Com S 336
Fall 2020
Homework 3

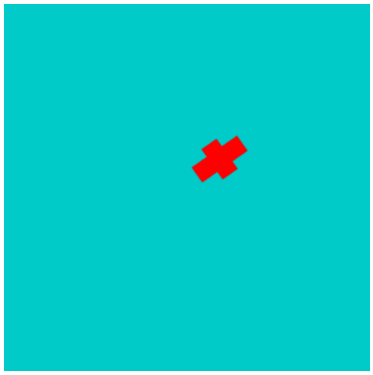
Please submit an archive on Canvas containing the files indicated at the beginning of each problem.

1. (*Please turn in your modified version of RotatingSquare.js*)

The example transformations/RotatingSquare.html has the square from GL_example1 rotating about the origin in a circle of radius .75. (Take a look.) Modify the .js file so that it draws the figure below, in which the square is drawn twice, first scaled by (.3, .1) and then by (.1, .2). Then, modify it so that instead of the circle around the origin, the figure follows the lemniscate or "figure-eight" path given by the equations

$$x = \frac{a\sqrt{2}\cos(t)}{\sin^2(t)+1}; \qquad y = \frac{a\sqrt{2}\cos(t)\sin(t)}{\sin^2(t)+1}$$

where a is (see the Wikipedia page on "Lemniscate of Bernoulli" for pictures). The figure is traced out once as t ranges from 0 to 2π, just as for the circular path. Finally, modify the code so that the figure itself is rotated to follow the direction of the curve, i.e., is aligned with the tangent line. (You do not need to calculate a derivative! It is easier, and sufficient, just to store the x and y values from the previous frame and use the slope between the new center and old center to find the angle of rotation (arctangent of slope). Make a special case for vertical slope (90 or -90 degree rotations) when the old x and new x are very close.) Hint: TRS: scale, then rotate, then translate. Figure is shown below at $t = \pi/3$ using $a = 0.5$. See
http://web.cs.iastate.edu/~smkautz/cs336f20/homework/hw3/lemniscate.mp4
for a little animation.



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

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 (or invert) the same transformation using a different sequence of rotations about the coordinate axes. In Rotations.js, you'll find that there is some code so that if the 't' key is pressed, the following sequence of rotations is applied:

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

If you start with RotateY(-45) * RotateX(-45) (i.e. press shift-XXX and then shift-YYY in "extrinsic" mode), then this sequence of rotations will approximately transform the figure back to its original position. If you switch to intrinsic mode, the keys 'p', 'q', and 'r', in that order, will perform the same three transformations one at a time, to better see what's going on. Edit the view matrix to see it from different angles.

The angles are not exact, but were deduced by trial and error. Note the order of the rotation axes is ZYZ (matching the order of the three matrices). We also did YZY in class (`RotateY(90) * RotateZ(45) * RotateY(-45)`).

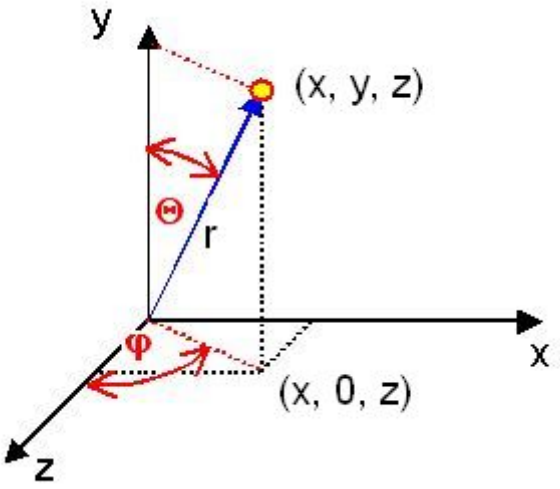
Determine how to perform the same transformation using the orderings:

a) ZXZ
b) XZX
c) ZYX
d) YXZ

For each one, determine the approximate angles as in the example in the code, and try them out by modifying Rotations.js. Hook them up to the keys 'a', 'b', 'c', and 'd' for testing, as we did for the 't' key in the example. (*Hint: (b) is similar to YZY and (a) is similar to ZYZ. Both (c) and (d) should end up with some combination of plus or minus the same three angles as ZYZ.*)

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

In class we suggested that a reasonable convention for Euler angles is "intrinsic head, pitch, roll", or YXZ. 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 rotation RotateY(phi) * RotateX(theta). See illustration 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 for pitch and head. Modify the controls so that the 'p' and 'P' keys increase or decrease the pitch angle by 5 degrees, and the 'h' and 'H' 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 theta and phi. 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.)

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

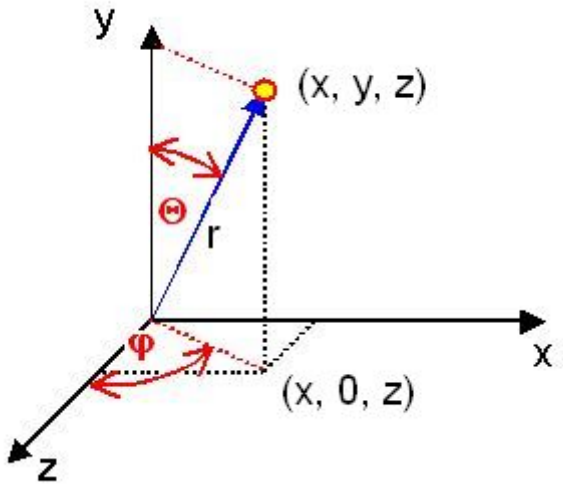
4. (*Please turn in your modified version of CS336Object.js*)

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 RotatingCubeWithObject that is set up to use this class to manage the

transformations of a colored cube. It won't do anything until you implement CS336Object, but take a look at the function handleKeyPress to see exactly what the key presses are supposed to be doing; also note the call to getMatrix on line 352. 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 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.