

# Geological Data Visualization

## Week 4

Andrew Pellett  
6/25/08

### Progress

The original MATLAB code uses the `print()` function to capture the images after they've been drawn. Results of rendering 10 frames at 150 dpi using `print()` and a contour level of `.5`:

```
Iteration: 1
predraw: 0.2679 | draw: 13.1176 | iteration: 13.3883
Iteration: 2
predraw: 0.15411 | draw: 12.3722 | iteration: 12.5293
Iteration: 3
predraw: 0.15047 | draw: 12.1873 | iteration: 12.3385
Iteration: 4
predraw: 0.15312 | draw: 12.4637 | iteration: 12.6176
Iteration: 5
predraw: 0.15111 | draw: 12.4683 | iteration: 12.6228
Iteration: 6
predraw: 0.15108 | draw: 12.069 | iteration: 12.2208
Iteration: 7
predraw: 0.15458 | draw: 11.7453 | iteration: 11.9006
Iteration: 8
predraw: 0.15075 | draw: 11.775 | iteration: 11.9311
Iteration: 9
predraw: 0.15068 | draw: 11.4857 | iteration: 11.6372
Iteration: 10
predraw: 0.15047 | draw: 11.4644 | iteration: 11.6156
Average iteration: 12.1571 seconds.
Colorbar draw time: 2.4745 seconds.
Initialization time: 0.2323 seconds.
Total elapsed time: 125.5708 seconds.
```

On average, this method takes over 12 seconds to draw a single frame. By profiling the program, I confirmed that `print()` is what's taking up the majority of the run time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
<a href="#">ncepvis4</a>	1	124.425 s	1.700 s	
<a href="#">print</a>	10	99.260 s	0.000 s	
<a href="#">graphics/private/render</a>	10	72.064 s	67.763 s	
<a href="#">graphics/private/prepare</a>	10	22.252 s	15.402 s	
<a href="#">specgraph.contourgroup.refresh</a>	30	19.889 s	0.788 s	
<a href="#">...ph.contourgroup.schema&gt;LdoDirtyAction</a>	40	19.848 s	0.000 s	
<a href="#">...ontourgroup.schema&gt;LdoMarkDirtyAction</a>	150	18.656 s	0.021 s	
<a href="#">...raph.contourgroup.refresh&gt;LdrawFilled</a>	20	11.753 s	11.753 s	
<a href="#">graphics/private/fireprintbehavior</a>	20	9.079 s	1.596 s	
<a href="#">hggetbehavior</a>	43437	7.369 s	4.104 s	
<a href="#">contours</a>	20	7.089 s	6.758 s	
<a href="#">graphics/private/preparehg</a>	10	6.809 s	0.041 s	
<a href="#">graphics/private/restore</a>	10	4.633 s	0.010 s	
<a href="#">graphics/private/restorehg</a>	10	4.612 s	-0.000 s	
<a href="#">imwrite</a>	11	3.441 s	0.021 s	
<a href="#">imagesci/private/writepng</a>	11	3.400 s	0.062 s	
<a href="#">hggetbehavior&gt;localPeek</a>	43437	3.244 s	3.244 s	
<a href="#">imagesci/private/png (MEX-function)</a>	11	3.182 s	3.182 s	
<a href="#">getframe</a>	1	1.482 s	1.482 s	
<a href="#">graphics/private/prepareui</a>	10	1.213 s	1.088 s	
<a href="#">findall</a>	151	1.192 s	1.192 s	
<a href="#">contourf</a>	10	1.099 s	0.031 s	
<a href="#">newplot</a>	10	0.902 s	0.010 s	
<a href="#">graphics/private/clo</a>	10	0.871 s	0.757 s	
<a href="#">newplot&gt;ObserveAxesNextPlot</a>	10	0.871 s	0.000 s	

I figured that there must be a quicker way to do this, so I explored the available image-related functions in MATLAB. `imwrite()` from the image toolbox is a lower level call that writes images given an m-by-n-by-3 array or an m-by-n array with a colormap. It is possible to grab images in this format using `getframe()`, but `getframe()`'s resolution is limited to the size of the image at the time of capture, which is an annoying limitation. The `getframe()/imwrite()` method produced better times, though.

```
Iteration: 1
predraw: 0.39004 | draw: 7.6496 | iteration: 8.0424
Iteration: 2
predraw: 0.18984 | draw: 6.3739 | iteration: 6.5645
Iteration: 3
predraw: 0.18876 | draw: 6.238 | iteration: 6.4276
Iteration: 4
predraw: 0.19385 | draw: 6.3952 | iteration: 6.6077
Iteration: 5
predraw: 0.18753 | draw: 6.2981 | iteration: 6.4864
Iteration: 6
predraw: 0.20145 | draw: 7.2321 | iteration: 7.4343
Iteration: 7
predraw: 0.18384 | draw: 6.2413 | iteration: 6.4259
Iteration: 8
predraw: 0.18826 | draw: 6.5113 | iteration: 6.7003
Iteration: 9
predraw: 0.19596 | draw: 6.3203 | iteration: 6.5228
Iteration: 10
predraw: 0.195 | draw: 6.4644 | iteration: 6.6602
```

**Average iteration: 6.7872 seconds.**

Colorbar draw time: 5.0751 seconds.

Initialization time: 0.29307 seconds.

Total elapsed time: 73.3196 seconds.

This method cuts the average draw time nearly in half. The run time is now dominated by the getframe call, but it's not as significant as the print() call.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
<a href="#">ncepvis3</a>	1	72.167 s	1.928 s	
<a href="#">getframe</a>	11	45.201 s	45.191 s	
<a href="#">specgraph.contourgroup.refresh</a>	30	21.073 s	0.854 s	
<a href="#">...ph.contourgroup.schema&gt;LdoDirtyAction</a>	40	21.051 s	0.022 s	
<a href="#">...ontourgroup.schema&gt;LdoMarkDirtyAction</a>	150	19.769 s	0.033 s	
<a href="#">...raph.contourgroup.refresh&gt;LdrawFilled</a>	20	12.048 s	12.048 s	
<a href="#">contours</a>	20	7.974 s	7.579 s	
<a href="#">contourf</a>	10	1.446 s	0.033 s	
<a href="#">graphics/private/clo</a>	10	1.216 s	1.106 s	
<a href="#">newplot</a>	10	1.216 s	0.000 s	
<a href="#">newplot&gt;ObserveAxesNextPlot</a>	10	1.216 s	0.000 s	
<a href="#">imwrite</a>	11	1.216 s	0.033 s	
<a href="#">imagesci/private/writepng</a>	11	1.128 s	0.055 s	
<a href="#">netcdf.subsref</a>	176	1.008 s	0.044 s	
<a href="#">imagesci/private/png (MEX-function)</a>	11	0.986 s	0.986 s	
<a href="#">ncvar.subsref</a>	176	0.789 s	0.164 s	
<a href="#">colorbar&gt;make_colorbar</a>	1	0.646 s	0.000 s	
<a href="#">colorbar</a>	1	0.646 s	0.000 s	
<a href="#">scribe_colorbar_colorbar</a>	1	0.624 s	-0.000 s	
<a href="#">ncmex</a>	1704	0.471 s	0.383 s	
<a href="#">...rgroup.schema&gt;LdoUpdateChildrenAction</a>	20	0.438 s	0.405 s	
<a href="#">ncvar.ncsize</a>	310	0.416 s	0.110 s	
<a href="#">scribe_colorbar_init</a>	1	0.350 s	0.000 s	
<a href="#">contours&gt;fixrounding</a>	40	0.350 s	0.274 s	
<a href="#">legendcolorbarlayout</a>	6	0.285 s	0.000 s	