1. a) In the euler_angles project, there are controls for adjusting the head, pitch, and roll angles of the model. Incorporate the head and pitch controls into simple_spinning_cube to orient the cube using those two Euler angles. For this step, suppress the cube’s spinning. Draw a magenta-colored line through the center of the cube, 2 units long, corresponding to the transformed y-axis. (This line should always penetrate the cube in the center of the blue face and the center of the green face. Take a look at the picture in the section “Reminders about spherical coordinates” at the end of this document. The arrow labeled “r” corresponds to the location of the magenta line.)

b) Modify the code from (a) so that the cube always spins about the magenta line. That is, it should perform a y-rotation with respect to the transformed frame given by the Euler angles. (Note: this is not difficult, since you have the Euler angles given explicitly.)

c) Add controls ‘v’ and ‘V’ to “zoom out” or “zoom in” by increasing or decreasing the size of the view volume by 0.2 units in each direction. Instead of the matrix labeled projection in the code, use the matrix

\[
\begin{align*}
\text{projection} & = \text{Ortho}(-\text{size}, \text{size}, -\text{size}, \text{size}, -\text{size}, \text{size}); \\
\end{align*}
\]

where \(\text{size}\) is half the size of the desired view volume (initially size = 1 for the default).

d) Modify the code from (c) so that the spinning cube rotates in a circular orbit in the x-z plane, of radius 3, about the origin. The rotation speed for the orbit should be one-tenth that of the cube’s own rotation. Keep the controls for inclining the cube’s own axis of rotation using the head and pitch angles. Note that you can’t just place the cube 3 units from the origin and then apply a rotation about the origin, since the rotation about the origin would interfere with the rotation about its own axis. Instead, calculate a new center point for the cube and use a translation: if \(\alpha\) is the angle of rotation measured from the positive z-axis in the x-z plane, the new center would be

\[
\begin{align*}
z & = 3 \cos(\alpha) \\
x & = 3 \sin(\alpha)
\end{align*}
\]

e) Add a second cube, same as the first but scaled to one-fifth the size, that orbits around the first cube in the first cube’s own x-z plane. Its rotation speed should be the same as the original cube but its orbit speed should be twice the rotation speed. (Note: this should not require a new VAO or shader, just a different transformation.)
(If you are successful in completing (a) – (e), you can just turn in one project incorporating all the changes, otherwise you might want to keep them separate so we can tell which parts were correctly done.)

2. a) Generalize 1(b) as follows: write a function \texttt{RotateAxis(float degrees, float phi, float theta)} that returns a matrix for performing a rotation of \texttt{degrees} degrees about the axis described by head angle \texttt{phi} and pitch angle \texttt{theta}. (The axis of rotation corresponds to the blue arrow labeled “r”, as pictured in the section “Reminders about spherical coordinates” at the end of this document. The matrix should be simply constructed from standard transformation matrices. You can try it out in your problem 1(b).

b) Write a similar function \texttt{RotateAxis(float degrees, vec3 direction)} in which the axis is specified as a direction vector, rather than by two Euler angles. (Hint: you’ll just need to convert to spherical coordinates first; see end of this document. Note: On p. 18 of the text, the author gives a general and complicated-looking matrix for accomplishing this same purpose. In contrast, your solution should actually make sense, in that it is just a product of standard transformation matrices.)

3. We have two ways of constructing a perspective matrix: the general function \texttt{Frustum}, which takes 6 arguments, and the more specialized function \texttt{Perspective} which requires only four arguments:
   - \texttt{fovy} – vertical field of view, in degrees
   - \texttt{aspect} – aspect ratio
   - \texttt{zNear} – near clipping plane
   - \texttt{zFar} – far clipping plane

(see mat.h and try the examples in the ortho project). Since \texttt{Frustum} is more general, we should be able to implement \texttt{Perspective} by making one call to \texttt{Frustum} using appropriate arguments. Do so. You should be able to verify (write a short test program) that you get the same matrix using your implementation and using the one in mat.h. Note that the C++ “\texttt{<<}” operator is overloaded for the \texttt{mat4} type so you can just dump the matrices to \texttt{cout} and look at them.
Reminders about spherical coordinates

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

If you also give a radial distance $r$ from the origin, the three values $r$, $\theta$, and $\phi$ 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 $\theta$ for the azimuth and $\phi$ for the polar angle, the opposite what we’re using here.)

**Conversion of spherical to rectangular:**

\[
\begin{align*}
y &= r \cos \theta \\
r' &= r \sin \theta \quad \text{(Note that $r'$ is the radial distance from (0, 0, 0) to (x, 0, z))} \\
x &= r' \sin \phi \\
z &= r' \cos \phi
\end{align*}
\]

**Conversion of rectangular to spherical:**

You can get $r$ using the `length()` function from vec.h (which is just 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 C 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 $[-\pi, \pi]$, so we conventionally add $\pi$ to negative values to shift it to the range $[0, 2\pi]$. Then convert to degrees as needed.