

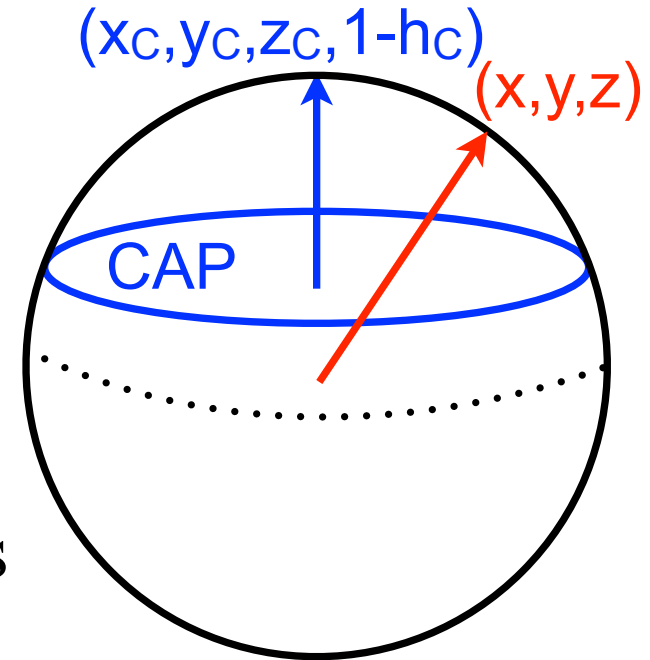
General Masking

Masking and Astronomical Surveys

- There are 3 main purposes to *Mangling* the sphere. The two purposes we will study are
 - to determine which objects in the sky lie within an arbitrary region defining a survey footprint
 - to populate that region with a catalog of random points to model the conditions of an *ideal* survey
 - These two purposes are often referred to as “masking” or “creating a mask”
 - The third purpose is to determine the *area* of the intersecting polygons used to model surveys
 - this requires a little extra math (applied to caps)
 - Areas are stored in a Mangle polygon as `str`
-

Spherical Cap Constraints

- The spherical cap formalism makes it easy to determine if points lie in intersections of polygons
- Consider whether a point (x,y,z) lies in the cap $(x_c,y_c,z_c,1-h_c)$
- Remember, $1-h_c=1-\cos\theta_c$ where θ_c is the “angular radius” on the sphere codified by the cap’s size (called the *cap constraint*)
- Take the dot product between the cap and the point
 - $(x_c,y_c,z_c).(x,y,z) = |1||1|\cos\theta_{\text{between cap and point vectors}}$
- Now if $\theta > \theta_c$ then the point lies outside of the cap and if $\theta < \theta_c$ then the point lies within the cap



Spherical Cap Constraints

- Algorithms to determine if points lie in spherical caps are very rapid, because they only require linear algebra
 - i.e. it is never necessary to use trigonometric functions (because if $\theta > \theta_c$ then $\cos\theta < \cos\theta_c$)
 - In the *pymangle* version of *Mangle* that we installed last week, any mask (and by extension, any cap or polygon) can be tested against a set of (RA, dec) coordinates using *contains*, e.g.
 - *mask* = *pymangle.Mangle*("file.ply")
 - *ra* = *np.array*([47.3, 152.7, 23.3, 280.4])
 - *dec* = *np.array*([-11.2, 12.2, 88.7, -39.2])
 - *good* = *mask.contains*(*ra*, *dec*)
-

Random Catalogs

- In the *Areas on the Sphere and HEALPix* notes, I provided equations for populating the sphere in equal-area angular projection with random points
 - To construct such a random catalog in a collection of polygons, we could populate the entire sphere in equal-area projection and then use *mask.contains* to find just the points that lie in the mask
 - *pymangle* implements a similar method for making random catalogs in masks called *genrand*
 - We already used *genrand* in the previous lecture, e.g.
 - *mask = pymangle.Mangle("file.ply")*
 - *ra_rand, dec_rand = mask.genrand(10000)*
-

Random Catalogs and weights

- One part of a *Mangle* polygon I have yet to discuss in detail is the `weight`
 - The reason for different `weights` is that when *genrand* creates a random catalog it will create proportionately more random points in polygons with more `weight`
 - This has real applications to astronomy. Consider taking spectroscopy of a plate of targets in the sky, for which good spectra are obtained for 80%
 - If that plate is modeled as a polygon then `weight=0.8`
 - The `weight` of a second plate might also be 0.8 but the weight of the *intersection* of the two plates should be higher, *as more objects can be observed in the overlap*
-

Python tasks

1. Create a polygon in a *Mangle* file consisting of 4 caps that define a “lat-lon rectangular” field bounded in RA by 5^h and 6^h and in declination by 30° and 40°
 - Use the *ra_cap* and *dec_cap* functions you wrote as part of the *Spherical Caps* lecture
 - In the *Areas on the Sphere and HEALPix* notes, I showed that the area of a “lat-lon rectangle” in steradians is $(\alpha_2^{\text{radians}} - \alpha_1^{\text{radians}})(\sin\delta_2 - \sin\delta_1)$
 - *Calculate* the correct `str` area for your polygon, add it to the file, and give the polygon a `weight` of 0.9
 2. Add a second polygon to your file for a field bounded in RA by 10^h and 12^h and in declination by 60° and 70°
 - add `str` for this polygon, and give it a `weight` of 0.2
-

Python tasks

3. Create a random catalog of 1 million objects distributed over the entire sphere
 - See, e.g., the *Areas on the Sphere and HEALPix* notes
 4. Use *mask.contains* to determine which points in your random catalog lie within the “lat-lon rectangular” polygons in your file
 - plot the entire random catalog, and over-plot just the points that lie within the polygons in a different color
 5. Use *mask.genrand* to generate 10,000 random points within your polygons
 - Is the density of random points that *genrand* creates the same in each of your polygons? Why or why not?
-