Assignment - 1

[1]Arjun Verma : IMT2017008
T2-20-CS 606: Computer Graphics
Course Instructors : Prof. Jaya Sreevalsan Nair
Prof. T.K Srikanth

2D Rendering and Learning WebGL

Abstract. *This technical report contains a brief overview of the methodology involved in solving the given problem statement along with the solutions to the questions posted with the problem statement.*

1. Problem Statement

The given problem statement is to create an application called the AbstractArtistApplication. The suggested application has three modes for the user to toggle with. A user begins in the default drawing mode (Mode 0). In this mode, a user can opt to either draw a rectangle, square or circle on the screen with the help of a mouse click. The point of click serves as the centre for the shape opted to draw. The user can then toggle to the next mode which is the instance transformation mode (Mode 1). Over here, a user can pick any of the shapes present on the screen and then transform the object by either translating it or scaling it. Optionally, a user can also delete the object in this mode. In the final mode which is the scene transformation mode (Mode 2), a user can rotate the entire scene either in clockwise or anti-clockwise directions. The application can be quit at any time by pressing the Esc key.

2. Methodology

In this section, we discuss the distinct characteristics of the approaches involved in designing the application.

2.1. Application Instructions

The application throws out occasional instructions and messages in the browser console for the ease of use of user. These are simple console.log messages and a sample of it can be seen below with the welcome display message 1.

2.2. Mode 0

This is the drawing mode of the application. You can choose whether you want to draw a circle, rectangle or square by pressing 'c', 'r' or 's' respectively. The user must click on the screen to draw a shape, where the point of click serves as the centroid of the shape. The shapes are created around (0,0,0) and then translated to the centroid provided. These make the application of transformations much easier. The sizes of the shapes are fixed. The rectangle maintains a side ratio of 1:2 . All shapes drawn on the screen are stored in a *primitives list*. The sample shapes are shown below 2.

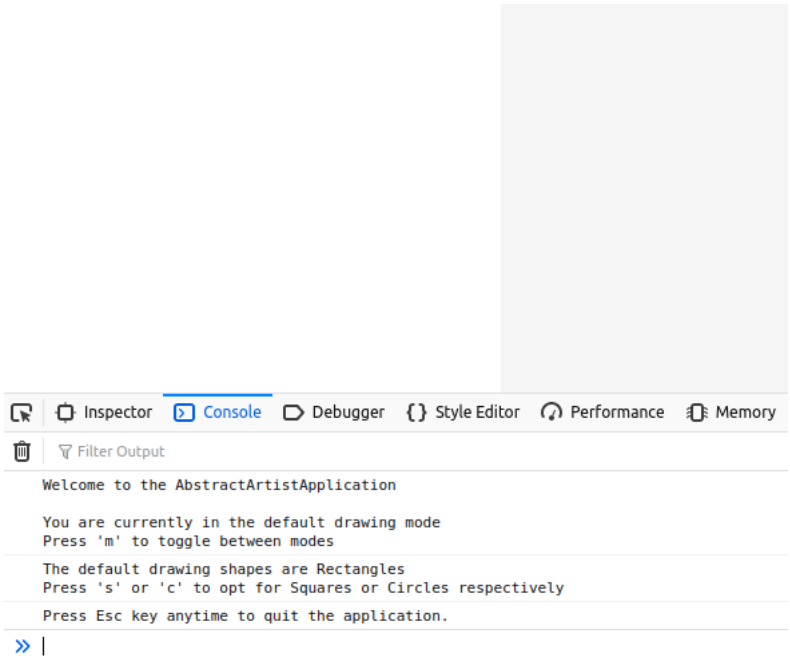

Figure 1. Welcome Instructions

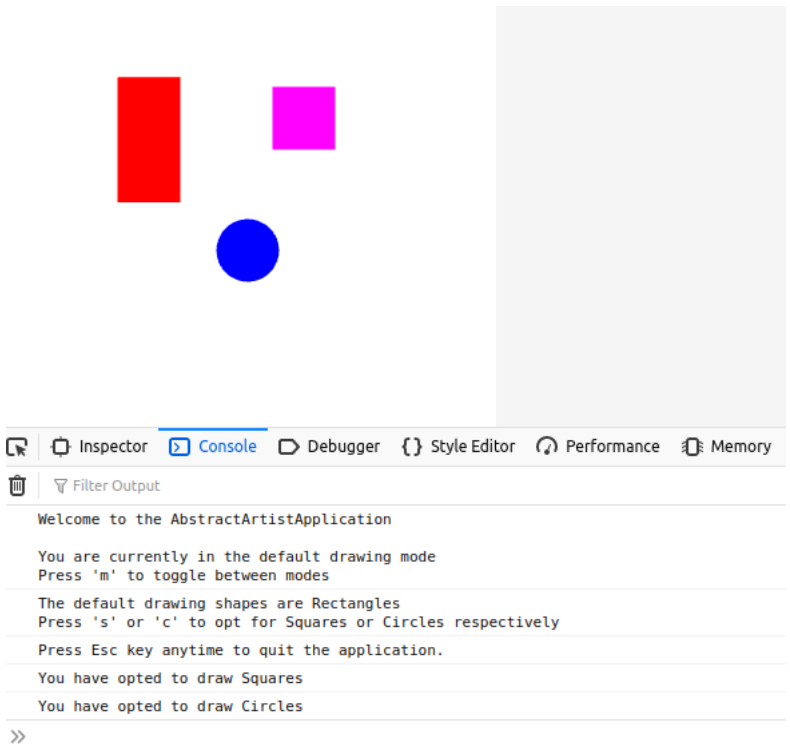

Figure 2. Sample shapes in Mode 0

2.3. Mode 1

This is the instance transformation mode. The two main components in these mode involve picking an object and transforming it. Let us look at the two components separately.

2.3.1. Picking

The objects to be chosen to transform are picked with the help of mouse clicks. When a user clicks the screen, the program computes the distances between the point of click and the centroids of all the shapes in the *primitives list*. The shape with its centroid nearest to the mouse click is shown to be selected. The picked object is depicted by changing its color to black. A sample of the picked shape is shown below 3.

2.3.2. Transforming Instances

The object that is picked can now be transformed in multiple ways. It can be translated in the right, left, up and down directions by pressing the appropriate arrow keys. It can also be scaled up or down by pressing the '+' and '-' keys respectively. The objects are transformed by maintaining a Model Transformation Matrix. As and when the user chooses the transformation options, the transformation settings are set and the Model Transformation Matrix is updated. The user can see the immediate effects on the screen. Few sample transformation displays are shown below 4, 5.

2.3.3. Deleting Instances

A user can also delete picked instances by pressing 'x' on the keyboard. When a user does this, the object is simply removed from the *primitives list* and the shape is no longer rendered. A deletion sample is shown below 6.

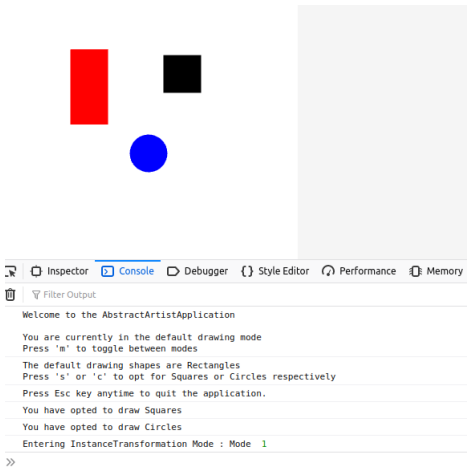

Figure 3. Picked shape is highlighted in black

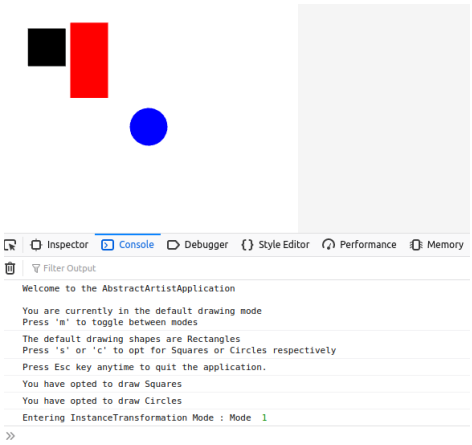

Figure 4. Picked shape is translated to the left

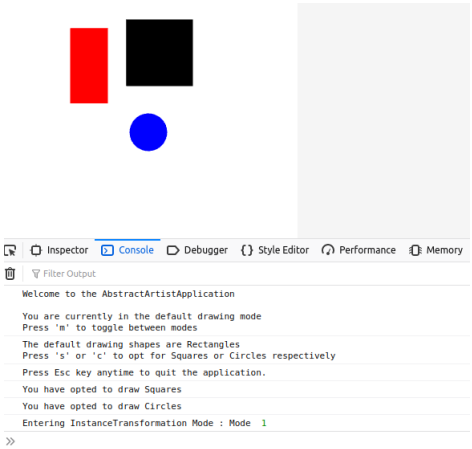

Figure 5. Picked shape is scaled up

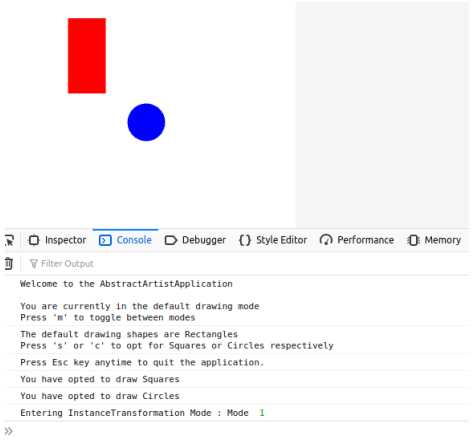

Figure 6. Picked shape is deleted

2.4. Mode 2

This is the scene transformation mode. In this mode the user can rotate the entire scene in the clockwise or anti-clockwise direction by pressing the left or right arrow keys respectively. This is done by rotating the shapes present on the screen around the centroid of the bounding box in which they are contained. The centroid is computed by keeping a track of the minimum and maximum, x and y coordinates for all the shapes on the screen. We then choose the minimum and maximum, x and y coordinates {MinX, MaxX, MinY, MaxY} from all the tracked coordinates to serve as the coordinates for the bounding box. The final centroid coordinates are thus,

$$((MinX + (MaxX - MinX)/2), (MinY + (MaxY - MinY)/2)) \tag{1}$$

All objects on screen are first translated to this point, rotated about it and then translated back to their original position. This is done by maintaining a separate World Transformation Matrix. This preserves the model transformations while a scene is being rotated. As one can see, the final position of any shape at a given time is computed as the following in the vertex shader,

$$WorldTransformation \cdot ModelTransformation \cdot Vec4Position \tag{2}$$

When a user toggles back to Mode 0 from Mode 2, the rotated scene is reset to its original position. A sample rotation and its resetting is shown below 8, 9.

2.5. Quitting the Application

A user can choose to quit the application at any time by pressing the 'Esc' key. On doing this, the screen is rendered clear and an exit message is displayed on the browser console. A sample of this is shown as below 7.

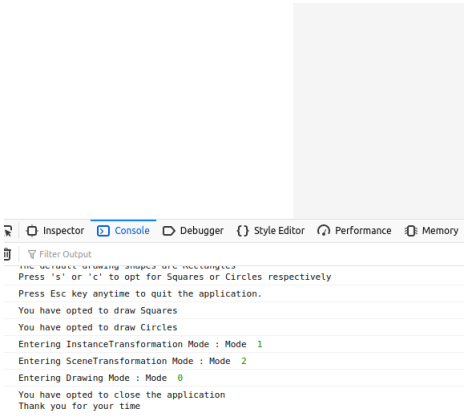

Figure 7. Quit Application

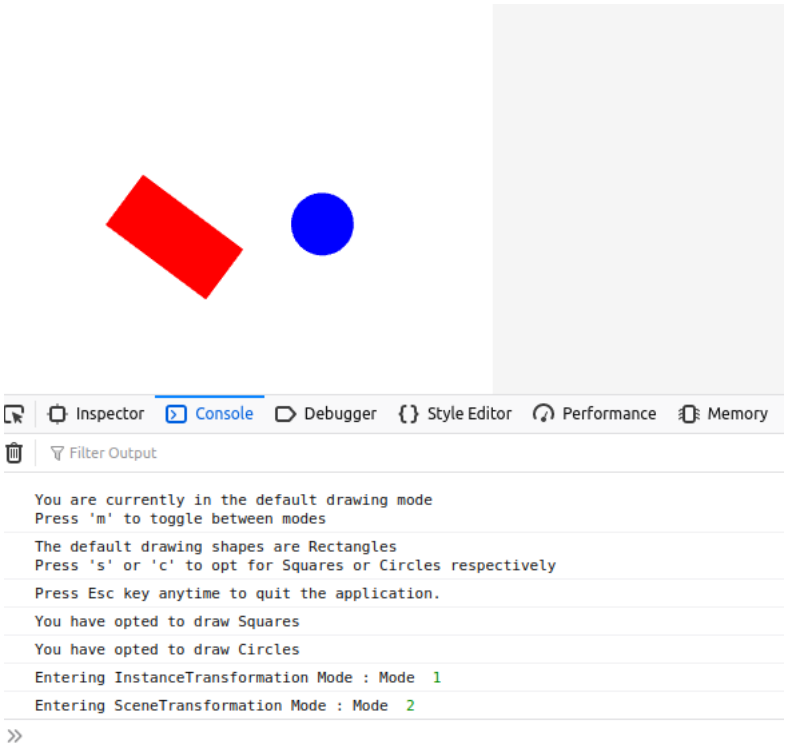

Figure 8. Scene rotation in Mode 2

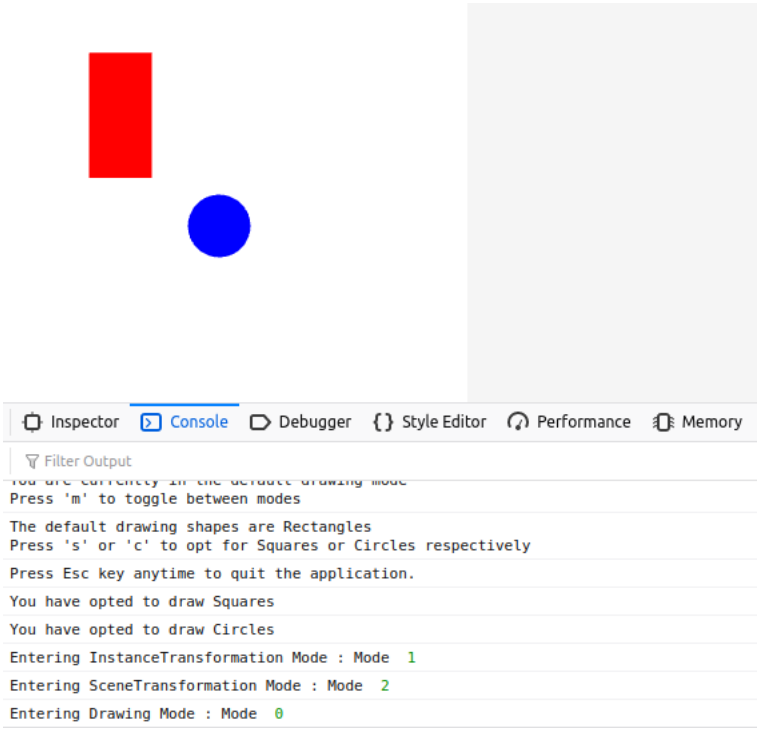

Figure 9. Scene reset in Mode 0 after rotation in Mode 2

3. Experiments

A few of the approaches used that were later improved upon in the final version of the coded are mentioned in this section.

- All drawings and transformations were first separately tried out with triangles before creating the application.
- The objects were initially translated and scaled by updating the centroids and the consequent points again and again. This was later improved upon by using transformation matrices.
- All optional implementations mentioned in the problem statement such as the inclusion of circles have been incorporated into the code.

4. Questions

The following section addresses the questions posted in the problem statement.

(1) *How did you program separate transformation matrices for all the object instances (primitives), and the scene?*
Answer. A separate transformation class was made which takes care of all the transformations for an object of a particular shape. Each shape object has its own transformation object. Each transformation object maintains two transformation matrices, *Model Transformation Matrix* and *World Transformation Matrix*. The *Model Transformation Matrix* takes care of the transformations applied to a shape object (used for translation, scaling in our application). The *World Transformation Matrix* stores a shape object's position with respect to the scene (used for scene rotation in our application).

(2) *What API is critical in the implementation of "picking" using mouse button click?*
Answer. The application designed as per this report uses the *addEventListener()* function to implement picking. The code submitted simply captures the mouse click using the aforementioned function and then computes distances between the centroids of the primitves on screen and the mouse click. The primitive with the minimum distance is then picked.

(3) *What would be a good alternative to minimize the number of key click events used in this application? Your solution should include how the mode-value changes are incorporated.*
Answer. A good way to reduce key click events by simply creating variables that keep a track of the number of times a key event occurs. This is very well captured in the way the mode change is implemented, where a single variable keeps track of the mode it is in, with each key event. Otherwise, one would have had to assign separate keys (say 0,1,2) to represent each mode. This implementation can be followed while deciding to draw objects as well. Rather than have separate 'c', 'r' and 's' keys for the different shapes,

one can implement just 1 ”change shape” key and keep a track of the number of times its been pressed with a single variable and set a shape accordingly.

(4) *Why is the use of centroid important? (Hint: Consider it with respect to rotation and scaling.)*
Answer. The importance of centroid can only be appreciated when one starts to rotate or scale a primitive. The cumulation of only a shape's centroid and its matrix transformation serves as the representation of the object. That means given only the current centroid of a shape along with its transformation matrix, you are capable of rendering the shape anywhere on screen. If one doesn't understand the importance of a centroid, he/she would inevitably end up recreating vertex arrays to render the shape for the updated vertices with each transformation of the object which is not optimal. With a centorid, one only needs to update the centroid along with its transformation matrices.

5. Conclusion

In conclusion, the report has successfully elaborated on all the required deliverables. The methodology section briefly discusses the approaches behind the implementation of the application and also displays sample images for the same. All the questions entailed in the problem statement have also been addressed. The trials and errors that were encountered have also been briefly discussed in the experiments section.