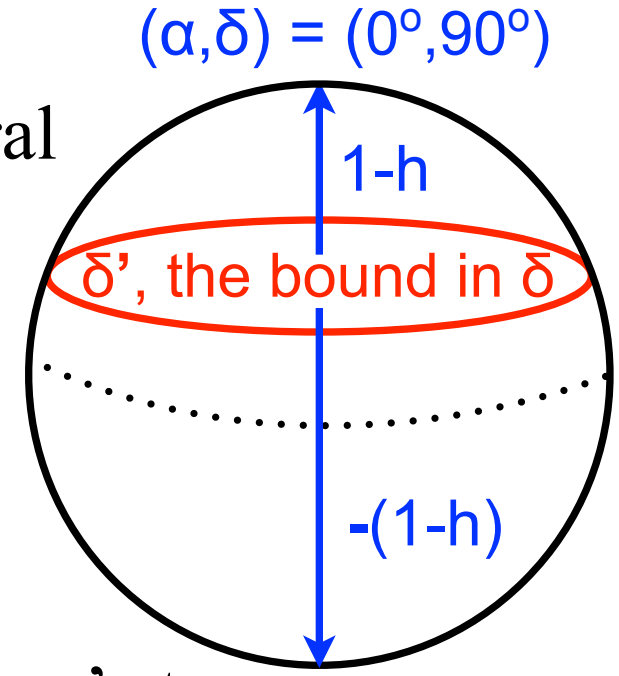
Mangle

Intersections of multiple caps

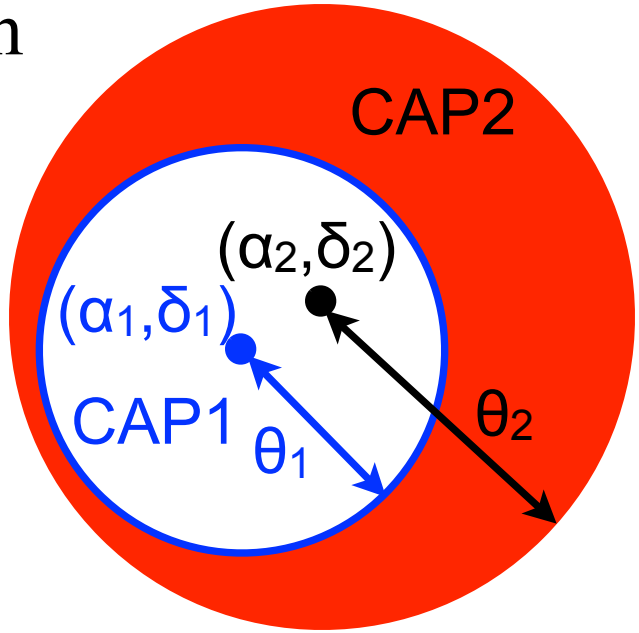
- By intersecting *multiple* spherical caps it is possible to construct general shapes on the surface of the sphere
- So, e.g., a "square" field bounded by (α_{min}', α_{max}', δ_{min}', δ_{max}') can be represented by the intersection of 4 spherical caps



- One cap bounded by α_{min}', one by α_{max}' etc.
- But, for this sort of scheme, it is sometimes necessary to represent the sphere on the *other side of the cap*
- This is achieved simply in our convention by flipping the sign of (1-h) so that (x,y,z,(1-h)) → (x,y,z,-(1-h))

Intersections of multiple caps - *Polygons*

- Consider the red area in the diagram
- It consists of CAP1 and CAP2, but we want the intersection of everything that is *in* CAP2 with everything that is *not in* CAP1
- We can create everything that is *not in* CAP1 by flipping the sign of the cap *constraint* (the cap size)



- So, if CAP2 is represented by $(x_2,y_2,z_2,1\text{-}h_2)$ and CAP1 by $(x_1,y_1,z_1,1\text{-}h_1)$, then the red area is represented by the intersection of $(x_2,y_2,z_2,1\text{-}h_2)$ and $(x_1,y_1,z_1,-(1\text{-}h_1))$
- We will refer to shapes represented by the intersection of multiple caps as *polygons*

Basic Python implementation of Mangle

- Software called *Mangle* has been developed to apply the spherical cap formalism. Our *astroconda* build has a basic Python implementation that can be imported with *import pymangle* (e.g. see the syllabus links)
- *Mangle* expects input files in the following format

```
1 polygons
polygon 1 ( 3 caps, 1 weight, 0 pixel, 0 str):
  -0.96592582629 0.25881904510 0 1
  0 0 1 0.4122147470752686
  0.209389006 0.781450409 0.587785253 0.00015230
```

- The first line states how many polygons are in the file note that `polygon(s)` and `cap(s)` are *always plural!*
- In the second line, `caps` specifies how many caps are in the polygon and `str` lists the area of the polygon in steradians (which you can leave as `0 str` for now)

Basic Python implementation of Mangle

- Note that if you wanted to create multiple polygons you could specify several blocks of caps, e.g.

```
2 polygons
polygon 1 ( 3 caps, 1 weight, 0 pixel, 0 str):
  -0.96592582629 0.25881904510 0 1
  0 0 1 0.41221474770752686
  0.209389006 0.781450409 0.587785253 0.00015230
polygon 2 ( 2 caps, 1 weight, 0 pixel, 0 str):
  0.087155742747 -0.9961946980 0 1
  0.785012134782  0.0686796625  0.615661475 1
```

- A collection of multiple polygons is typically referred to as a *mask* (or, occasionally, a *footprint*)
- The `weight` allows polygons to be weighted differently when creating catalogs of random coordinates (a `weight=1` polygon would contain twice as many random points as a `weight=0.5` polygon)

Python tasks

Please save all the plots you create as .png files in SVN!

1. Use your *circle_cap* function from the previous lecture to create two caps, one of radius $\theta=5^o$ centered at $(\alpha,\delta) = (76^o, 36^o)$, which I will call *cap 1* and one of radius $\theta=5^o$ centered at $(\alpha,\delta) = (75^o, 35^o)$, which I will call *cap 2*

2. Create a file containing these two caps as a single polygon in the *Mangle* format. The file should be called *intersection.ply*. Create a second *Mangle* file containing these two caps as two different polygons (i.e. polygon 1 should contain *cap 1* and polygon 2 should contain *cap 2*). This second file should be called *bothcaps.ply*

   - Note that the *Mangle* convention is to use an extension of *.ply* for files containing polygons

Python tasks

3. *import pymangle* and read in each of your masks, e.g.
   - *minter = pymangle.Mangle("intersection.ply")*

   then use *genrand* (see the syllabus link) to populate each mask with 10,000 random points. Plot the random points corresponding to each mask on a single plot with each mask's points plotted in a different color
   - do you understand exactly how and why the masks are different? If not, then ask Adam
4. Consider the file *intersection.ply* and its mask (*minter*)
   - Flip the sign of the constraint on *cap 1* and read it in as a mask (*mflip1*)
   - Plot *minter* and *mflip1* on one plot in different colors

Python tasks

5. Now make the constraint on *cap 1* positive again, and instead flip the constraint on *cap 2*
   - Read in this new mask as *mflip2*
   - Plot *minter, mflip1* and *mflip2* in different colors on a single plot
   - Do you understand what happened when the caps' constraints became negative? If not, then ask Adam
   - How would the plot you've just made compare to the mask in the file *bothcaps.ply*?
6. Make *both* cap constraints in *intersection.ply* negative. Read in this mask and generate 1 *million* random points in it. This will take a suspiciously short amount of time.
   - Plot the random catalog. Does this mask make sense?