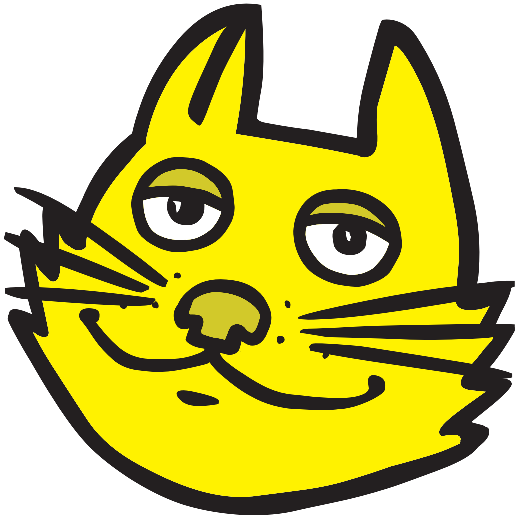
TOPCAT: Working with data & working with users

Mark Taylor (University of Bristol)

ADASS XXVII
Santiago de Chile

October 2017



$Id: talk2-lite.tex,v 1.2 2017/10/24 15:12:07 mbt Exp $

Outline

Session topic:

Human-Computer Interaction, UI Design Guidelines & Interfaces to Big Data Sets

Talk outline:

- TOPCAT introduction
- Technical Questions (with answers)
  - ▷ external services, data access, scalability, implementation platform, ...
- Human Questions (with ideas)
  - ▷ requirements, user engagement, user interfaces, marketing, ...
- Summary

Outline

Session topic:

Human-Computer Interaction, UI Design Guidelines & Interfaces to Big Data Sets

Talk outline:

- TOPCAT introduction
- Technical Questions (with answers)
  - ▷ external services, data access, scalability, implementation platform, ...
- Human Questions (with ideas)
  - ▷ requirements, user engagement, user interfaces, marketing, ...
- Summary

IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPUs, UP FROM THE OLD LIMIT OF 1,024.
DO YOU HAVE SUPPORT FOR SMOOTH FULL-SCREEN FLASH VIDEO YET?
NO, BUT WHO USES THAT?

xkcd.com

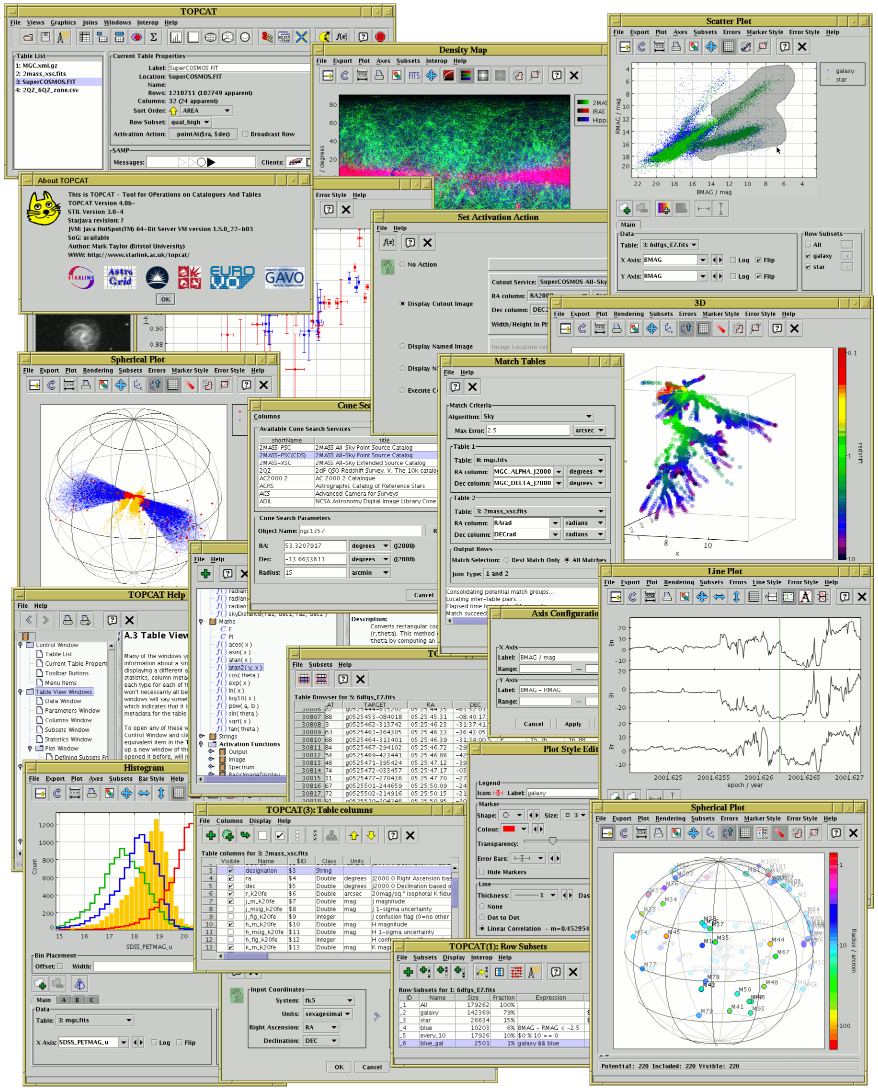
TOPCAT

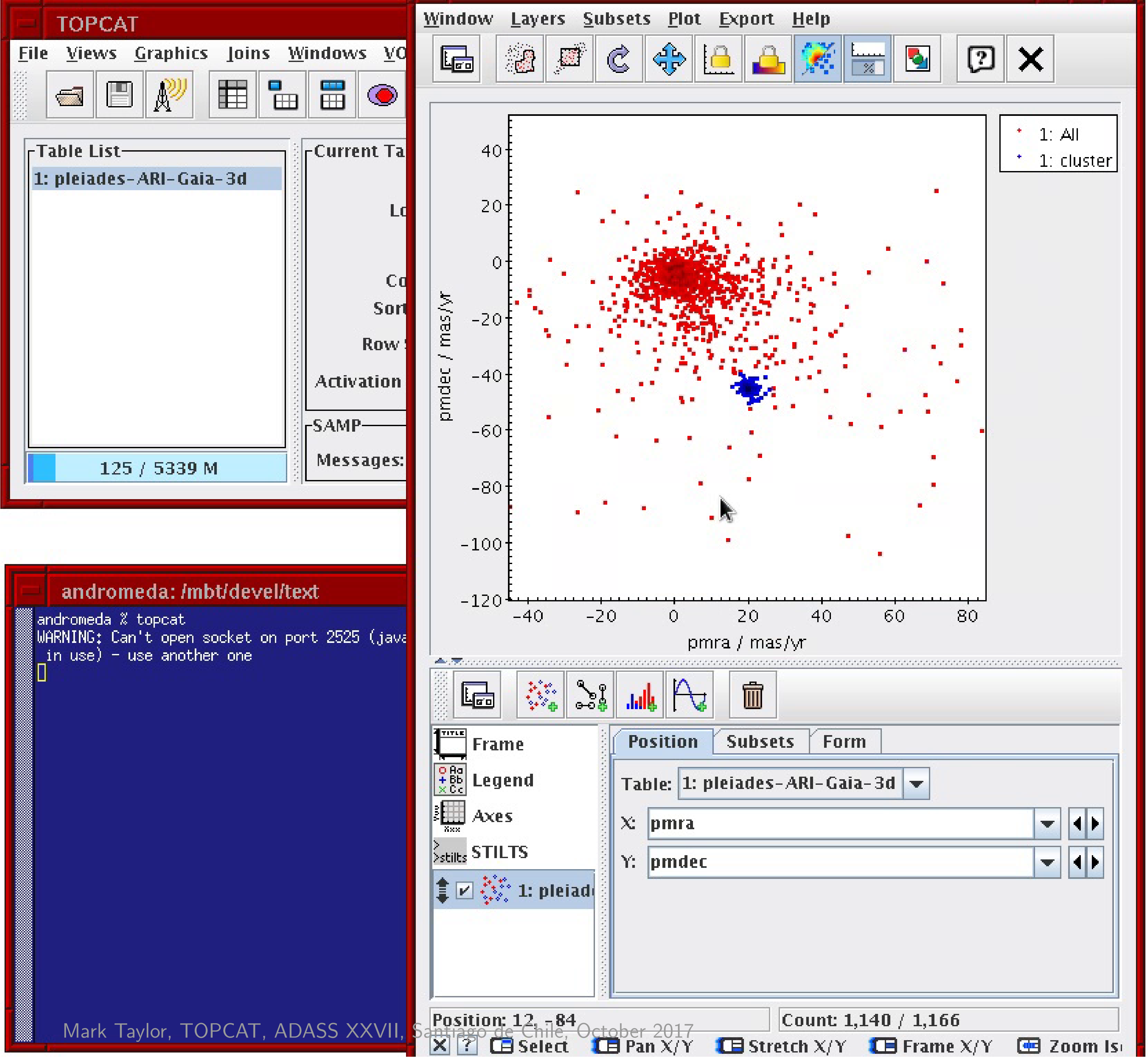
TOPCAT: Does what you want with tables

Aims to do all the mechanical things that astronomers need to work with source catalogues (and other tables), so they can concentrate on science

- Under development since 2003
- $O(10^3)$ active users
- ~ 400 citations

STILTS provides a command-line interface to the same functionality.





```
andromeda: /mbt/devel/text
andromeda % topcat
WARNING: Can't open socket on port 2525 (java
 in use) - use another one
```

Table: 1: pleiades-ARI-Gaia-3d

X: pmra

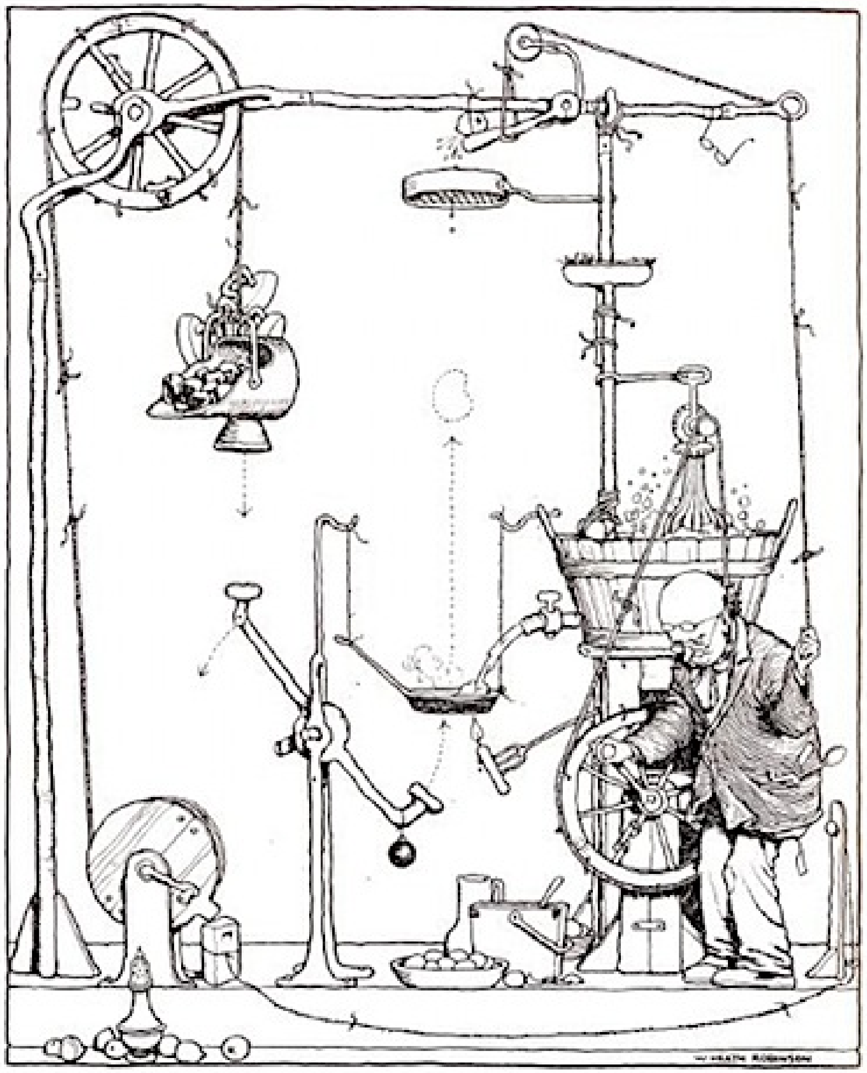
Y: pmdec

Position: 12, -84

Count: 1,140 / 1,166

Technical Questions

- External service use
- Data access model
- Local I/O
- Scalability
- Platform



W. Heath Robinson

External Services

Much of TOPCAT's power comes from working with other software

- **Virtual Observatory**
  - ▷ Table access: **TAP**, Cone Search, Simple Image Access, Simple Spectral Access, ...
  - ▷ Data discovery: Registry
  - ▷ Desktop interoperability: SAMP
  - ▷ Standardisation means effort is manageable
    - ○ Developer benefit: Implement once for many services
    - ○ User benefit: Consistent user interface for many services

External Services

Much of TOPCAT's power comes from working with other software

- **Virtual Observatory**
    - ▷ Table access: **TAP**, Cone Search, Simple Image Access, Simple Spectral Access, ...
    - ▷ Data discovery: Registry
    - ▷ Desktop interoperability: SAMP
    - ▷ Standardisation means effort is manageable
        - ○ Developer benefit: Implement once for many services
        - ○ User benefit: Consistent user interface for many services
- A few non-standard data access services:
    - ▷ VizieR, GloTS, CDS X-Match — *too useful to be without*
    - ▷ BaSTI, Millennium simulations — *historical relics*

Data Access Model

There are smart ways to work with big data:

- Bring the computation to the data
- Progressive downloads (HiPS)
- ...

Data Access Model

There are smart ways to work with big data:

- Bring the computation to the data
- Progressive downloads (HiPS)
- ...

TOPCAT does it the dumb way:

- Download a static table to the client, and then use it
    - ▷ Disadvantage: user needs to pre-select data if input dataset is very large
        - ○ In practice, usually works fine; many tables are small(ish) anyway, and pre-selection down to a few million rows is usually acceptable
    - ▷ Advantage: low-tech approach, not much to go wrong
        - ○ No server-side component required
        - ○ Robust against network issues

Local I/O

Traditional data access model needs good I/O to local files

- Users can store large datasets locally
- Application may download large datasets from VO services
  (for immediate use or save and later reload)

Requirements:

- Fast load, efficient access
- Random access
- Preserve metadata

Solution:

- FITS BINTABLE: Compact, preserves data, data laid out predictably
- Memory mapping: Instant load, fast read, random access, caching delegated to OS, does not use Java's limited heap space
- Some customisation of FITS format:
  - ▷ Store rich metadata in VOTable format in unused primary HDU *("FITS-plus")*
  - ▷ Column-oriented storage option for large/wide tables *("colfits")*
  - ▷ Private convention for >999 columns

Local I/O

Predictable layout + memory mapping
⟶ Fast random access to large files

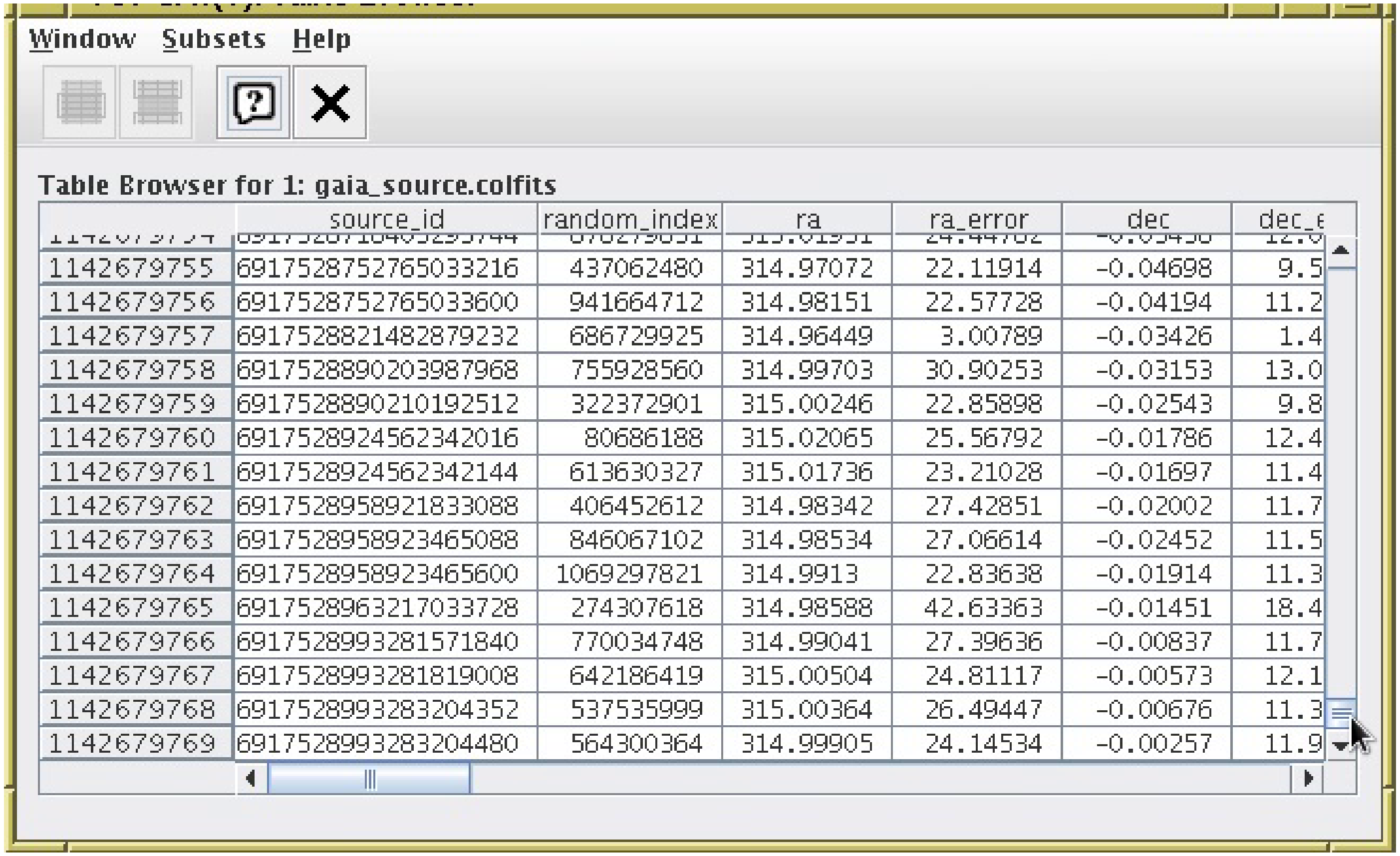


Table Browser for 1: gaia_source.colfits

	source_id	random_index	ra	ra_error	dec	dec_e
1142679755	6917528752765033216	437062480	314.97072	22.11914	-0.04698	9.5
1142679756	6917528752765033600	941664712	314.98151	22.57728	-0.04194	11.2
1142679757	6917528821482879232	686729925	314.96449	3.00789	-0.03426	1.4
1142679758	6917528890203987968	755928560	314.99703	30.90253	-0.03153	13.0
1142679759	6917528890210192512	322372901	315.00246	22.85898	-0.02543	9.8
1142679760	6917528924562342016	80686188	315.02065	25.56792	-0.01786	12.4
1142679761	6917528924562342144	613630327	315.01736	23.21028	-0.01697	11.4
1142679762	6917528958921833088	406452612	314.98342	27.42851	-0.02002	11.7
1142679763	6917528958923465088	846067102	314.98534	27.06614	-0.02452	11.5
1142679764	6917528958923465600	1069297821	314.9913	22.83638	-0.01914	11.3
1142679765	6917528963217033728	274307618	314.98588	42.63363	-0.01451	18.4
1142679766	6917528993281571840	770034748	314.99041	27.39636	-0.00837	11.7
1142679767	6917528993281819008	642186419	315.00504	24.81117	-0.00573	12.1
1142679768	6917528993283204352	537535999	315.00364	26.49447	-0.00676	11.3
1142679769	6917528993283204480	564300364	314.99905	24.14534	-0.00257	11.9

`gaia_source` table (Gaia DR1)

10^9 rows × 33 columns

180 Gbyte colfits file

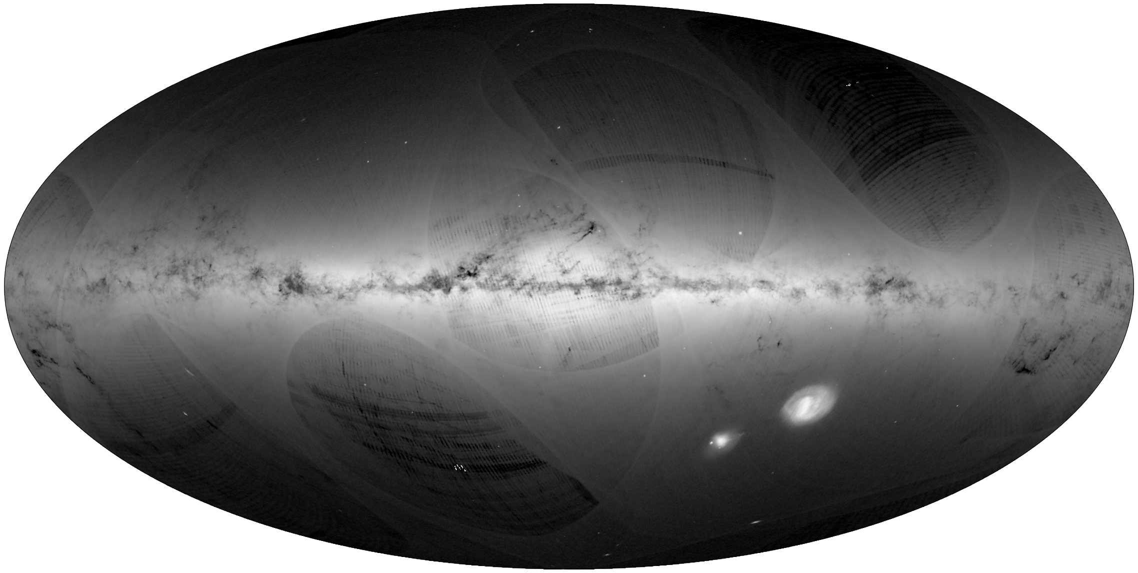
Scalability

Most basic functions work with unlimited row count

- Interfaces not arrays
  - ▷ Can be backed by mapped file, stream from URL, array, ...
- `long`s (64-bit) not `int`s (32-bit)
  - ▷ $2^{31} \sim 2$ billion; Gaia DR2 source list ~ 2 billion
- Implement using iterated not indexed access where possible
  - ▷ Random access required for some algorithms but not most
- Avoid memory usage that scales with row count
  - ▷ E.g. store pixel grids not point lists when plotting

In practice:

- TOPCAT
  - ▷ 1M rows no problem
  - ▷ 10M rows not bad
  - ▷ 100s Mrows possible
- STILTS
  - ▷ No limits for most things



Gaia DR1 source density: 1.1 Grow, 128 Mb RAM, 1 CPU, 25 mins.
STILTS plot from single colfits file

Platform: Desktop Java

Build/deploy benefits:

- Easy deployment: single jar file to download + run, user just needs Java Runtime

  + a few other optional possibilities: Un*x script, WebStart, DMG file, Debian package
- No user build/library issues
- No OS/platform-dependent issues (well, almost none)
  - ▷ Stick rigidly to pure Java (no native/C-based libraries)
  - ▷ Avoid libraries relying on system-dependent behaviour
- No platform version issues
  - ▷ Java SE backward compatibility is excellent (so far)
  - ▷ Currently target Java SE 1.6 (very old) so any java runtime will work

Other nice features:

- Static typing/picky compiler, good libraries, concurrency support, javadocs, ...

Disadvantages:

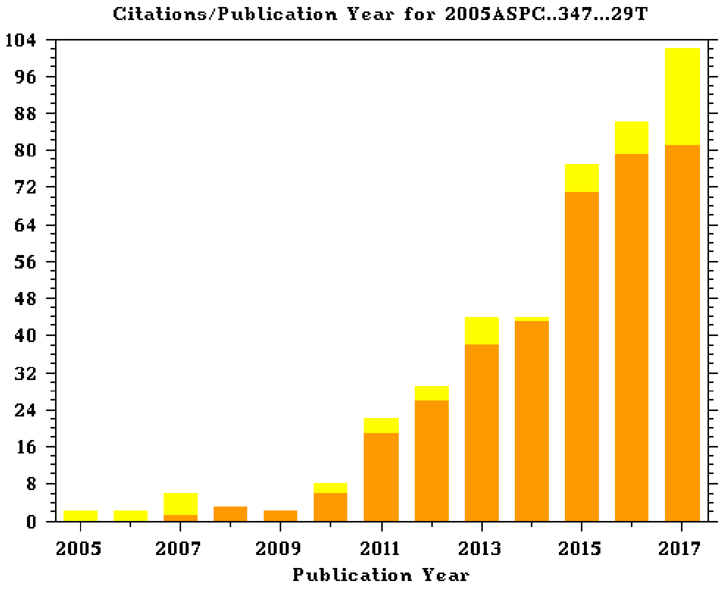
- Some system-dependent features unavailable (GPUs, multi-touch, ...)
- C-based libraries unavailable (HDF5)
- OS/Desktop integration not always perfect, Swing is a bit ugly
- Some people hate Java
  - ▷ Too bad!

Human Questions

- Take-up
- Gathering requirements
- User engagement
- Prioritising implementation
- GUI design

Take-Up

- Getting people to use software is hard
    - Writing good software is not enough!
    - I'm lazy and hate learning new ways to do things; probably other people are the same
    - Human-Computer Interaction doesn't start when the user runs an application,
      it starts when she thinks about how or whether to run it (installation, go to a web page, ...)
- What helps?
    - Low barrier to initial use (installation and startup as easy as possible)
    - First impressions (beginning use as easy & rewarding as possible)
    - Documentation? (tutorial and reference documents, FAQs, examples, videos?)
    - Tutorials, conferences
    - Word of mouth
- There are usually external factors
    - politics, geography, ...
- I don't have a magic bullet
- It's a long job ...



ADS

Gathering Requirements

Ask the users?

> "If I had asked my customers what they wanted, they would have said faster horses."
> — *(mis?)attributed to Henry Ford*

- Users don't know what they want.
- It's not really their job to know, and often they don't like being asked.

Top-down visionary design?

- Nice idea, but I'm not smart enough.
- I don't know what users want either, though it is my job to know.
- Defining requirements is hard. It's especially hard when the landscape of what's possible (available data, available services) keeps changing.

Incremental development, informed by user engagement

- Short development cycle *(agile?)*
- Provide some basic functionality, let users play with it, see how it works, see how or whether to improve it

User Engagement

- Encourage how-to queries and bug reports
  - They can give you a good idea what people are doing or trying to do, which sometimes suggests missing functionality or opportunities for improvement
- Have a public mailing list.
  - Sometimes users answer each others questions.
  - It's a good encouragement to write good replies when it's in public.
  - (but sometimes users prefer to discuss things in private)
  - Social media?
- Preparing and delivering demos and tutorials is a good discipline.
  - It can give a user's-eye view of missing functionality
  - It's a strong motivation to fix what's embarrassingly bad.
- Have contact with multiple projects
  - Get funded by a succession of different projects?
  - VO community has been good for communication

Prioritising Implementation

How to prioritise/select from the to-do list?

- Do the easy things first!
- Do things that several people have asked for
- Features must be discoverable
  - New features must have a comprehensible UI
  - Avoid degrading existing UIs (but sometimes you have to)
  - Avoid demoware (expert-only controls, undiscoverable functions, data-specific functionality)
- Beware feature creep
  - TOPCAT is for tables; adding spectrum/time-series functions is tempting but may complicate the UI

    > "Every program attempts to expand until it can read mail.
    > Those programs which cannot so expand are replaced by ones which can."
    > — *Zawinski's Law of Software Development*

  - SAMP helps (delegate to Aladin, DS9, SPLAT, ...)
- Have good communication between *design*, *implementation* and *user support* teams
  - That's easy for me to say :-)

GUI Design Principles

Aims:

- Simplicity
- Flexibility
- Responsiveness
- Visibility of status
- Recognition not recall
- Error reporting/error prevention
- Documentation

Obstacles:

- Screen real estate
- Simplicity vs. flexibility
- Representing many-element status
- Recognition for unfamiliar functions
- Responsiveness for large datasets

GUI Design Principles

Aims:

- Simplicity
- Flexibility
- Responsiveness
- Visibility of status
- Recognition not recall
- Error reporting/error prevention
- Documentation

Obstacles:

- Screen real estate
- Simplicity vs. flexibility
- Representing many-element status
- Recognition for unfamiliar functions
- Responsiveness for large datasets

The problem:

- *It's hard* to control many options from a comprehensible UI

  ... and gets harder the more capable the software is
- If you get it wrong, the implementation is pointless
  - ▷ People probably won't use a feature that's hard to understand
  - ▷ People definitely won't use a feature they don't know is there

GUI Design in Practice

Users are lazy when operating software

- They don't read manuals
- They don't like having to think about the user interface
- They don't like making difficult decisions

This is because they are thinking about astronomy

Require minimal user effort:

User action required	Acceptability
None	Good
Something obvious	OK
Multiple choice	Not bad
Fill in a blank field	Avoid

Explorable interface (for the visualisation GUI):

- Initially display something, not nothing
- Put common controls somewhere obvious
- Put other controls somewhere discoverable
- Fiddling with any control should do something visible, immediately

GUI Examples

Show something not nothing

- Plot window opens with a plot visible

Defaults give something reasonable

- Axis auto-range determination
- Representation makes sense for dense or sparse data

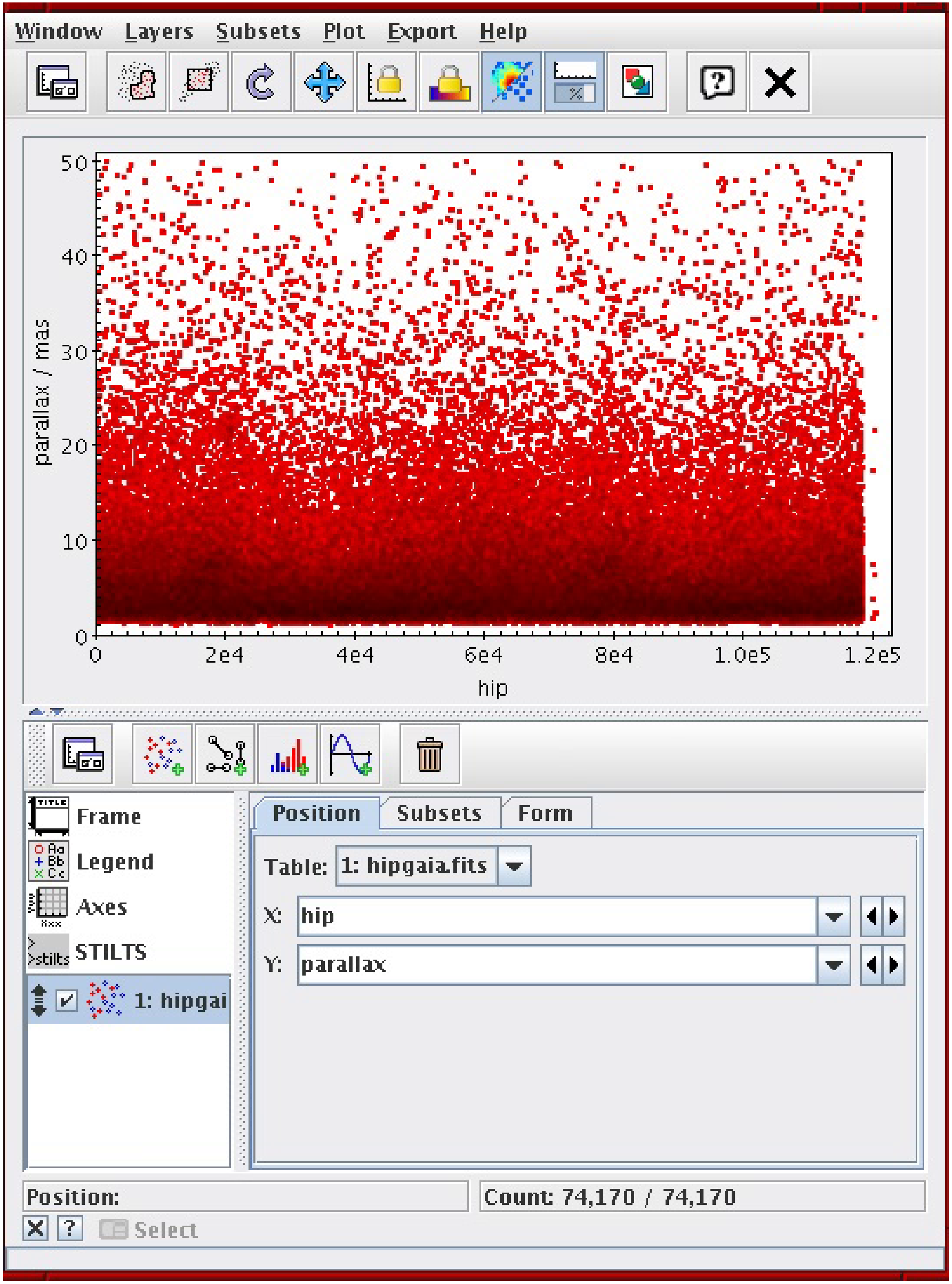


Table: 1: hipgaia.fits

X: hip

Y: parallax

Position:

Count: 74,170 / 74,170

GUI Examples

Obvious how to do basic things

- Plotted quantity controls are prominent

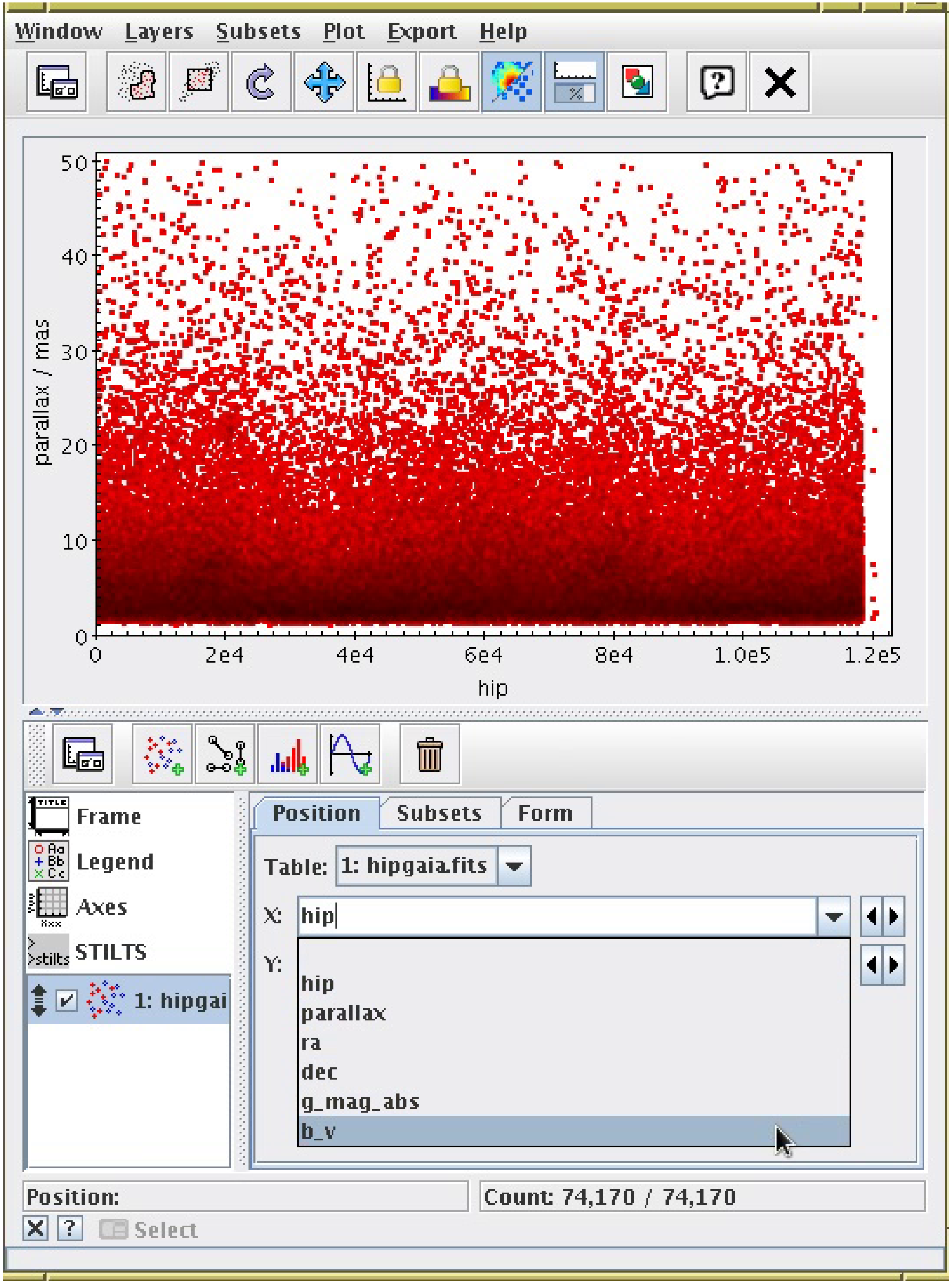


Table: 1: hipgaia.fits

X: hip

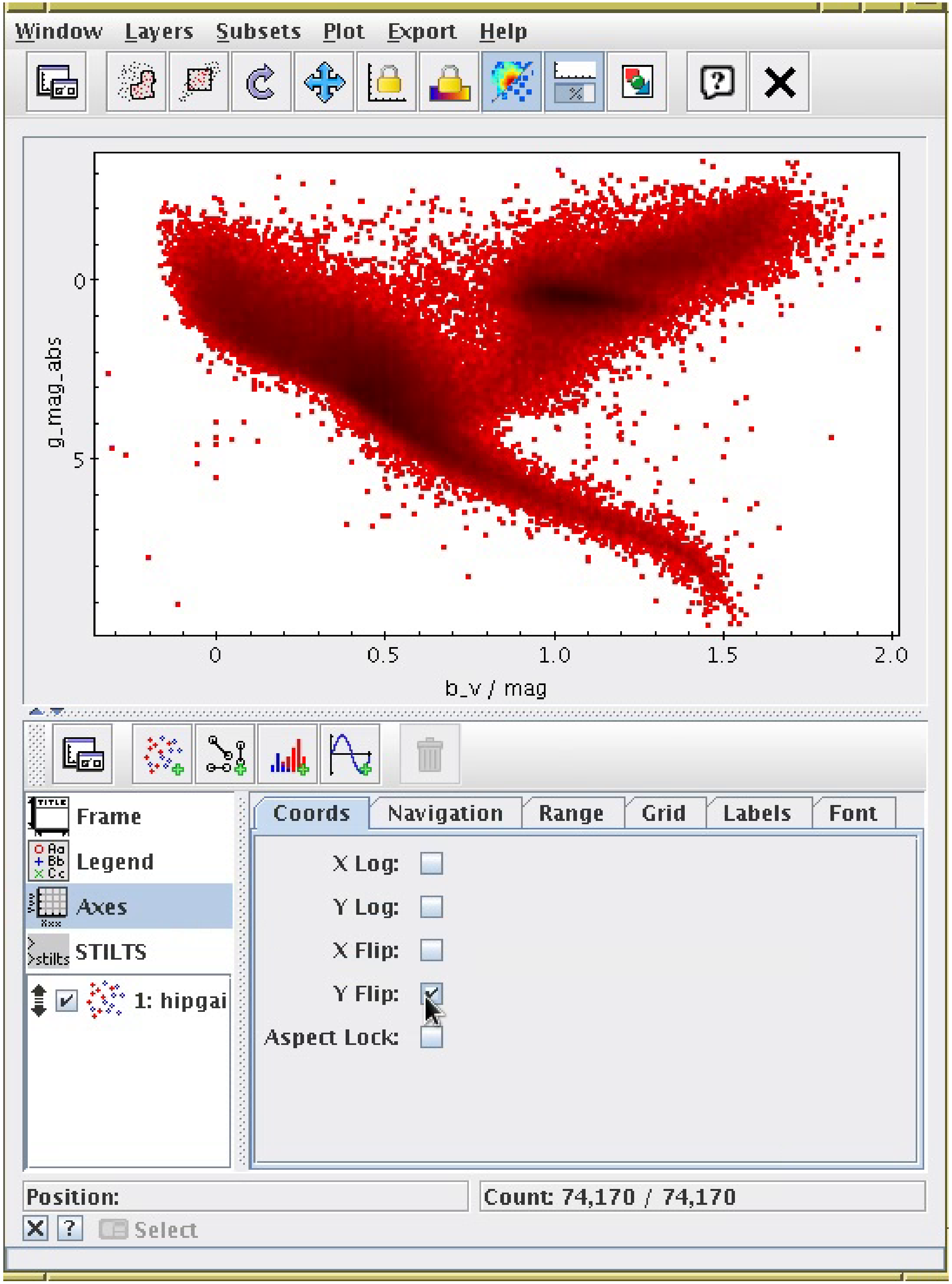
Y:

Position:

Count: 74,170 / 74,170

GUI Examples

Clues visible for
how do to less
obvious things



X Log: □

Y Log: □

X Flip: □

Y Flip: ☑

Aspect Lock: □

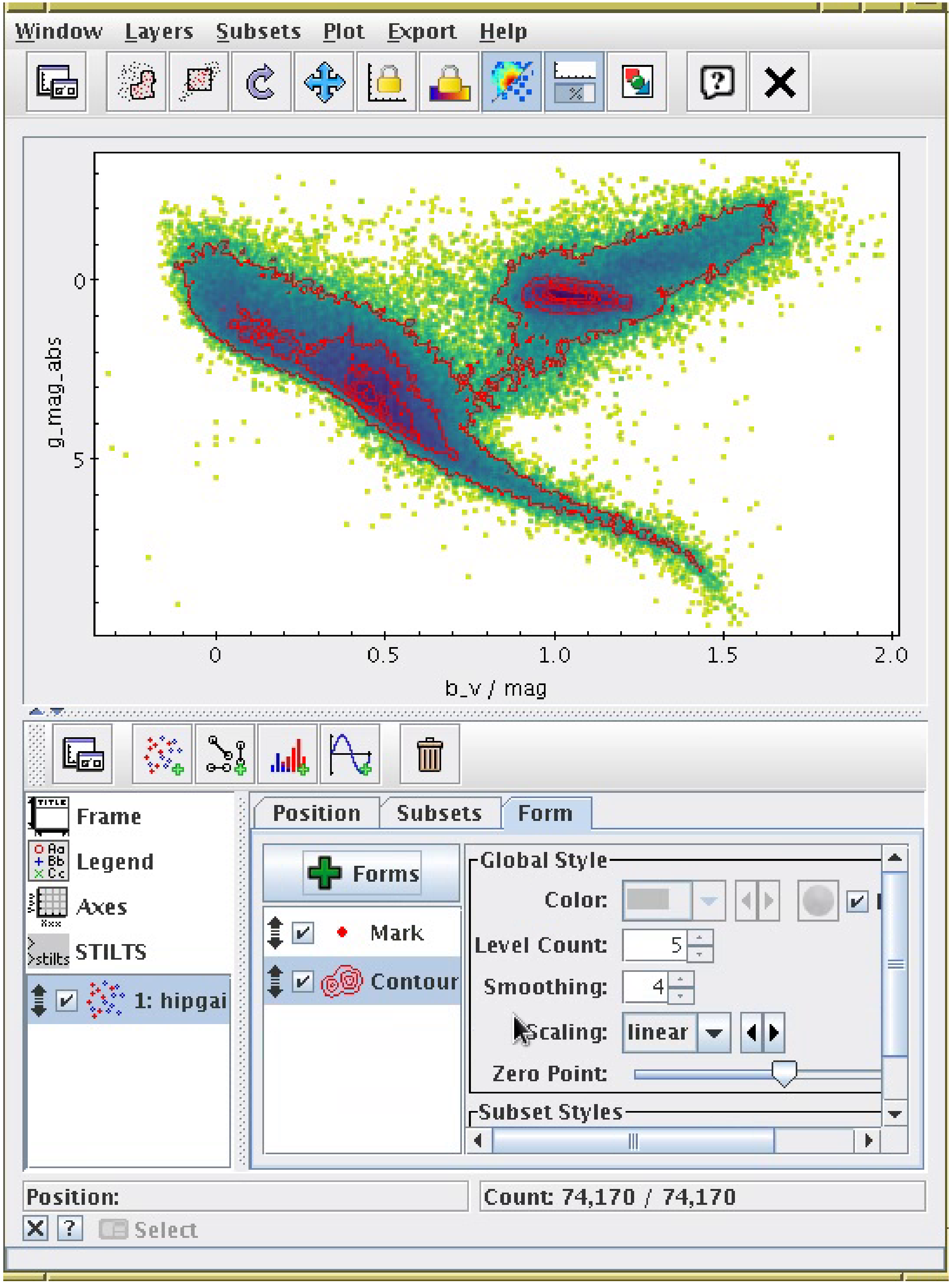
Position:

Count: 74,170 / 74,170

GUI Examples

Many more sophisticated options available if you go looking

- Controls should give an idea what they will do
- A comprehensible amount of options should be visible at any one time



Color:

Level Count: 5

Smoothing: 4

Scaling: linear

Zero Point:

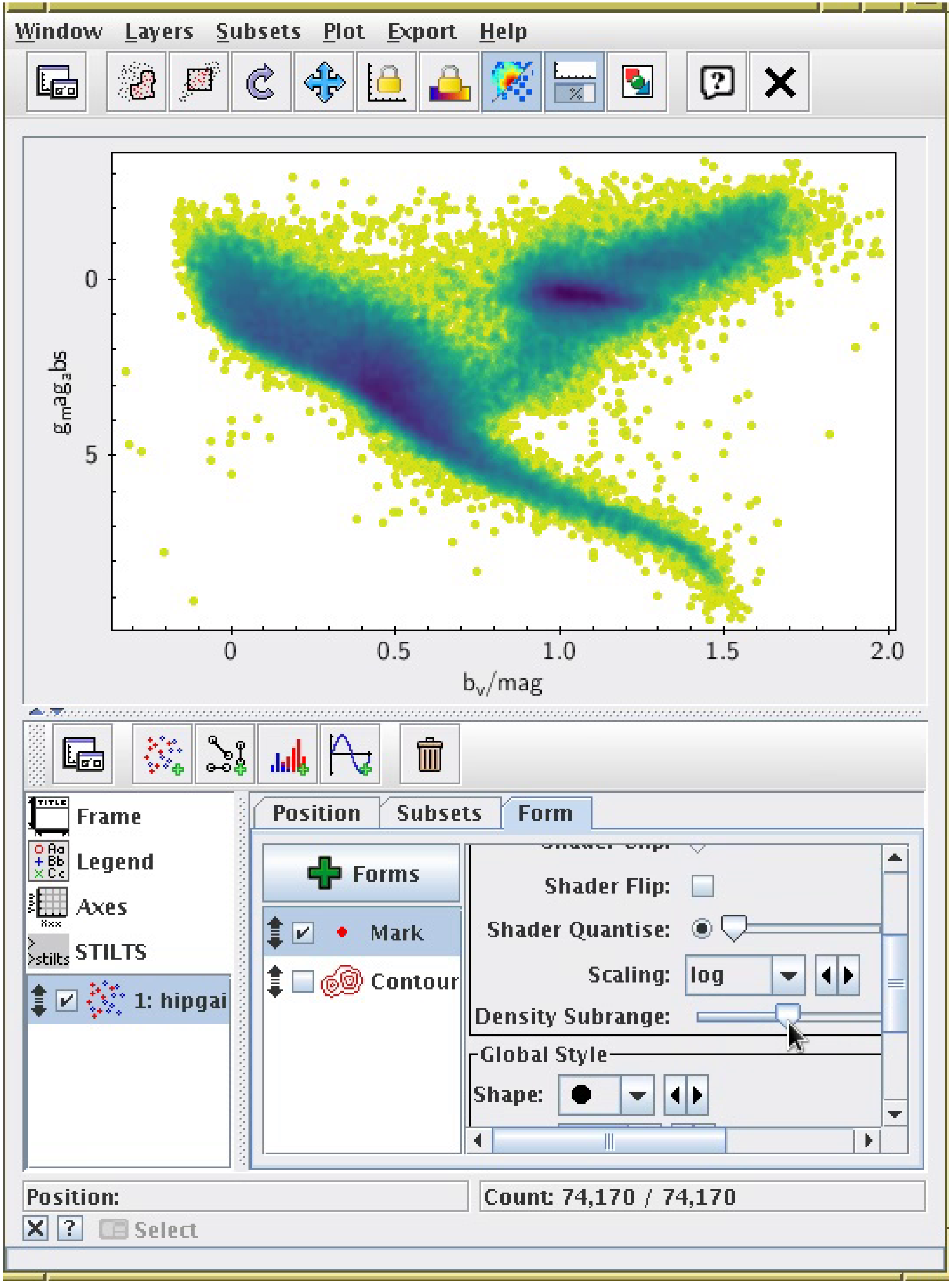
Position:

Count: 74,170 / 74,170

GUI Examples

Instant feedback when adjusting controls

- Easy to find out what controls do by trying them out
- No “replot” button



Shader Flip: □

Shader Quantise: ●

Scaling: log

Density Subrange:

Shape: ●

Position:

Count: 74,170 / 74,170

Final Comments

TOPCAT's Top Tips for application development:*

- It is possible to write client-side software for quite large datasets
  - Build scalability in from the bottom up; understand bottlenecks
  - Use memory-mapped files
- Steal judiciously
  - Use other people's libraries/services if they do what you want
  - Have complete control over core functions; that may mean reinventing wheels
- Persuading people to use (even good) software is hard
  - Make installation and beginning use *really* easy
  - Accommodate user laziness
- Defining detailed requirements is hard
  - Short development cycles informed by user input/support
  - Try to think like a user
  - Don't do anything clever! Leave the astronomy to the astronomers.
- GUI design is really hard
  - Provide working defaults wherever possible
  - Explorable interfaces — obvious things easy, other things discoverable
- Project management is easy for a 1-person project

*: YMMV

Extra Slides

Exploratory/Reproducible Modes

TOPCAT: GUI tool

- Suitable for exploring data

STILTS: command-line suite

- Scriptable
- Can work with streams (arbitrarily large tables)
- Various flavours: Un*x shell, Jython front-end, CGI-BIN interface, API, ...
- Steeper learning curve; citations suggest STILTS usage much lower than TOPCAT

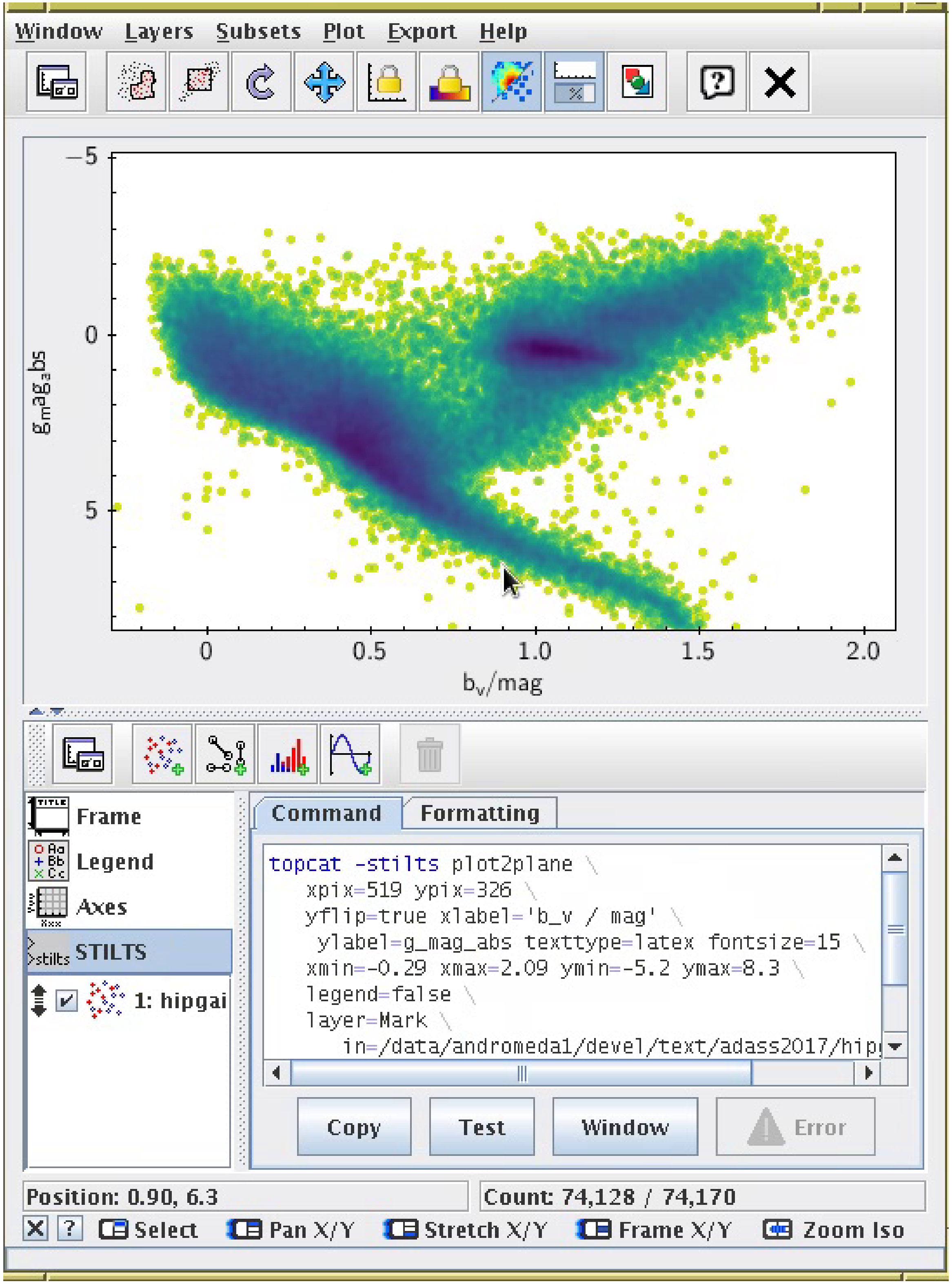
GUI/CLI integration

- From v4.5 (Sept 2017) TOPCAT can export STILTS plot commands

GUI Examples

Provide scripted equivalents

- New in TOPCAT 4.5:
  STILTS plot command export



```
topcat -stilts plot2plane \
   xpix=519 ypix=326 \
   yflip=true xlabel='b_v / mag' \
    ylabel=g_mag_abs texttype=latex fontsize=15 \
   xmin=-0.29 xmax=2.09 ymin=-5.2 ymax=8.3 \
   legend=false \
   layer=Mark \
      in=/data/andromeda1/devel/text/adass2017/hip
```

Position: 0.90, 6.3

Count: 74,128 / 74,170

Platform: Why not a web application?

Web app benefits:

- Very easy for users to start up

Client-server issues:

- scalability (requires centralised resources, scaling with users)
- availability (only works online, when services are running)
- implementation complexity (client-server coordination complicates things)

Implementation issues:

- Browser (+version) dependency

GUI issues:

- lots of windows - doesn't fit well into a browser

Sandboxing issues:

- local data access
- (no `mmap`)