1. Processing CSK from SLC

COSMO-SkyMed data are delivered either as processed imagery or as raw data. TerraSAR-X and RadarSAT-2 (and the future ALOS-2) data are always delivered as processed imagery; there is no possibility to process from Level 0. We saw in Lab 6 how to process CSK data from Level 0. In this lab, we will see how to set up ISCE for the three data sets that are or can be delivered as SLC data.

We saw in Lab 6 that the naming convention for the CSK data sets encodes all different data levels. The HDF5 files contain all the information needed to describe the radar data, the processing parameters of the imagery, the orbit and the geolocation of the products. Thus setting up the input files for CSK SLC data is essentially the same as it is for CSK raw data. In the lab7 directory, you will find two `.h5` data sets and an `insar.xml` input file.

```
> cd
> cd data
> cd lab7
> cd csk
> ls
CSKS2_SCS_U_HI_05_VV_RD_SF_20110525020512_20110525020519.h5
insarApp.xml
CSKS3_SCS_U_HI_05_VV_RD_SF_20110526020511_20110526020518.h5
```

The product type is seen to be SCS_U, which indicates SLC data (sometimes the name is SCS_B for CSK SLCs). We have created an input file with the right elements in it for these data to process:

```
> cat insarApp.xml
<insarApp>
<component name="insar">
    <property name="Sensor name">
        <value>COSMO_SKYMED_SLC</value>
    </property>
    <component name="master">
        <property name="OUTPUT">20110526.slc</property>
        <property
name="HDF5">CSKS3_SCS_U_HI_05_VV_RD_SF_20110526020511_20110526020518.
h5</property>
    </component>
    <component name="slave">
        <property name="OUTPUT">20110525.slc</property>
        <property
```

```
name="HDF5">CSKS2_SCS_U_HI_05_VV_RD_SF_20110525020512_20110525020519.
h5</property>
    </component>
<!--
    <component name="Dem">

<catalog>/u/proj8/dev/isce_regression/dem/demLat_N35_N37_Lon_W121_W12
0.dem.wgs84.xml</catalog>
    </component>
-->
    <property name="posting">
        <value>20</value>
    </property>
    <property name="unwrap">
        <value>False</value>
    </property>
    <property name="unwrapper name">
        <value>snaphu_mcf</value>
    </property>
    <property name="doppler method">
        <value>useDOPCSKSLC</value>
    </property>
</component>
</insarApp>
```

The beginning of the file should be familiar now, specifying the master and slave images. Since we are starting with SLC images, it is less confusing to set the OUTPUT names to end in .slc (but you can actually use .raw). The DEM specification is commented out, so ISCE will look to the internet database of SRTM data to download a DEM. The posting and unwrap properties should also be familiar from previous labs. What's new is the specification of the "`doppler method`" property to have a value of `useDOPCSKSLC`. Because the imagery are already processed, we do not have the freedom to specify the Doppler centroid for processing. For each data type that is delivered processed, we must specify a method to interpret and utilize the Doppler history in the imagery. If this parameter is not specified, the processing run fails because the Doppler is unspecified. Though the .h5 files have knowledge of their Doppler history, it is possible that the user might want to post-process the image with a different Doppler history. Allowing a method to be specified on input enables this possibility (though it is not implemented in `insarApp.py`). There is another possible value for this called `useDEFAULT`. This has the same effect as `useDOPCSKSLC`.

Once the input is set up properly, one can run `insarApp.py` as usual.

```
> insarApp.py insarApp.xml --steps
```

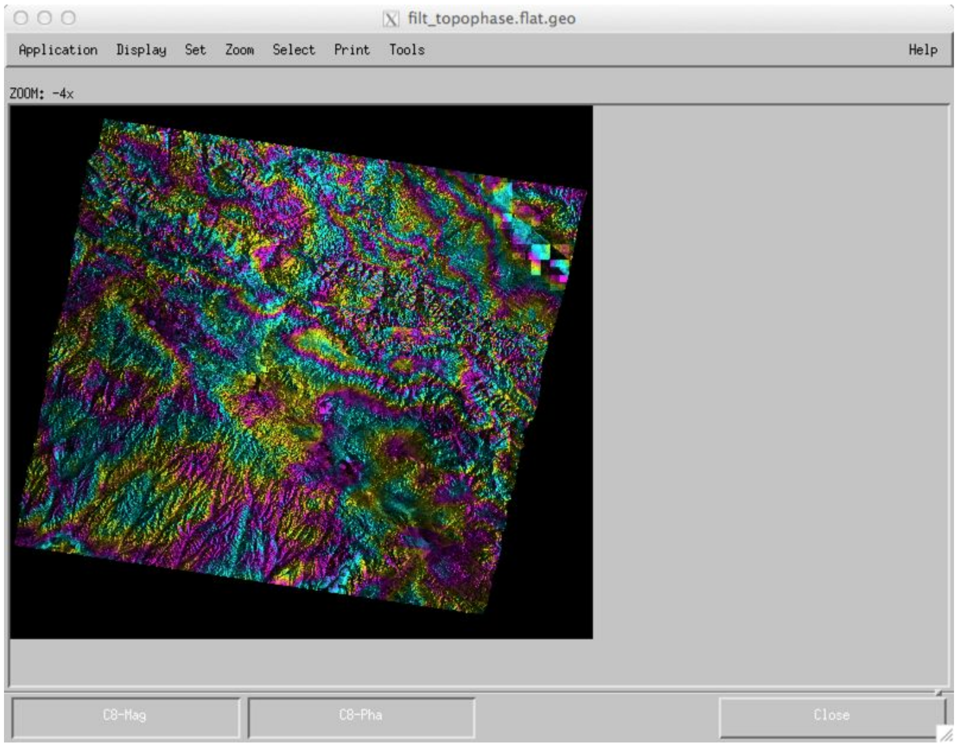
You'll note in the screen output that the steps are the same as for raw data even though formslc doesn't actually form an slc.

```
2014-07-29 20:30:18,466 - isce.insar - INFO - ISCE VERSION = 2.0.0,
RELEASE_SVN_REVISION = 1544,RELEASE_DATE = 20140724,
CURRENT_SVN_REVISION = 1525:1549M
ISCE VERSION = 2.0.0, RELEASE_SVN_REVISION = 1544,RELEASE_DATE =
20140724, CURRENT_SVN_REVISION = 1525:1549M
The Dem specified was not properly initialized. An SRTM Dem will be
downloaded.
Processing steps
self.step_list =  ['startup', 'preprocess', 'verifyDEM',
'pulsetiming', 'estimateHeights', 'mocompath', 'orbit2sch',
'updatepreprocinfo', 'formslc', 'offsetprf', 'outliers1',
'prepareresamps', 'resamp', 'resamp_image', 'mocompbaseline',
'settopoint1', 'topo', 'shadecpx2rg', 'rgoffset', 'rg_outliers2',
'resamp_only', 'settopoint2', 'correct', 'coherence', 'filter',
'unwrap', 'geocode', 'endup']
.
.
.
```

This is because `insarApp.py` is designed to preserve the overall interferometric flow, and `formslc` for an SLC input is essentially a resampling step since the image is already formed.

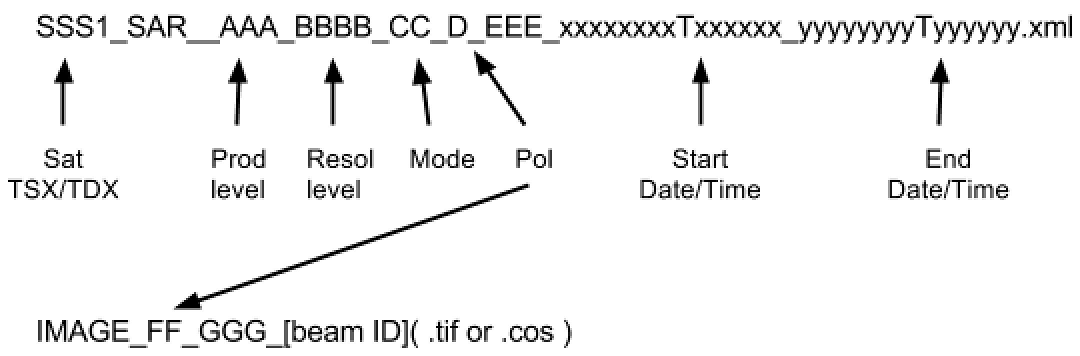
Configurability and setting of input file parameters and properties is possible for data sets starting from SLC as well. Certain parameters, for example the number of patches to process in formslc, obviously have no meaning when starting from SLCs, and the usual caveats about arbitrarily configuring parameters that should not be configured obtain. The final output of this CSK data over Parkfield looks pretty noisy from atmosphere and overly-smoothed decorrelation, but the interferogram has quite good fringe visibility because the time interval is only 1 day.

```
> mdx.py filt_topophase.flat.geo
```


filt_topophase.flat.geo
Application
Display
Set
Zoom
Select
Print
Tools
Help
ZOOM: -4x
C8-Mag
C8-Pha
Close

2. Processing TSX from SLC

The TerraSAR-X team has a similarly lengthy and descriptive file name as CSK.


SSS1_SAR__AAA_BBBB_CC_D_EEE_xxxxxxxxTxxxxxx_yyyyyyyyTyyyyyy.xml
Sat
TSX/TDX
Prod
level
Resol
level
Mode
Pol
Start
Date/Time
End
Date/Time
IMAGE_FF_GGG_[beam ID](.tif or .cos)

This XML file captures a directory-based hierarchy of files that support the processing, including imagery, annotation, thumbnails, and other ancillary files. It is buried a few levels down in a directory tree for the data. We have set up the `master.xml` and `slave.xml` files to show you how to construct the path to the xml files.

```
> cd
> cd data/lab7/tsx
> cat master.xml
<component>
    <property name="XML">
dims_op_oc_dfd2_204281662_1/TSX-1.SAR.L1B/TSX1_SAR__SSC______SM_S_SRA
_20091205T042212_20091205T042220/TSX1_SAR__SSC______SM_S_SRA_20091205
T042212_20091205T042220.xml
    </property>
    <property name="OUTPUT">20091205.raw</property>
</component>
```

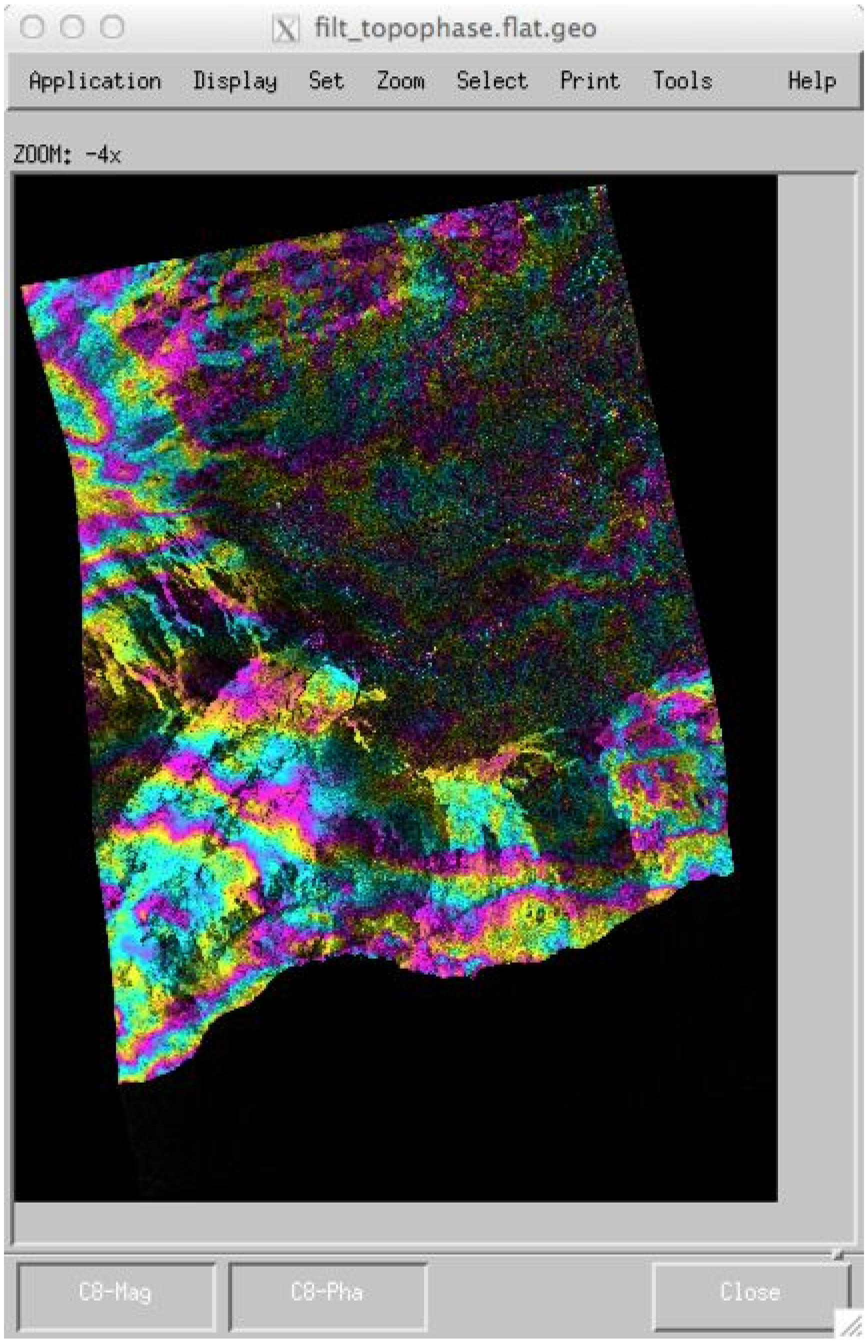
The top level directory is `dims_op_oc_dfd2_204281662_1/TSX-1.SAR.L1B.`
The next level directory is `TSX-1.SAR.L1B.`
The next level directory is
`TSX1_SAR__SSC______SM_S_SRA_20091205T042212_20091205T042220.`
And finally, the xml file can be found in this directory:
`TSX1_SAR__SSC______SM_S_SRA_20091205T042212_20091205T042220.xml.`

The `insarApp.py` input file is quite simple. It points to the master and slave catalogs, and specifies the posting and sensor name. As with CSK, the doppler method must also be specified, this time as `useDOPTSX`. It could also be `useDEFAULT`.

```
> cat insar_091205_091216.xml
<insarApp>
    <component name="insar">
        <property name="posting">
            <value>20</value>
        </property>
        <property name="Sensor Name">TerraSARX</property>
        <property name="doppler method">useDOPTSX</property>
        <component name="master">
            <catalog>20091205.xml</catalog>
        </component>
        <component name="slave">
            <catalog>20091216.xml</catalog>
        </component>
    </component>
</insarApp>
```

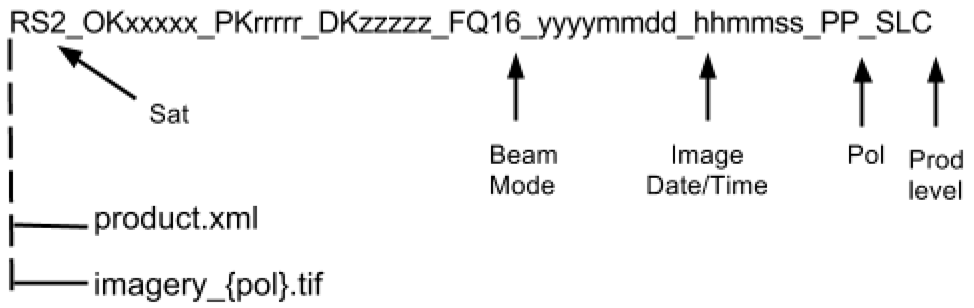
After running `insarApp.py` with this input file, you can display the output.

```
> mdx.py filt_topophase.flat.geo
```


filt_topophase.flat.geo
Application
Display
Set
Zoom
Select
Print
Tools
Help
ZOOM: -4x
C8-Mag
C8-Pha
Close

3. Processing RadarSAT-2 from SLC

By now you should know the drill pretty well: the SLC input files have long and complicated names but the data are nicely organized and self-contained, making the setup of the input files relatively straightforward. Let’s do the same for RadarSAT-2. The data are typically delivered in a zipped file with the long complicated name as described here:



but upon unzipping the file, two simply named files are created, `product.xml` and `imagery_XX.tif`, where XX stands for the polarization state of the radar transmitter and receiver for this product. The xml description has a great deal of information about the nature of the data and how it was processed. The tif file is the imagery itself, in geotiff format.

Note in this lab, we have organized the master and slave data sets into two directories named by date:

```
> cd
> cd data/lab7/rs2
> ls 2010*
20100402/:
imagery_HH.tif  product.xml

20100707/:
imagery_HH.tif  product.xml
```

which makes it easy to setup the insarApp.py input file:

```
> cat insarApp.xml
<insarApp>
    <component name="insar">
        <property name="unwrap">
```

```
            <value>True</value>
        </property>
        <property name="doppler method">useDEFAULT</property>
        <property name="Sensor Name">
            <value>RADARSAT2</value>
        </property>
        <property name="slc offset method">ampcor</property>
        <property name="posting">30</property>
        <component name="master">
            <property name="xml">
                <value>20100707/product.xml</value>
            </property>
            <property name="tiff">
                <value>20100707/imagery_HH.tif</value>
            </property>
            <property name="OUTPUT">
                <value>20100707.raw</value>
            </property>
        </component>
        <component name="slave">
            <property name="xml">
                <value>20100402/product.xml</value>
            </property>
            <property name="tiff">
                <value>20100402/imagery_HH.tif</value>
            </property>
            <property name="OUTPUT">
                <value>20100402.raw</value>
            </property>
        </component>
    </component>
</insarApp>
```

This file follows a familiar pattern and exploits some properties that perhaps we have not seen before, for example `slc_offset_method`, which specifies a particular component to use, and the `doppler method` is now explicitly set to `useDEFAULT` rather than something specific to RadarSAT-2.

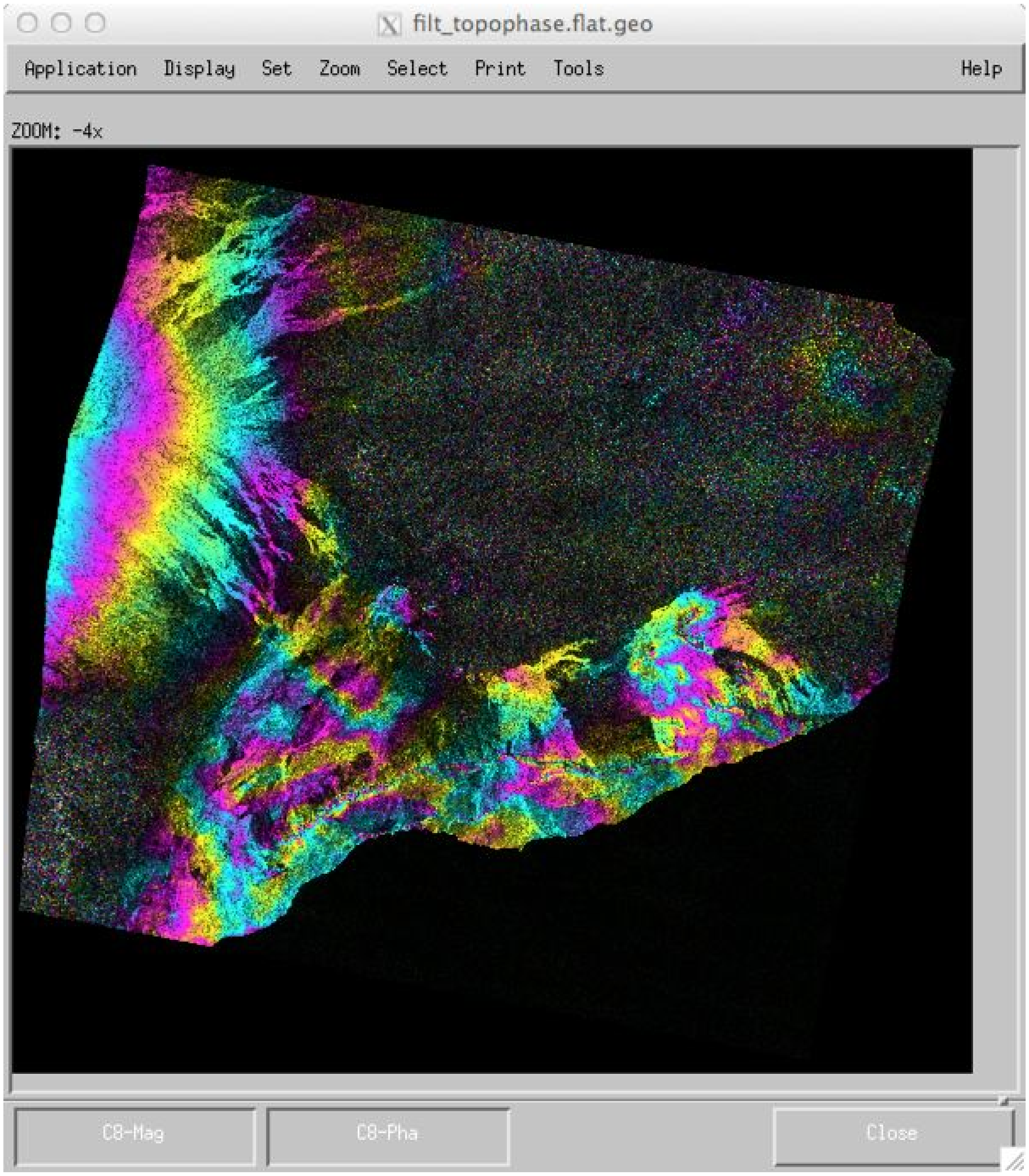
After running this example,

```
> insarApp.py insarApp.xml
```

we can display the output

```
> mdx.py filt_topophase.flat.geo
```

(see below). Note that unwrapping was turned on, but the output data (filt_topophase.unw.geo) is blank, so clearly it didn't work. This is because the default unwrapping scheme was used, and it doesn't work well when the middle of the scene, where the unwrapping origin is placed, is very noisy. Moving the origin to a less noisy region would likely fix this. Can you figure out how to do it?


filt_topophase.flat.geo
Application
Display
Set
Zoom
Select
Print
Tools
Help
ZOOM: -4x
C8-Mag
C8-Pha
Close

4. Running ISCE modules outside insarApp.py

New lab exercise from Piyush

```
> cd lab5/env/20100502_20100328/
> python3
Python 3.2.3 (default, Feb 27 2014, 21:31:18)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
enabling readline
>>> import isce
>>> import isceobj
>>> img = isceobj.createDemImage()
>>> img.load('filt_topophase.flat.geo.xml')
>>> img.length
2560
>>> img.width
2009
>>> img.renderEnviHDR()
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>>
> more filt_topophase.flat.geo.hdr
ENVI
description = {Data product generated using ISCE}
samples = 2009
lines   = 2560
bands   = 1
header offset = 0
file type = ENVI Standard
data type = 6
interleave = bip
byte order = 0
coordinate system string =
{GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID
["W
GS_1984",6378137,
298.257223563]],PRIMEM["Greenwich",0],UNIT["Degree",0.01745
329
25199433]]}
map_info = {Geographic Lat/Lon, 1.0, 1.0, -116.09083333333334,
33.48750000000
000
4, 0.0008333333333333334, 0.0008333333333333334, WGS-84,
units=Degrees}
>
```