# Project 2 Documentation

09/07/18

Andrew Eissen

CMSC 405 6381 (2188)

**Section I: Deliverables**

As per the Project 2 rubric requirements, the author's Project 2 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 eleven (11) Java source files (namely `Driver.java`, `Application.java`, `ScenePanel.java`, `SceneObject.java`, `Floor.java`, `Cube.java`, `FiveSidedPyramid.java`, `HexagonalPrism.java`, `TriangularPrism.java`, `Star.java`, and `TenSidedPolygon.java`), 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**

From the outset, the author wanted to include more in the way of user-mediated manipulation in his Project 2 submission. Unlike the first project, which simply played through a set of predetermined transformation operations when a button was pressed, the author's Project 2 submission was deliberately designed to allow the user to manipulate the scene through the use of a series of accepted keystrokes. Though a predetermined animation displaying preselected transformations was included in the form of the "Video" `JToggleButton`, allowing the user (and author) to test the program's response to certain transformations, the scene is set by default at the program initialization to be manually manipulated by the user via a standard keyboard.

As per the precedent set by the first assignment, the author began the Project 2 construction process with the design and development of a hand-coded user GUI for the display of the `GLJPanel`-extending class instance, the status log, and the requisite manipulation buttons. The proper sizing of the entire window in order to facilitate the inclusion of a scene panel with the required 4:3 aspect ratio was

a major issue, as all the author's attempts to override the various layout managers' default sizes were unsuccessful. Thus, the author made use of a set of helper utility methods to assist in calculating the widths and heights of the non-`ScenePanel` elements, the values of which were added to the 640x480 sizing of the scene to calculate the required total size of the window. This admittedly "janky" approach could probably have been replaced with a more dynamic means of preserving the aspect ratio, but the author is unsure of how specifically to implement this without ditching the layout managers completely.

As per the first project, the author has once again included a set of interactive manipulation buttons and a user log for the display of status messages and error entries. After much fiddling with the initial Project 1-inspired GUI design, the author decided to move the buttons panel from the top of the GUI to the left of the status log, compressing the buttons' sizes and shrinking the status log width accordingly. This was done to waste as little space as possible on the interface elements and direct the user's attention and focus to the 3D scene. As a result of this shrinkage, the font size of the status log was decreased from the standard 12 pt. font to a slightly smaller 11 pt. Monospaced font.

The buttons included were the "About" button, the "Reset" button, the "Clear" button, and the "Video" button. As the name implies, the "About" button displays a popup modal in the form of an HTML-laden `JOptionPane` informing the user of what keystrokes to use to manually manipulate the 3D scene via the keyboard. The "Reset" button resets the scene to the pre-manipulation "default" perspective, reassigning constant values to the various `ScenePanel` fields and repainting accordingly. The "Clear" button does what it did in the Project 1 submission; specifically, it clears the user status log of messages. Finally, as mentioned above, the "Video" button plays through a predetermined set of transformation animations, demonstrating six different scene transformations as per the project rubric requirements. This button in particular was specifically designed to play into the author's test plan, providing a means by which the author could ensure the transformations were occurring properly and in

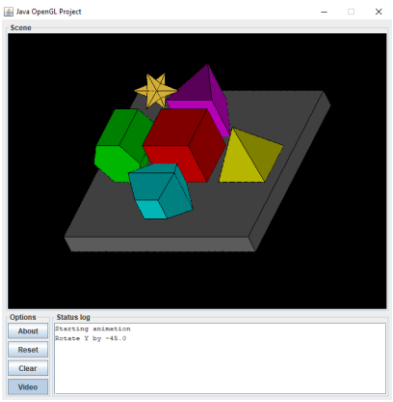
accordance with user expectations. Its use during the unit testing phase simplified the process of testing the scene transformation operations significantly for the author, as discussed below.

As far as the author's test plan was concerned, the author knew he had to ensure that if the user was allowed to interact with the scene in real time, the assorted transformation operations would have to behave in accordance with the user's expectations and not result in strange, unforeseen transformations. Thus, the "Video" button proved invaluable for testing certain transformations as the need arose over the lifetime of the program development. The author's approach generally began with a bit of mental storyboarding and doodling on paper, illustrating which way, based on the OpenGL coordinate system that displays the positive z-axis as thrusting out of the screen towards the eye, the scene is supposed to move in accordance with the transformation. Once the author had mapped out the progression, the animation was run and the results compared against the author's expectations.

**Section III: Test cases**

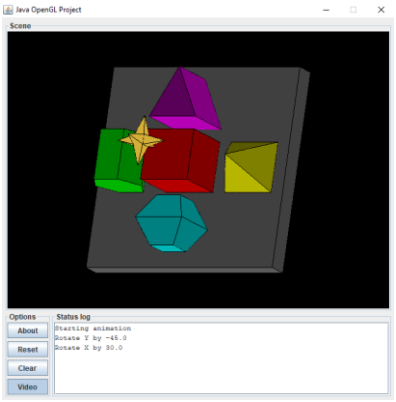
| **Transformation** | **Method(s) tested** | **Testing process/method** | **Result of testing** |
|---|---|---|---|
| **Rotation**: -45.0 on y-axis | ScenePanel .performTransformation() | ("RotateY", -45.0); | Scene swung on y-axis |
| **Rotation**: +30.0 on x-axis | ScenePanel .performTransformation() | ("RotateX", 30.0); | Scene swung on x-axis |
| **Scale**: 0.2 (zoom in) | ScenePanel .performTransformation() | ("Scale", 0.2); | Scene expands in viewport |
| **Rotation**: +75.0 on y-axis | ScenePanel .performTransformation() | ("RotateY", 75); | Scene swung on y-axis |
| **Translation**: -0.2 on z-axis | ScenePanel .performTransformation() | ("TranslateZ", -0.2); | Scene shifts up on z-axis to user |
| **Scale**: -0.5 (zoom out) | ScenePanel .performTransformation() | ("Scale", -0.5); | Scene shrinks in viewport |
| **Button**: *About* button | Application.displayDetails() | displayDetails(); | Popup modal displays |
| **Button**: *Reset* button | ScenePanel.resetScene() | resetScene(); | Scene reset to default transforms |
| **Button:** *Clear* button | JTextArea.setText() | logTextArea.setText(""); | Status log entries erased |
| **Keystroke:** *Shift* key during animation | Application.toggleButtonHandler() SceneKeyListener.keyPressed() | toggleButtonHandler (true); *~ Press of "Shift" key ~* | Error log entry displayed |
| **Keystroke:** *Shift* key | SceneKeyListener.keyPressed() | *~ Press of "Shift" key ~* | Error log entry displayed |

*Test Case 1: Rotate y-axis -45.0*



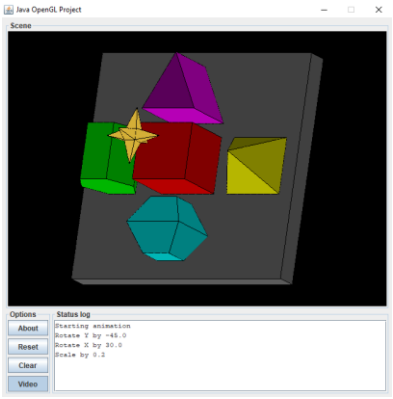

The first test case tests the program's reaction to an altered `rotateY` value of -45.0, a new value which is added to the previous `rotateY` value and set accordingly as the new composite value. As expected, the scene is rotated about the y-axis as seen in the associated screenshot.

*Test Case 2: Rotate x-axis 30.0*



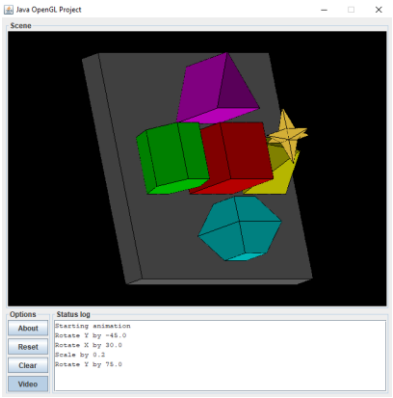

The second test case again tests the rotation transformation functionality, this time testing the `rotateX` x-axis transformation. A new value of +30.0 is added to the previous default value and set as the new `rotateX` value. As expected, the scene is rotated along the x-axis as seen in the accompanying screenshot.

*Test Case 3: Scale scene 0.2*



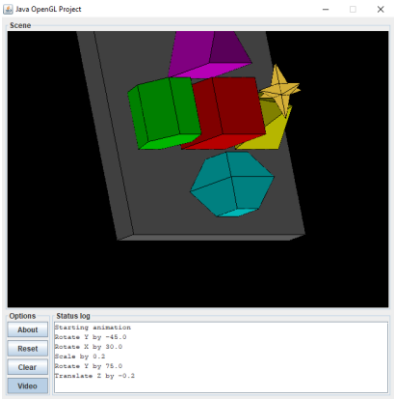

The third test case is one of two such test cases that focus on the scaling transformation function, used to zoom in or out of the scene. This particular test case scales the scene by a positive value of 0.2, in effect zooming in on the scene slightly. As before, the new value is added to the old and reassigned.

*Test Case 4: Rotate y-axis 75.0*



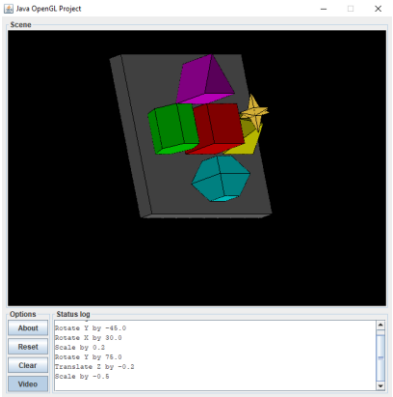

The fourth test case is a second test of the `rotateY` transformation functionality, this time adjusting the previously saved value of the field by +75.0. As expected, the scene is swung around on the y-axis to face the user's right as illustrated in the accompanying screenshot.

*Test Case 5: Translate z-axis -0.2*



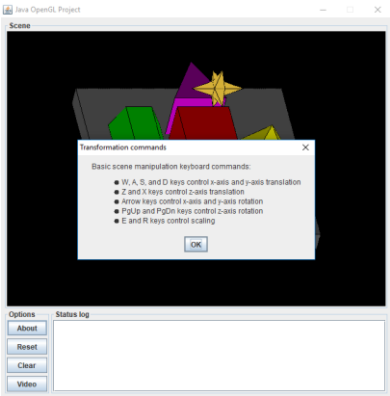

This test case is another related to the testing of required transformations, though unlike the former, this one handles cases related to translation. The scene is translated along its z-axis by -0.2, displayed in the viewport as a shift of the scene in its present state towards the top of the window on the z-axis.

*Test Case 6: Scale scene -0.5*



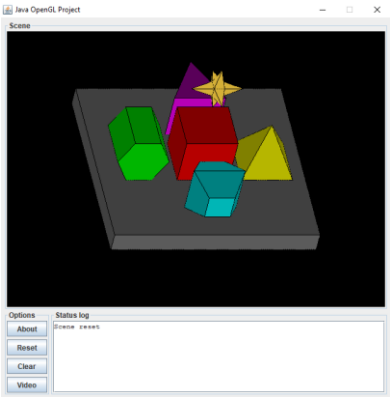

This test case is the second test case to test the scene-scaling function, zooming out and adjusting the perspective by -0.5. As seen in the accompanying screenshot, the scene appears significantly further away from the viewer prior to the scene reset that occurs at the end of the complete animation cycle.

*Test Case 7: "About" button*



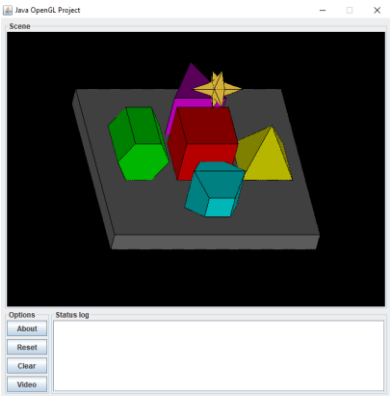

This test case is one of four that simply test the reaction of the various button presses available to the user. This one displays a `JOptionPane` containing HTML info pertaining to what keystrokes are used to manipulate what part of the scene. Loading and displaying the window is a little slow, but it works as expected.

*Test Case 8: "Reset" button*



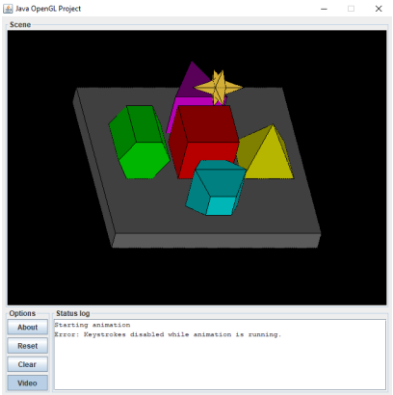

As its name implies, this test case takes a look at the "Reset" button, used to undo whatever changes to the scene perspective the user has managed to undertake and restore the scene to the perspective seen at the start of the program. As seen in the image, the scene is reset to its default slightly-skewed perspective.

*Test Case 9: "Clear" button*



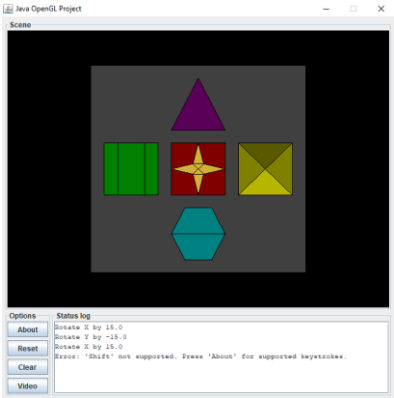

As with the author's Project 1 submission which included similar functionality related to the user GUI status log, this test case displays the program's response to presses of the "Clear" button, removing all previous log entries from the easily-cluttered log and replacing the log's contents with an empty `String` as expected.

*Test Case 10: Keystrokes & "Video" button*




This test case was used to display the message shown to the user in the event that he/she attempts to manually manipulate the scene with keystrokes while the Video animation is playing. As this is not acceptable behavior in the middle of the video, the program adds a log entry informing the user of this.

*Test Case 11: Illegitimate keystrokes*




This final test case displays the manner in which the program handles the pressing of keys without a designated function in the program. As the user should be informed that only certain keystrokes will perform the desired transformation operations, a message is displayed in the log prompting the user to press "About."

**Section IV: Lessons learned and potential improvements**

The entire Java OpenGL learning experience was one of "trial by fire" for the author, as he found the specifics of how to think in a manner required to implement the required functionality to be more difficult to grasp than the code and the implementation of the assorted OpenGL classes/methods themselves. The author has never been a particularly artsy sort, and the entire process of developing a scene in three dimensions while making use of an unfamiliar API was inordinately difficult for him. However, several hours of external research and assorted experimentation with the package template files was very useful in allowing him to personally interact with the material on a firsthand basis. Though

he is sure his code is rife with inefficiencies and unoptimized hacks evidencing a subpar understanding of the material, he is pleased that he can at least think in an elementary manner about the best way to approach the design and development of a three-dimensional scene.

As far as desired improvements are concerned, the author did not have the time or the mental fortitude to look into more advanced functionality not required by the project rubric requirements themselves, namely lighting, shadows, and the like. The author managed to implement pseudo-shadows by way of coloring certain `SceneObject` faces different shades and hues of the parameter base color based on those faces' positions relative to an unseen light source on the positive z-axis, but this was hardly a substitute for the real thing. If he had had more time to explore the Java OpenGL API functionality over the course of the entire semester, he would have liked to have added support for these sorts of features to this program in particular.

After a look over the `UnlitCube.java` template file from the Project 2 templates, the author decided to go with the second approach evidenced therein, wherein objects are assembled from multidimensional vertex and face arrays called Indexed Face Sets. While this approach worked for the author in this context, the process of assembling the vertex lists and the faces they formed by hand was often tedious and time-consuming. Though the author followed the convention that the ordering of vertices occur in a counterclockwise direction based on the front-facing face, the implementation required a fair bit of graph paper and time to compile together, resulting in the production of rather simple shapes and objects. As this process had gotten significantly easier for the author as the program developed towards production release and submission, he would have likely striven to create more complex shapes and designs from indexed face set collections if he had had more time to spend on it.

Furthermore, as per usual for the author in particular, he is unsure if all his methods are actually properly optimized and in accordance with best coding conventions for JOGL, with particular focus

placed on the `SceneObject` methods `constructObject` and `drawShape` and the entirety of the `SceneGLEventListener` inner listener class. Though the methods implemented worked as expected and did not generate errors in the course of their use, the author's grasp of how best to implement their contents was shaky at best through most of the program development process, leading him to consider that his approaches evidenced therein were perhaps less than ideal.