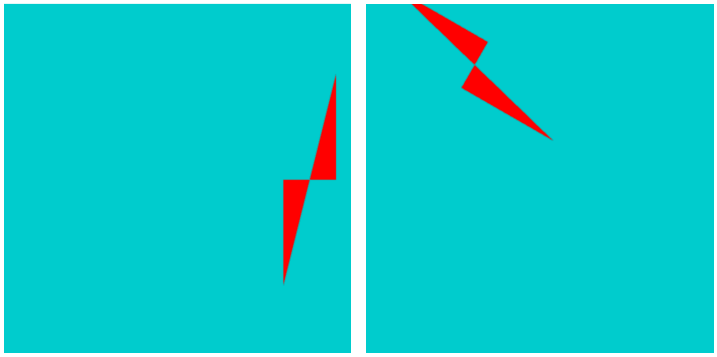
Com S 336
Fall 2023
Homework 3

Please submit an archive on Canvas containing the files indicated at the beginning of each problem.

1. (*Please turn in two files Propeller.html and Propeller.js*)

Modify the rotating square from homework 1 so that instead of the square, it draws the little triangle from Transformations1.js. Then modify it so that it draws the figure below, in which the little triangle is drawn twice, scaled by .5 in the x direction and 3 in the y direction. Then, modify it so that in addition to the rotation about the origin, the figure rotates 4 degrees per frame clockwise about its center (where the two triangles meet in the figure). There is also an animation you can see here:

https://stevekautz.com/cs336f23/misc/hw3/propeller.mp4



(*Hint*: Use the strategy TRS: Scale, then rotate, then translate. You'll need to get familiar with the three.js Matrix4 type. You can see the three.js documentation, but here is a summary of common operations:

Pseudocode	Javascript
M = Translate(dx, dy, dz)	`M = new THREE.Matrix4().makeTranslation(tx, ty, tz);`
M = Scale(sx, sy, sz)	`M = new THREE.Matrix4().makeScale(sx, sy, sz);`
M = RotateX(degrees)	`M = new THREE.Matrix4().makeRotationX(toRadians(degrees));`
M = RotateY(degrees)	`M = new THREE.Matrix4().makeRotationY(toRadians(degrees));`
M = RotateZ(degrees)	`M = new THREE.Matrix4().makeRotationZ(toRadians(degrees));`
A = AB	`A.multiply(B);`
A = BA	`A = new THREE.Matrix4().copy(B).multiply(A);` `A.premultiply(B);  // alternate form`
C = AB	`C = new THREE.Matrix4().copy(A).multiply(B);`
M = Translate(1, 2, 3) * Scale(4, 5, 6)	`M = new THREE.Matrix4().makeTranslation(1,2,3)` `  .multiply(new THREE.Matrix4().makeScale(4,5,6));`

2. *(Please turn your modified version of Rotations.js.)*

TL;DR: Your task is to do something similar to what we did in class on arts-and-crafts day, this time using the starting configuration

```
RotateZ(-45) * RotateY(-45)
```

Find the Euler angles needed to invert the above rotation using the orderings:

a) XZX
b) XYX
c) ZXY
d) XZY

All the angles involved are plus or minus 35, 55, 30, or 60 degrees[1] as in our original example. Use a physical model to figure out the direction of rotation and approximate angle you want, then add a code block to the **`handleKeyPress`** function (similar to what we have for the examples, described below), get the exact angles and verify that it works. When you go on to the next one, just comment out the previous code block. *So, in the end, what you turn in will be Rotations.js with these four additional commented out code blocks inserted around line 183.*

Further details and warmup: In our class exercise we looked at the rotation RotateY(-45) * RotateX(-45) and investigated some of the ways to use Euler angles to perform (more specifically, to invert) that transformation using a different sequence of rotations about the coordinate axes. In particular one of the sequences we looked at was the ZYZ ordering:

```
RotateZ(-35) * RotateY(60) * RotateZ(55)
```

[1] These can be calculated analytically, of course. More precise values are -35.264, 60.0, and 54.736.

In Rotations.js, you can try this:

1. Select the radio button for “Intrinsic”
2. Enter 45 into the text box labeled “Rotation increment”
3. Press Shift-Y then Shift-X
4. Then, you should be able to invert it by pressing ‘p’, ‘q’, then ‘r’.

(Note that the ‘p’, ‘q’, and ‘r’ keys ignore the contents of the text box for the rotation increment.)

Pressing the ‘d’ or ‘D’ key will change the camera direction by plus or minus 15 degrees so you can look around, and the ‘e’ key will toggle the camera elevation from 45 degrees to 0 degrees (i.e. aligned with the x-z plane) and back.

The general pattern is that if you are doing some order ABC, your first rotation (about the model A axis) needs to align the B axis with one of the coordinate planes, and the second rotation (about the model B axis) needs to align the C axis with the world C axis.

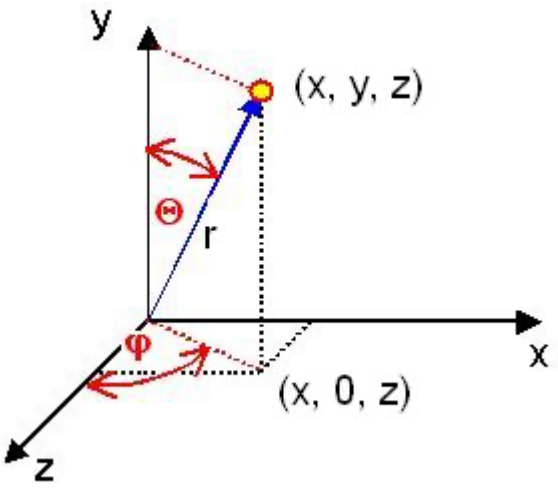
Try this:

1. Reload the page to restore everything to its initial state.
2. Click the radio button for “intrinsic” mode.
3. Enter 45 in the text box for “rotation increment” (and click away from it)
4. Press Shift-Y then Shift-X.
5. Rotate the camera by pressing ‘d’ four times, which should line up your viewpoint along the x-y plane.
6. Enter 5 in the increment box and click away
7. Press shift-Z 7 times to perform RotateZ(35). *Notice that the model y-axis (green line) moves almost exactly into the x-y plane.*
8. Move the camera down by pressing ‘e’ to align your viewpoint with the x-z plane.
9. Press ‘y’ 12 times. *This should move the model z-axis (blue line) into the x-z plane.*
10. Press ‘e’ and then shift-D four times to restore the original view point. You can see that the model z-axis is lined up with the world z-axis.
11. Press ‘z’ eleven times, which performs RotateZ(55). *Notice that the model y-axis is almost perfectly vertical.*

The code that hooks up the ‘p’, ‘q’, and ‘r’ keys is in the function handleKeyPress on lines (approximately) 150-165 of Rotations.js. Next, try commenting out those lines and uncommenting the next block, roughly lines 166-180, which do something similar using the ordering YZY.

3. (*Please turn in your modified version of RotatingCube.js*)

One interpretation of the pitch and head angles (assuming roll is zero) is that these two angles determine a *direction*. This direction is denoted by the vector labeled r in the figure below, which is where the y-axis ends up after applying the Euler angle transformation RotateY(head) * RotateX(pitch) (or RotateY(phi) * RotateX(theta) as illustrated below).



(These two angles are related to the so-called spherical coordinates; see the end of this document for more details.)

Make a modified version of transformations/RotatingCube.js. Remove all the existing rotation behavior, and then set it up like this:

a) The orientation of the cube is determined by the two angles theta and phi. Modify the controls so that the 'x' and 'X' keys increase or decrease the pitch angle by 5 degrees, and the 'y' and 'Y' keys increase or decrease the head angle by 5 degrees.

b) Draw a black line representing the vertical centerline of the cube, that is, aligned with the vector r determined by your pitch and head angles. This line should always go through the exact center of the top square (green) and the center of the bottom square (dark green). (Hint: just redraw the y-axis, but apply the two rotations for head and pitch. This is easy to do: the three.js **`Vector3`** class has a method **`applyMatrix4(m)`** that multiplies the vector on the left by matrix **`m`**.)

c) Make the cube spin about its vertical centerline (the black line drawn above).

Note: There are two ways to go about this. The Matrix4 function setRotate() will create a rotation matrix about any axis, given as a coordinate vector (see RotatingCube.js around line 421). So one approach is to find the coordinates of the vector **r** in the illustration above (and normalize it to be sure it is a unit vector). This is not difficult, see the section on spherical coordinates at the end of this document.

However, it is probably simpler to just intrinsically rotate about the cube's y-axis *before* applying the head and pitch.

There is a brief animation you can see here:

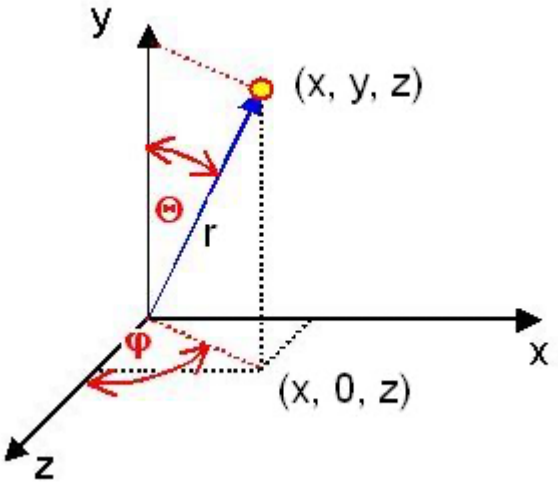
https://stevekautz.com/cs336f23/misc/hw3/problem3.mp4

4. (*Please turn in your modified version of CS336Object.js*)

The class CS336Object (see homework3/CS336Object.js) encapsulates a scale, rotation, and translation and exposes a suite of convenient methods for manipulating these values. It is partially implemented; see the TODO markers and read the documentation carefully. Your task is to finish the implementation. The code to write is generally very simple, but you may have to get out your pipe cleaners and think about the transformations. There is an application homework3/RotatingCubeWithObject that uses this class to manage the transformations of a colored cube. Take a look at the function handleKeyPress to see exactly what it's doing; also note the call to getMatrix around line ~210. The getMatrix method (which is already implemented in CS336Object) takes the encapsulated data and constructs a transformation matrix based on a scale, then rotation, then translation. There are keyboard controls for the operations to be applied to the cube. See the html file for these. You can use this application to get a rough test of whether your implementation is working. *Note that key actions connected to the "orbit" operations are not expected to do anything sensible unless the object's negative z-axis is facing the origin.*

Notes on spherical coordinates

A direction can be described using two angles, an *inclination* θ from the vertical, also called the *polar angle*, and a rotation φ about the vertical, also called the *azimuth*. Note that this direction corresponds exactly to the orientation of the y-axis after rotations by pitch angle θ and head angle φ.



If you also give a radial distance r from the origin, the three values r, θ, and φ determine a unique point and are called *spherical coordinates* for the point. (Note that in calculus books, the z-axis is vertical and the azimuth is the x-axis, and they typically use symbol θ for the azimuth and φ for the polar angle, the opposite what we're using here.)

Conversion of spherical to rectangular:

$y = r \cos\theta$
$r' = r \sin\theta$ (Note that r' is the radial distance from (0, 0, 0) to (x, 0, z))
$x = r' \sin\varphi$
$z = r' \cos\varphi$

Conversion of rectangular to spherical:

You can get r using the distance formula.

- If x, y, and z are all zero, then phi and theta are both undetermined.
- If x and z are both zero but y isn't, then phi is undetermined and theta is zero.
- If x and z are not both zero, then phi is arctan(x/z), and theta is arccos(y/r).

To compute the arctangent, it is easiest to use the function `atan2(a, b)`, which will give the correct value of arctan(a/b) even when b is zero. The arctangent function always returns a value in the range [-π, π], so we conventionally add π to negative values to shift it to the range [0, 2π]. Then convert to degrees as needed.