CMPE360

Project 7

WebGL_2

Section 02

Erkan Sancak - Elif Aysu Kürşad

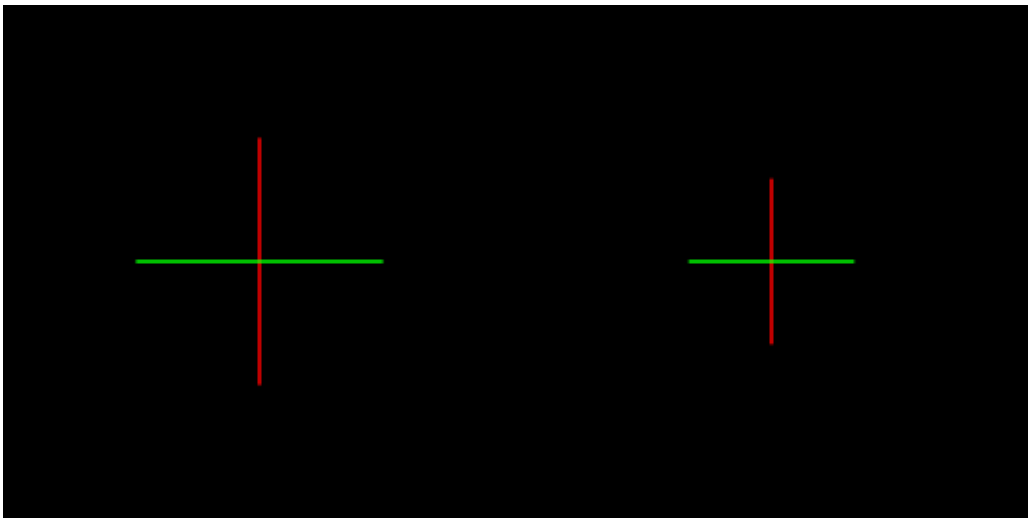
442935667 - 14162766452

Table of Contents

Part I

1. Commenting out the lookAt() call and replace it with translate().

Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12 Server at localhost Port 80



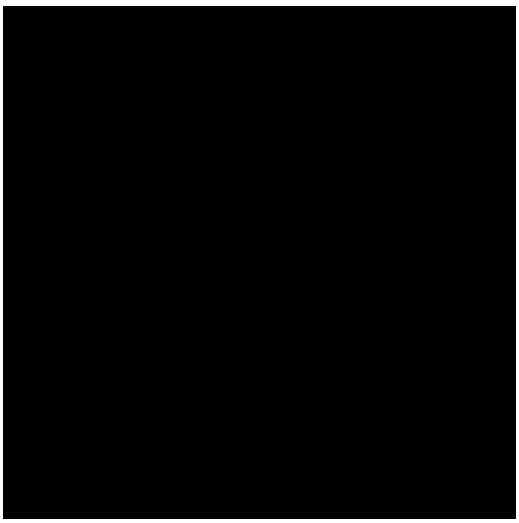
```
//mv = lookAt(eye,at,up);
mv = translate(0,0,-15);
```

After changing the view matrix to a translation with **mv = translate(0, 0, -15)**, the axes appeared smaller since they are being rendered further away from the camera's original position.

The **lookAt()** function sets up a view matrix based on an eye position, a center of interest, and an up vector, defining where the camera is and where it's looking. When it is replaced by a translation, the camera no longer looks at a point from a specific position instead, the scene is moved along the z-axis.

The effect of this translation is that the perspective of the axes relative to the camera has changed.

2. Commenting out both the lookAt() and translate() lines.



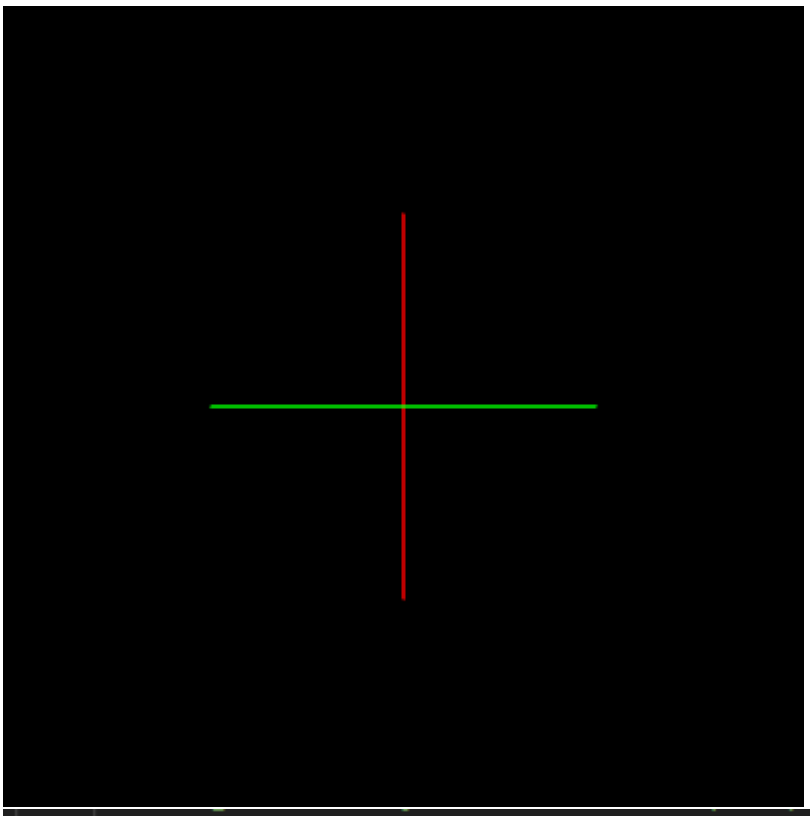
```
//mv = lookAt(eye,at,up);
//mv = translate(0,0,-15);
```

When both the **lookAt()** and **translate()** lines are commented out, the camera's view matrix is not being set, so end up with the default view matrix, which corresponds to the identity matrix.

This means the camera is positioned at the origin of the world space, looking down the negative z-axis. If axes or any other objects are not within the view frustum from this position, they will not be visible, resulting in a black canvas.

3. Restoring the lookAt() call

4. Perspective() call and Canvas Dimensions



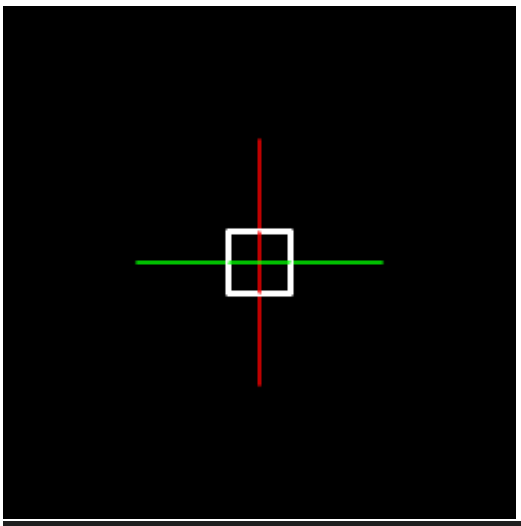
```
<canvas id="gl-canvas" width="400" height="400">
```

Changing the canvas to **400x400** while keeping the aspect ratio parameter in the **perspective()** function as **1** is consistent because the aspect ratio of the canvas width to height remains **1:1**. This means that the projection will not distort the shapes, and they will appear correctly proportioned on the canvas and will maintain the square shape of the canvas. Since the canvas size has increased to 400x400, everything within the scene will appear larger on the screen, but the relative proportions will be maintained.

If the canvas dimensions are changed to **256x256,** the canvas remains square, and the aspect ratio is still **1**. The rendering result should look identical to the previous case in terms of proportions. However, because the canvas size is smaller, the entire scene will appear smaller on the screen.

In both cases, as long as the aspect ratio parameter in the **perspective()** call matches the actual aspect ratio of the canvas, no distortion will occur, and the rendered scene should appear correctly proportioned.

5. Drawing a wireCube centered (0,0,0) relative to the axes.



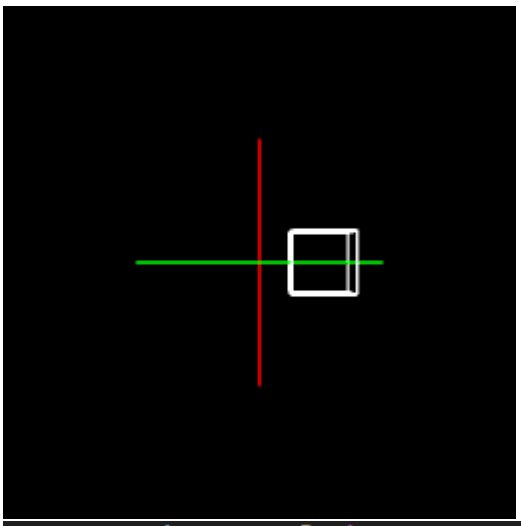
```
for (var i = 0; i < wireCubeLookups.length; i += 5) { //draws each face of the cube separately
    gl.uniformMatrix4fv(mvLoc, gl.TRUE, flatten(transpose(mv)));
    gl.drawArrays(gl.LINE_STRIP, shapes.wireCube.start + i, 5); //using LINE_STRIP
}
```

To create the wireCube, a series of line strips were drawn between the defined vertices. The **LINE_STRIP** drawing mode in WebGL was used, which connects a series of points with line segments.

A loop was implemented to draw each face of the cube using the **gl.drawArrays** function. This loop iterates through the **wireCubeLookups** index array, drawing a connected line for each set of vertices that define a face.

The vertex data for the cube was uploaded to the GPU using buffer objects. In the rendering loop, the shader program was configured to use the vertex positions and the appropriate uniform matrices to render the cube in its proper location within the scene.

6. Moving wireCube so that it is centered as (1,0,0) relative to the axes.



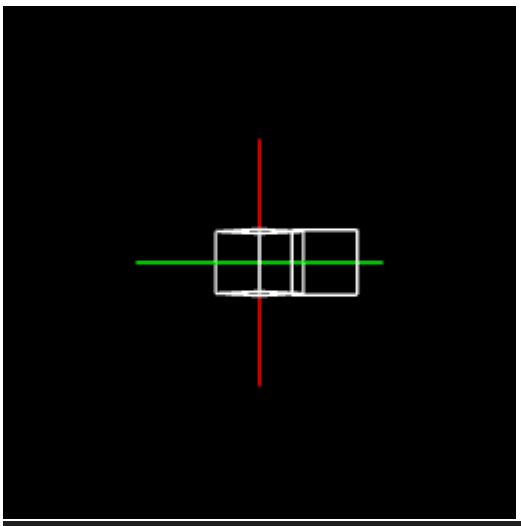
```
var mvCube = mult(mv, translate(1.0, 0.0, 0.0));
```

A translation matrix with the values (1,0,0) was created, which indicates that the cube should be moved 1 unit along the x-axis, and 0 units along the y and z axes.

The translation matrix was then combined with the model-view matrix using matrix multiplication. This alters the position where the cube will be rendered.

With the updated model-view matrix (mvCube), the cube is then drawn at its new position. The vertices of the cube are transformed by this matrix, which results in the cube appearing at the location (1,0,0) in the scene.

7. Drawing a second cube and rotating it 45 degrees around y-axis.



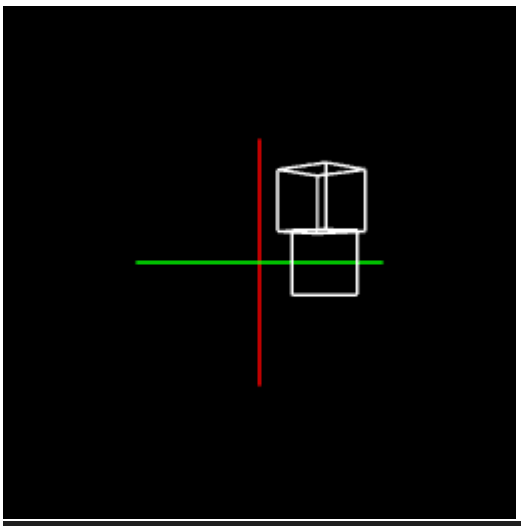
```
mvSecondCube = mult(mvSecondCube, rotate(45, [0, 1, 0])); // rotates the cube 45 degrees around the y-axis
for (var i = 0; i < wireCubeLookups.length; i += 5) { //draws each face of the cube separately
    gl.uniformMatrix4fv(mvLoc, gl.TRUE, flatten(transpose(mvSecondCube)));
    gl.drawArrays(gl.LINE_STRIP, shapes.wireCube.start + i, 5); //using LINE_STRIP
}
```

A rotation matrix for a 45-degree rotation around the y-axis was generated using a rotation function (e.g., rotate(45, [0, 1, 0])). This matrix was then multiplied with the model-view matrix to apply the rotation to the second cube.

With the model-view matrix now containing the rotation for the second cube, the cube was rendered using the **gl.drawArrays** function.

The vertices were processed by the vertex shader using the model-view matrix, which positioned and oriented the cube correctly in the scene.

8. Placing this rotated cube directly above the first cube.

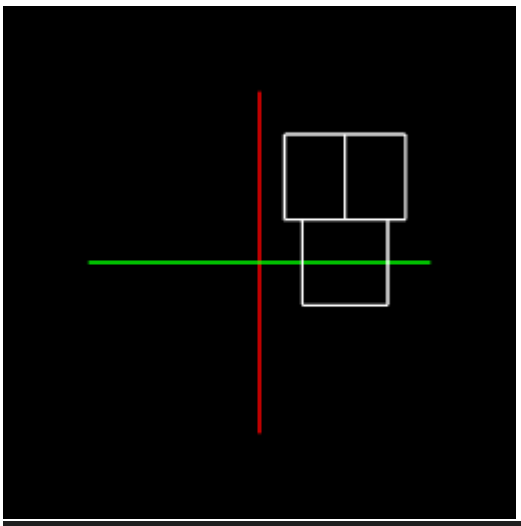


```
var mvSecondCube = mult(mv, translate(1.0, 1.0, 0.0));
```

The **translate(1.0, 1.0, 0.0)** function creates a translation matrix that moves the second cube 1 unit along the x-axis and 1 unit along the y-axis, with no change along the z-axis. The resulting position is directly above the first cube, which is already centered at (1,0,0).

The transformations are combined then, using matrix multiplication **(mult)** to ensure the cube is at desired position. The second cube is drawn directly above the first cube, rotated as expected.

9. Using orthographic projection instead of the perspective call.



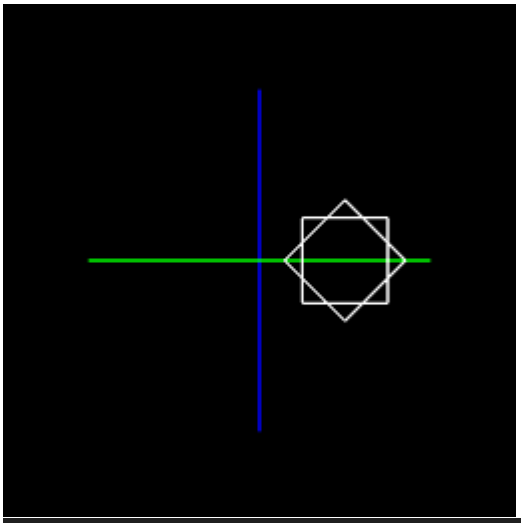
```
//Set up projection matrix
p = ortho(-3.0, 3.0, -3.0, 3.0, -1.0, 100.0);

gl.uniformMatrix4fv(projLoc, gl.FALSE, flatten(transpose(p)));
//left,right,bottom,top,near,far
```

The rendering of the scene was shifted from a perspective projection, which simulates the depth perception of the human eye, to an orthographic projection, which displays all objects at the same scale regardless of their distance from the camera. This change was made to address the awkward appearance of the two cubes when viewed with a perspective projection, which could distort their proportions.

The orthographic projection was implemented using WebGL's **ortho** function, which defines a box-shaped viewing volume where each side is parallel to the corresponding axes. The parameters of the ortho function specify the coordinates for the left, right, bottom, top, near, and far clipping planes of this volume.

The result is a scene where the cubes are displayed without perspective distortion, appearing directly comparable in size and aligned with the axes. The cubes' edges remain parallel, and their relative positions are accurately represented, providing a clear, undistorted view.

10. Rotating everything so that looking down at the top of the boxes and seeing the z-axis.



```
mv = mat4();
// Rotation around the x-axis to look down at the top of the boxes
mv = mult(mv, rotate(-90, [1, 0, 0]));
```

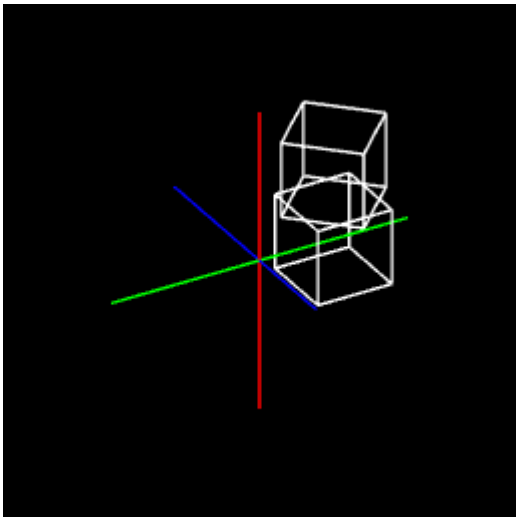
A rotation matrix was created to rotate the scene around the x-axis by -90 degrees. This transformation aligns the y-axis with the view direction (pointing directly into the screen) and makes the z-axis point upwards.

With the model-view matrix updated to reflect the top-down orientation, the axes and cubes are drawn using this transformed view. As a result, the blue z-axis appears vertical on the screen, and the y-axis is invisible from this perspective.

The model-view matrix is applied to each object in the scene. First, it positions the first cube at (1,0,0), and then a second model-view matrix is computed to position the second cube above the first and rotated by 45 degrees around the y-axis.

The final rendered image displays the scene with a top-down orthographic view. The two cubes are shown in correct relative position and scale, with the blue z-axis prominently displayed and the red y-axis effectively hidden from view

11. Rotating everything so that all three axes along with the cubes can seen.



```
// Rotations
mv = mult(mv, rotate(-30, [1, 0, 0]));
mv = mult(mv, rotate(-30, [0, 1, 0]));
```

The scene's orientation was altered using two key rotations. These rotations were applied to the model-view matrix (mv), which determines how objects are transformed in the scene.

The first transformation was a rotation around the x-axis by -30 degrees. This tilts the scene, effectively lowering the camera's angle relative to the horizontal plane, thereby making the top faces of the cubes and the z-axis more visible.

The second transformation was a rotation around the y-axis, also by -30 degrees. This rotation pivots the scene around the vertical axis, bringing the x-axis into a more prominent view and ensuring that all three axes can be seen together with the cubes.

The combined effect of these rotations positions the one can see the scene from a vantage point that reveals the three axes intersecting at the origin point. Both cubes are also clearly visible, with their spatial relationship to the axes and to each other clearly presented.