Longitude, Latitude, and Great Circle Distances... Part 1: The Basic Idea

Later Parts: Common Great Circle Forumlas and their derivations

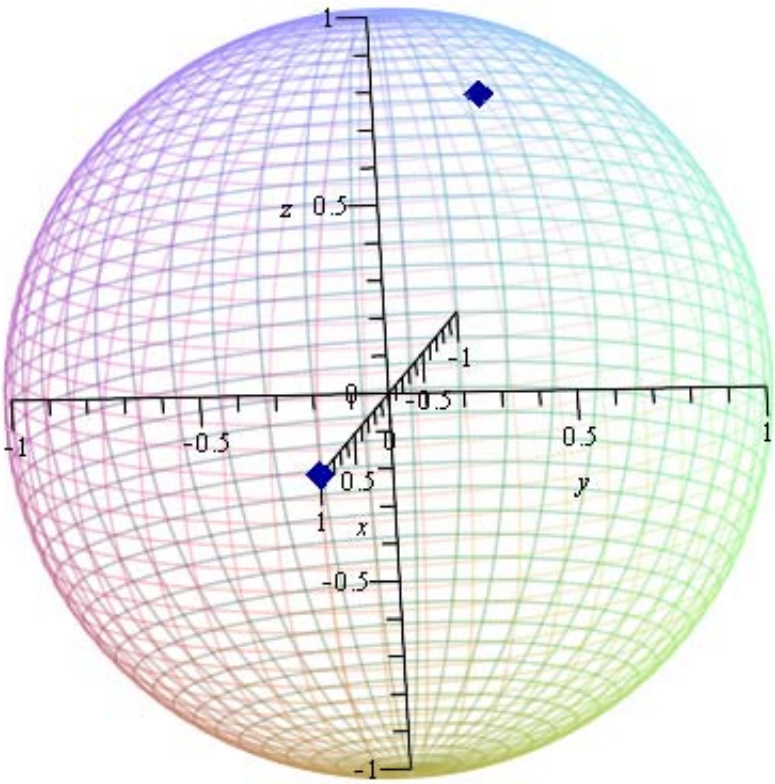
Brought to you by spatial.BurkeyAcademy.com ©2014

1. Why Using standard Cartesian distance formulas is wrong
2. What longitude and latitude mean
3. Visualizing Great Circles
4. The solution is in finding the angle between the vectors! (A simple example of how a distance formula might work)

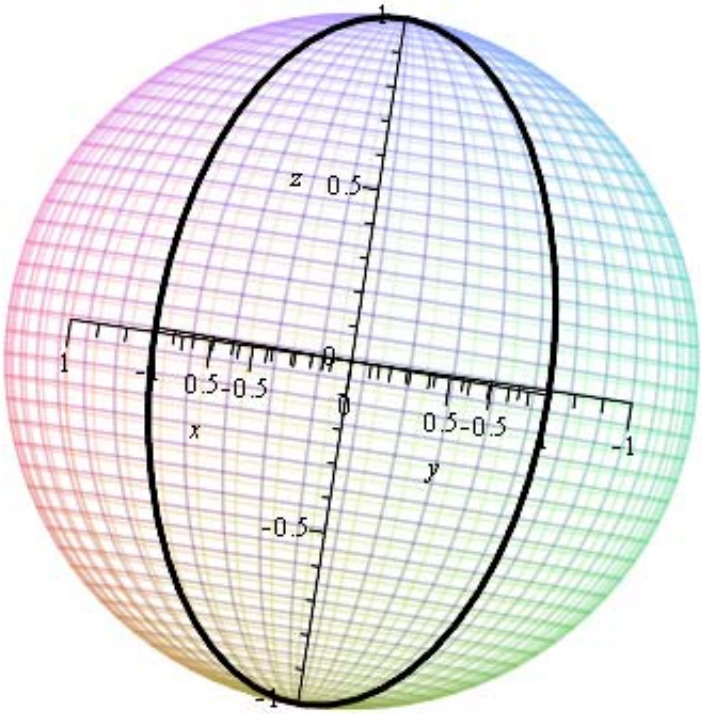
Sometimes people are tempted to calculate distances using longidue and latitude as if they were on a Cartesian grid, i.e. using the standard distance formula. You CANNOT do this! While a change in 1° latitude (North and South) always takes you 1/360th around the Earth (around 69.1 miles), a change of 1° longitude takes you 1/360th of the way around a circle that is not a "Great Circle"- this circle does not go all the way around the Earth. Latitude=distance, Longitude= ANGLE!

NOTE: It is much easier to do all of our calculations on a sphere with radius=1 (UNIT SPHERE). Earth is A) larger, and B) not a perfect shpere. We will ignore B), since the Earth only deviates from shperical a tiny bit (radius min-max 6353-6384 km, less than .5% difference). The average radius is 6371 km (3959 miles). We will adjust our calculations at the end to get the distance in miles or km, but will stick with radius = 1 until then. The circumference= 2Pi*r=2Pi for now.

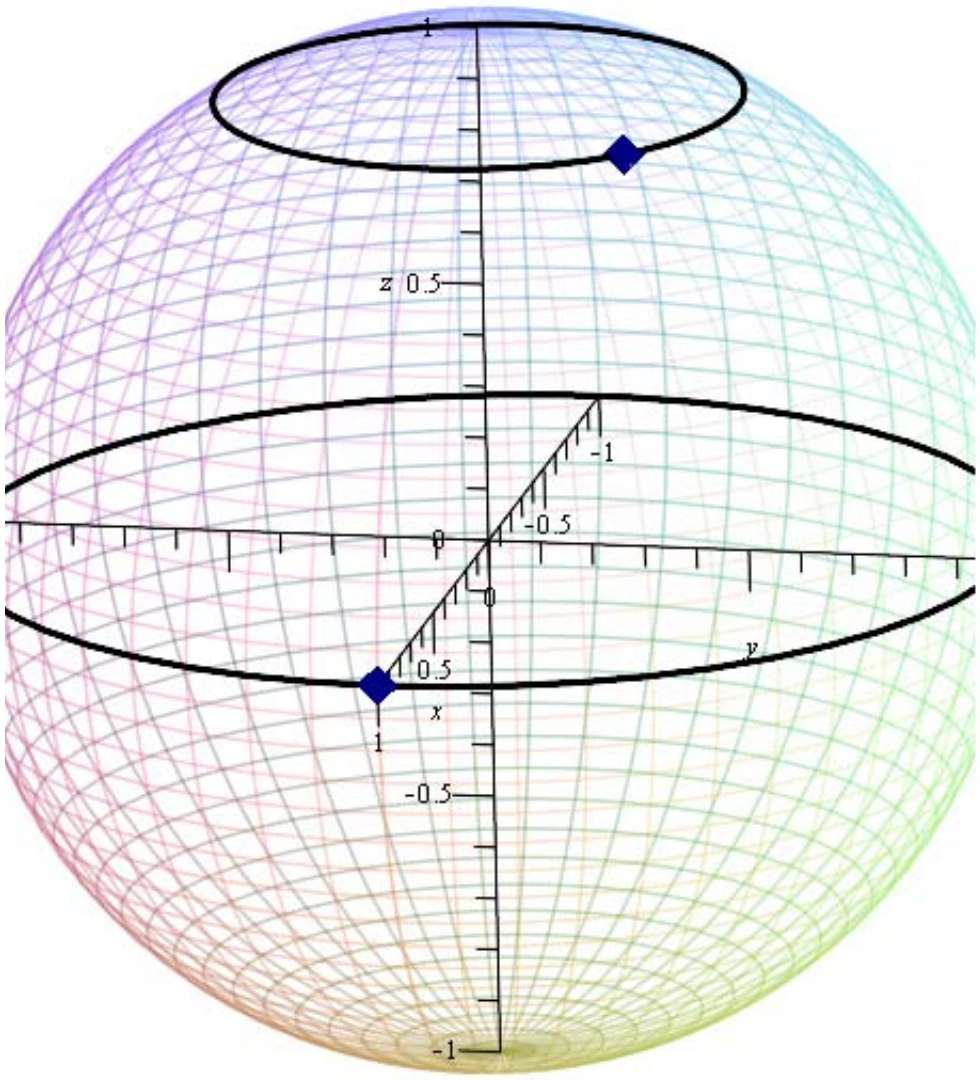
```
> restart; with(plots):
  sphere1 := sphereplot(1, θ = 0..2π, ϕ = 0..π, scaling = constrained, style = line, axes = normal):
      vec5 := Vector(3, [1, 0, 0]): vec6 := Vector(3, [.3536, .3536, .866]): points1
       := pointplot3d({vec5, vec6}, color = blue, symbolsize = 20):
      #These points correspond to longitude and latitude (0,0) and (45,60)
  display(sphere1, points1);
```



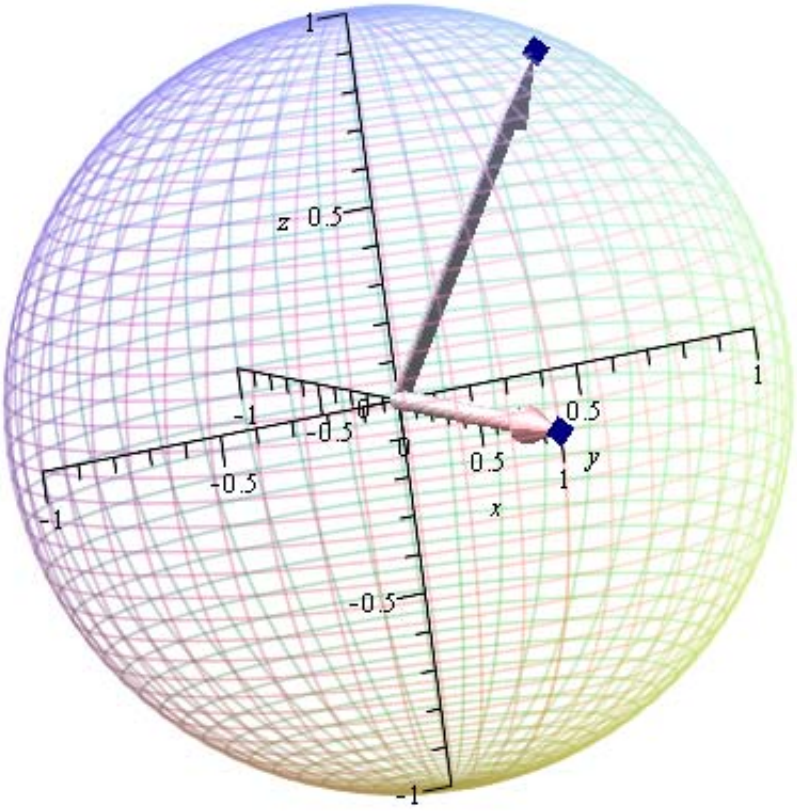

```
> gc4 := spacecurve( [0, cos(s), sin(s) ], s = 0 ..2·Pi, axes = normal, scaling = constrained,
      thickness = 3, color = black) : display( [sphere1, gc4] );
```




```
> gc1 := spacecurve( [cos(s), sin(s), 0], s = 0..2 * Pi, axes = normal, scaling = constrained,
  thickness = 3, color = black) : gc3 := spacecurve( [.5· cos(s), .5· sin(s), 0.866], s = 0..2
  * Pi, axes = normal, scaling = constrained, thickness = 3, color = black) : display( [sphere1,
  gc1, gc3, points1] );
```

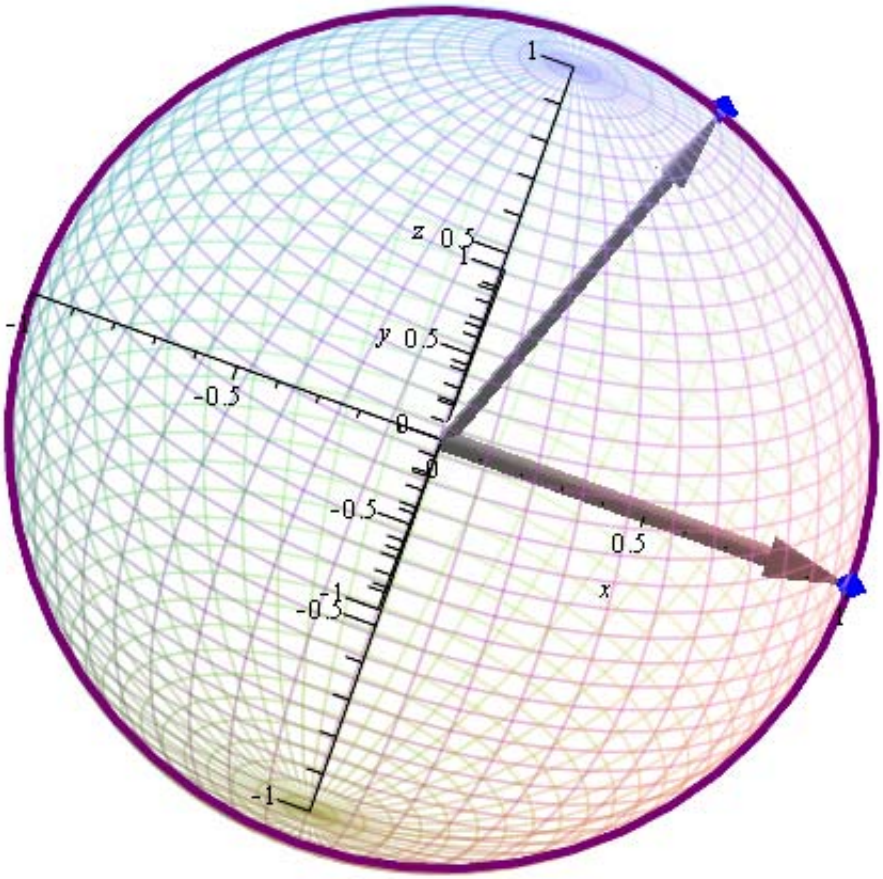

1
z 0.5
-1
-0.5
0
0
0.5
y
x
1
-0.5
-1

```
> display(sphere1, points1, arrow(vec5), arrow(vec6));
```



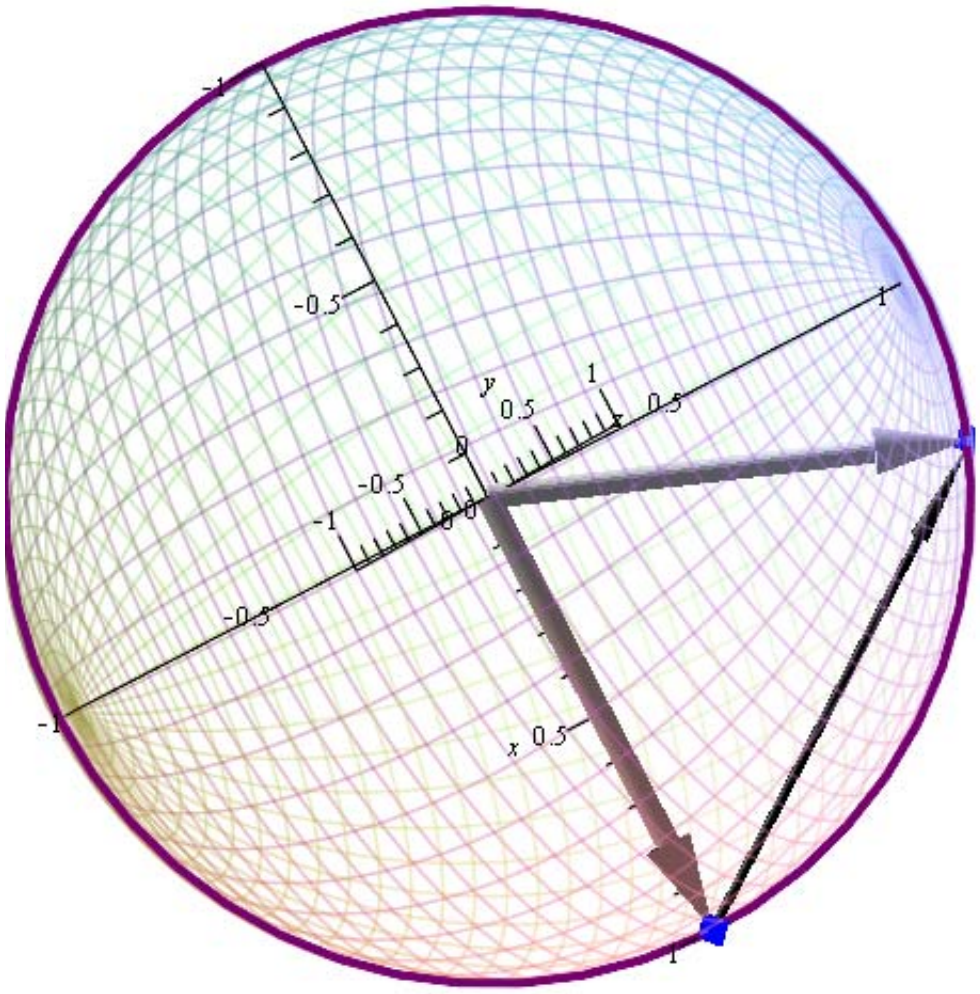
> $with(linalg) : b := evalf\left(scalarmul\left(crossprod(vec5, vec6), \frac{1}{dotprod(crossprod(vec5, vec6), crossprod(vec5, vec6))^{.5}}\right)\right) : gc2 := spacecurve\left(\left[b[1] \cdot b[3] \cdot \cos(s) - \frac{b[2] \cdot \sin(s)}{\left(b[1]^2 + b[2]^2\right)^{.5}}, b[2] \cdot b[3] \cdot \cos(s) + \frac{b[1] \cdot \sin(s)}{\left(b[1]^2 + b[2]^2\right)^{.5}}, -\cos(s) \cdot \left(b[1]^2 + b[2]^2\right)^{.5}\right], s = 0 ..2 * \mathrm{Pi}, axes = normal, scaling = constrained, thickness = 4, color = purple\right) : display(sphere1, points1, arrow(vec5), arrow(vec6), gc2);$

#this command creates a unit length vector that is perpendicular to the green and yellow ones. This is needed in order to draw a great sphere that goes through them, seen in the creation of gc2 below


1
z 0.5
1
y 0.5
-0.5
0
0
-0.5
-1
-0.5
0.5
x
1
-1

```
> dotprod(vec5 - vec6, vec5 - vec6)^.5;
      #find the distance between the two points, i.e. the third side of the triangle
> thirdside := arrow(vec5, vec6 - vec5, color = black, width = .02) : display(sphere1, points1,
      arrow(vec5), arrow(vec6), gc2, thirdside);
```

1.13702327152966


-1
-0.5
0
0.5
1
y
x
z