In the InitGraphics() function:

In the Animate() function:

const int MAXIMUM_TIME_MILLISECONDS = 1000 * MAXIMUM_TIME_SECONDS;
const int MAXIMUM_TIME_SECONDS = 10 * 60;  // I decided to use 10 minutes

At the top of the program:

.glutIdleFunc( Animate );
.glutPostRedisplay();
.glutSetWindow(MainWindow);
// force a call to Display();
// note that Time goes from 0. to however many seconds you asked for, not 0. to 1.

Time = (float)ms / 1000.f;  // seconds
ms %= MAXIMUM_TIME_MILLISECONDS;  // [0, MAXIMUM_TIME_MILLISECONDS-1]
int ms = glutGet( GLUT_ELAPSED_TIME );  // milliseconds

Warning: these are the actual numbers for our solar system.
You would need to change them to your exaggerated numbers!

Earth Transformations

Steps to transform the Earth-eye-viewing system into Solar System Coordinates:

1. Put the eye-position at the corner of its Equator and Prime Meridian (xe = EARTH_RADIUS_MILES, ye = 0., ze = 0.)
2. Translate the Earth by EARTH_SPIN_TIME_SECONDS * ONE_FULL_TURN
   - This makes our view-vector (from the eye-position to the look-at position) tangent to the Earth's surface, which is a good way to start.
3. Revolve the Earth by EarthSpinAngle

Note: all we have to do is transform these 3 locations/vector into Solar System Coordinates
(In case to call them “World Coordinates” here...)
Moon Transformations

Let's make this fairly straightforward. In model coordinates:

1. Put the eye-position at the corner of the Equator and Prime Meridian
   \( x = \text{MOON\_RADIUS\_MILES}, \ y = 0, \ z = 0 \)

2. Set the look-at position to be on a straight-line east of the eye-position
   \( x = \text{MOON\_RADIUS\_MILES}, \ y = 0, \ z = 1000 \)

3. Set the up-vector to be: \( x = 1000, \ y = 0, \ z = 0 \)

   This makes our main-vector (from the eye-position to the look-at position) tangent to the Moon's surface, which is a good way to start.

Now all we have to do is transform those 3 locations/vectors into Solar System Coordinates. I hate to call them "World Coordinates" here...
What if You Want the Eye at Corvallis (or some other arbitrary location)?

Corvallis sits at Latitude 44.57° N x Longitude -123.27° W

Treating lat-long as spherical coordinates and solve for x, y, and z:

\[ y = \sin(44.57°) \]
\[ xz = \cos(44.57°) \]
\[ x = xz \cdot \cos(123.27°) \]
\[ z = xz \cdot \sin(123.27°) \]

Then multiply x, y, and z by EARTH_RADIUS_MILES

Let's assume that we want to look straight east from where we are:

- The eye-position will be \((x, y, z)\)
- By the law of perpendicular-to-a-circle, the eye-to-look vector (i.e., look-eye) = \((z, 0, -x)\)
- So, the look-at position will be \(\text{eye-position} + \text{eye-to-look-vector} = (x+z, y, z-x)\)
- The up vector is \((x, y, z)\)

Note that the dot product of these two vectors is 0., which proves they are perpendicular.

This works for any point on the globe, not just Corvallis.

Put this in the Display() function:

```cpp
glm::mat4 earth = MakeEarthMatrix();
glm::mat4 moon = MakeMoonMatrix();
glEnable(GL_TEXTURE_2D);
glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, EarthTex);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glMultMatrixf(glm::value_ptr(earth));
    glCallList(EarthList);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, MoonTex);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glMultMatrixf(glm::value_ptr(moon));
    glCallList(MoonList);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, EarthTex);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glMultMatrixf(glm::value_ptr(earth));
    glCallList(EarthList);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, MoonTex);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glMultMatrixf(glm::value_ptr(moon));
```