**Kaitlyn Zahn**

***Assignment 4 Report***

**March 18, 2021**

**\_\_**

**CS 4610 Computer Graphics 1**

**\_\_**

**Ye Duan**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Link to YouTube Video:**

https://youtu.be/s14-iPLbvxw

**First-Person Camera Controller**

***Add perspective projection matrix***

/\*index.html line 15 and index.js lines 110-116 and index.js lines 192-209\*/

This code creates a projection matrix and calls the function perspective (declared in Common) that takes in the angle, the ratio of the canvas width to the height, and the clipping size. This shows that at .01 or 1000, the cube will begin to be clipped. Then, the matrix is sent to the shader. The slider and fovText elements are identified and an event listener is created for when the slider changes. When it does, the label is changed, and the perspective function is called with the correct fov. Then, it is sent to the shader to update.

uniform mat4 uProjection;

// set up projection matrix

// takes the angle, the ratio of width & height, and nearest and farthest thing you're able to see

var projection = perspective(90, canvas.width / canvas.height, 0.01, 1000);

// send projection matrix to shader

var projectionLoc = gl.getUniformLocation(program, "uProjection");

gl.uniformMatrix4fv(projectionLoc, false, flatten(projection));

// create variable for the slider and text

var slider = document.getElementById("FOVSlider");

var fovText = document.getElementById("FOVText");

// change the text when the slider is moved

slider.addEventListener("input", function (event) {

var fov = slider.value;

fovText.innerHTML = "FOV: " + fov;

var projection = perspective(fov, canvas.width / canvas.height, 0.01, 1000);

// send projection matrix to shader

var projectionLoc = gl.getUniformLocation(program, "uProjection");

gl.uniformMatrix4fv(projectionLoc, false, flatten(projection));

});

***Add view matrix to simulate the position and rotation of a virtual camera***

/\*index.html line 16 and index.js lines 119-128\*/

This code creates the cameraPosition and cameraRotation matrix. Then, it creates the view matrix by multiplying the two of them (this give you the view of the “camera”). Because we’ll move the cube and not the camera, we use the inverse function to give the illusion that the camera is moving in the opposite direction. Finally, we send the view matrix to the shader.

uniform mat4 uView;

var cameraPosition = mat4();

var cameraRotation = mat4();

// set up view matrix

var view = mult(cameraPosition, cameraRotation);

view = inverse(view);

// send view matrix to shader

var viewLoc = gl.getUniformLocation(program, "uView");

gl.uniformMatrix4fv(viewLoc, false, flatten(view));

***Move the mouse to rotate the camera***

/\* index.js line 64 and lines 225-242\*/

This code creates an event listener for when the mouse moves. When the mouse goes down, the pointer locks. If it moves, it checks whether the pointer is locked or not, and if it is, the cameraRotation value is changed based on the movementX and movementY values. When the mouse is released, the pointer lock is exited.

canvas.requestPointerLock = canvas.requestPointerLock || canvas.mozRequestPointerLock;

// mousemove event listener

canvas.addEventListener("mousemove", function(event) {

if(document.pointerLockElement === canvas || document.mozPointerLockElement === canvas) {

if(event.buttons == 1) {

console.log(event.movementX + " " + event.movementY);

cameraRotation = mult(rotateY(-event.movementX), cameraRotation);

cameraRotation = mult(rotateX(-event.movementY), cameraRotation);

}

}

})

canvas.addEventListener("mousedown", function(event) {

canvas.requestPointerLock();

})

canvas.addEventListener("mouseup", function(event) {

document.exitPointerLock();

})

***Use WASD to move forward/left/back/right***

/\* index.js lines 134-162\*/

This code creates the global vectors using a vec4 (with a w value) for right, forward and up and then translates those to vec3 values, as they need to be send to the mult() function. It uses a series of conditional statements and the KeyIsPressed functionality to identify which key is pressed. Then, the cameraPosition matrix is changed by multiplying it with the translation of the direction the key makes the cube go. This ensures that it moves in the correct local/global direction. Then, as shown in the following page, the view is updated.

var right = mult(rotation, vec4(.05, 0, 0, 1));

right = vec3(right[0], right[1], right[2]);

var forward = mult(rotation, vec4(0, 0, .05, 1));

forward = vec3(forward[0], forward[1], forward[2]);

var up = mult(rotation, vec4(0, .05, 0, 1));

up = vec3(up[0], up[1], up[2]);

// move forward using W

if(KeyIsPressed("KeyW")) {

console.log("W");

cameraPosition = mult(translate(forward[0], 0, forward[2]), cameraPosition);

}

// move left using A

if(KeyIsPressed("KeyA")) {

console.log("A");

cameraPosition = mult(cameraPosition, translate(right[0], right[1], right[2]));

}

// move back using S

if(KeyIsPressed("KeyS")) {

console.log("S");

cameraPosition = mult(translate(-forward[0], 0, -forward[2]), cameraPosition);

}

// move right using D

if(KeyIsPressed("KeyD")) {

console.log("D");

cameraPosition = mult(cameraPosition, translate(-right[0], -right[1], -right[2]));

}

***Use Space/LeftShift to move up/down***/\* index.js lines 163-172 and 173-180\*/

This code uses conditional statements and the KeyIsPressed functionality to identify if “Space” or “ShiftLeft” is clicked. If it is, the cameraPosition is multiplied by the translation (up and down) and then the view is calculated by multiplying the position and rotation, it is inversed, and then finally it is updated.

// move up using space

if(KeyIsPressed("Space")) {

console.log("Space");

cameraPosition = mult(translate(0, 0.02, 0), cameraPosition);

}

// move down using left shift

if(KeyIsPressed("ShiftLeft")) {

console.log("Shift Left");

cameraPosition = mult(translate(0, -0.02, 0), cameraPosition);

}

// set up view matrix

var view = mult(cameraPosition, cameraRotation);

view = inverse(view);

// send view matrix to shader

var viewLoc = gl.getUniformLocation(program, "uView");

gl.uniformMatrix4fv(viewLoc, false, flatten(view));

**Figures:**

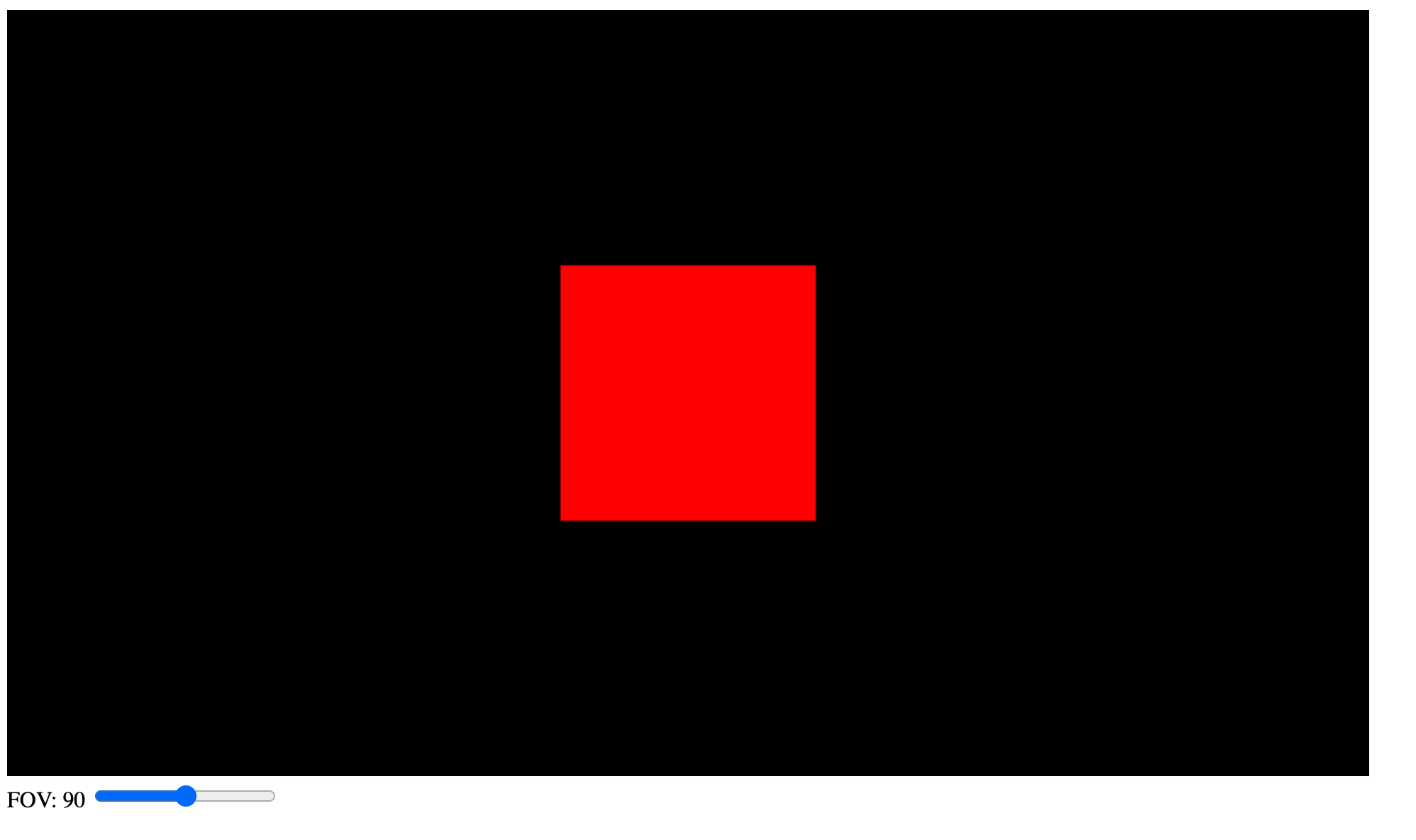
****

Figure 1 Original Cube

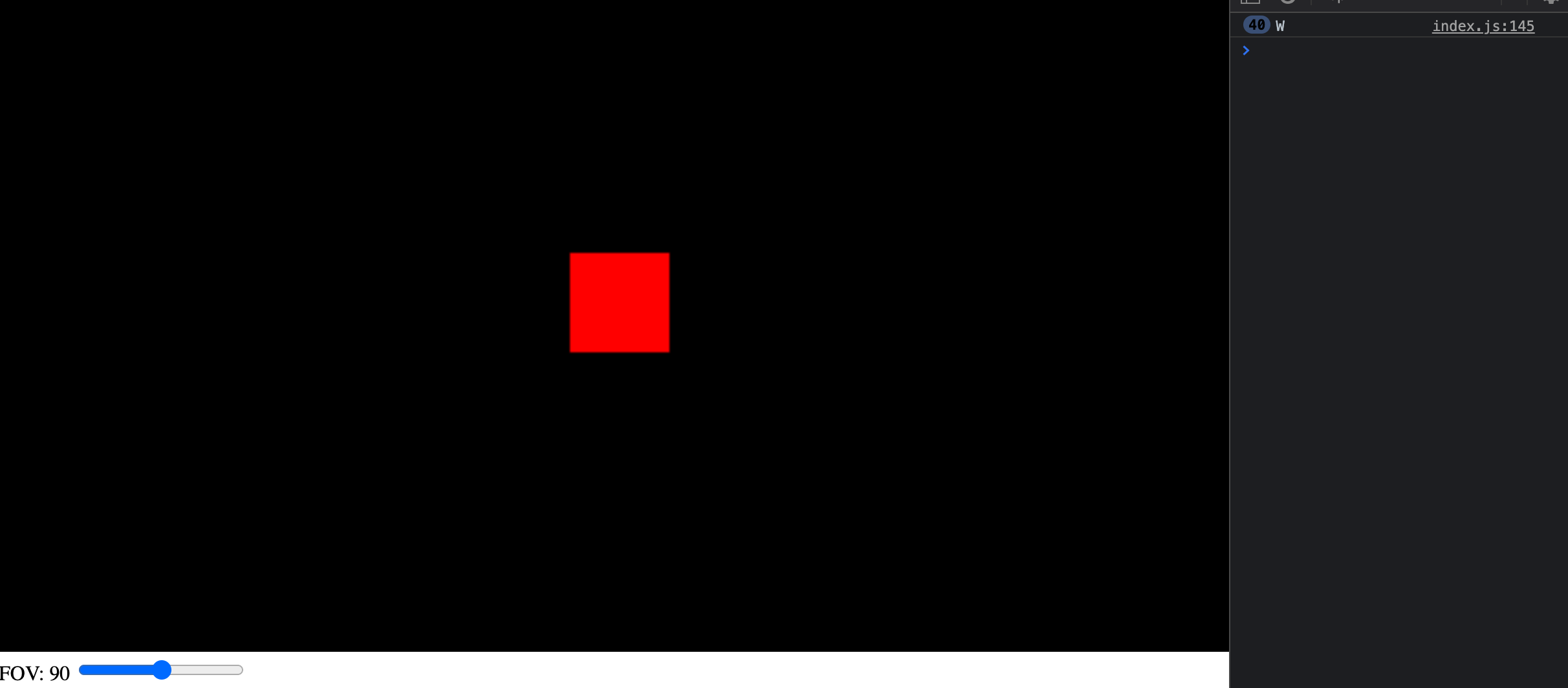
****

Figure 2 W Functionality Cube

****

Figure 3 A Functionality Cube

****

Figure 4 S Functionality Cube

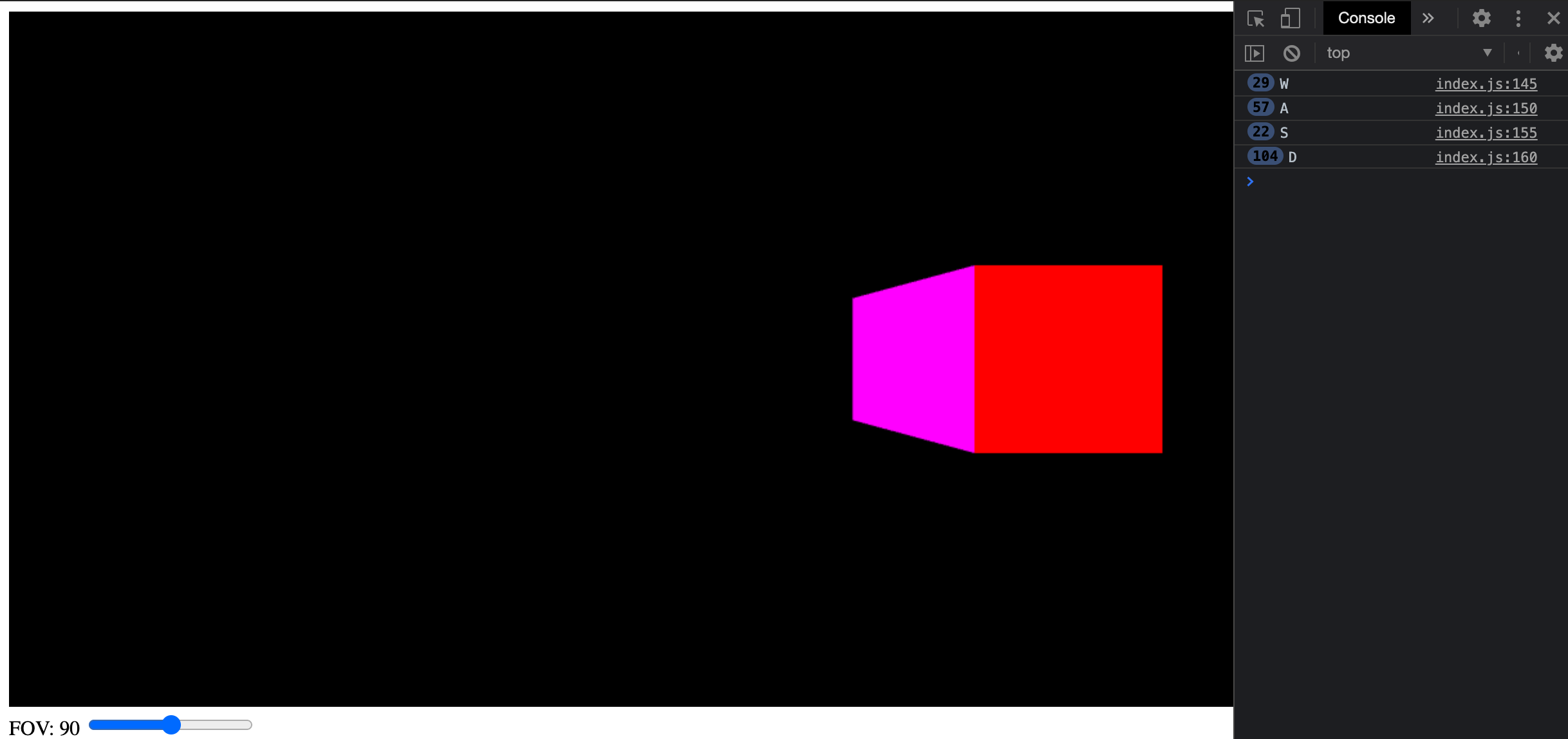
****

Figure 5 D Functionality Cube

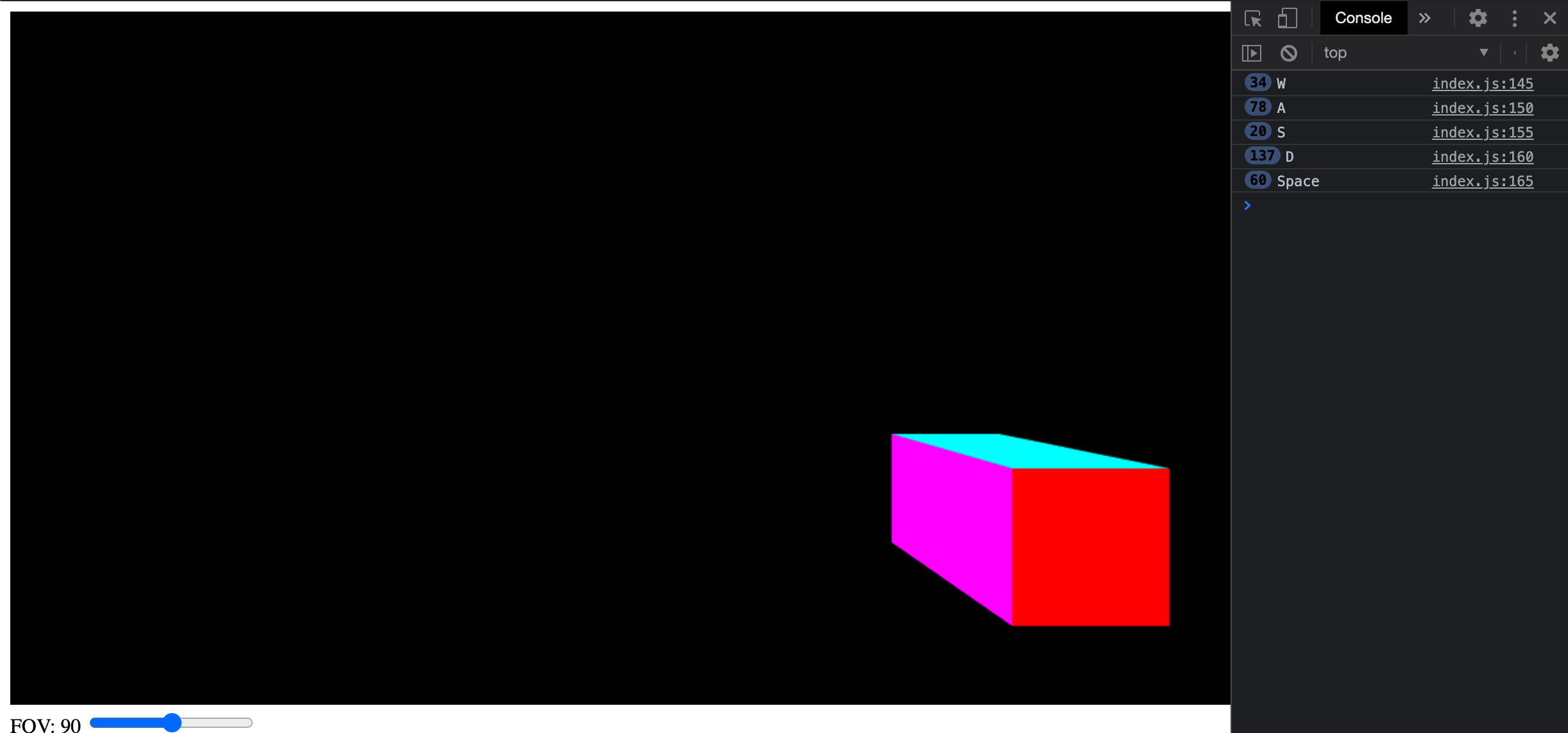
****

Figure 6 Space Functionality Cube

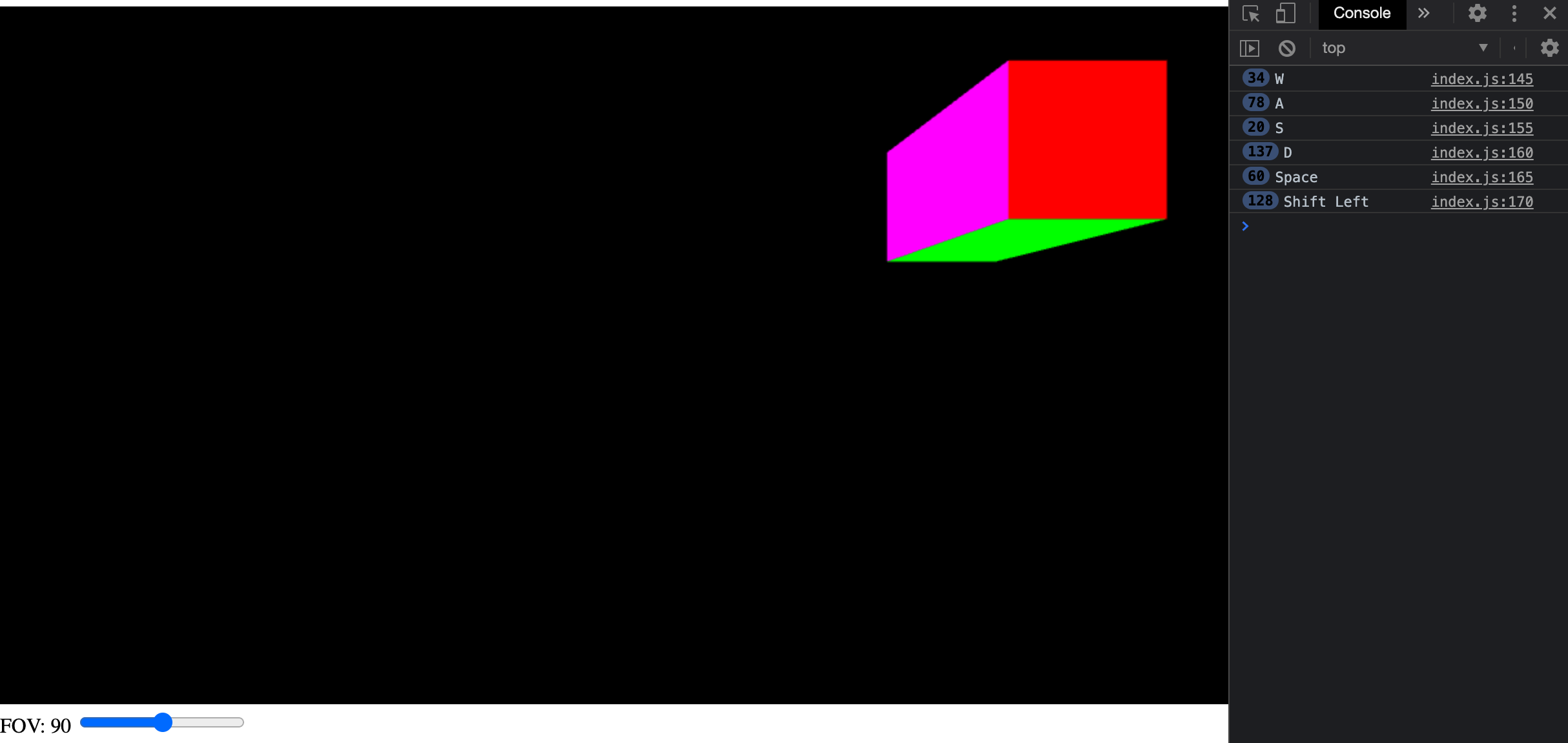
****

Figure 7 Left Shift Functionality Cube

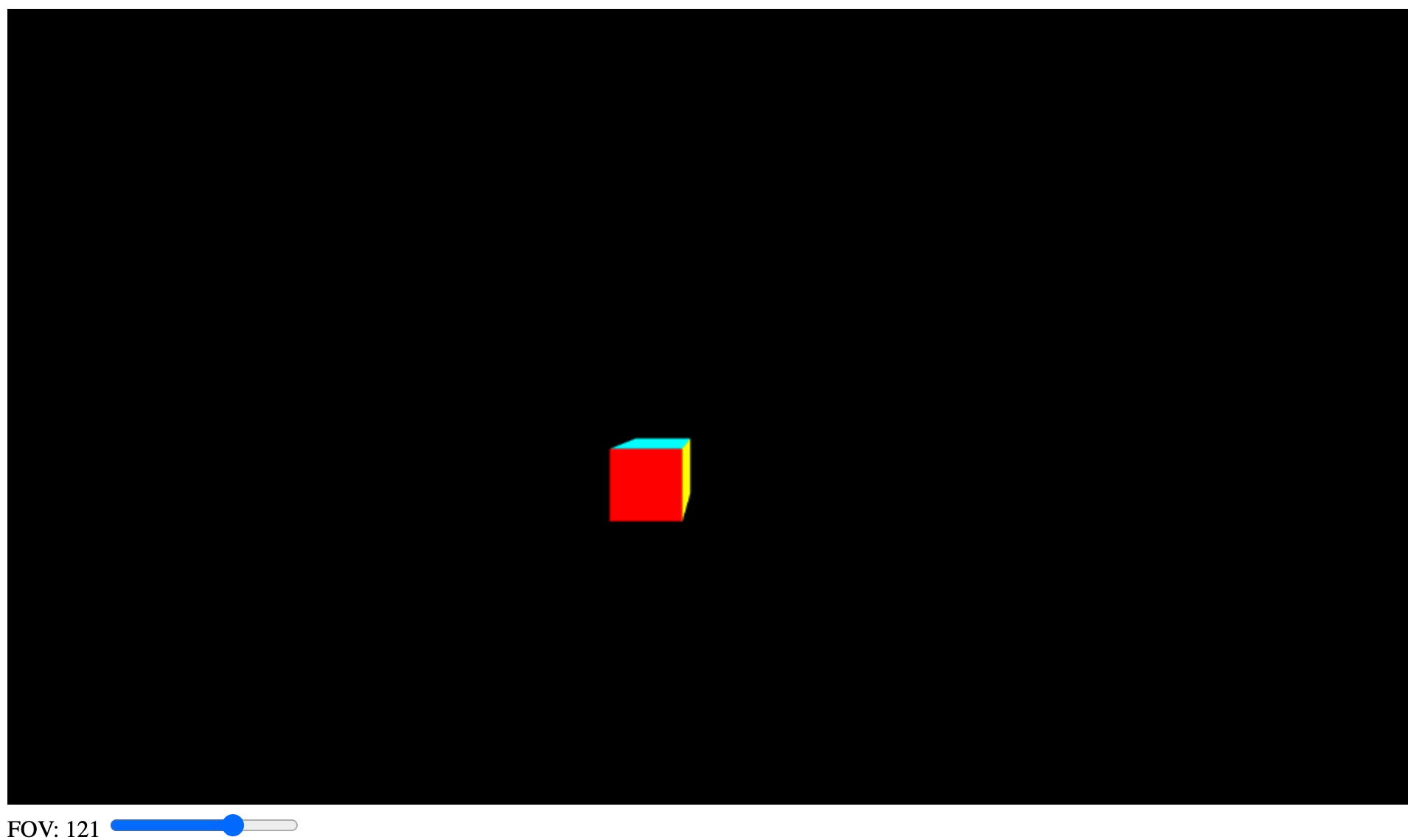
****

Figure 8 FOV High

****

Figure 9 FOV Low

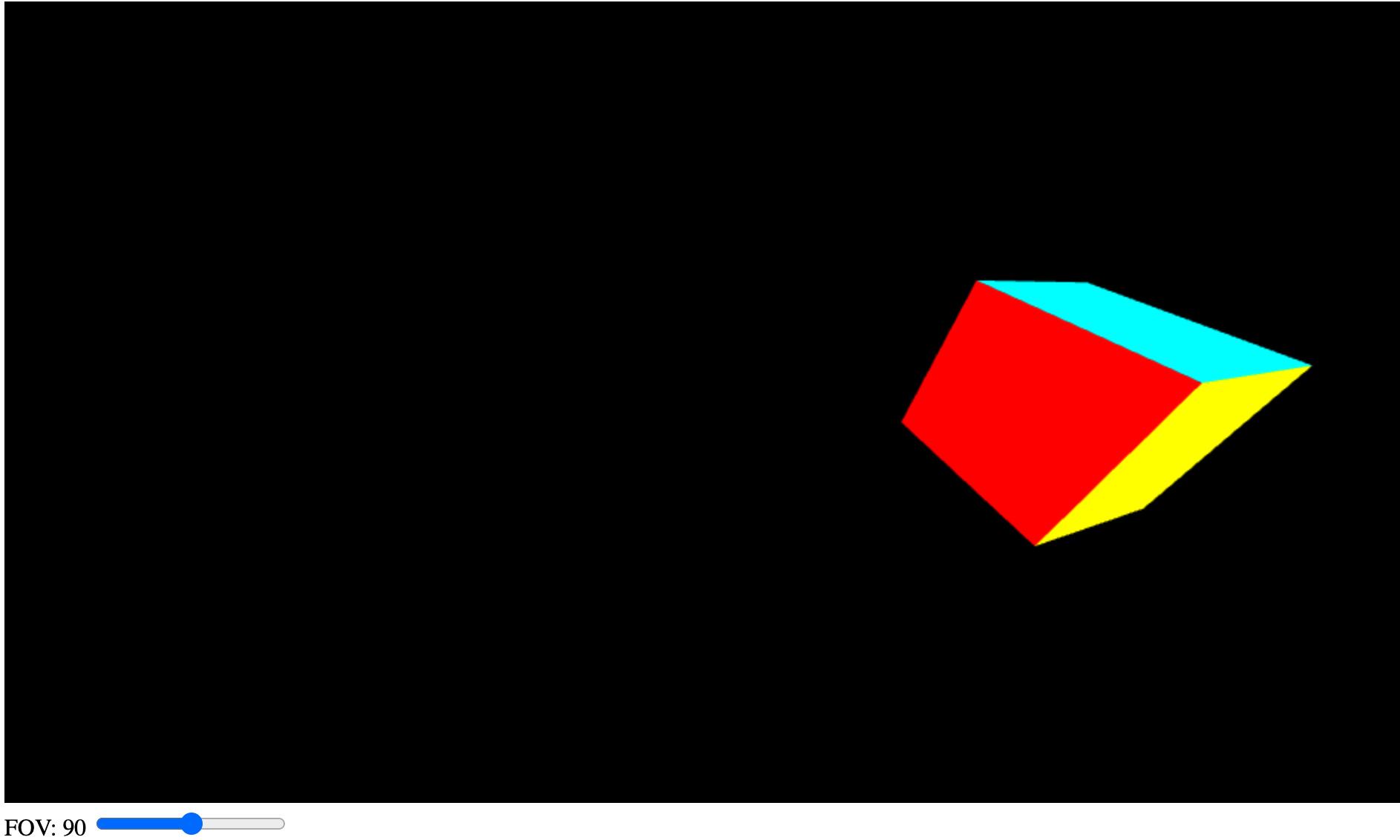
****

Figure 10 Mouse Move Function 1

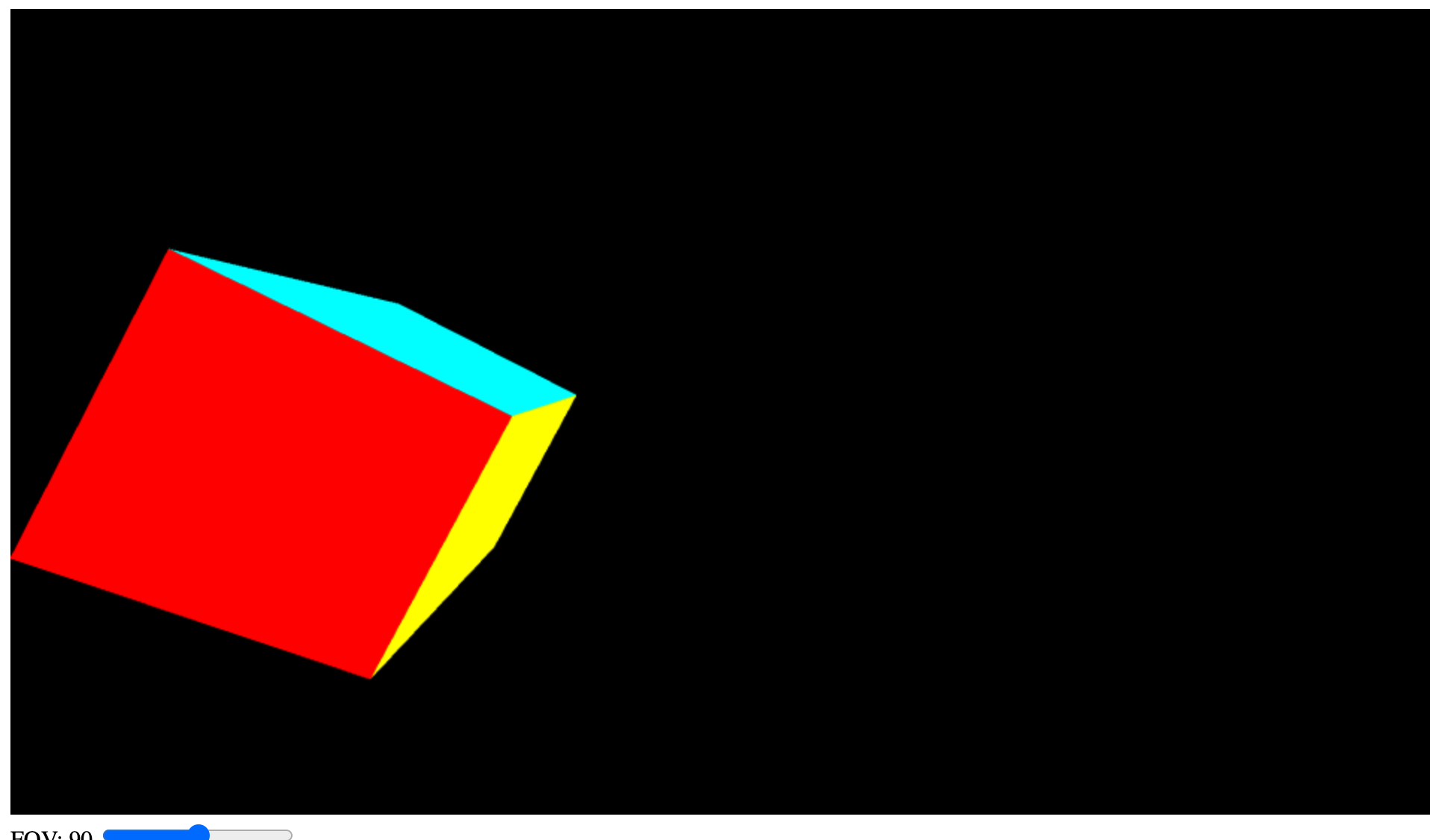
****

Figure 11 Mouse Move Function 2

**Issues I Faced:**

Originally, I didn’t realize that the cameraPosition and cameraRotation matrices essentially replaced the translation and rotation matrices from Homework 3. I was trying to figure out how to work with all three of them and in my confusion, was trying to initialize cameraPosition and cameraRotation and then send them to the shader. When I realized that they just needed to be declared and would be sent to the shader via the view, things began to make a lot more sense.