Chapter 5. Rotation of a point or rigid triangle around an arbitrary axis

5.1 Introduction

We have now learned how to model the rotation of a point on the surface of a globe that is rotating around its north spin axis, which coincided with the Z axis of our reference coordinate system. Now we want to develop the ability to model a rotation of one or more points around an axis of any orientation. We have already developed all of the computational tools we need to perform this task.

5.2 Coordinate systems

In the last chapter, we had a reference point that was located in two different coordinate systems at some initial time. One coordinate system was fixed to the globe and hence to the reference point, and the other coordinate system was fixed to the room in which an observer was strapped to a chair bolted to the floor. When the globe rotated in a positive (counter-clockwise) direction around its north-polar axis as observed from the room, the "room" coordinate system rotated in a negative direction relative to an imaginary observer on the globe.

We are now going to model a rotation on a unit-sphere globe around an axis of rotation that is not located at the north pole. We use the word ***pole*** to refer to the point on the surface of the sphere where a rotational axis intersects the sphere. A ***positive pole*** is a pole around which motion is counter-clockwise as observed looking down at the pole from above the sphere.

Here are a few user-defined functions from earlier chapters that will be useful.

```
unitVect3D[vect_] :=
  {(vect[[1]] / Norm[vect]), (vect[[2]] / Norm[vect]), (vect[[3]] / Norm[vect])};

zRotation[w_, dT_] := {{Cos[(w dT) Degree], -Sin[(w dT) Degree], 0},
   {Sin[(w dT) Degree], Cos[(w dT) Degree], 0}, {0, 0, 1}};
```

The location vector to the north pole is

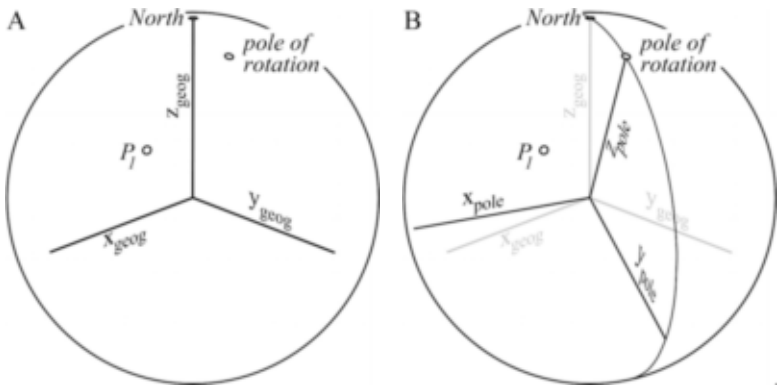
```
zGeog = {0, 0, 1};
```

The other unit vectors along the positive axes of the Cartesian ***geographic coordinate system*** are

```
xGeog = {1, 0, 0};
```

where the positive xGeog axis passes through latitude 0, longitude 0, and

```
yGeog = {0, 1, 0};
```



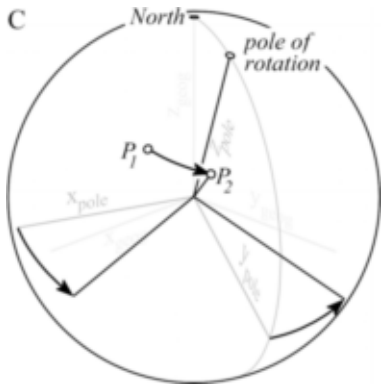


Figure 5-1. A. Geographic coordinate system in which zGeog passes through the north pole and xGeog passes through latitude 0, longitude 0. A pole of rotation is located at latitude 65°N and longitude 70°E. The initial position of reference point P1 is at latitude 35°N, longitude 30°N. B. Pole coordinate system in which zPole passes through the pole of rotation, xPole = aPole × zGeog, and yPole = zPole × xPole. C. Reference point P1 with an initial position at latitude 35°N, longitude 30°E, rotates 30° around the zPole axis. That is, the coorinate system fixed to the point rotates 30° relative to the pole coordinate system whose axes are xPole, yPole, and zPole.

We will choose a positive pole located at latitude 65°N (in the northern hemisphere) and longitude 70°E (in the eastern hemisphere). In the Cartesian geographic coordinate system, the pole's unit location vector is given by

```
poleLat = 65;

poleLong = 70;

zPole = {(Cos[poleLat Degree] Cos[poleLong Degree]),
    (Cos[poleLat Degree] Sin[poleLong Degree]), (Sin[poleLat Degree])};
```

The vector **zPole** coincides with the positive Z axis of the ***pole coordinate system***, which remains fixed to the geographic coordinate system in which the position vectors to rotational poles and reference points are initially located. The positive X axis of the pole coordinate system (**xPole**) is found by taking the cross product of the rotational pole and the north pole of the Cartesian geographic coordinate system

```
xPole = unitVect3D[Cross[zPole, zGeog]];
```

and the positive Y axis (**yPole**) is found by taking the cross product of the rotational pole and **xPole**

```
yPole = unitVect3D[Cross[zPole, xPole]];
```

The vector **zPole** also coincides with the Z axis of a third coordinate system that is fixed to the reference point(s) whose motion we will track. We will refer to this third coordinate system as the ***body coordinate system***, in the sense that it is fixed to the rigid body whose motion we are modeling. The axes of the body coordinate system and the pole coordinate system are identical at the initial time of our model run, and their respective Z axes remain coincident throughout the model time interval.

The matrix that allows us to transform from the Cartesian geographic coordinate system (**xGeog**, **yGeog**, **zGeog**) to the pole coordinate system (**xPole**, **yPole**, **zPole**) is

```
j1 = ( Dot[xGeog, xPole]  Dot[yGeog, xPole]  Dot[zGeog, xPole]
       Dot[xGeog, yPole]  Dot[yGeog, yPole]  Dot[zGeog, yPole]
       Dot[xGeog, zPole]  Dot[yGeog, zPole]  Dot[zGeog, zPole] );
```

and the matrix that allows us to transform from the pole coordinate system (**xPole**, **yPole**, **zPole**) back to the Cartesian geographic coordinate system (**xGeog**, **yGeog**, **zGeog**) is

```
j3 = Inverse[j1];
```

To check that we have not made any immediate mistakes in creating the matrices, we set the rotational angle θ to zero and make sure that the input location vector to our reference point is the same as the output vector. It should be the same, because we have rotated it 0° from its starting position.

```
θ = 0;
```

Matrix **j2** is the rotational matrix we developed in chapter 4.

```
j2 = ( Cos[θ Degree]  -Sin[θ Degree]  0
       Sin[θ Degree]   Cos[θ Degree]  0
            0               0         1 );
```

We will choose an arbitrary reference point located at latitude 35°N and longitude 30°E. In the Cartesian geographic coordinate system, the reference point's initial unit location vector is given by

```
p1 = {(Cos[35 Degree] Cos[30 Degree]),
    (Cos[35 Degree] Sin[30 Degree]), (Sin[35 Degree])};
```


The variable **p2** is the location vector of the reference point after a rotation of θ degrees around the rotational pole that we specified above.

Core algorithm for circular finite motion

We begin with the location vector to a reference point (**p1**) as defined in the geographic coordinate system. We also have the location vector to the positive pole of rotation (**zPole**). We define the pole coordinate system based on **zPole** and the north pole of the geographic coordinate system (**zGeog**). In defining the pole coordinate system, we are also defining the initial position of the body coordinate system to which our reference point is fixed. The body coordinate system rotates with respect to the pole coordinate system around their common Z axis.

The **j1** matrix transforms vector coordinates from the geographic coordinate system to the pole coordinate system. The **j2** matrix takes the transformed vector and rotates it (and the body coordinate system) around the Z axis, yielding a new set of vector coordinates expressed in the pole coordinate system. The **j3** matrix is the inverse of the **j1** matrix, and so it transforms from the pole coordinate system back to the geographic coordinate system. The result is the location vector to the displaced reference point (**p2**) expressed in the geographic coordinate system. Hence, this process is computed in *Mathematica* as follows:

```
p2 = j3.j2.j1.p1
```

For θ=0, the input location vector (**p1**) should be the same as the output location vector (**p2**). The numerical values of the input vector coordinates are

```
N[p1]
```

```
{0.709406, 0.409576, 0.573576}
```

while the numerical values of the output vector coordinates are

```
p2 = j3.j2.j1.p1;
```

```
N[p2]
```

```
{0.709406, 0.409576, 0.573576}
```

Bueno, but that just means that we have successfully modeled the case of no motion. As achievements go, this is not particularly impressive. Yet...

5.3 Rotating a point around an arbitrary axis

Now that we have all of the pieces together, let's input a non-zero rotational angle and see what we get. Set the value of θ to compute where the reference point will be after a rotation of 30° counter-clockwise around the pole.

```
θ = 30;
```

Changing the value of θ changes the values of the four most important elements in the rotation matrix **j2**.

```
j2 = {{Cos[θ Degree], -Sin[θ Degree], 0}, {Sin[θ Degree], Cos[θ Degree], 0}, {0, 0, 1}};
```

Variable **p2** is the location vector to the reference point after a rotation of 30° around the pole.

```
p2 = j3.j2.j1.p1;
```

```
N[p2]
```

```
{0.557858, 0.676488, 0.480789}
```

```
N[p1]
```

```
{0.709406, 0.409576, 0.573576}
```

We can check our output data in a couple of ways. First, the output vector **p2** should be the same distance away from the pole as the input location vector **p1** was.

```
N[VectorAngle[zPole, p1] (180 / π)]
```

```
38.2763
```



```
N[VectorAngle[zPole, p2] (180 / π)]
```

```
38.2763
```

The first check was successful. Second, the angle between the great-circle arc from the pole to the initial point and the great-circle arc from the pole to the displaced point should be equal to 30 degrees.

```
checkθ = N[VectorAngle[Cross[zPole, p1], Cross[zPole, p2]] (180 / π)]
```

```
30.
```

Again, our check is successful. We appear to have a procedure that generates correct answers.

Exercise 5.1 Explain the *Mathematica* expression we just used to find the angle θ that the reference point had been rotated around the rotational pole: **checkθ=N[VectorAngle[Cross[pole,rpInitial],Cross[pole, rpDisplaced1]](180/π)]**

5.4 Time-dependant rotation given an angular velocity

Now we need to modify our procedure as we did in chapter 4, so that it makes use of angular velocities and is able to provide answers for a range of times and for multiple reference points. We borrow some ideas from the user-defined function **circMotion** from chapter 4, adding the transformation matrices that we developed above

```
circMotion[x_, m1_, w_, dT_] := Module[{m2, m3, answer}, m2 = zRotation[w, dT];
    m3 = Inverse[m1];
    answer = m3.m2.m1.x;
    answer];
```

and use an angular velocity of 15°/hour.

```
ω = 15;
```

We then use the built-in *Mathematica* function **Table** to generate a table of vectors called **results** for time=0 to time=23 hours in half-hour steps.

```
results = Table[circMotion[p1, j1, ω, deltaT], {deltaT, 0, 23, 0.5}];
```

We adapt the graphics code we used in chapter 4 to plot our data on a sphere.

```
graphicTry1 = Graphics3D[Sphere[{0, 0, 0}, 1],
    AspectRatio → 1, BoxRatios → {1, 1, 1}, PlotRange → All,
    PlotRangePadding → 0.1, ColorOutput → GrayLevel, Lighting → "Neutral"];

graphicTry2 = ListPointPlot3D[results, AspectRatio → 1, BoxRatios → {1, 1, 1},
    ColorOutput → GrayLevel, PlotRange → All, PlotRangePadding → 0.1];
```

We add the rotational pole to the list of red marker dots that are plotted on the sphere.

```
markers1 = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1},
    {p1[[1]], p1[[2]], p1[[3]]}, {zPole[[1]], zPole[[2]], zPole[[3]]}};

graphicTry3 = ListPointPlot3D[markers1, AspectRatio → 1, BoxRatios → {1, 1, 1},
    PlotStyle → Red, PlotRange → All, PlotRangePadding → 0.1];
```



```
Show[graphicTry1, graphicTry2, graphicTry3]
```

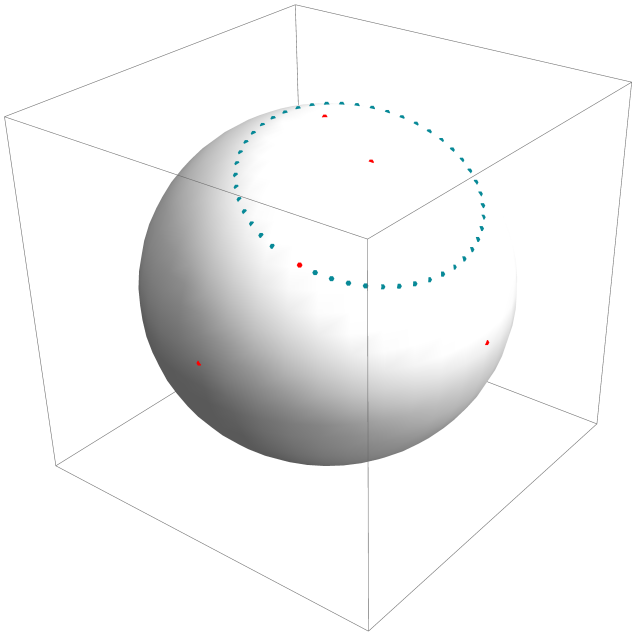


Figure 5-2. The graphic displayed above can be manipulated with the computer's mouse or trackpad. The red dots are plotted where the positive X, Y and Z axes intersect the sphere, at the pole of rotation located at latitude 65°N longitude 70°E, and at the initial location of a reference point at latitude 35°N longitude 30°E. The blue dots are the locations of the reference point seen at 0.5 hour intervals as it rotates around the pole with an angular velocity of 15°/hour.

5.5 Rotating a rigid body

We will consider a rigid body to be defined by three or more points on the surface of our sphere that do not move relative to each other. We will save a more substantial discussion of this for later, because our interest at this moment is to develop the ability to move more than one point around the surface of a unit sphere at the same time, such that the reference points are not in motion with respect to each other.

First, we will specify the unit location vector to each of three points that will constitute the vertices of a triangle. Our goal is to learn how to move that triangle around the rotational pole we used in the previous section, such that the triangle remains the same size and shape as it is displaced around the pole. The triangle remains a rigid body, without internal distortion. Let's define the map coordinates of vertex 1 at latitude 35°N, longitude 10°W; vertex 2 is latitude 40°N, 5°W; and vertex 3 is latitude 37°N, longitude 0°.

```
vertex1 = {(Cos[35 Degree] Cos[-10 Degree]),
    (Cos[35 Degree] Sin[-10 Degree]), (Sin[35 Degree])};

vertex2 = {(Cos[40 Degree] Cos[-5 Degree]),
    (Cos[40 Degree] Sin[-5 Degree]), (Sin[40 Degree])};

vertex3 =
  {(Cos[37 Degree] Cos[0 Degree]), (Cos[37 Degree] Sin[0 Degree]), (Sin[37 Degree])};
```

Second, we make a list of vectors called **triangle**.

```
triangle = {vertex1, vertex2, vertex3};
```

Third, we use the circMotion function act on each of the three vectors in triangle and generate vectors to points at different times during the rotation, creating a list of vectors called **prelimResults**.

```
prelimResults =
  Table[circMotion[triangle[[i]], j1, ω, deltaT], {i, 1, 3}, {deltaT, 0, 23, 4}];
```

Fourth, we reduce the dimension of the **prelimResults** matrix so that it is a simple *n*-row, 3-column matrix where *n* is the number of time steps multiplied by the number of reference points. With **deltaT** specified as it is above, ranging from 0 to 23 in steps of 4, we have 6x3=18 columns.

```
outResults = Flatten[prelimResults, 1];
```

Finally, we create the appropriate output graphics files to display the results. The initial position of the vertices are marked in red, and the displaced triangles are in blue.

```
graphicTry1 = Graphics3D[Sphere[{0, 0, 0}, 1],
    AspectRatio → 1, BoxRatios → {1, 1, 1}, PlotRange → All,
    PlotRangePadding → 0.1, ColorOutput → GrayLevel, Lighting → "Neutral"];

graphicTry2 = ListPointPlot3D[outResults, AspectRatio → 1, BoxRatios → {1, 1, 1},
    ColorOutput → GrayLevel, PlotRange → All, PlotRangePadding → 0.1];
```

We add the initial position of the respective vertices to the list of red marker dots that are plotted on the sphere.

```
markers2 =
  {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}, {vertex1[[1]], vertex1[[2]], vertex1[[3]]},
   {vertex2[[1]], vertex2[[2]], vertex2[[3]]}, {vertex3[[1]],
    vertex3[[2]], vertex3[[3]]}, {zPole[[1]], zPole[[2]], zPole[[3]]}};

graphicTry3 = ListPointPlot3D[markers2, AspectRatio → 1, BoxRatios → {1, 1, 1},
    PlotStyle → Red, PlotRange → All, PlotRangePadding → 0.1];

Show[graphicTry1, graphicTry2, graphicTry3]
```

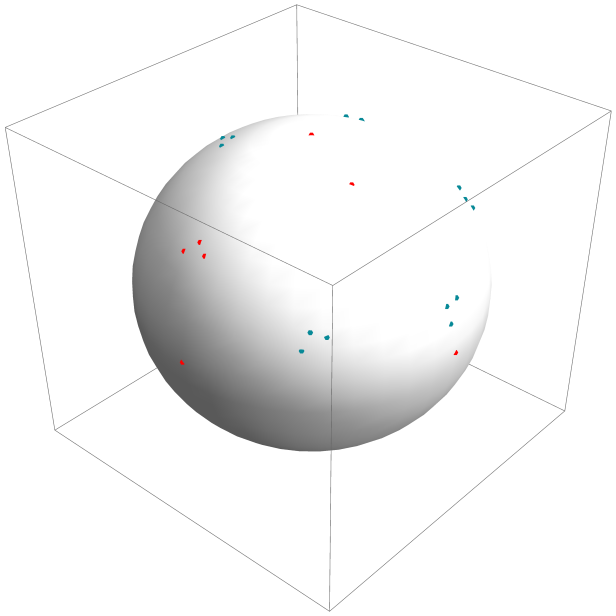


Figure 5-3. The graphic displayed above can be manipulated with the computer's mouse or trackpad. The red dots are plotted where the positive X, Y and Z axes intersect the sphere, the pole of rotation located at latitude 50°N longitude 30°E, and at the initial locations of the vertices of a triangle. The blue dots are the locations of the triangle seen at 4 hour intervals as it rotates around the pole with an angular velocity of 15°/hour.

Exercise 5.2 Write a *Mathematica* notebook that creates and rotates a polygon with more than 3 sides around a pole that is not coincident with the X, Y or Z axes of the Cartesian geographic coordinate system. Give the rotation an angular velocity of 10°/hour and produce a graphic showing the polygon every 4 hours for 32 hours. Make the polygon large enough so it can be clearly seen, but small enough so that its image at one time does not overlap with its image at another time.

