Basic plotting and animation with VCS

What is VCS

- *Visualisation and Control System*
- VCS came before CDAT
- Python interface
- Only 1D or 2D graphics
- Plots or animations
- Outputs to screen, GIF, EPS, PS etc.

VCS Concepts and Terminology

Basic concepts:

- **VCS Canvas** – where the plots are drawn
- **Graphic Methods** – how the data is rendered (the plot type). E.g. “boxfill”, “isofill”, “isoline”, “vector”, etc,. Multiple **projections** may be available for a given Graphic Method.
- **Templates** – define the location where things are drawn on the canvas (data, legend, title, comments, units, etc..)
- **Primitives** – additional secondary items such as lines, polygons, text, markers, etc.,

The VCS Canvas

- VCS canvas needs to be initialized (created)

```
>>> x=vcs.init() # without any arguments
```

- Up to 8 Canvases at once.

- Canvas as “magic board”, can clean or destroy it:

```
>>> x.clear()
>>> x.close()
```

- Can have multiple plots on a single Canvas (not covered here)

VCS Help

- Basic help on VCS can be obtain inline via:
  ```
  >>> vcs.help()
  ```
- This will list available function in vcs, notably functions to create/get/query vcs objects…

VCS First Help

This also can be used to get help on a specific command:

```
>>> vcs.help('createboxfill')
```

Function: createboxfill

```
# Construct a new boxfill graphics method
```

Description of Function:

Create a new boxfill graphics method given the name and the existing boxfill graphics method to copy the attributes from. If no existing boxfill graphics method name is given, then the default boxfill graphics method will be used as the graphics method to which the attributes will be copied from.

If the name provided already exists, then a error will be returned. Graphics method names must be unique.

Graphic Methods Concepts (1)

- Essentially a graphic method represents **HOW** data are going to be plotted (the WHERE will be determined via templates, see later)

- There are 13 type of graphic methods in VCS:
  - 2D Graphic Methods
    - Boxfill, Isofill, Isoline, Meshfill, Vector, Outfill, Outline, Continents , (Taylordiagrams)
  - 1D Graphic Methods
    - Yxvsx, Xyvsy, XvsY, Scatter

Specifying a Graphic Method

To specify a graphic method use a “create” function, then use plot:

- For example, for a boxfill:

```
>>> gm = x.createboxfill('name')
>>> x.plot(data, gm)
```

- If a graphic method already exists, use “get” functions:

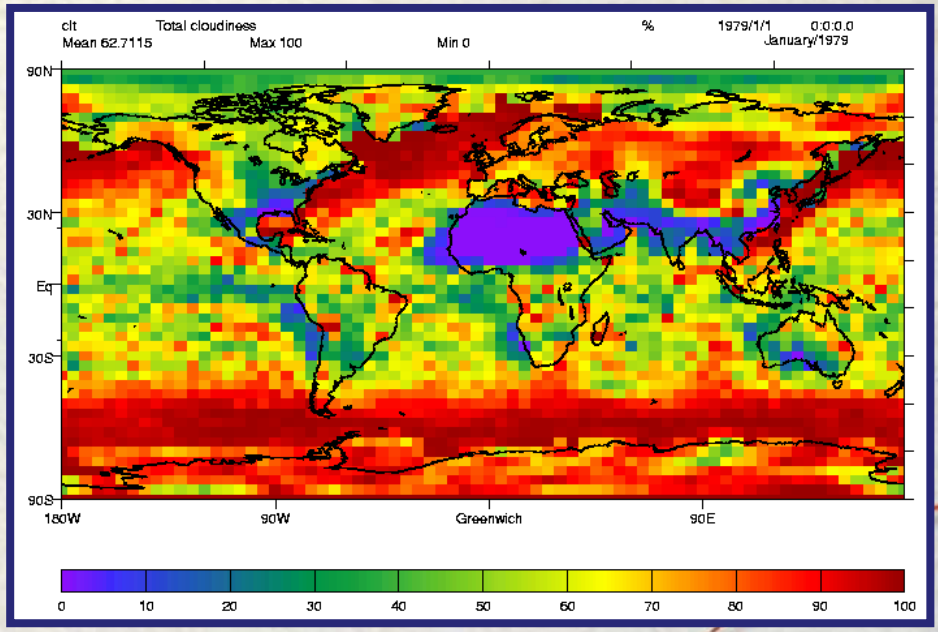
```
>>> gm = x.getboxfill('name')
>>> x.plot(data, gm)
```

- Replace ‘boxfill’ in the above for other methods.

2D - “boxfill”

- The boxfill graphic method takes a 2D array and represents it by filling a “box” (determined by the bounds on each axis values) with a color linked to the array’s value at this location:

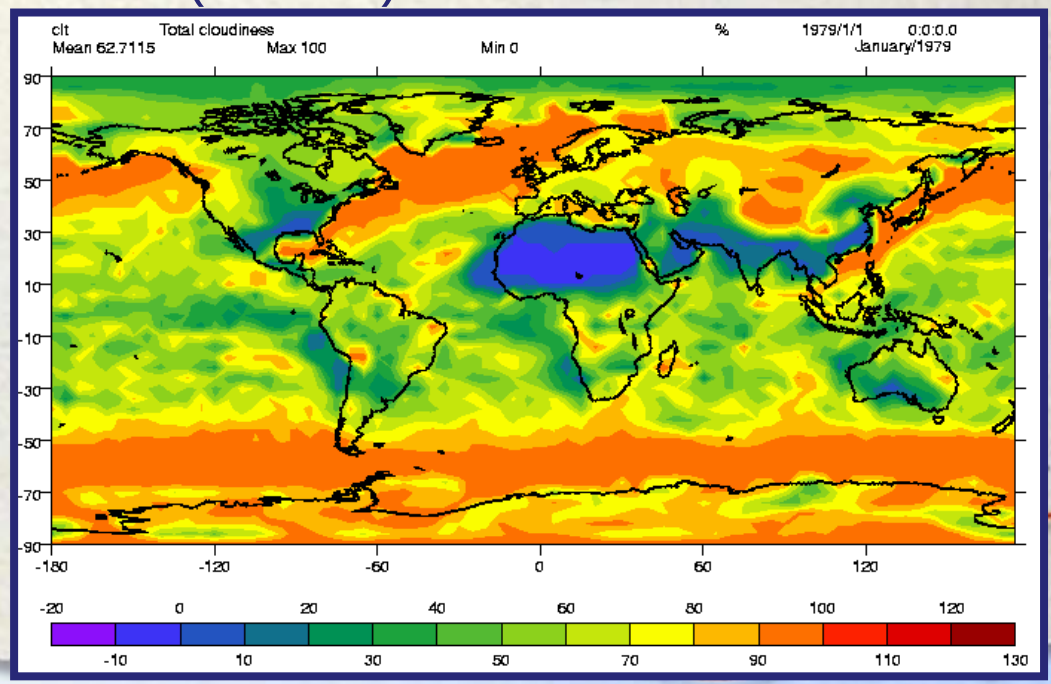
```
>>> box=x.createboxfill('new')
>>> x.plot(data,box)
```



2D -“isofill”

- Isofill graphic methods draws filled isocontour
- They are extremely similar to boxfill “custom” type
- Known Limitation:
  - No control on labels position
  - No control on isolines “Smoothness”

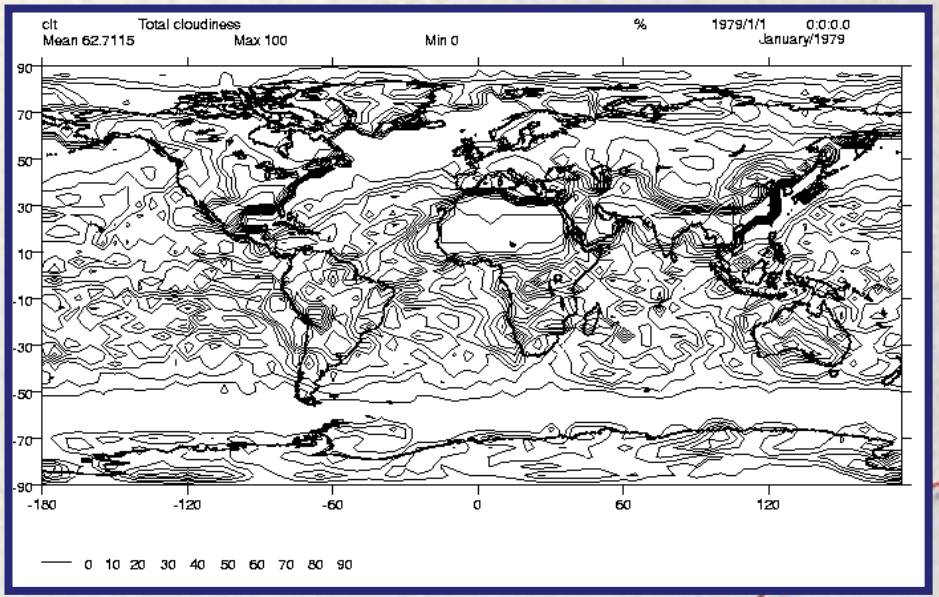
```
>>> iso=x.createisofill('new')
```



2D - “isoline”

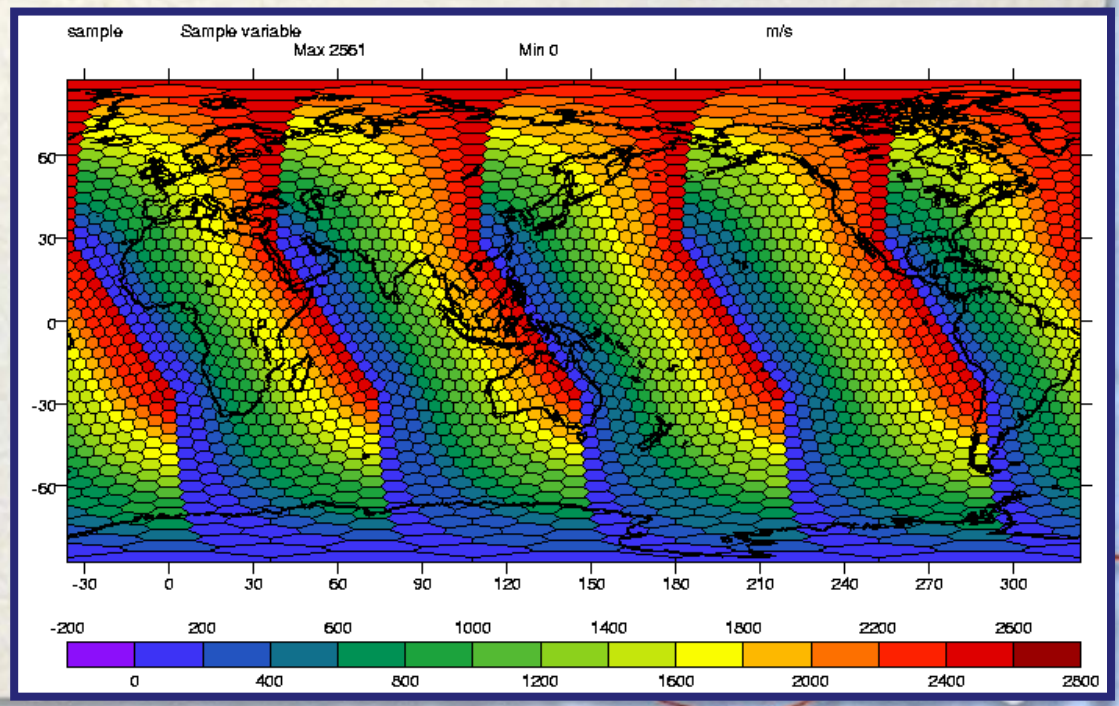
- Isoline, draws isocontours, color, style, can be controlled.
- Limitation:
  - No control on the labels location
  - No control of “smoothness”

```
>>> iso=x.createisoline('new')
```



2D – “meshfill” – for generalised grids

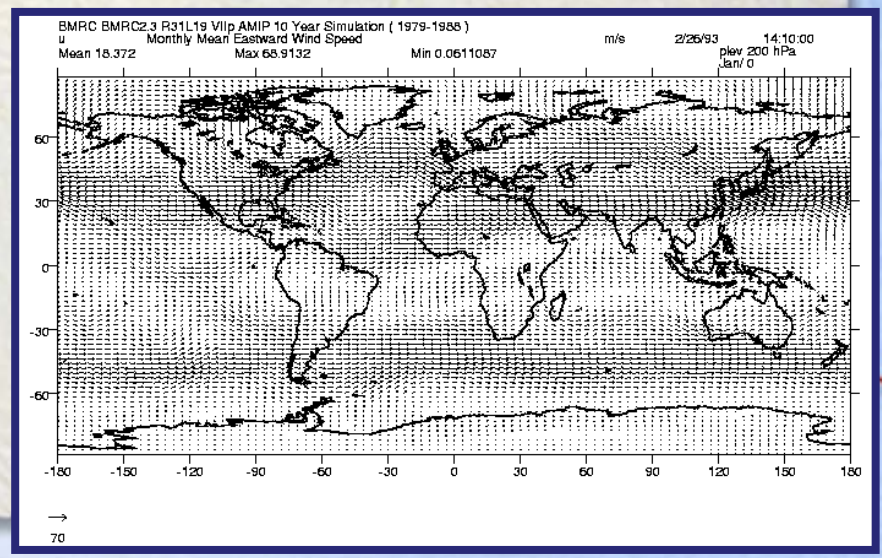
- Meshfill is similar to boxfill “custom” but allows representation of **generalized grids**, i.e. instead of filling a box, “meshfill” fills cells, of “n” points
- Requires an additional array to be passed to represents the “mesh”
- This array is generated automatically for Transient Variables recognized by cdms.
- Allows very creative 2D plots.



2D - “vector”

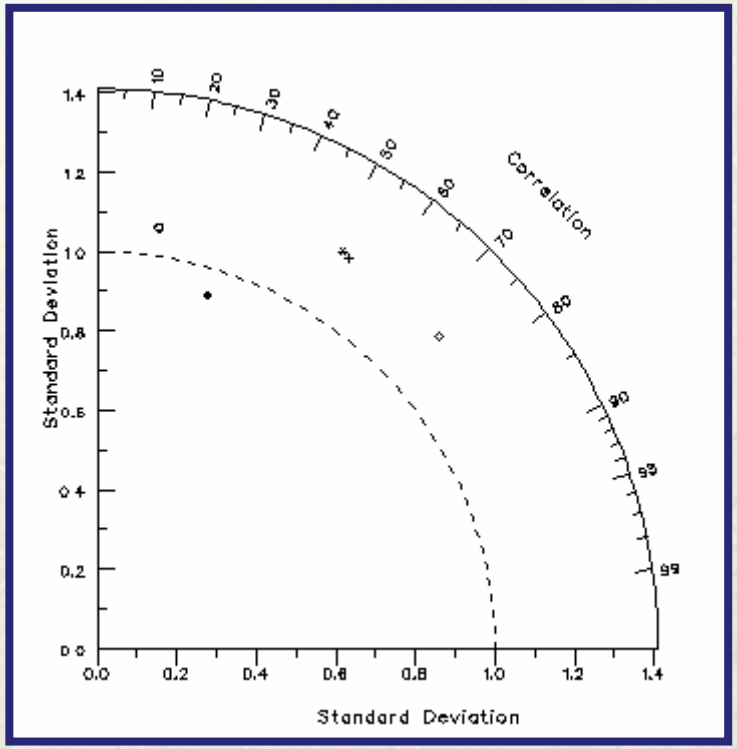
- The “Vector” graphic method represents the combination of 2 arrays, via “vector” the first array representing the “X” axis component and the second array representing the “Y” axis component.

```
>>> f=cdms.open('sample_data/clt.bc')
>>> u=f('u')
>>> v=f('v')
>>> vec=x.createvector('new')
>>> x.plot(u,v,vec)
```



Taylor Diagram – comparing climate runs

- With the right set up you can create Taylor diagrams:

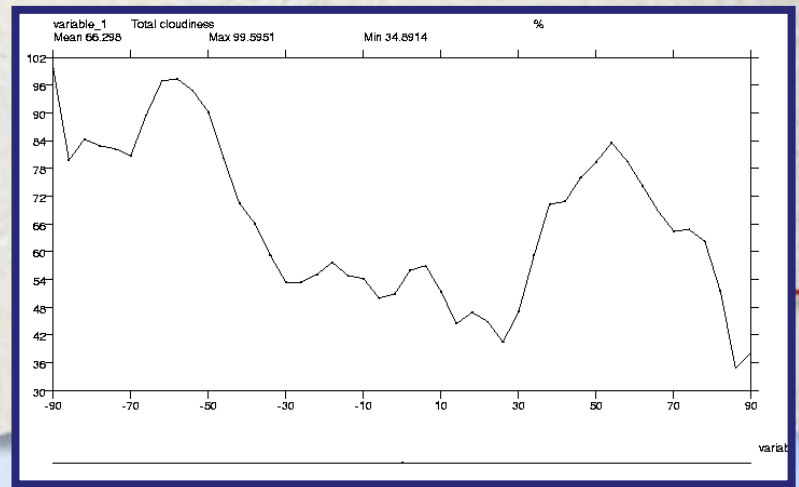


See the on-line tutorial: `taylordiagram_tutorial.py`

1D – Y(x) vs x

- All 1D plots in VCS basically work the same way. There are 4 types of 1D graphic method, we'll start with the basic: Yxvsx, which stands for Y(x) vs x
- This graphic method draws a 1D array (Y) as a function of its 1D axis (x)
- Example zonal mean of the first time point of our data array

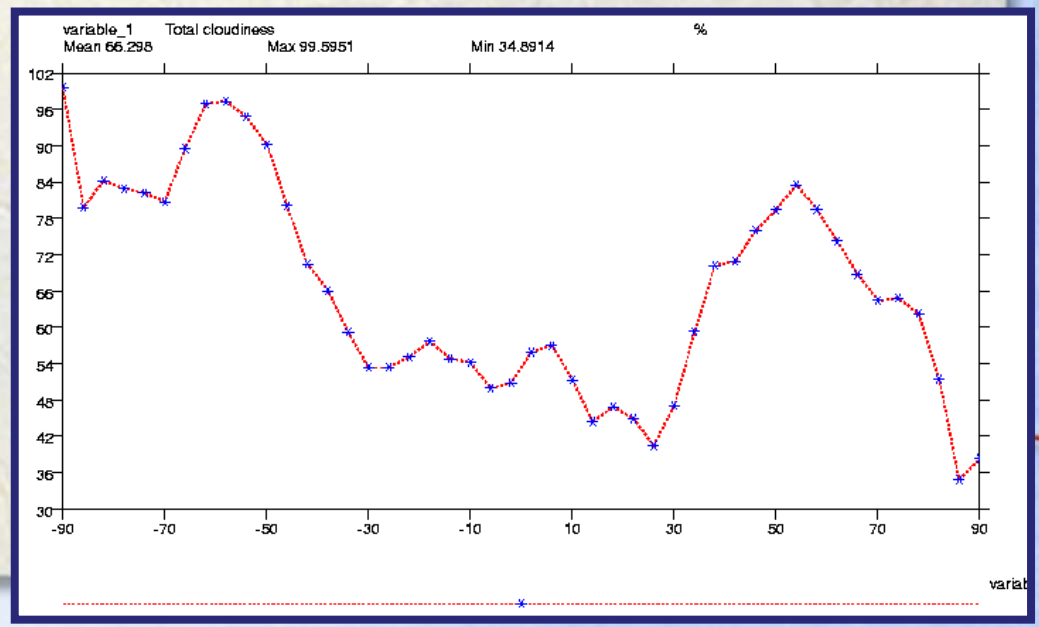
```
>>> zm=MV.average(data[0],1) # Zm.shape is (46,)
>>> x.plot(zm) # knows to plot 1D with yxvsx
>>> yx=x.createyxvsx('new')
>>> x.plot(zm,yx) # same
```



1D - “VCS” Yxvsx attributes

- As in isoline, or vector, line, linecolor, linewidth, determine the line.
- marker, markercolor, markersize, determine the markers to be drawn:

```
>>> yx.line='dot'
>>> yx.linecolor=242
>>> yx.linewidth=2
>>> yx.marker='star’
>>> yx.markercolor=244
```



1D - “VCS” Xyvsx, Xvsy, scatter

Other 1D graphic method, work very similarly:

- **Xyvsy** does the same thing except the X and Y axes are flipped relatively to a Yxvsx.
- **XvsY** is the same thing as YxvsX except it takes 2 data arrays (X and Y) as arguments, therefore the values on the horizontal axis are not taken from the “axis” definition of the Y array but from the values of a first (X) array.
- **Scatter** basically works as XvsY except both X and Y MUST have the same Axis!

Graphic Methods Attributes

```
>>> b=x.createboxfill('new_one')
>>> b.list()
```

----------Boxfill (Gfb) member (attribute) listings ----------

Canvas Mode = 1

graphics method = Gfb # indicates the graphic method type: Graphic Filled Boxes (Gfb)

name = new # Name of the specific graphic method

projection = linear # projection to use (see projection section)

xticlabels1 = * # 1st set of tic labels, '*' means 'automatic'

xticlabels2 = * # 2nd set of labels (pos determined by template)

xmtics1 = # 1st set of sub ti for details)

xmtics2 =

yticlabels1 = *

…

…

Graphic Methods plot functions

Graphic methods may also be selected by replacing the 'plot' method with the graphic method name:

```
>>> x.boxfill(data)
>>> x.isofill(data)
>>> x.isoline(data)
>>> x.vector(data)
>>> x.yxvsx(data)
etc...
```

- Note: an error is returned if the data structure(s) incompatible with method.

The importance of *squeezing* data

- A common problem is confusion about the shape of a data slab:
  - You think you have 1D data but you have 4D data
  - shape is (1, 1, 1, 4)
  - array is [[[[13,15,18,14]]]]
  - VCS uses first index of every leading dimension until it gets a 2D array (if 2D specified)

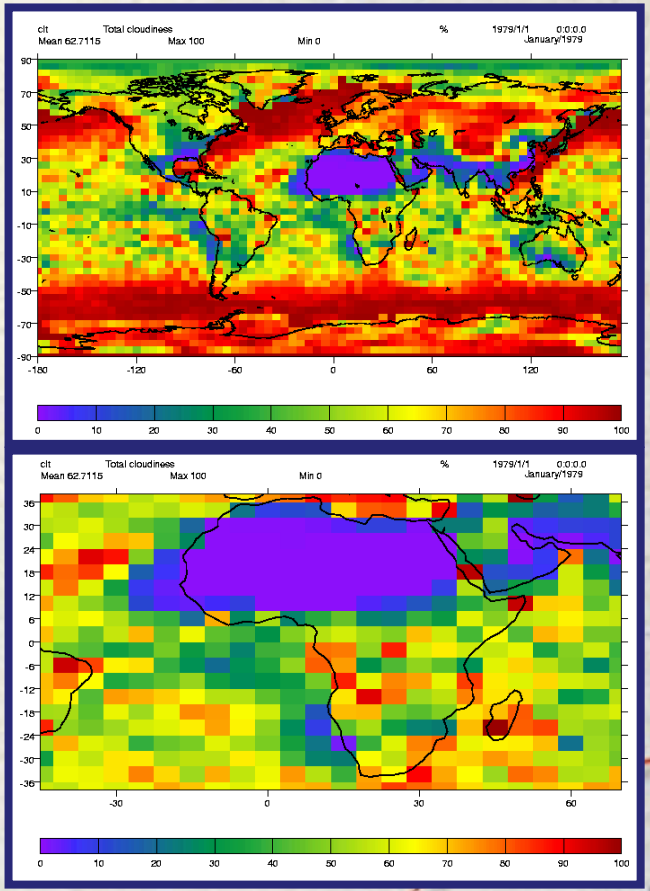
Squeeze is the key:

```
>>> data=data(squeeze=1) # OR MAYBE...
>>> data=bigdata(lat=(90,0), squeeze=1)
>>> x.plot(data)
```

2D - World Coordinates – example attributes

- **worldcoordinate** attributes are present on all graphic methods.
- can select a subset area.
- for example to visualise Africa:

```
>>> b.datawc_x1 = -45.
>>> b.datawc_x2 = 70.
>>> b.datawc_y1 = -38.
>>> b.datawc_y2 = 38.
>>> x.plot(s,b)
```

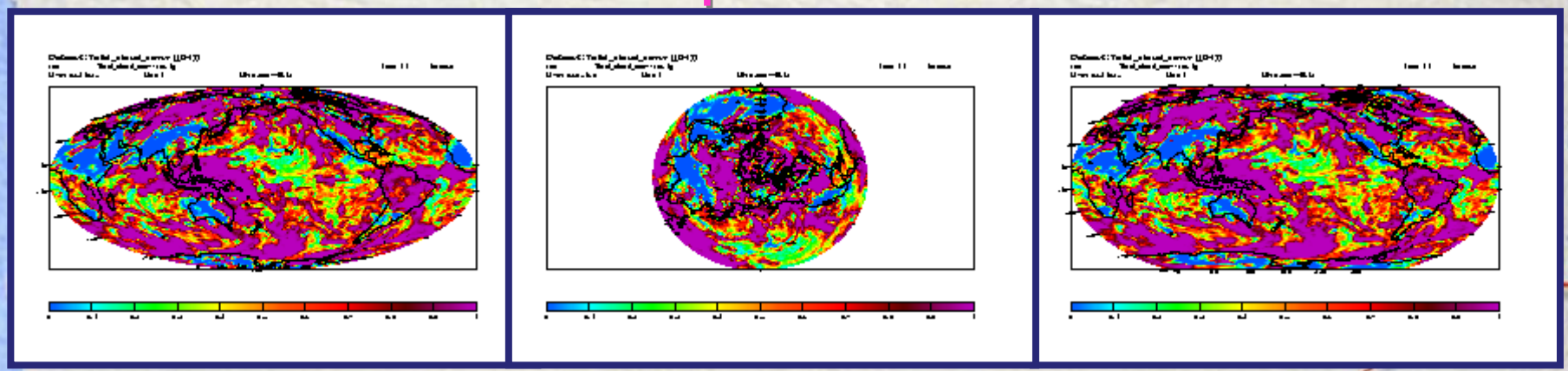


Projection attributes of Graphic Methods

Each graphic method may have a number of projections, the full list is available from:

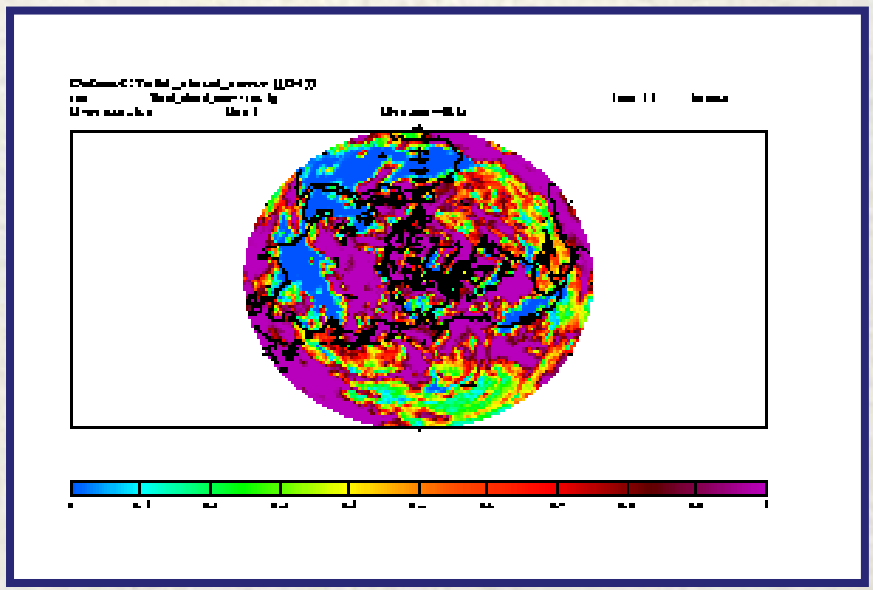
```
>>> v.show('projection')
************************Projection Names List**************************
(   1):               default                linear              mollweide
(   4):              robinson            polyconic                  polar
(   7):               lambert          orthographic               mercator

************************End Projection Names List***********************
```

mollweide polar robinson



Selecting a projection

```
>>> import vcs
>>> x=vcs.init()
>>> iso=x.createisofill('newone', 'ASD')
>>> iso.projection="polar"
>>> x.plot(var1, iso)
```

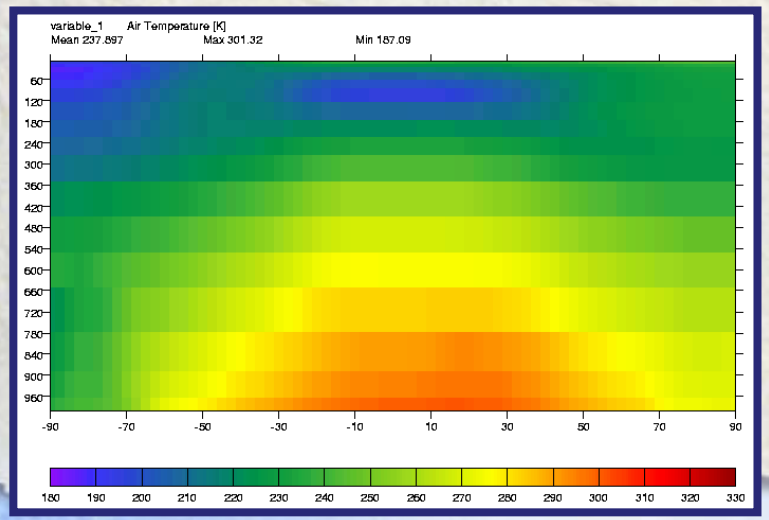


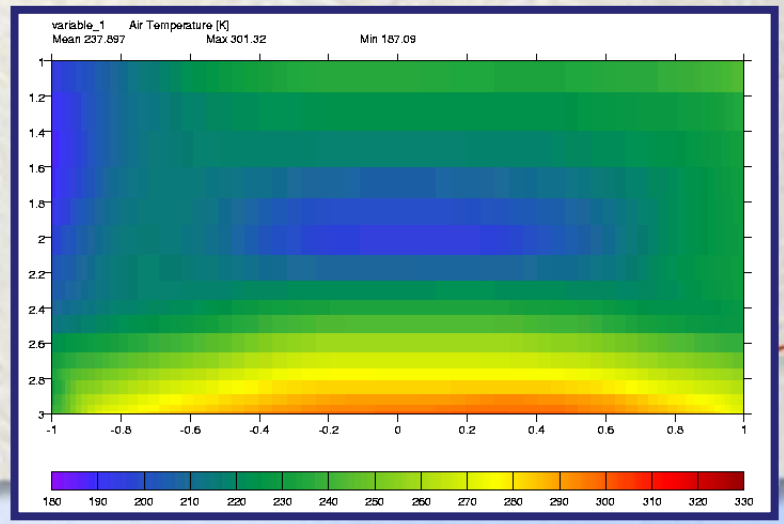
2D - Axes Transformations

Axis transformations allowed are:

- area-weighted, ln, log10, exp.

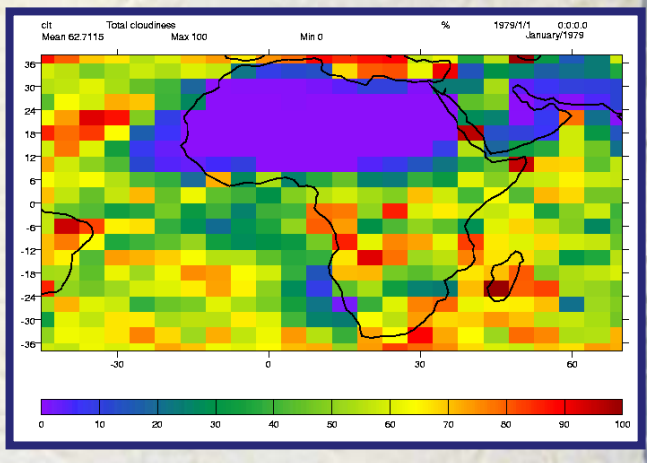
```
>>> box=x.createboxfill('new')
>>> x.plot(data2, box)
>>> box.xaxisconvert='area_wt' # Area weighted
        representation of latitudes
>>> box.yaxisconvert='log10' # Log of P
        repres of Pressure dim
>>> x.plot(data2, box)
```



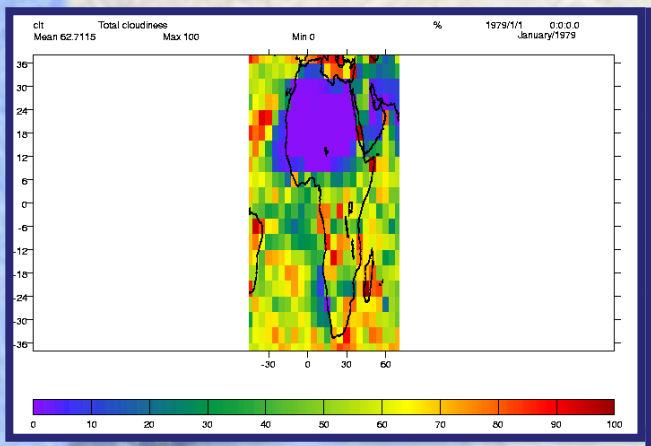


2D - Controlling the Y/X ratio of a plot

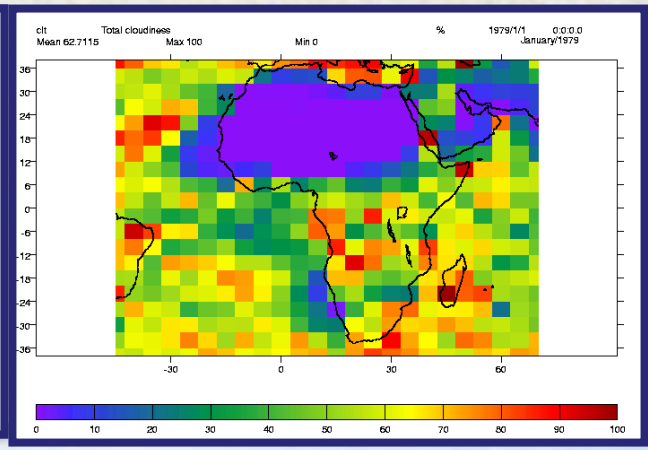
- ratio controls the Y ratio relative to X
- ratio=2 means Y will be twice X
- For data with spatial grids, the ‘**auto**’ value can be passed.
- To adjust the box and tick marks use ‘**autot**’



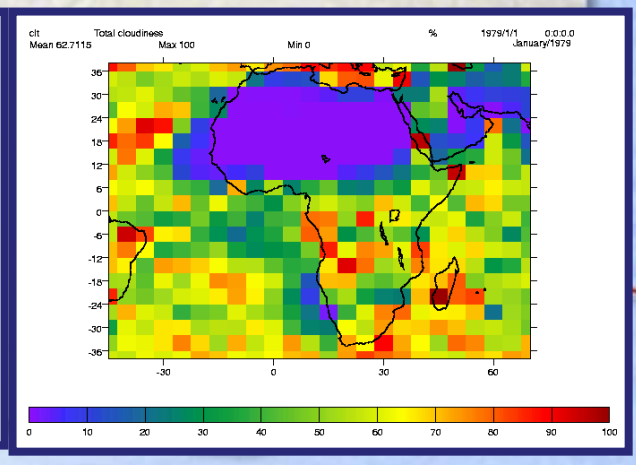
x.plot(data,b,ratio=2)



x.plot(data,b,ratio=‘auto’)



x.plot(data,b,ratio=‘autot’)

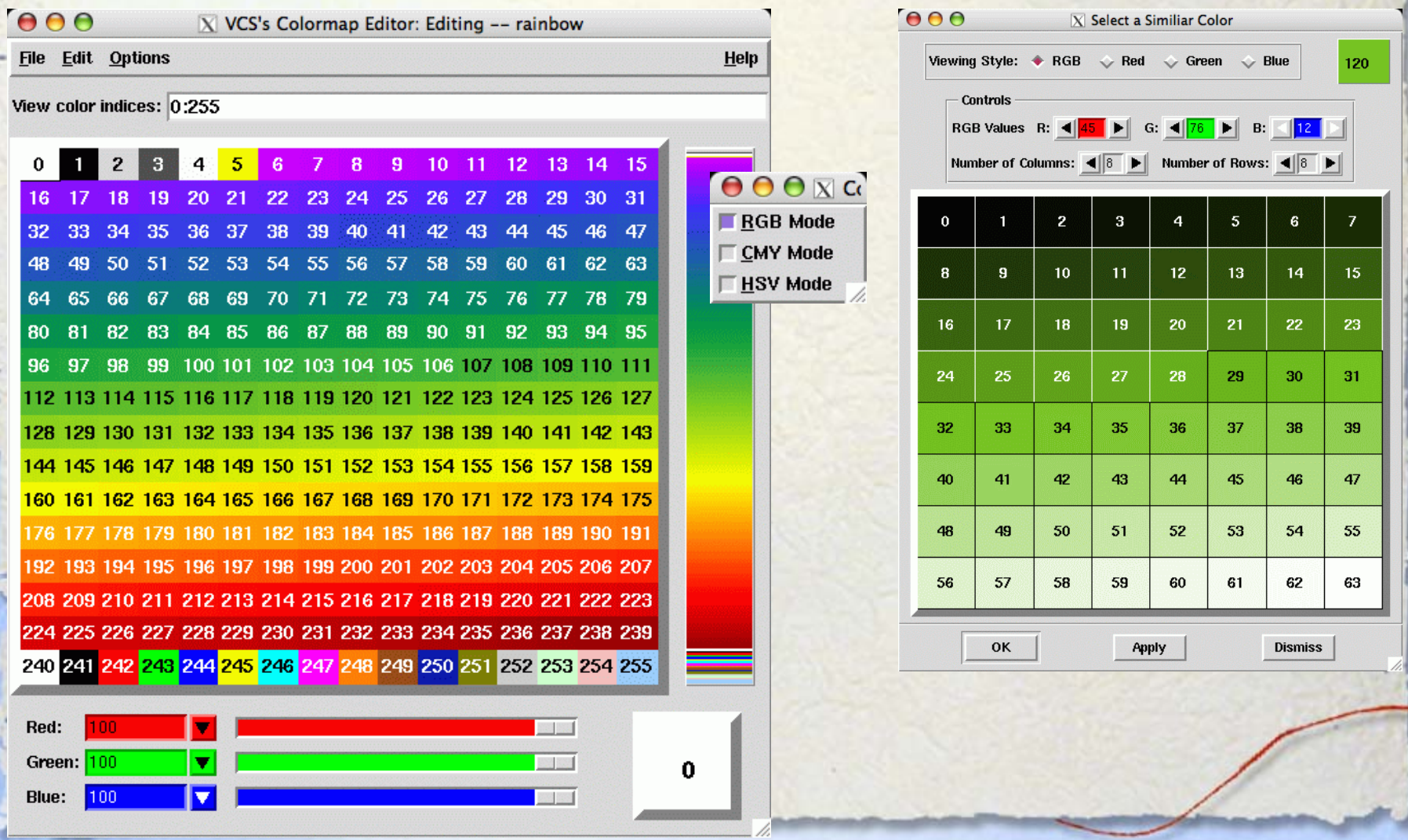


Colormaps

- The colormap is the range of colours used in any plot or animation.
- Colormap in VCS consist of 256 colors.
- Each colormap has a unique name associated with itself.
- VCS is currently limited to **ONE** colormap at a time, but if you clear your plot, you can use multiple colormaps during the same session.

Colormaps from VCDAT

- The Colormap GUI let you easily pick/change colors and color model



Introducing Templates (1)

- Template tell VCS **WHERE** to draw objects on the canvas (whereas Graphic Methods specify HOW to draw them).

- 4 aspects of the plot controlled templates:
  - Text location for things like *title*, *comments*, *name*, etc..
  - Data area
  - Tick marks and label locations
  - Legend

Introducing Templates (2)

- Each element of the template (e.g “data”) can be turned on/off or moved on top/below other elements on the page via its “**priority**” attribute
- Template object are created via the “createtemplate” command

```
>>> t=x.createtemplate('new')
```

- All elements of a template object can be listed via the list() function

```
>>> t.list()
```

- Alternatively, a single elements’s attributes can be listed, e.g.:

```
>>> t.data.list()
member = data
  priority = 1
  x1 = 0.78400000234
  y1 = 0.92734249999 ...
```

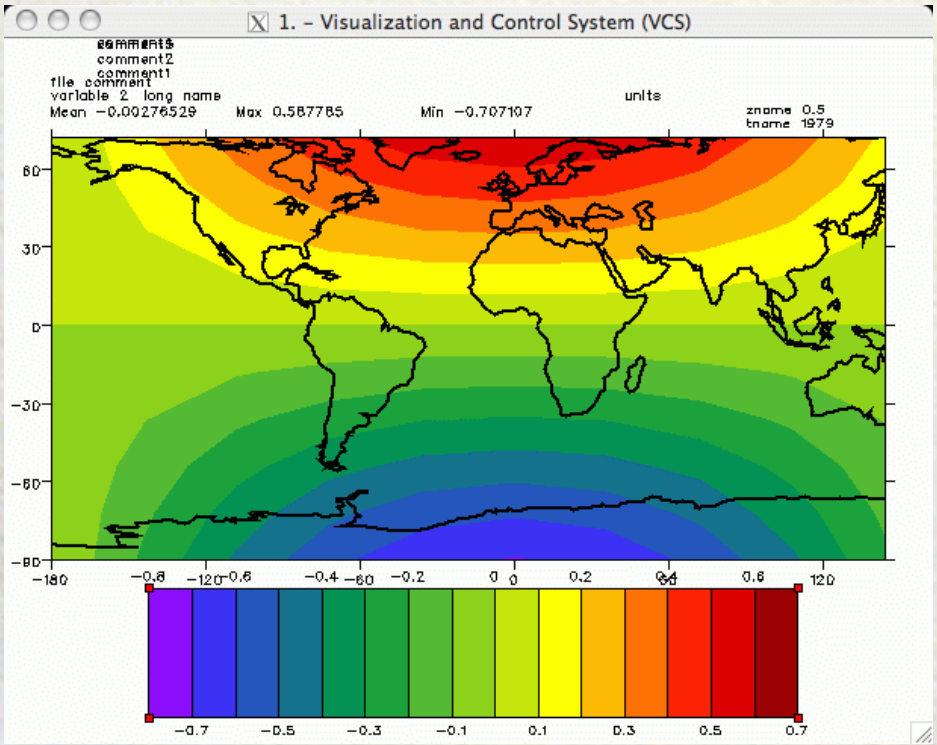
VCS Templates – useful methods

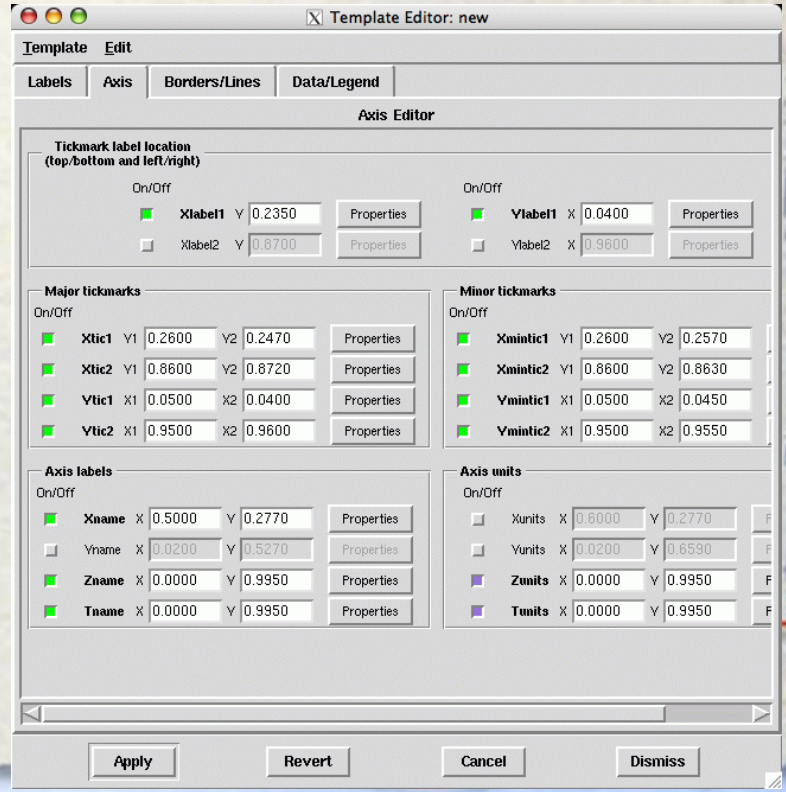
- Command line manipulation of entire templates:

```
>>> templ.move(percent, axis)
>>> templ.moveto(x,y) # move the
  data.x1,data.x2 corner to x/y
>>> templ.scale(scale,axis='xy',font=-1)
  # scale the template along x,y,xy can
  also scale fonts (automatic for xy
  direction only)
```

VCS Templates
Simplifying your life!

- Setting up templates can be tedious!
- Recommend that you construct your template once, save it and reuse it for ever.
- Use the VCDAT template editor to create them!

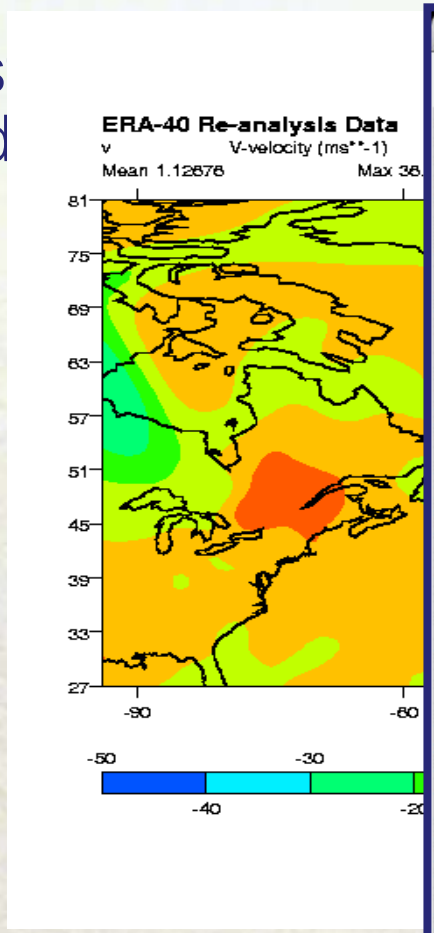


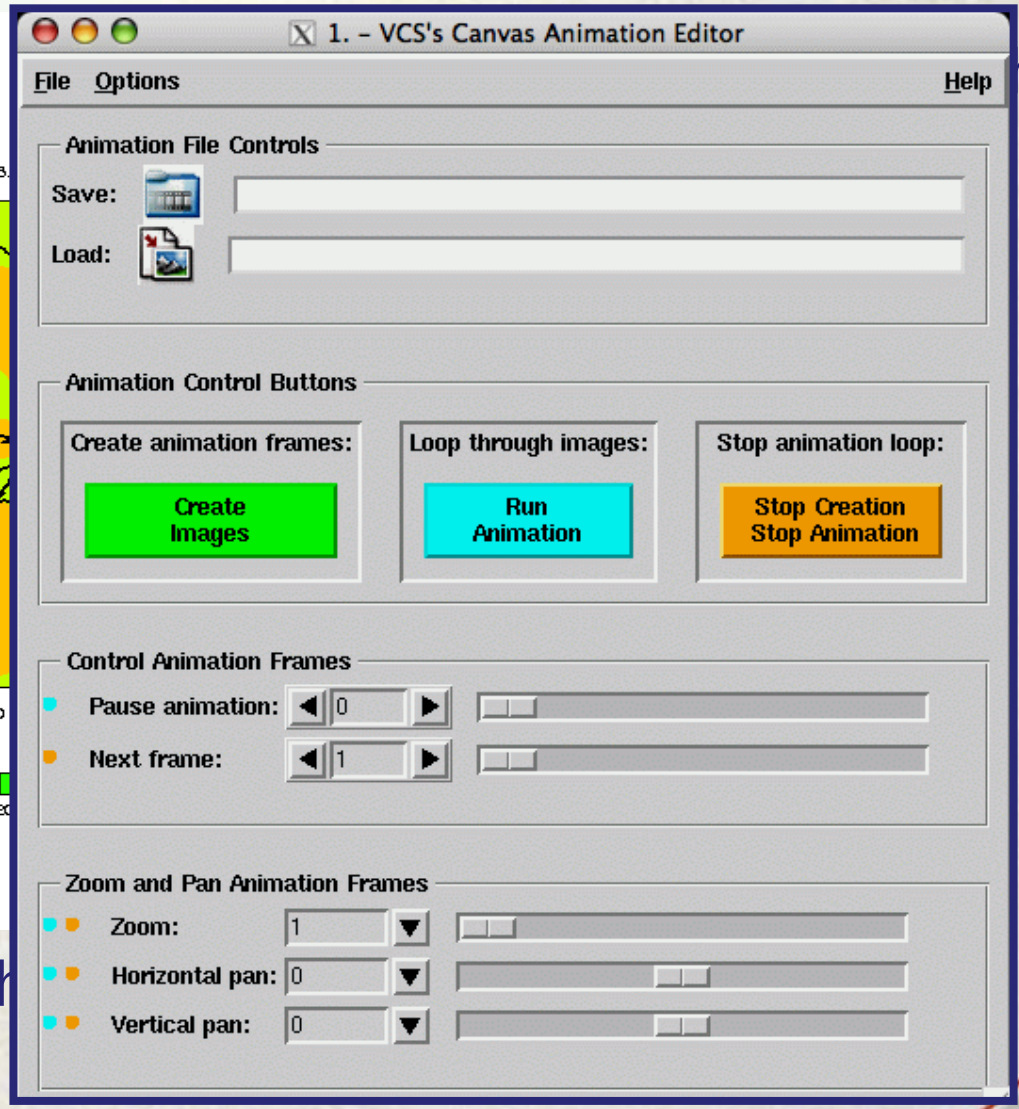


Animating at the command line

Jus … hen
d

```
>>>
>>>
>>>

>>>
. . .
. . .
. . .
```





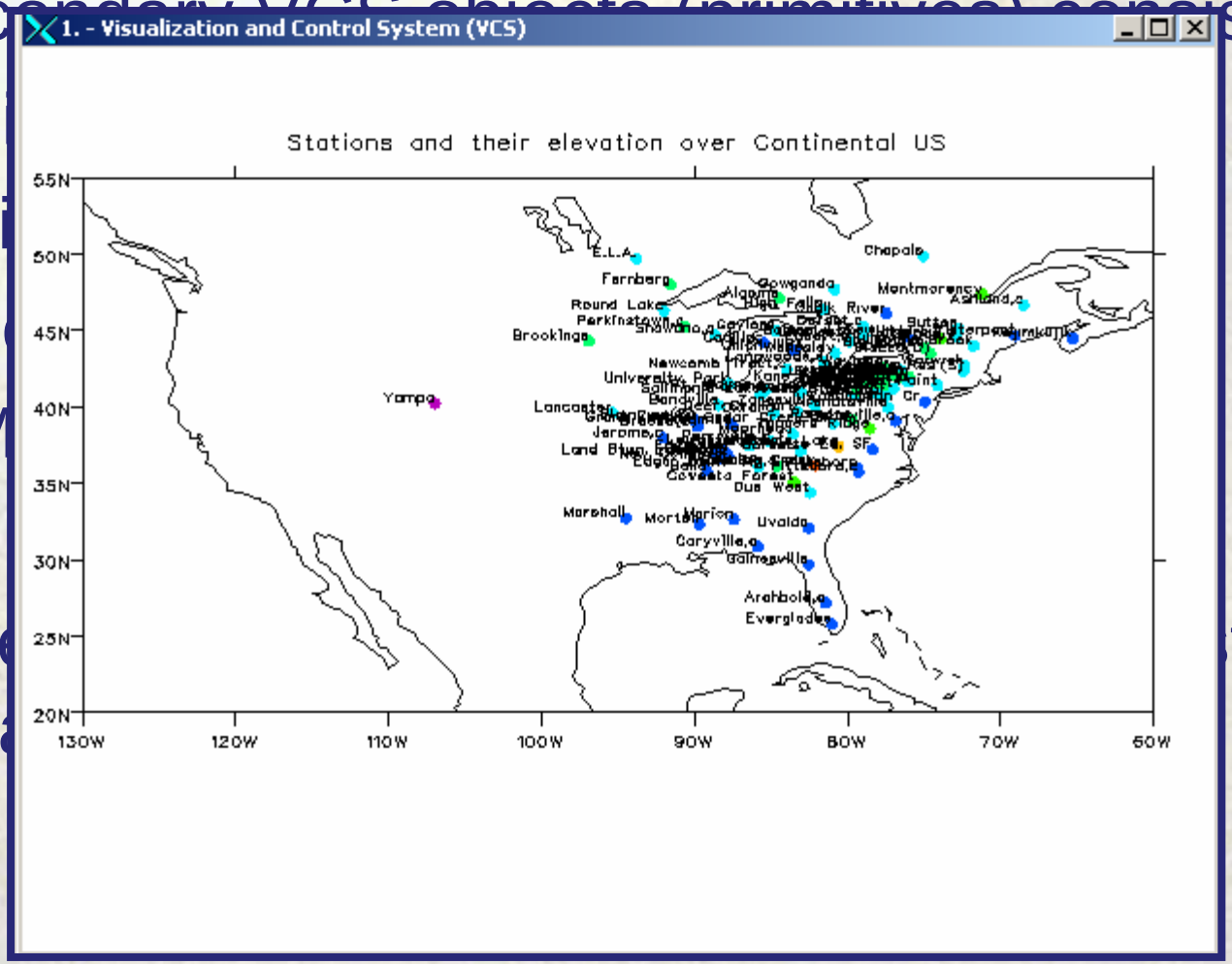
Or call up GUI with … \

“Secondary” VCS Objects

Secondary VCS objects (primitives) consist of:

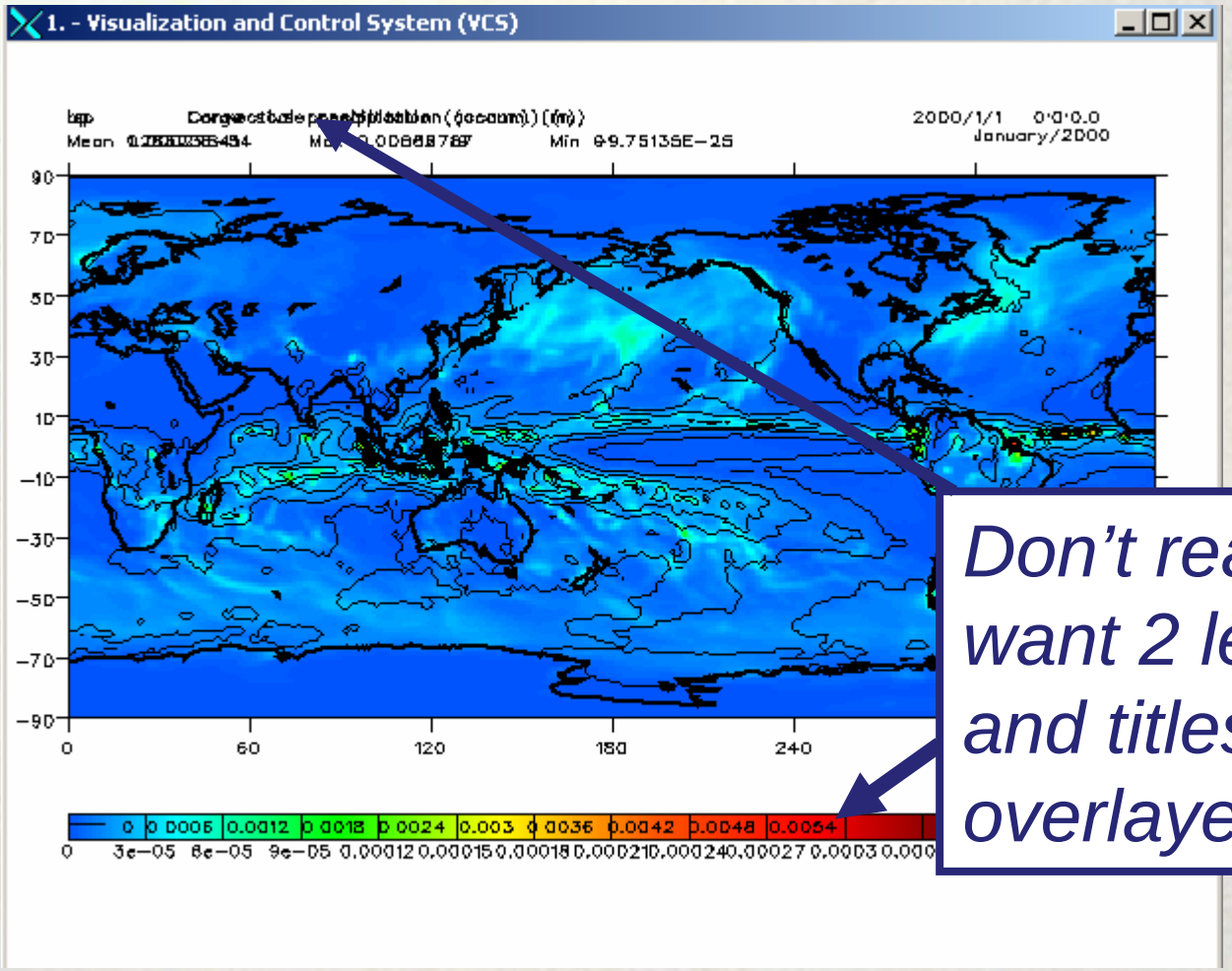
- F
- Li
- T
- M

An e tation
da



Overlaying images

- Overlaying is simple, just don't clear the canvas between plots, templates may need modifying…



Other graphics packages

CDAT/Python objects can also be visualised in:

- xmgrace (1D plots) – ships with CDAT
- PyNCL (NCAR Command Language)
- PyNGL (NCAR Graphics Language)
- MatPlotLib (MatLab equivalent for Python)
- IAGraph (which is a layer on top of VCS)