Subject: Finding all angles within a range of directions; an algorithm question
Posted by tbowers0 on Thu, 11 Apr 2002 21:01:30 GMT
View Forum Message <> Reply to Message

Say I have a 3D array of observation data of 'Brightness' over the hemisphere of theta,phi angles (theta=0 is straight up, phi=0 is North, and 3rd dim is timesteps).

data = findgen(3,4,10) ;simple example with 3 thata angles, 4 phi angles, 10 timesteps
theta = [0,30,60] ;a *very* simple example
phi = [0,90,180,270]
timestep = indgen(10)

If I have a flat plate facing an arbitrary angle, how do I find all brightnesses that fall on its surface. In other words, all brightness[*,*,*] that come from angles within +- 90 degree hemisphere of plate's surface (of course, all angles are really in radians, but listed here as degrees for clarity). The only solution I come up with requires:

1) reforming the brightness data to a list of theta,phi,timestep,brightness quadruplets (now a 2D array [4,n_thetas*n_phis*n_timesteps]; about 7 or 8 lines of code reform()ing and transpose()ing). Doing this cause I'll use where() in a moment

2) convert the theta,phi polar coordinates to x,y,z cartesian coordinates by:
brightnessAnglesCartesian = sin(brightness[0,*]) * cos(brightness[1,*]), sin(brightness[0,*]) * sin(brightness[1,*]), cos(brightness[0,*])]

3) convert the plate's theta,phi polar coordinate facing direction (its 'normal') to x,y,z cartesian coordinate by same formula to create plateAngleCartesian, a 3-element vector

4) compute all angles psi that plate normal (plateAngleCartesian) makes with all brightness angles (brightnessAnglesCartesian) by:
psi = acos(plateAngleCartesian # brightnessAnglesCartesian) ;acos(dot product of 3x1 plate angle vector and 3xN brightness angle vectors)

5) indices = where(psi le !pi/2.0)

6) Now I can work with these angles: brightness[3,indices]) = 0.0 ;brightnesses in 4th column

This seems awfully circuitous. I'm using an interactive interface to

rotate the plate with the mouse so calculation speed is critical and I think my method is way too slow (and not very elegant either). Could anyone please advise on a better way to do this? One thing I've learned is that when dealing with angles and rotations, there are usually very quick and clever alternatives than my usual brutish approach.

Many thanks in advance

Subject: Re: Finding all angles within a range of directions; an algorithm question
Posted by Craig Markwardt on Thu, 18 Apr 2002 01:16:00 GMT
View Forum Message <> Reply to Message

tbowers0@yahoo.com writes:

> Wow Struan! Your explanation was a bit beyond me. So, instead of just
> replying "Uhh.. Huh?" I did some surfing on rotation matrices and
> stuff and found the Matrix and Quaternion FAQ at
> http://skal.planet-d.net/demo/matrixfaq.htm. Educated myself a bit,
> but I'm still unclear. If I understand:

Hi Todd--

You definitely want to use the dot-product approach. It's fast, and there's no room for screwing up the signs like there is in rotation matrices. And, as you say, you can make a different cut, other than 90 degrees.

But, if you *did* want to use quaternions, then I have a whole library for you on my web page :-)

Craig

Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Finding all angles within a range of directions; an algorithm question
Posted by Struan Gray on Thu, 18 Apr 2002 09:27:13 GMT
View Forum Message <> Reply to Message

Craig Markwardt, craigmnet@cow.physics.wisc.edu writes:
>
> tbowers0@yahoo.com writes:
>>
>> Wow Struan! Your explanation was a bit beyond me. So, instead of just
>> replying "Uhh.. Huh?" I did some surfing on rotation matrices and
>> stuff and found the Matrix and Quaternion FAQ at
>> http://skal.planet-d.net/demo/matrixfaq.htm. Educated myself a bit,
>> but I'm still unclear. If I understand:
>
> You definitely want to use the dot-product approach. It's fast, and
> there's no room for screwing up the signs like there is in rotation
> matrices. And, as you say, you can make a different cut, other than
> 90 degrees.

I am more than happy to bow to Craig's superior knowledge, but in case you want to do it the hard way here's a more coherent (I hope) explanation of what I meant.

Rotations are coordinate transformations, and if you have an array of coordinates you can transform them all at once by doing a matrix multiplication between the array of coordinates and a rotation matrix. In deciding which method to use you have to weigh the possible speed penalty of doing a whole series of dot products against the difficulties of constructing the correct rotation matrix.

The rotation you want is the one which will rotate the current theta=0 direction into the surface normal of your plate. Find that, and you then just rotate all the other coordinates and find the ones who have a new theta greater than zero. The rotation is about the vector formed by the cross product of the plate normal and the original zenith, with an angle found by vector geometry.

As Craig said, the real problem is keeping track of all your sign conventions for rotations and directions. This is a common problem in astronomy (it is roughly the same as asking which stars will be above the 'horizon') so some of the astro libraries (or Craig's) may have already done the hard work in your particular case.

Another place the hard work has been done is the IDL object graphics libraries. IDLgrModel objects have a method called GetCTM which returns the correct transformation matrix to map coordinates in the Model's internal coordinate system to those of any of the models higher up the tree, and finally the View object itself. In this case all coordinates are cartesian, so you would still have to do a sperical-polar to cartesian conversion, but IDL would handle the signs properly.

Once you have the transformation matrix you apply it to an array which

lists the theta and phi values for each data entry in your matrix. You then have to find those that match a criterion for visibility, which varies with the coordinate system you've ended up in: i.e. theta<pi/2 or z>0. Finding elements of an array which match a condition is an old chestnut here in the newsgroup, and there are essentially three ways of doing it.

First, and most obvious, is WHERE. Here the only difficulty is taking the one dimensional indices WHERE returns and reforming them to index your original two or more dimensional array.

The second technique uses HISTOGRAM's useful REVERSE_INDICES keyword to identify all elments with a particular value or which lie in a range of values. This is usually faster then WHERE, but you have to parse the REVERSE_INDICES array correctly.

The third technique uses a direct compare on the whole array at once, i.e. if z is an array of tranformed z-values, just write:

```
visible = z>0
```

and visible will be an array of the same size as z with 1's where z is greater than zero and 0's where it's not. This is faster yet, but whether it is useful depends on what you want to do. It makes it very easy to increment or add up those values:

```
new_z = z + visible*increment
conditional_total = total(z*visible)
```

So there you go, more than you ever wanted to know. Happy hunting and let us know what you end up with.

Struan
