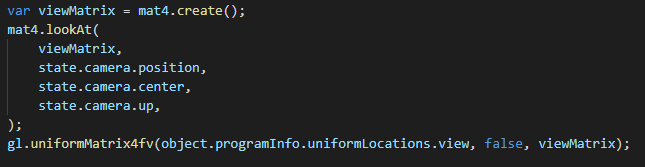
**View and view change**

* **Data**: This is set up using state. We can directly modify the state.camera’s position and center, and change the way we view the objects on our screen. Also, we can modify the viewMatrix to match the key event presses.
* **Shaders**: The vertex shader affects the way we view because depending on the value set for the uViewMatrix, our gl\_Position changes. Our uViewMatrix is what we defined as:



This sets up the view.

* **Key event handling**: The following cases interactively changes the view.  
  a and d — translate view left and right along view X

w and s — translate view forward and backward along view Z

q and e — translate view up and down along view Y

A and D — rotate view left and right around view Y (yaw)

W and S — rotate view forward and backward around view X (pitch)

These cases modifies the eye, lookAt and lookUp vectors used to form the viewing transformations. So when the user presses those keys, it will change the view according to the description listed beside the events.

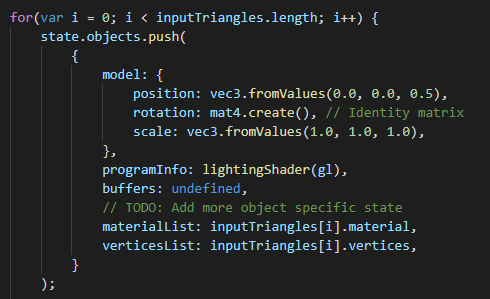
* **Render**: While we render, we are updating the camera’s position using a uniform3fc function call. This will apply any modifications (if there were any) to change how we view the objects.

**Light and light change**

* **Data**: To set this up we had to save the material’s list within our state. We can then use the material’s values by calling each object’s material list that was saved. The values will be accessed and used when the objects are being rendered.   
  We use uniform functions to bind the retrieved data from the json to the variables existing within the shaders. The data will be used within the shaders to determine the object’s color and lighting.
* **Shaders**: Within the shaders, we handle the lighting and coloring within the fragment shader (fsSource). The values are passed in through uniform values and are used to work out the math for the lighting and colors. uLight0Position is used to figure out the light direction. uLight0Colour is used to figure out the ambient, diffuse and specular. uLight0Strength is used to determine the strength of the light.
* **Key event handling:** Within the case “KeyN,” we handle the increase of the specular integer exponent. When the user presses the N key on the keyboard, the exponent value of the selected object increases by 1 (wraps from 20 to 0). This affects the light specular and spread it gets on the object.  
  Within the cases where the user presses 1, 2, and 3 keys, the object’s ambient, diffuse and specular (respectfully) changes by adding a 0.1 value to the vec3’s components. This value wraps from 1.0 to 0.0.
* **Render**: While we render the scene, we are calling the function “drawScene” and this function updates the lights using gl.uniform functions which stores the values. Then at a later time, the values will be bound to a variable that exists within the shaders and passes on the value to the shaders.

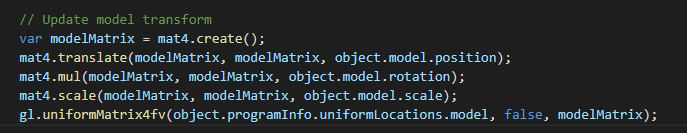
**Model selection and modeling transformation (for selected model)**

* **Data**: We use the state again to set this up. In the doDrawing function, we push the model’s information which contains the position, rotation and scale information. These will be used to modify the modeling transformation during the key event handling.



To set up the model selection part, we did not do much since we will be using the selectionIndex within the key event handling to select the respective object.

* **Shaders**: The vertex shader uses the uModelMatrix which is bound to our model matrix and affects the gl\_Position. The model matrix is updated when we render with the following code:



* **Key event handling**: The following cases interactively transforms the triangles.

k and ; — translate selection left and right along view X

o and l — translate selection forward and backward along view Z

i and p — translate selection up and down along view Y

K and : — rotate selection left and right around view Y (yaw)

O and L — rotate selection forward and backward around view X (pitch)

I and P — rotate selection clockwise and counter-clockwise around view Z (roll)

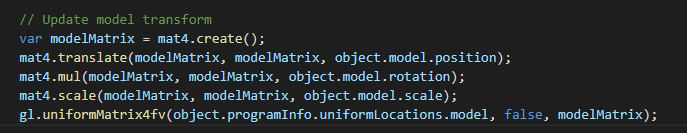
These event key presses will modify the object.model.position/ object.model.rotation based on the selected object (using selectedIndex)

left and right — select and highlight the next/previous triangle set (previous off)

space — deselect and turn off highlight

These events affect the selection of the object. Left and right uses the arrow keys to increase/decrease the selectedIndex. Space deselect the object and we can tell by uniformly scaling the selection by 20%. And when we turn highlighting off, we remove that scaling.

* **Render**: When we render the objects, we render per frame which allows for updates to the model matrix.



This code within the drawScene function updates the transformation of the models with every frame. So if any changes were made during event key presses, those code will update the information and transform the models.