CS 606: Computer Graphics
Report for Assignment 1

Swasti Shreya Mishra (IMT2017043)
International Institute of Information Technology Bangalore
Email: SwastiShreya.Mishra@iiitb.org

1. Introduction

The aim of this assignment is to develop an application that achieves 2D planar rendering with 2D translation, rotation and scaling. The geometric shapes implemented in this project are—rectangle, square and circle, which can be created and rendered using mouse click, when the user is in the *drawing mode*. Objects can be selected based on the mouse click position and be translated in all four directions or scaled, when the user is in the *instance transformation mode*. Finally, all the objects can be rotated about the center of the bounding box that tightly fits all the objects on the scene, when the user is in the *rotation mode*.

2. Methods

The entire application is implemented using WebGL. According to the given specifications, drawing mode has *mode_value* $= 0$, instance transformation mode has *mode_value* $= 1$ and rotation mode has *mode_value* $= 2$.

When the user is in *mode_value* $= 0$, the user can select the *shape_mode* $= r$ for creating a rectangle object, *shape_mode* $= s$ for creating a square object and *shape_mode* $= c$ for creating a circle object.

When the user is in *mode_value* $= 1$, the user can "pick" (Section 2.1) an object and can translate it in all four direction and scale it, using arrow keys and $+/-$ keys respectively. The transformations done in this mode are retained for the objects. The user can also delete the object in this mode by clicking on the x key.

When the user is in *mode_value* $= 2$, the user can rotate all the objects collectively about a single point which is the centroid of the "bounding box" (Section 2.2) that tightly fits all the objects present in the scene. The objects can be rotated clockwise or anticlockwise, using left and right arrow keys respectively. The transformations made in this mode are undo-ed when the user switches to a different mode.

2.1. Picking

The picking of an object is defined as—given that there exists one or more than one objects in the scene, if the mouse click coordinates lie on an object, that object is selected; if the mouse click coordinates do not lie on any object, the object nearest (in distance) to the mouse click coordinates is selected.

Given that we maintain a list of objects in the scene in the order which they are added to the scene, the picking described above is implemented as follows:

We iterate over the list of objects from the end to the start. This is because we want to select the topmost object first. While iterating over the list of objects, we obtain the *transformed* corner positions (retained after changes made in *mode_value* $= 1$) of the object. Note that, the transformed corner positions are calculated by applying the corresponding object's transformation matrix (or the MVPMatrix) over the four different coordinates individually.

To compute the minimum distance of mouse click from a circle object, we just compute the distance between the mouse click coordinates and the transformed center and subtract the transformed radius from it. If this minimum distance is negative, it means that the mouse click coordinates lie on the object. Therefore, in that case, we return the corresponding object as the selected object.

To compute the minimum distance from a rectangle/square objects we consider the following four cases–

1) When the mouse click coordinates lie completely inside the object, we return the corresponding object as the selected object.
2) When the mouse click $x-$coordinate lies inside the left and the right edge of the object but the mouse click $y-$coordinate doesn't lie inside the top and bottom edge of the object. We set the minimum distance of mouse click from the object, as the minimum of the distances of the mouse click $y-$coordinate from the top and bottom edges.
3) When the mouse click $y-$coordinate lies inside the top and bottom edge of the object but the mouse click $x-$coordinate doesn't lie inside the left and the right edge of the object. We set the minimum distance of mouse click from the object, as the minimum of the distances of the mouse click $x-$coordinate from the left and right edges.
4) When the mouse click coordinates does not satisfy any of the above conditions, we calculate the minimum distance of mouse click from the object, as

the minimum of the distances of the mouse click coordinates from all the corners of the object.

While calculating the minimum distances for all the objects from the mouse click coordinates, we maintain the object with the least distance from the mouse click and return the object after the end of the iteration. Note that, if the mouse click coordinates lie inside the object, the function immediately returns the selected object.

2.2. Bounding Box

The bounding box for a scene is defined as—the rectangle which tightly fits all the objects and is aligned with the principal axes (x and y).

We iterate over the list of objects and maintain the minimum as well as the maximum of the $x-$coordinates and $y - coordinates$ of the corners of the objects individually, among all the objects. While doing so, we obtain the *transformed* corner positions (retained after changes made in $mode\_value = 1$) of the object. The corners for a circle object is computed as the left-most, right-most, top-most and the bottom-most points that lie on the circle object.

Finally, to compute the centroid of the bounding box, we take the mean of the minimum and maximum $x-$coordinates, this gives us the *centroidX*; then we take the mean of the minimum and maximum $y-$coordinates, this gives us the *centroidY*. Therefore, the centroid of the box on the 2D place is given by (*centroidX*,*centroidY*).

2.3. Transformations

All the transformations for the objects have been implemented using *mat4* matrix algebra functions. A transform class (in javascript) is defined to maintain a transform object for all the objects individually. These contain all the transformation operations/functions including—translation, scaling and rotation. The *MVPMatrix* gives the final transformation, after all the transformations have been combined.

Translations are implemented using the *mat4.translate()* function. Whereas, the scaling and rotation operations have to be combined with translation to achieve the desired transformation.

The scaling of an object is implemented about its center. To do so, first the origin is moved to the center of the object. Then the object is scaled. Finally, the origin is moved back to (0,0).

Similarly, for rotation of the objects is done about the centroid of the bounding box. To do so, we again move the origin to the bounding box centroid, rotate the object and move the origin back to (0,0).

3. Answers To Assignment Questions

3.1. How did you program separate transformation matrices for all the object instances (primitives), and the scene?

An Object Oriented approach is taken while implementing the program. A transform class is implemented, where we define/update attributes such as the translation coordinates, scale, rotation axis and rotation angle. This class has an attribute *MVPMatrix*, which gives the final transformations (as mentioned in Section 2.3). This class also has a method to update the *MVPMatrix* when any of the other attributes are modified.

A class has been defined for each of the primitive object—rectangle and circle. Square is just a special case of the rectangle and hence its object type is rectangle. These primitive objects have their own instance of the transform class (i.e. a transform object) as a class attribute. The transformations are stored for the objects by their corresponding transform instances. This is why, the program can separate transformation matrices for all the object instances (primitives), and the scene.

3.2. What API is critical in the implementation of "picking" using mouse button click?

The main APIs that are critical in the implementation of "picking" (Section 2.1) are:

- The API which takes mouse click coordinates from the event (from the eventListener) as input and gives us the clipped coordinates. Here, first we convert the position from pixels to make them lie between -1.0 and 1.0.
- The API that provides us with the transformed vertex positions for the objects. If we do not take the transformed vertex positions into consideration, the mouse click might correspond to some object which was previously present in that position.

Both the above APIs are critical, as otherwise there will be a discrepancy between where we click verses which object is picked.

3.3. What would be a good alternative to minimize the number of key click events used in this application?

A few alternatives to minimize the number of key click events used in the application are as follows:

- We can make every particular option implemented in the application to correspond to a single key. Since in this application, the total options are not too large, this can serve to minimize switching between modes and using more keys to achieve a goal. But one downside of this approach is that, it might not scale well when the number of total options increase. It also can get a bit confusing if every particular action corresponds to a new key.

- We can make the selection of options by using GUI features. This makes the application more interactive and intuitive for a user. We can switch between different modes (both application mode and shape mode) by clicking on their respective buttons on the screen. Once an object is selected, we can transform the objects by using sliders that map them to translation, scaling and rotation.

3.4. Why is the use of centroid important?

The use of centroid is very important as they help us implement transformations correctly. As mentioned in Section 2.3, while scaling an object, we need to move the origin to the centroid of the object in order to achieve uniform scaling of the object. Similarly, in the case of rotation, if we want to rotate an object about its center, we need to translate the origin to the centroid of the object and then rotate it and translate it back to the original position. Maintaining a centroid for every objects makes it easier for us to implement transformations.

4. Discussion

The implementation of the program has been described in Section 2. However, the implementation of "closing of the application" when the users presses the key *Escape* is different from the one specified in the assignment. The browser (both Chrome and Firefoc) versions that have been used for the implementation, do not support closing of the tab or browser through the javascript application. Therefore, when the user presses the *Escape* key, the user is redirected to the home page (which doesn't contain the drawing application). The details are mentioned in the video-demo submission.

5. Conclusion

This assignment helped us learn the basics of WebGL and programming in JavaScript. It improved our understanding of transformation operations using matrix algebra. We also learnt to use mouse and keyboard events and implemented functions that execute when an event occurs. We can now build more complex 3D objects and render them using WebGL, as implementation in a 3D space is just an extension of the 2D space, and the fundamentals stay the same.