contents, including the suggested templates `diskworld-2.html` and `bumpmap.html`, among others. From here, prior to attempting any optimizations regarding configuration object arrays, he first constructed the scene objects and worked his way around the various functions, coming to understand the best means by which transformations could be undertaken and the most efficient ways to build multiple objects with their own unique properties. From here, despite having lost nearly five days to overhasty optimization, he was able to mostly rebuild the program functionality from the ground up, despite some issues with the application of textures from image files that persist into the present build.

For reasons unknown to the author, the application of image textures to the texture-friendly object template models could not be accomplished, despite the author’s many varied attempts to fix this problem. He tried many different approaches aimed at loading the images both synchronously and asynchronously, from hardcoding the image elements into the body of the DOM prior to the loading of any JS related to grabbing the textures by id to render blocking the JS application until the necessary boolean flags had been thrown indicating the texture had been loaded. The attempt included in the submission build of this project makes use of a technique encouraged by many StackOverflow threads and the Mozilla tutorial on WebGL; that is, apply a temporary color texture to the object while the image texture loads. Once image loading is complete, the color texture is replaced dynamically with the image. This way the object may be used in the scene while its image texture is being retrieved. However, though the program does successfully add a color texture, the images are not applied to the objects.

Initially, prior to the whole unforeseen over-optimization screw-up, the author had intended to build a scene illustrating the stable Lagrange points in a celestial orbit around a star. The scene was initially that of Jupiter (with rings) rotating around the sun with a set of Jupiter Trojan asteroids located at the stable Lagrange points L3, L4, and L5. These asteroids, taking the form of many small polyhedrons of various shapes, colors, and types, orbited the sun in a locked rotation with Jupiter at the same rate as their parent planet. However, in order to have them behave like asteroids, the author designed the scene-rendering function to iterate through the asteroid polyhedrons with each move around the sun, rotating them about their own individual axes at different rates to give the appearance of listlessly floating asteroids bereft of a planetary rotational orbit of their own.

This scene was lost with the author's decision to abandon this build iteration to start anew and avoid the pitfalls of overzealous optimization encountered during the first few days. However, many of the same ideas first implemented in the orbit model were retained in the final submission's scene. Though the author went for a more straightforward approach for the rewrite, abandoning all the overly-complex optimization code and object configuration, the rotation of the various asteroid polyhedra about their own axes was retained in much the same way as before, though bereft of the overreaching ambition the author initially possessed. Like asteroids, the polyhedra continue to float listlessly in orbit around the sun, though not in the original way the author had intended at specific Lagrange points.

As per the rubric requirement to include at least ten different shapes in the scene displaying different lighting and coloring properties of their own, the author created a set of ten polyhedron model templates from which multiple instances could be created (such as the pair of spheres present in the scene). These shapes can be colored or textured (in theory) any way as desired, with their specific specular colors and exponent properties altered to make them more or less shiny when light is reflected off their surfaces from the camera perspective. In the default non-debug scene, the ring and sphere y-axis objects and three of the four centermost objects, namely the icosahedron, dodecahedron, and

Sierpinski structure, located close at hand to the sun display high levels of shininess when sunlight or viewpoint light is reflected off their copper surfaces and into the user's eyes. The fourth element in the center, the octahedron, displays medium shininess due to its dimmer, muted gray specular color and associated exponential factor. The golden outermost elements located furthest from the center, that is, the frame cylinder, the cube, the pyramid, and the tetrahedron, possess no shininess due to their dark specular color values.

As far as unit testing was concerned, the author was initially going to implement a tester method similar to those evidenced in his Projects 2 and 3 submissions, wherein the entire scene is rotated according to a set of transformations to test the axes. However, given that in the animation itself, all objects rotate about the origin and the "sun" in a negative y-axis direction at the same rate but rotate around their own axes in disparate, often unpredictable ways, the author figured it was most important to make sure that the individual objects' rotations about their own axes were consistent with viewer expectations. As such, the author initially made use of the toggle object visibility functionality activated via the checkboxes to zoom in on a single object and make sure its rotational transformations were actually happening as required by that object's configuration properties. However, as manually removing all but one object from the scene for each new test became tedious quickly, the author implemented some debug-specific functionality to allow this to be more readily undertaken.

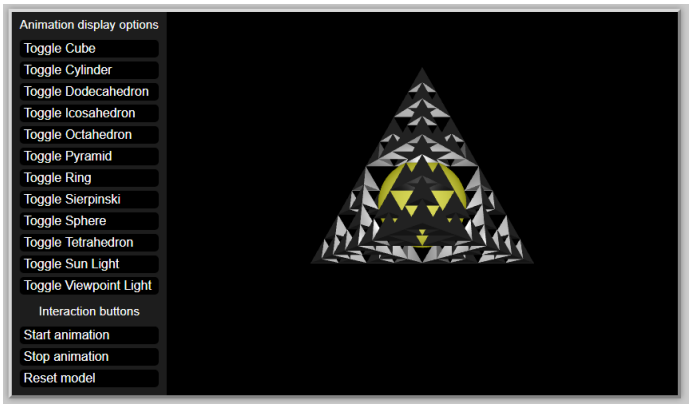
To that end, a special scene objects data array was created to house a single test object located at the center of the scene. This object, a Sierpinski triangular structure unaffected by the usual orbital rotation around the sun in the negative y-axis direction, was the perfect means by which the author could check x, y, and z-axis rotations in turn to make sure the rotations were happening in the right direction and in accordance with the author's and the user's expectations. The included animated test cases related to object transformations, though difficult to visualize and demonstrate in single-frame picture screenshots, illustrate the way in which the test object's axial rotations are undertaken.

As far as testing the light sources was concerned, the author made use of the shiny white sphere orbiting below the scene objects' main plane to determine whether or not the light sources were emanating from the proper directions. By shifting around the scene with the mouse, the author followed the light reflection on the sphere to make sure the viewpoint light was situated properly and the sunlight source reflection was moving out of view depending on the camera position. Testing of the emission light and attenuation elements was mainly done by "eyeballing" the scene with the viewpoint light switched off. The attenuation factor was adjusted manually in the appropriate light source configuration object and given different values to make sure coloring at a distance was undertaken logically and in accordance with viewer expectations.

Section III: Test cases

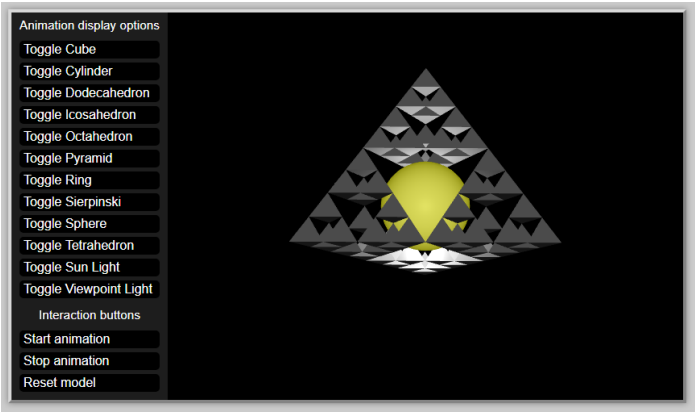
Transformation	**Method(s) tested**	**Testing process/method**	**Result of testing**
Rotation: positive x-axis	mat4.rotateX()	mat4.rotateX(modelview, modelview, (frameNumber * 5) / 180 * Math.PI);	Sierpinski swung forward on x-axis
Rotation: negative x-axis	mat4.rotateX()	mat4.rotateX(modelview, modelview, (frameNumber * -5) / 180 * Math.PI);	Sierpinski swung backward on x-axis
Rotation: positive y-axis	mat4.rotateY()	mat4.rotateY(modelview, modelview, (frameNumber * 5) / 180 * Math.PI);	Sierpinski spins right on y-axis
Rotation: negative y-axis	mat4.rotateY()	mat4.rotateY(modelview, modelview, (frameNumber * -5) / 180 * Math.PI);	Sierpinski spins left on y-axis
Rotation: positive z-axis	mat4.rotateZ()	mat4.rotateZ(modelview, modelview, (frameNumber * 5) / 180 * Math.PI);	Sierpinski spins left on camera axis
Rotation: negative z-axis	mat4.rotateZ()	mat4.rotateZ(modelview, modelview, (frameNumber * -5) / 180 * Math.PI);	Sierpinski spins right on camera axis
Toggle: Viewpoint light off	ProjectFourModule .inaccessible .handleLightSourceCheckboxChanges	Press of “Toggle Viewpoint Light”	Sunlight alone lights the scene
Toggle: Sun light off	ProjectFourModule .inaccessible .handleLightSourceCheckboxChanges	Press of “Toggle Sun Light”	Viewpoint light alone lights the scene
Toggle: Sierpinski & octahedron	ProjectFourModule .inaccessible .handleSceneElementCheckboxChanges	Presses of “Toggle Sierpinski” and “Toggle Octahedron”	Sierpinski and octahedron are hidden

Test Case 1: Rotation +5 on x-axis



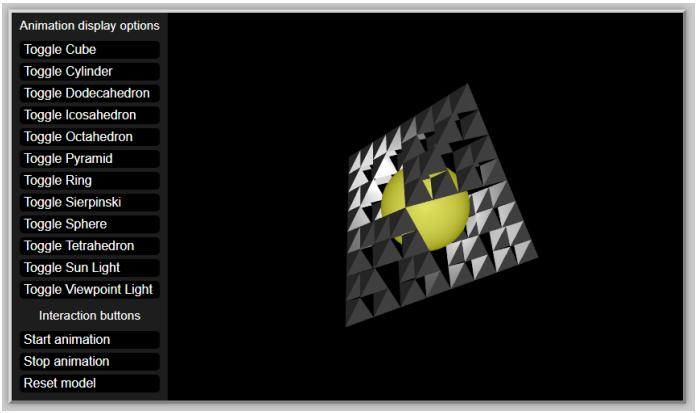
The first test case handles the test object's reaction to a required axial rotation along the positive x-axis at a rate of 5. This results in the Sierpinski's upper pyramid tip being spun forward towards the viewer as it rotates on its axis.

Test Case 2: Rotation -5 on x-axis



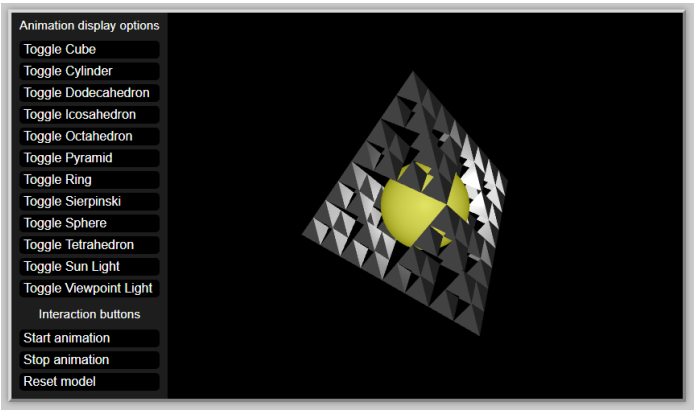
Related to the first test case, the second test case once again tests the program's reaction to a required rotation along the x-axis, this time along the negative x-axis at a rate of 5. As expected, the Sierpinski's tip rotates away from the camera on the x-axis.

Test Case 3: Rotation +5 on y-axis



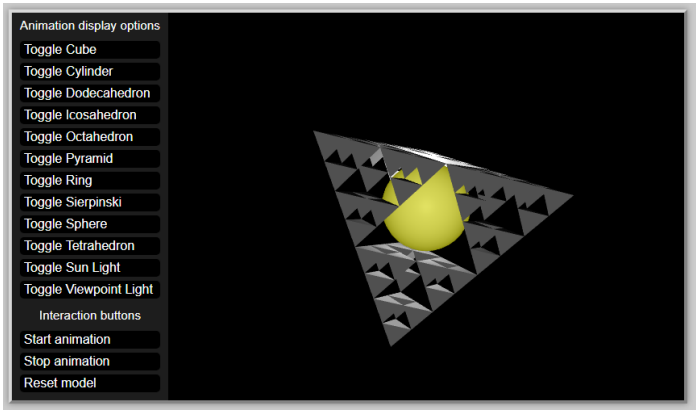
The third test case moves on to test the rotation of the Sierpinski about the positive y-axis at a constant rate of 5. As expected and evidenced in the screenshot, the structure is shifted to the right to face away from the camera on the positive y-axis.

Test Case 4: Rotation -5 on y-axis



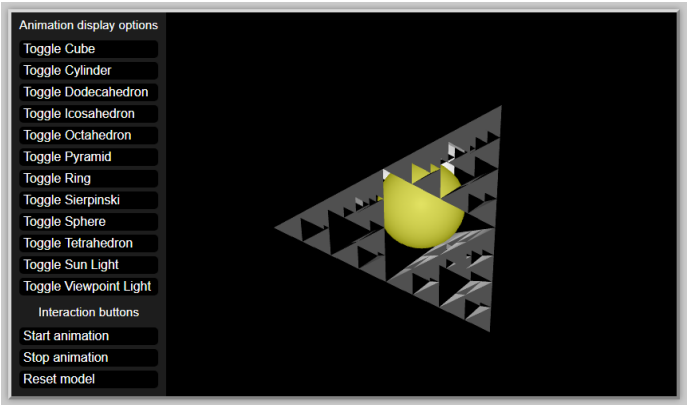
The fourth test case handles similar shifts along the y-axis as the previous test, this time moving the test shape on the negative y-axis a rate of 5. As expected, the Sierpinski shape is swung away to the left to face away from the viewpoint perspective.

Test Case 5: Rotation +5 on z-axis



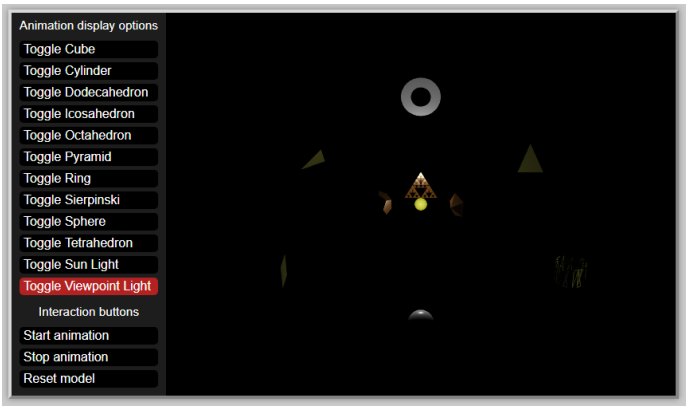
The second to last of the rotation transformation tests, this case handles shifts along the z-axis, in particular a move along the positive axis a rate of 5. As seen in the image, the Sierpinski test image is swung to the left on the z-axis as expected.

Test Case 6: Rotation -5 on z-axis



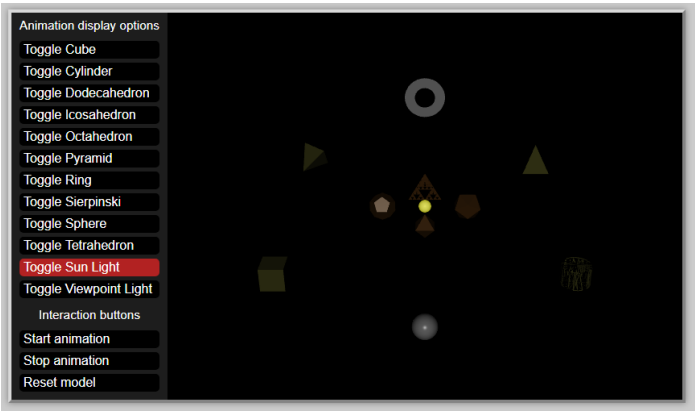
The final rotation transformation test case handles similar rotations as above, specifically a rotation of the object on the negative z-axis a rate of 5. As seen before in the previous test case, the Sierpinski is moved to the right on the z-axis.

Test Case 7: Viewpoint light toggled off



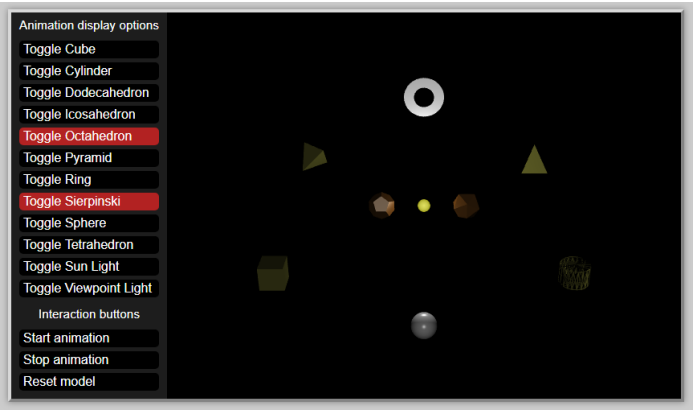
The first of the two light-toggling cases, this test case handles toggles of the viewpoint light, leaving only the central sun as the light of the scene. As can be seen, the sunlight evidenced on distant objects dissipates the further they sit from the origin.

Test Case 8: Sun light toggled off



The second of the light-toggling test cases, this test case displays the result of the toggling of the sun light, leaving only the flat, dim-gray camera viewpoint light as the source of illumination in the scene. Though properly lit, all objects are flatly illuminated.

Test Case 9: Sierpinski and octahedron toggled off



The final test case displays the result of toggling all instances of a shape type or multiple shape types off via the sidebar checkboxes. If pressed, all instances of the included shape type are hidden from view, though they can be restored with a press.

Section IV: Lessons learned and potential improvements

As discussed in the preceding sections in some detail, the greatest lesson learned (again) by the author was the inherent dangers of preemptive optimization. While there is nothing inherently wrong with wanting to clean up a template file so as to make it both better-coded and more easily followed, the overzealous approach the author evidenced during the project's early days was ill-advised, and served as an excellent illustration of the programmers' paradigm "premature optimization is the root of all evil" as first espoused by Sir Tony Hoare and Donald Knuth.

Additionally, the author learned much from the project about the specifics of the C-like GLSL shader language and the nature of its interface with JavaScript to render a set of scene objects and properties. Though he is by no means an expert now, he believes he coded his vertex and fragment shaders well enough to permit the use of multiple lighting effects and textures that respond to light in various ways. His ability to at least add a color texture to an object means that he was at least mostly successful in implementing code written in this language. If he had more time to study the specifics in the tutorials he has amassed over the week's time, he would have liked to have added some additional GLSL-mediated functionality to the project.

If he had not lost much time to the aforementioned preemptive optimizations, the author would have liked to have included some additional functionality. Originally, the author wanted to get rid of the `TrackballRotator` used to define model-view context and handle mouse-mediated scene manipulation in favor of the more dynamic keystroke-performed approach evidenced in the author's previous two submission. Though the author did expand on the rotator via the inclusion of a scaling/zooming function applied via the middle mouse button scrollwheel, he would have preferred to reduce the program's reliance on outside modules by implementing n internal, keyboard-specific approach. Additional functionality the author would have liked to add would have been an orbital speed

slider allowing the user to control how quickly the polyhedra rotate about the sun. This would have not likely been that difficult to add, but since the author was working down to the submission deadline to complete the project and was primarily focused on more pressing matters, this functionality never actually materialized in the final submission build.

Additionally, there are a few “sketchy” areas of the JS program that the author would have liked to have been able to debug if he had not lost time to his own errors. For example, on certain presses of the light-toggling checkboxes, the center sun light is rendered in inexplicably bright white tones until the canvas element is pressed or clicked by the user. The author noticed this behavior a few hours prior to project submission, and thus did not have the time to spend investigating the reason for this bug.