```
                    width=1.6E7, height=1.2E7)
draw_map(m)
```

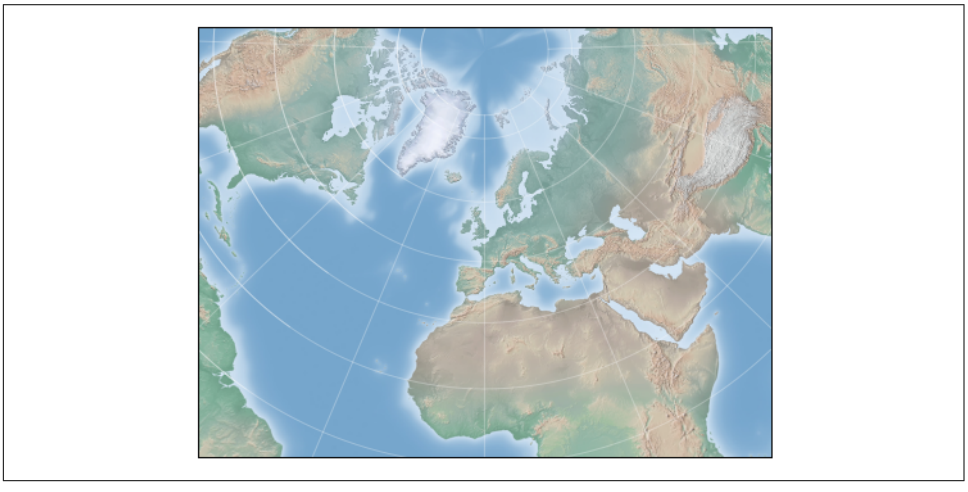

*Figure 4-107. The Albers equal-area projection*

### Other projections

If you're going to do much with map-based visualizations, I encourage you to read up on other available projections, along with their properties, advantages, and disadvantages. Most likely, they are available in the Basemap package. If you dig deep enough into this topic, you'll find an incredible subculture of geo-viz geeks who will be ready to argue fervently in support of their favorite projection for any given application!

## Drawing a Map Background

Earlier we saw the `bluemarble()` and `shadedrelief()` methods for projecting global images on the map, as well as the `drawparallels()` and `drawmeridians()` methods for drawing lines of constant latitude and longitude. The Basemap package contains a range of useful functions for drawing borders of physical features like continents, oceans, lakes, and rivers, as well as political boundaries such as countries and US states and counties. The following are some of the available drawing functions that you may wish to explore using IPython's help features:

- Physical boundaries and bodies of water

  `drawcoastlines()`
  : Draw continental coast lines

  `drawlsmask()`
  : Draw a mask between the land and sea, for use with projecting images on one or the other

`drawmapboundary()`
: Draw the map boundary, including the fill color for oceans

`drawrivers()`
: Draw rivers on the map

`fillcontinents()`
: Fill the continents with a given color; optionally fill lakes with another color

- Political boundaries

`drawcountries()`
: Draw country boundaries

`drawstates()`
: Draw US state boundaries

`drawcounties()`
: Draw US county boundaries

- Map features

`drawgreatcircle()`
: Draw a great circle between two points

`drawparallels()`
: Draw lines of constant latitude

`drawmeridians()`
: Draw lines of constant longitude

`drawmapscale()`
: Draw a linear scale on the map

- Whole-globe images

`bluemarble()`
: Project NASA's blue marble image onto the map

`shadedrelief()`
: Project a shaded relief image onto the map

`etopo()`
: Draw an etopo relief image onto the map

`warpimage()`
: Project a user-provided image onto the map

For the boundary-based features, you must set the desired resolution when creating a Basemap image. The `resolution` argument of the `Basemap` class sets the level of detail in boundaries, either `'c'` (crude), `'l'` (low), `'i'` (intermediate), `'h'` (high), `'f'` (full), or `None` if no boundaries will be used. This choice is important: setting high-resolution boundaries on a global map, for example, can be *very* slow.

Here's an example of drawing land/sea boundaries, and the effect of the resolution parameter. We'll create both a low- and high-resolution map of Scotland's beautiful Isle of Skye. It's located at 57.3°N, 6.2°W, and a map of 90,000×120,000 kilometers shows it well (Figure 4-108):

```
In[9]: fig, ax = plt.subplots(1, 2, figsize=(12, 8))

       for i, res in enumerate(['l', 'h']):
           m = Basemap(projection='gnom', lat_0=57.3, lon_0=-6.2,
                       width=90000, height=120000, resolution=res, ax=ax[i])
           m.fillcontinents(color="#FFDDCC", lake_color='#DDEEFF')
           m.drawmapboundary(fill_color="#DDEEFF")
           m.drawcoastlines()
           ax[i].set_title("resolution='{0}'".format(res));
```

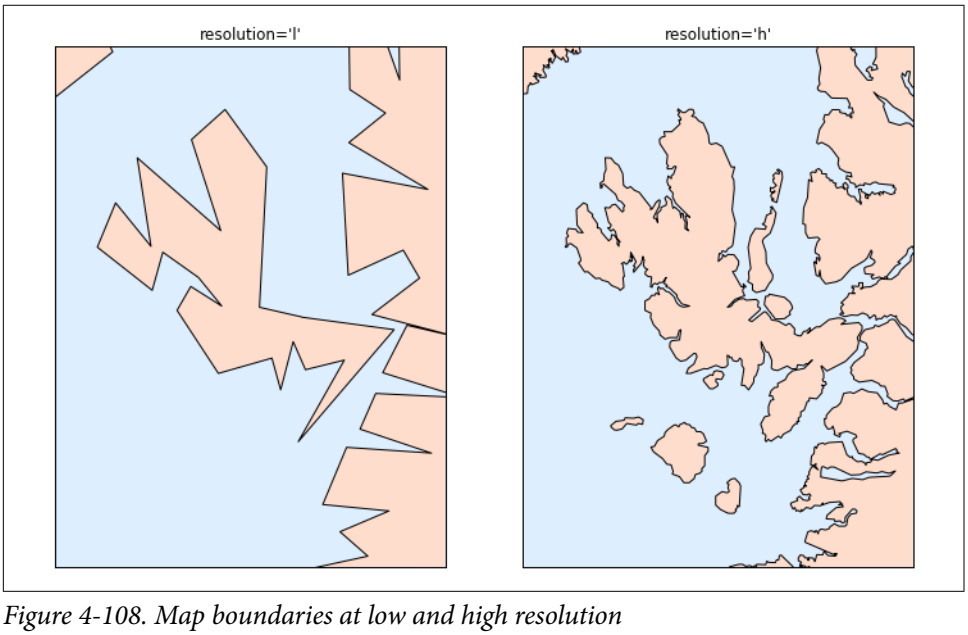


*Figure 4-108. Map boundaries at low and high resolution*

Notice that the low-resolution coastlines are not suitable for this level of zoom, while high-resolution works just fine. The low level would work just fine for a global view, however, and would be *much* faster than loading the high-resolution border data for the entire globe! It might require some experimentation to find the correct resolution

parameter for a given view; the best route is to start with a fast, low-resolution plot and increase the resolution as needed.

## Plotting Data on Maps

Perhaps the most useful piece of the Basemap toolkit is the ability to over-plot a variety of data onto a map background. For simple plotting and text, any `plt` function works on the map; you can use the `Basemap` instance to project latitude and longitude coordinates to `(x, y)` coordinates for plotting with `plt`, as we saw earlier in the Seattle example.

In addition to this, there are many map-specific functions available as methods of the `Basemap` instance. These work very similarly to their standard Matplotlib counterparts, but have an additional Boolean argument `latlon`, which if set to `True` allows you to pass raw latitudes and longitudes to the method, rather than projected `(x, y)` coordinates.

Some of these map-specific methods are:

`contour()`/`contourf()`
: Draw contour lines or filled contours

`imshow()`
: Draw an image

`pcolor()`/`pcolormesh()`
: Draw a pseudocolor plot for irregular/regular meshes

`plot()`
: Draw lines and/or markers

`scatter()`
: Draw points with markers

`quiver()`
: Draw vectors

`barbs()`
: Draw wind barbs

`drawgreatcircle()`
: Draw a great circle

We'll see examples of a few of these as we continue. For more information on these functions, including several example plots, see the online Basemap documentation.