Visualising Large Datasets in TOPCAT v4

ADASS XXIII, Hawaii, 1 October 2013

Mark Taylor
(University of Bristol)
What is TOPCAT?

TOPCAT — Tool for OPerations on Catalogues And Tables

“Does what you want with tables”

Desktop Java GUI application

Key capabilities include:

- Visualisation
- Crossmatching
- Row selections
- Linked views
- Powerful expression language
- Good communications (file formats, other tools, data services, VO)

For doing science with tables

Version 4 (2013): rewrite visualisation from scratch
Q: How do you visualise \(\sim 10,000,000\) 2D/3D points

... using \(<1,000,000\) pixels?
... so both large and small scale structure is visible?
... presenting multiple dimensions?

A: Key requirements:

- Fast and easy interactive navigation
- Single plot type that works at both high and low densities
- Extensive configurability
High/Low Density Plots
Hybrid Scatter/Density Plot

Convolve a single-pixel density map with multi-pixel marker shape

- Each plotted point paints a marker shape on pixel grid *(configurable marker shape)*
- Pixel color is determined by how many markers hit it *(configurable color map)*

Other ways to look at it:

- A single-pixel density map with a shaped 1-bit smoothing kernel
- A scatter plot with non-standard compositing

Result:

- Low density regions: it’s a single-color scatter plot
- High density regions: it’s a smoothed density map
Multi-Dataset Plots

Color maps:

- Full-color maps (e.g. rainbow) are good for single-component plots
  ... but not so good for multi-component (especially 3D)
  ... and no good for point color coding

- By default use darkening map
  - Scale (Hue, Saturation, Value) Value of dataset base color
  - Use asinh ramp auto-scaled from data

- Provide other options too

Mark Taylor, Visualising large datasets in TOPCAT v4, ADASS XXIII, Hawaii, 1 October 2013
2D gestures — fairly obvious

- Mouse drag to pan
- Mouse wheel to zoom around mouse position

Notes:

- No multi-touch support in J2SE 😞 would require native code
- Wheel zoom needs a wheel
- Wheel zoom currently only isotropic — anisotropic options TBD
- Other zoom options supplied (slider, enter explicit range), but less convenient
Navigation: 3D

3D — pan/zoom not obvious (2D gestures, 3D space)

- Left button: drag to rotate round center
- Wheel: zooms in/out round center
- Right button: click to re-center
  - Break screen-normal degeneracy using “center of mass” along line of sight. Click on dense blob or isolated point works. Generally does what you want.

Notes:

- Zoom around cursor would be possible, but slow and unstable if center of mass is determined each frame
- Not so easy to navigate to distant low-density regions — but do you want to?
Performance

Target:

- User makes tweaks to plot (pan, zoom, marker shape/color, contour levels, density/aux color map/clip, vector size scaling, error bar style, font size, ...)
  → Plot updates immediately (≪1 second)
- User tries a different plot (different coordinate values, adds vectors/ellipses, ...)
  → Plot updates quickly (≤ a few seconds),
  but if necessary take more time to cache data for later use

Lots of tricks

- Smart multi-level caching (column data, per-layer density maps, plot image)
- Plot-sensitive rendering schemes (monochrome, opaque, Z-buffer, ...)
- Pluggable data storage management (primitive array, table interface, mapped file, ...)
- Build pixel map/manage compositing by hand
- Pack coords into primitives (RGBA+z→\texttt{double}, RA+Dec→\texttt{int}[3])
- Careful thread management (multi-threading TBD)
- Marker pixel map pre-calculation
- ...

... work in progress
Many configuration options to manage

- Tens of options required to specify a given plot:
  - ~20 options for axes
    (coord log/flip, axis ranges, axis annotations, grid options, . . . )
  - ~10 options per data layer
    (marker color/size/shape, color map/scale, compositing options, label, . . .

⇒ usability challenge

- ~100 options (and counting) in the application as a whole

  - Several axis types:
    2D, 3D, Sky, Sphere, Time Series
  - Many layer types:
    Points, Vectors, Error bars, Elipses, Analytic functions, Lines, Contours,
    Text labels, Polygons, Spectrograms . . .

⇒ implementation challenge
Try to combine extreme flexibility with ease of use

- Everything is configurable
- ... but has sensible defaults
- Plots auto-populate and auto-range
Configuration Usability

Try to combine extreme flexibility with ease of use

- Everything is configurable
- ... but has sensible defaults
- Plots auto-populate and auto-range

Good:
Try to combine extreme flexibility with ease of use

- Everything is configurable
- ... but has sensible defaults
- Plots auto-populate and auto-range
**ConfigKey grouping:**
- 1 group for axes
- 1 group per data layer

Each **ConfigKey** has:
- Name
- Description
- Type
- Default
- GUI control
- value ↔ string mapping

Use them to:
- Build GUI
- Specify plot from GUI
- Specify plot from shell, API, SAMP, ...
- Build documentation

*Very easy to add new configuration options*
**ConfigKey grouping:**
- 1 group for axes
- 1 group per data layer

**Each ConfigKey has:**
- Name
- Description
- Type
- Default
- GUI control
- value $\leftrightarrow$ string mapping

**Use them to:**
- Build GUI
- Specify plot from GUI
- Specify plot from shell, API, SAMP, ...
- Build documentation

**Very easy to add new configuration options**
Current status:

- In public release, but not yet replacing old-style plots (hidden in Graphics menu)
- Many plot/layer types: vectors, ellipses, contours, analytic functions, all-sky, color/size coding, ...
- Performance: 1 Mrow: good; 10 Mrow: usable; 300 Mrow: reported

Coming soon:

- More plot/layer types (stacked time series, histogram, plot grid, regions, ...)
- External control (STILTS, public API, SAMP)
- Performance improvements (×10?)