HEALPix_MongoDB

July 3, 2019

1 MongoDB Cone Searches with HEALPix

In this notebook, we will go over how you can implement simple astronomical cone searches using HEALPix in a NoSQL database. We'll use MongoDB and the `BrownDwarf` collection we created in the first tutorial. Now, MongoDB *does* have spatial indexing capabilities, but we'll go over those in a future post. In this case, we will assume we don't know or can't use these and thus build our own way to quickly search for objects using sky coordinates. A write up of this topic is also available in my blog.

1.1 HEALPix

HEALPix stands for Hierarchical Equal Area isoLatitude Pixelisation and is an algorithm for dividing up the sky in equal-area pixels. This is in contrast to other algorithms like HTM, Hierarchical Triangular Mesh, which divide up the sky in triangles of equal-length. We'll stick to HEALPix since it's used as the basis for several IVOA standards like MOC and HiPS. There are two main Python packages for working with HEALPix- `healpy` and `astropy-healpix`. For simplicity, I'll only use the later code here. For more details, refer to the documentation.

```
In [1]: from astropy_healpix import HEALPix, pixel_resolution_to_nside
        from astropy.coordinates import ICRS, SkyCoord
        from astropy import units as u

        # nside required for chosen resolution
        resolution = 10 * u.arcsec
        nside = pixel_resolution_to_nside(resolution, round='up')
        print('NSide: ', nside)

        # HEALPix object with that resolution
        hp = HEALPix(nside=nside, order='nested', frame=ICRS())

        print(hp.pixel_resolution.to(u.arcsec))
        print('Number of pixels: ', hp.npix)
```

```
NSide:  32768
6.441537022157636 arcsec
Number of pixels:  12884901888
```

Note that HEALPix values could be equatorial, galactic, ecliptic and their numbering can follow two schemes- RING or NESTED. For consistency with other standards (namely MOC), we'll chose ICRS equatorial and NESTED. In the code above, I create a HEALPix object with pixel resolution that's at least 10 arcseconds or better. That does mean that objects closer than my resolution will be inside the same pixel. For practical applications, you'll probably want a resolution comparable to your PSF or instrument pixel size.

The nside is just a measure of resolution- it's the number of pixels along an edge of the highest level HEALPix pixels. The larger it is, the smaller the pixels, and the more precise your representation. At the chosen precision level, you can see that there are over 12 billion pixels, each represented with an integer value. For example, here's what pixel 42 at this resolution level corresponds to in real-sky coordinates:

```
In [2]: lon, lat = hp.healpix_to_lonlat([42])
        print(lon, lat)
        print(lon.degree, lat.degree)
```

```
[0.78523rad] [0.00016276rad]
[44.99038696] [0.00932548]
```

```
In [3]: coords = hp.healpix_to_skycoord([42])
        print(coords)
```

```
<SkyCoord (ICRS): (ra, dec) in deg
    [(44.99038696, 0.00932548)]>
```

```
In [4]: # Now, for the reverse
        coords = SkyCoord(ra=34*u.deg, dec=-23*u.deg)
        print(hp.skycoord_to_healpix(coords))
```

```
9542850888
```

With this you can probably start putting things togeter- if we can store object locations as integers, then it's just a matter of comparing lists of integers to see if my search area intersects any physical objects. Let's have a look at one more built-in method of `astropy-healpix`:

```
In [5]: coords = SkyCoord(ra=34*u.deg, dec=-23*u.deg)
        hp_to_search = hp.cone_search_skycoord(coords, radius=5 * u.arcmin)
        print(len(hp_to_search))
        print(hp_to_search[0:10])
```

```
6988
[9542850888 9542850889 9542850891 9542850890 9542850847 9542850845
 9542850839 9542850882 9542850883 9542850886]
```

This cone search function provides all HEALPix pixels that are in the specified area. Note that because of our chosen resolution level, as well as the size of the cone search, there can be a lot of pixels returned. In this example, nearly 7000 integers were returned. If you have an arbitrary shape, perhaps one defined by an STC-S polygon, it's not too hard to code up something that will return all pixels inside your shape.

1.2 MongoDB Updates

Now that we have prototyped how to create HEALPix values for target RA/Dec and for cone searches, let's update the `BrownDwarf` collection we previously generated in the first tutorial to include these values. For that, we'll loop over every document, load it as a dictionary, get the healpix value, and add it to that document. For more information about how to connect to MongoDB, I'll refer you to my first tutorial as well as the pymongo documentation.

```
In [6]: import pymongo

        client = pymongo.MongoClient()  # default connection (ie, local)

        db = client['test']  # database
        dwarfs = db.dwarfs  # collection
```

If you look closely, you'll notice my query uses `$exists`, which is a MongoDB operator that checks exactly what it says- if the specified field exists or not, the result is returned. With this query, I only select those cases without a healpix field. If you continue to build this database, using `$exists` ensures you don't re-calculate healpix values you already computed.

```
In [7]: dwarfs.update_many({}, {'$unset': {'coords.healpix': ''}})  # remove the coords.healpix

        # Loop over those without coords.healpix and set the value
        cursor = dwarfs.find({'coords.healpix': {'$exists': False}})
        for doc in cursor:
            coords = SkyCoord(ra=doc['coords']['ra']*u.deg, dec=doc['coords']['dec']*u.deg)
            healpix = int(hp.skycoord_to_healpix(coords))
            print(doc['source_id'], doc['coords']['ra'], doc['coords']['dec'], healpix)
            dwarfs.update_one({'_id': doc['_id']}, {'$set': {'coords.healpix': healpix}})
```

```
11 181.889 -39.548 6443584085
2 202.95387 -1.280556 6777358882
4 222.106791 10.533056 2188614871
7 219.868167 19.487472 2305451854
14 342.472709 0.734611 4945615057
15 332.05679 29.355972 3527258534
17 6.924875 5.061583 4759262121
19 327.068041 40.0665 3639926034
20 165.54097 -34.509869 10061678832
32 63.831417 -9.585167 5945842394
34 111.826001 17.167 5903955082
36 72.753833 -34.0375 9077122522
53 228.753459 48.794889 2960203509
61 191.309 -44.485477 11422991835
63 53.537667 -49.893944 8822707629
80 238.24591 29.81342 2460281099
82 278.90792 32.998497 3895121831
83 236.94662 -24.397028 11616859038
86 9.067376 18.352889 5205599093
```

```
91 79.692333 -27.946028 5504145867
96 42.170846 -16.856022 9604959407
98 40.297958 -3.449639 4647676099
```

Note that for purposes of this example, the first line does an `update_many()` command with the `$unset` operator to remove that field in the first place. The command `update_one()` is later used with `$set` to add the healpix value. Also, I explicitly set the type of healpix to be `int` as pymongodb appeared to have trouble with its default type (numpy.int64).

For good measure, we'll create an index out of this field, to make any searches a bit faster:

```
In [8]: if 'healpix' not in dwarfs.index_information():
            dwarfs.create_index([('coords.healpix', pymongo.ASCENDING)],
                                name='healpix', background=True)
```

1.3 Cone Search Function

Now that we have healpix values in the database, we can start using them for cone searches. Let's do a manual search first. Remember that due to our resolution level, we can only explore down to sizes of about 6 arcseconds and the list of pixels for much larger areas can quickly grow very large.

```
In [9]: coords = SkyCoord(ra=63.8*u.deg, dec=-9.6*u.deg)
        hp_to_search = hp.cone_search_skycoord(coords, radius=10 * u.arcmin)
        print(len(hp_to_search))
        print(hp_to_search[0:10])
```

```
27621
[5945845060 5945845061 5945845063 5945845062 5945845059 5945845057
 5945839595 5945839598 5945839599 5945839610]
```

```
In [10]: cursor = dwarfs.find({'coords.healpix': {'$in': [int(h) for h in hp_to_search]}})
         for doc in cursor:
             print(doc['source_id'], doc['coords'])
```

```
32 {'ra': 63.831417, 'dec': -9.585167, 'healpix': 5945842394}
```

Note that, as before, I had to convert the healpix values to integers. I also had to make use of the `$in` MongoDB operator, which matches any document whose field value is in the array. It's very important that you use the same healpix parameters (nside, frame, nested or ring ordering) as what you stored in the database. Otherwise the healpix values will be different and you'll never find a match.

Finally, let's put everything together and write a utility wrapper to perform these cone searches against the database without having the user calculate HEALPix values manually.

```
In [11]: def cone_search(ra, dec, radius, collection=dwarfs, field='coords.healpix', hp=hp):
             # Function to perform a simple cone search against a MongoDB collection
```

```
            # ra, dec in degrees; radius in arcseconds
            if radius > 1800.:
                raise RuntimeError('Only cone searches up to 1800 arcsec (0.5 deg) are supporte

            coords = SkyCoord(ra=ra*u.deg, dec=dec*u.deg)
            hp_to_search = hp.cone_search_skycoord(coords, radius=radius * u.arcsec)
            cursor = collection.find({'coords.healpix': {'$in': [int(h) for h in hp_to_search]}
            return cursor
```

```
In [12]: cursor = cone_search(181.9, -39.5, 180.)
         for doc in cursor:
             print(doc['source_id'], doc['coords'])
```

```
11 {'ra': 181.889, 'dec': -39.548, 'healpix': 6443584085}
```

And that's all there is to it. Note that I added a small check to ensure you don't supply very large search radii. I noticed that on my machine, the computation of healpix values took a bit of time so I wanted to make sure I didn't accidentally send a very large query.

You could alter the function to take in a SkyCoord object. Or you could go more generic and use the latitute and longitude functionality of the astropy-healpix package.

1.4 Final Thoughts

In this tutorial, we went over how to calculate HEALPix values for a particular region of the sky, from both a target and cone search perspective. We then updated our existing MongoDB database to contain this HEALPix values and built a simple cone search function that queries the database for a user supplied set of coordinates. Now, there are other ways to break up the sky, such as HTM, and more compact ways to represent regions with HEALPix, such as MOC. We're not covering those at the moment. Finally, before you go out and populate large databases with HEALPix values, I'd like to note that MongoDB has built-in spatial functionality. Setting that up takes a bit of time, and you'll have to do some extra conversions for astronomical work, but I hope to present a future tutorial going over MongoDB's geospatial queries.