If I am here and heading towards there and I've covered this much ground, where am I?

Asked 16 years, 1 month ago Modified 16 years, 1 month ago

Viewed 920 times

2

I need help writing the following method:

```
def get_new_location(current_location,
target_location, distance_travelled):
    ...
    ...
    return new_location
```

where all locations are (lat,long)

I realize that there are different models for the earth (WGS-84, GRS-80, ...) which take into account the fact that the earth is an ellipsoid. For my purposes, this level of precision is not necessary, assuming a perfect sphere is good enough.

UPDATE

I'm fine tuning my question taking into account some of the responses.

`benjismith` argues that my question cannot be answered because there is more than one shortest path between points on the globe. He has a lot of backing in the form of votes, so I guess there's something I don't understand, because I disagree.

> The midpoint between any two locations on a sphere is a circular arc.

I concede that this is true when two points are at complete opposites. By this I mean that both points, while remaining on the surface of the sphere, could not be any further away from each other. In this case there are infinite number of equidistant paths joining both points. This, however, is an edge case, not the rule. In all other cases, the vast majority of cases, there is a single shortest path.

To illustrate: if you were to hold a string which passed through two points, and pulled it tight, would there not be only one possible path on which the string would settle (except the edge case already discussed)?

Now, prior to asking the question, obtaining the distance between two points and the heading was not a problem.

I guess what I should have asked is if the following is valid:

```
def get_new_location(current_location,
target_location, percent_traveled):
    new_location.lon = (1-
percent_traveled)*current_location.lon+percent_trave
    new_location.lat = (1-
percent_traveled)*current_location.lat+percent_trave
    return new_location
```

If I were to follow this path, would I be following the great-circle, the rhumb line, ... or would I be completely off? (I know these terms now because of Drew Hall's answer.)

great-circle dead-reckoning

Share

Improve this question

Follow

edited Nov 13, 2008 at 5:06

Brad Gilbert
34.1k ● 11 ● 79 ● 130

asked Nov 12, 2008 at 4:32

carrier
33k ● 23 ● 79 ● 100

Could this be a homework question perhaps? – mwjackson Nov 12, 2008 at 5:23

Somewhere in the middle? – paxdiablo Nov 12, 2008 at 6:46

@jacko No it's not a homework question. :) Although I wouldn't mind being back in school. – carrier Nov 12, 2008 at 14:45

Oops. I was (obviously) wrong, and I'm glad you noticed. I've deleted my answer. – benjismith Nov 13, 2008 at 3:15

@carrier: I'm curious about the final solution. Did you get it to work? – James Eichele Nov 18, 2008 at 5:29

4 Answers

Sorted by: Highest score (default)

3

As BenjiSmith said, there are potentially several paths that connect any A & B on the globe, but the two most popular (by far!) are the "great circle" and "rhumb line" paths.

A great circle gives the shortest distance (by constructing a plane from the two points & the center of the earth & following a circular arc in that plane).

A rhumb line maintains a constant heading, trading some distance (can be extreme at high latitudes) for ease of use. That is, in a boat or plane, you simply point at the desired heading and go until you arrive at your destination (whereas with a great circle the heading changes continuously). In mid latitudes the distance penalty isn't too severe.

Be warned, both path types have discontinuities involving the poles and ambiguities when dealing with antipodal points (pts opposite each other on the sphere).

To build a great circle, you'll want to convert the points to 3D cartesian coordinates (I'll leave this step out but it's trivial for a spherical earth & found iteratively for an oblate earth model a la WGS-84).

> Let **a** be the unit vector pointing at the start point from the center of the earth.

> Let **b** be the unit vector pointing at the end point from the center of the earth.
>
> Let *r* be the radius of the earth.
>
> Let *d* be the (given) distance traveled.

Construct the unit vector normal to the G.C. plane by taking the cross product of the unit vectors **a** and **b**. That is, let **n = a x b**.

The (given) distance traveled is the length of the arc formed by sweeping the vector *r***a** around **n** by some angle *theta*. Recalling that the circumference of the full great circle is 2 * pi * *r*, we find *theta* = *d*/*r*.

The cartesian point corresponding to the new location is thus found by rotating *r***a** around **n** by *theta* radians. Convert that cartesian point to lat/long & you're done.

I won't derive the rhumb line math here, but will say that the Mercator map projection has the property that rhumb lines are straight. You can easily construct a rhumb line with the mercator projection formula, but you'll have to define some error tolerance so you can break the path up into short, straight segments.

Good luck!

Share Improve this answer Follow

edited Nov 13, 2008 at 4:18

answered Nov 12, 2008 at 5:53

Drew Hall
29k ● 12 ● 65 ● 81

2

Regarding your updated question:

What you seem to be doing is a linear interpolation of lat/lon coordinates. This is a valid path, but it's neither the great circle or the rhumb line. In fact, because meridians converge as latitude increases (in the northern hemisphere, at least), a smooth interpolation in the lat/lon sense would result in an oddly accelerating path on the ground.

If you were describing interpolation in cartesian coordinates, you'd at least be moving in the right plane but the path would cut through the surface of the earth (i.e. it'd be a chord on the great circle, rather than an arc).

Share Improve this answer Follow

answered Nov 12, 2008 at 20:42

Drew Hall
29k ● 12 ● 65 ● 81

2

Here is some sample code that should do the trick. The algorithm works for all cases and always follows the shortest great-circle path between the two locations. The math is essentially identical to Drew Hall's answer, but using percent_traveled and ignoring the radius of the earth.

For the sake of simplicity, this code assumes that latitude and longitude are stored in radians.

```
def get_new_location(current_location,
target_location, percent_traveled):

    # convert locations into cartiesian co-
ordinates
    current_vector =
location_to_vector(current_location)
    target_vector =
location_to_vector(target_location)
    # compute the angle between current_vector and
target_vector
    complete_angle =
acos(vector_dot_product(current_vector,
target_vector))
    # determine the current partial angle, based
on percent_traveled
    partial_angle =
percent_traveled*complete_angle
    # compute a temporary vector to simplify
calculation
    temporary_vector =
vector_cross_product(current_vector,
target_vector)
    temporary_vector =
vector_cross_product(current_vector,
temporary_vector)
    # calculate new_vector
    scalar_one = cos(partial_angle)
    scalar_two = -
sin(partial_angle)/sin(complete_angle)
    vector_one =
vector_multiply_by_scalar(scalar_one,
current_vector)
    vector_two =
vector_multiply_by_scalar(scalar_two,
temporary_vector)
    new_vector = vector_sum(vector_one,
vector_two)
```

function to convert from latitude and longitude to cartesian co-oridinates:

```
def location_to_vector(location)
    vector.x = cos(location.lat)*sin(location.lon)
    vector.y = sin(location.lat)
    vector.z = cos(location.lat)*cos(location.lon)
    return vector
```

function to convert from cartesian co-oridinates to latitude and longitude:

```
def vector_to_location(vector)
    location.lat = asin(vector.y)
    if (vector.z == 0):
        if (vector.x < 0):
            location.lon = -pi/2
        else:
            location.lon = pi/2
    else:
        if (vector.z < 0):
            if (vector.x < 0):
                location.lon =
atan(vector.x/vector.z) - pi
            else:
                location.lon = pi - atan(-
vector.x/vector.z)
        else:
            if (vector.x < 0):
                location.lon = -atan(-
vector.x/vector.z)
            else:
                location.lon =
atan(vector.x/vector.z)
    return location
```

function to compute the dot-product of two vectors:

```
def vector_dot_product(A, B):
    dot_product = A.x*B.x + A.y*B.y + A.z*B.z
    return dot_product
```

function to compute the cross-product of two vectors:

```
def vector_cross_product(A, B):
    cross_product.x = A.y*B.z - A.z*B.y
    cross_product.y = A.z*B.x - A.x*B.z
    cross_product.z = A.x*B.y - A.y*B.x
    return cross_product
```

function to multiply a vector by a scalar:

```
def vector_multiply_by_scalar(scalar, vector)
    scaled_vector.x = scalar*vector.x
    scaled_vector.y = scalar*vector.y
    scaled_vector.z = scalar*vector.z
    return scaled_vector
```

function to compute the sum of two vectors:

```
def vector_sum(A, B)
    sum.x = A.x + B.x
    sum.y = A.y + B.y
    sum.z = A.z + B.z
    return sum
```

Share Improve this answer Follow

edited Nov 14, 2008 at 18:06

answered Nov 14, 2008 at 16:35

James Eichele
119k ● 41 ● 182 ● 214

Your updated sample code would not always follow the correct path.

-1

For a quick example, consider the following two points on the equator in the middle of the pacific ocean:

- current_location: lat = 0, lon = -179
- target_location: lat = 0, lon = 179

These two points are very close together (only two degrees of longitude apart), but when percent_traveled is at 0.5, new_location would be: lat = 0, lon = 0, which is a point on the opposite side of the globe.

edit: it gets worse

Consider the follwing two points in the northern hemisphere:

- current_location: lat = 80, lon = 0
- target_location: lat = 80, lon = 180

The great circle path between these two points goes directly over the north pole, but new_location would move *around* the globe, remaining parallel to the equator.

Share Improve this answer Follow

edited Nov 12, 2008 at 23:41

answered Nov 12, 2008 at 23:17

James Eichele
119k ● 41 ● 182 ● 214

thanks for you input. you make a valid point. i already had a strategy for dealing with this though, i just didn't want to clutter the example code so that the focus would remain the issue i needed help with. – carrier Nov 13, 2008 at 2:12

Fair enough. I have posted a new answer with some sample code in an effort to answer the original question. Hope it helps! – James Eichele Nov 14, 2008 at 17:48