Making the Sky Searchable: Fast Geometric Hashing for Automated Astrometry

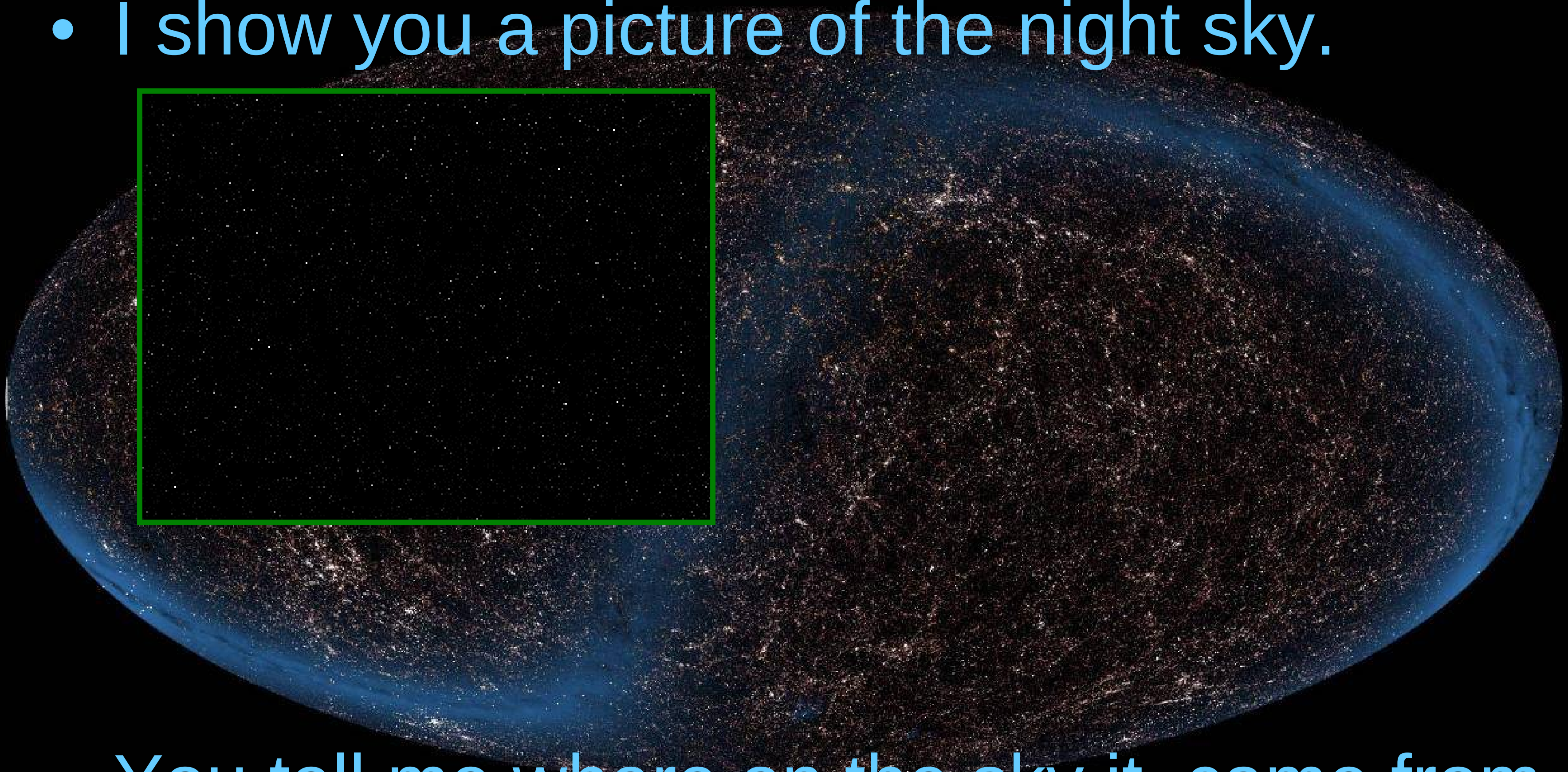
Sam Roweis, Dustin Lang & Keir Mierle
University of Toronto

David Hogg & Michael Blanton
New York University

roweis@cs.toronto.edu

Basic Problem

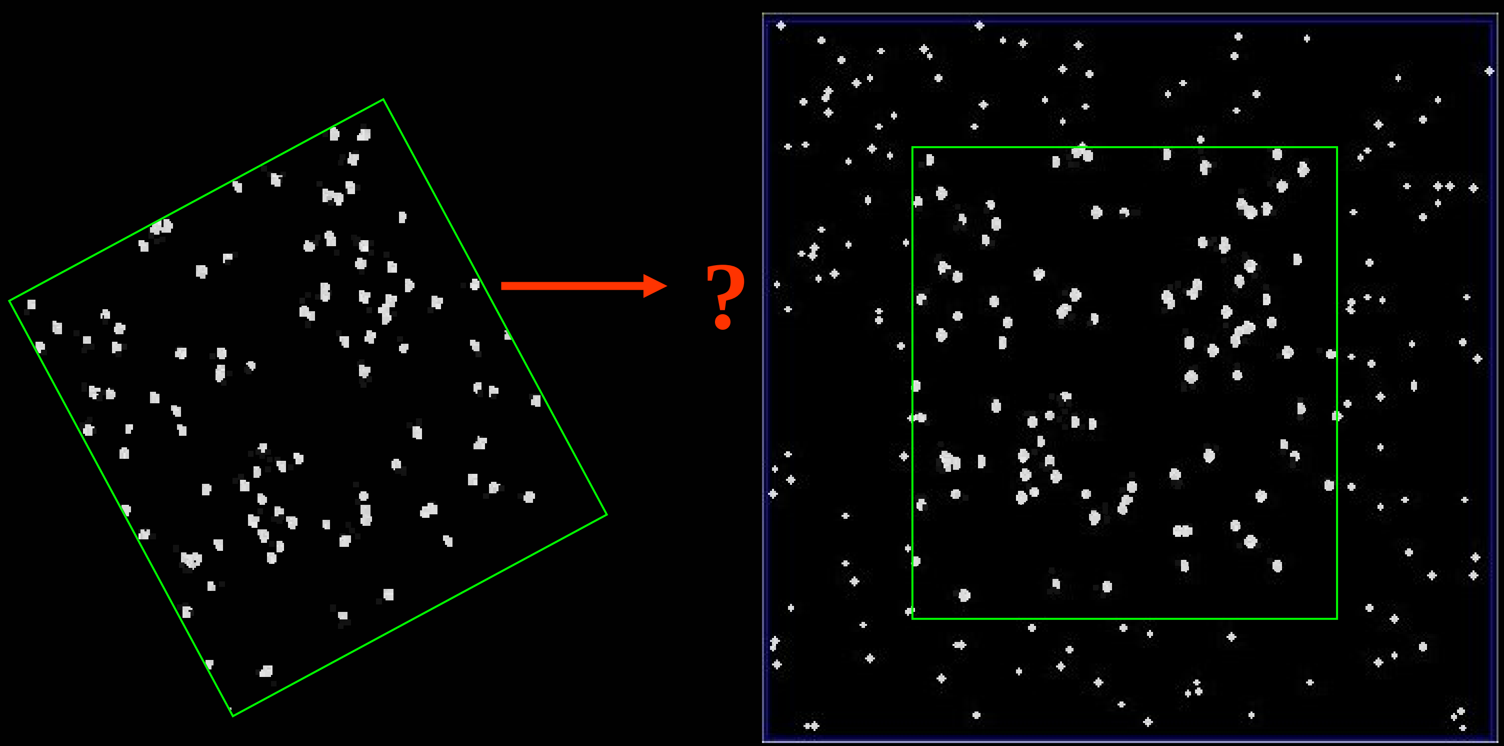
- I show you a picture of the night sky.



- You tell me where on the sky it came from.

Rules of the game

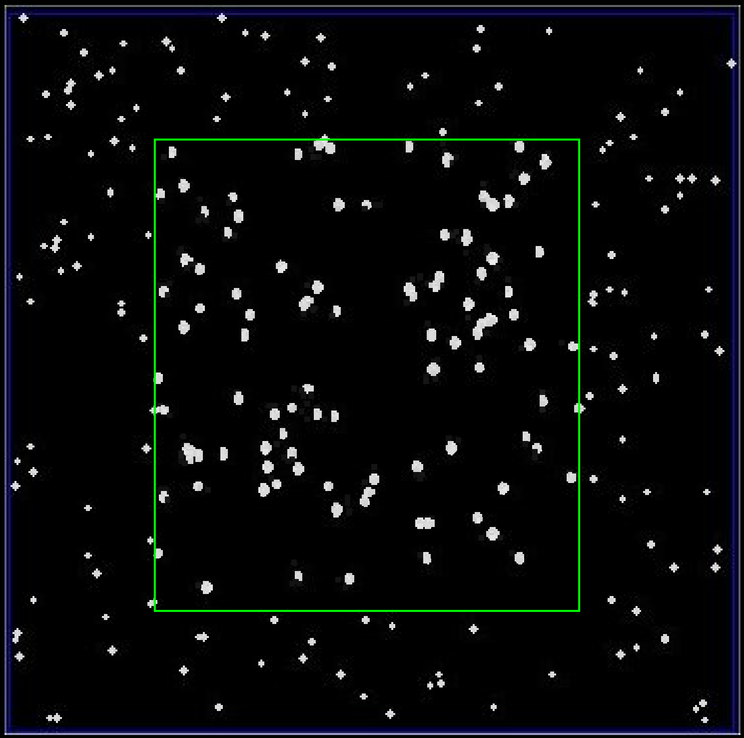
- We start with a catalogue of stars in the sky, and from it build an index which is used to assist us in locating ('solving') new test images.



Rules of the game

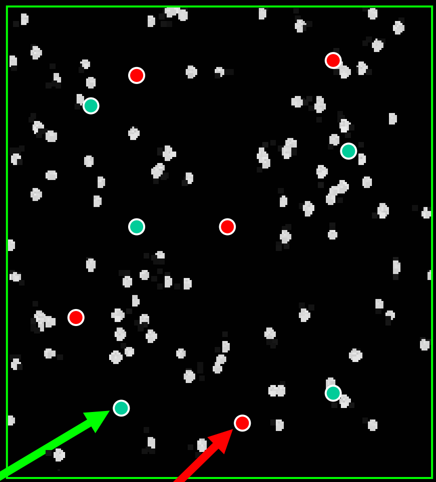
- We start with a catalogue of stars in the sky, and from it build an index which is used to assist us in locating ('solving') new test images.

- We can spend as much time as we want building the index but solving should be fast.
- Challenges:
  1) The sky is big.
  2) Both catalogues and pictures are noisy.



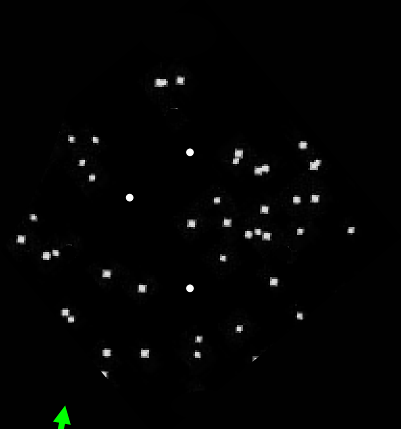
Distractors and Dropouts

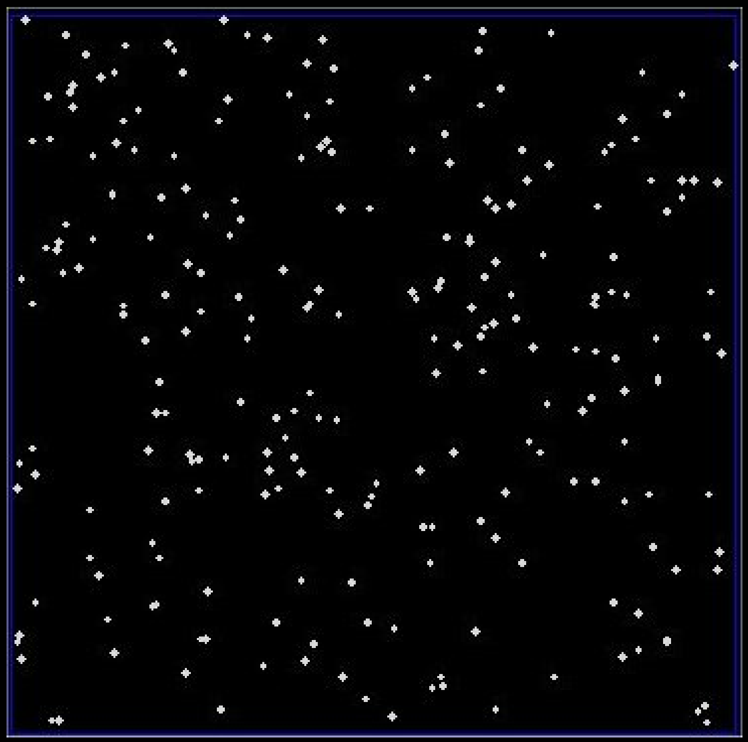
- Bad news:
  Query images may contain some extra stars that are not in your index catalogue, and some catalogue stars may be missing from the image.



- These "distractors" & "dropouts" mean that naïve matching techniques will not work.

You try

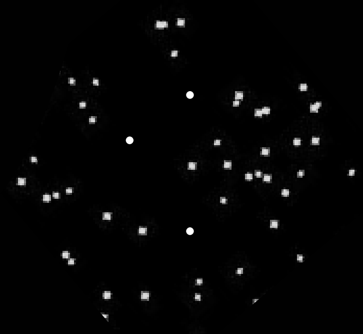


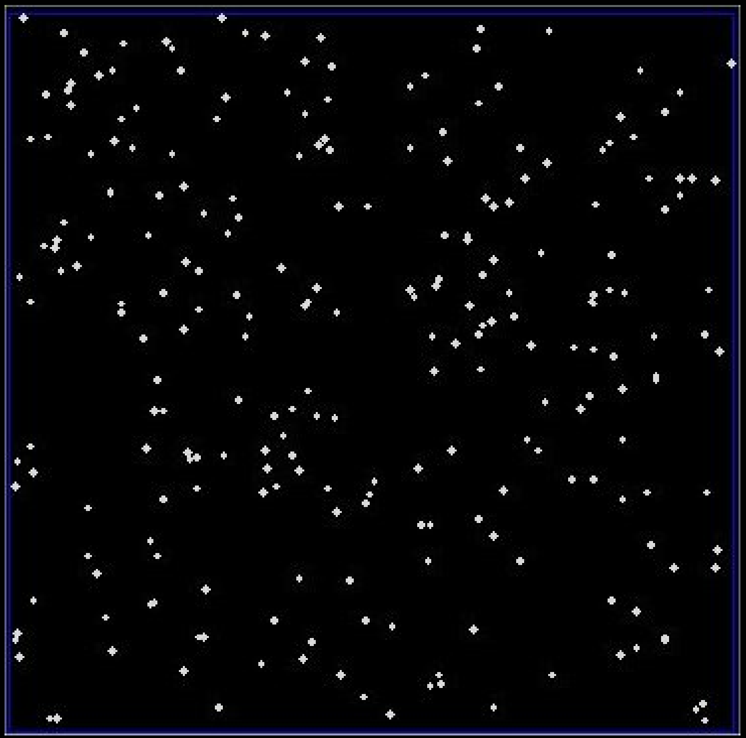


Find this "field" on this "sky".

You try

Hint #1: Missing stars.

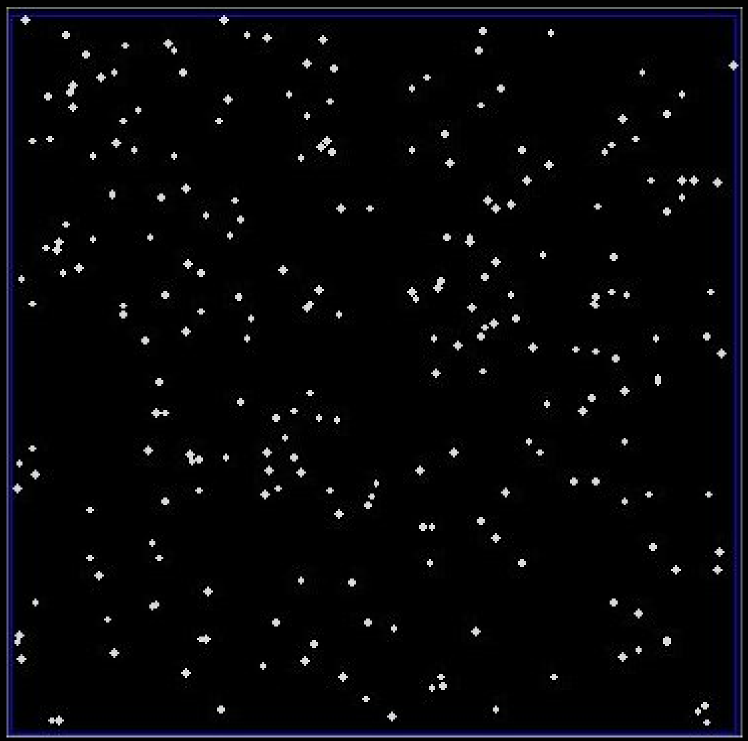




Find this "field" on this "sky".

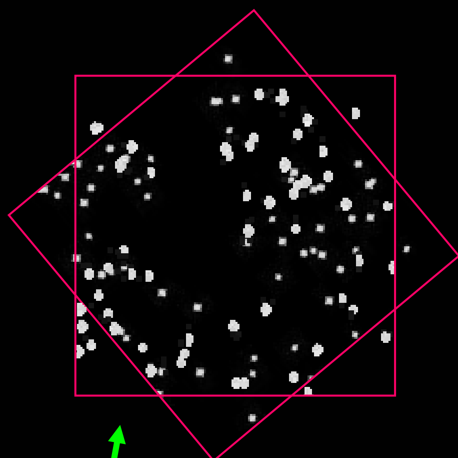
You try

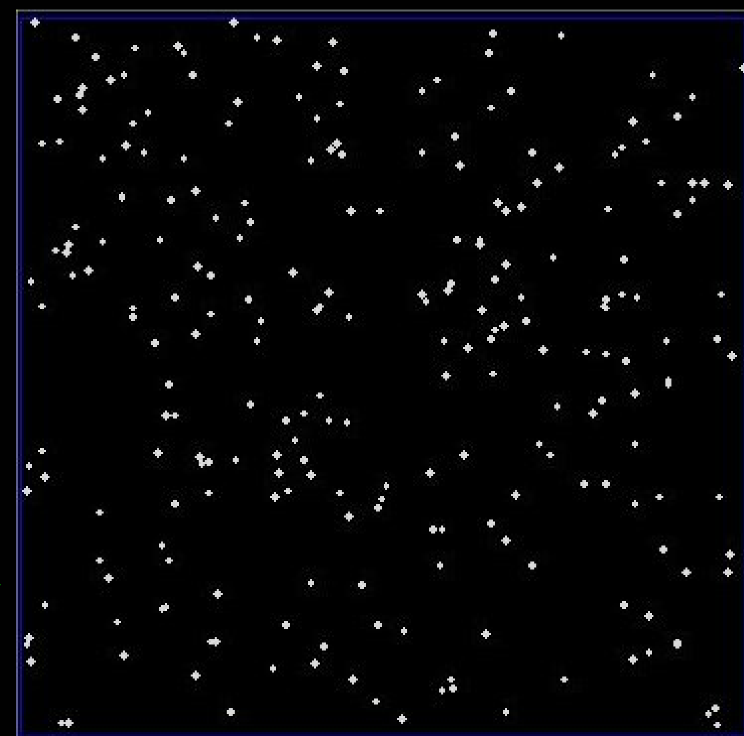
Hint #1: Missing stars.
Hint #2: Extra stars.



Find this “field” on this “sky”.

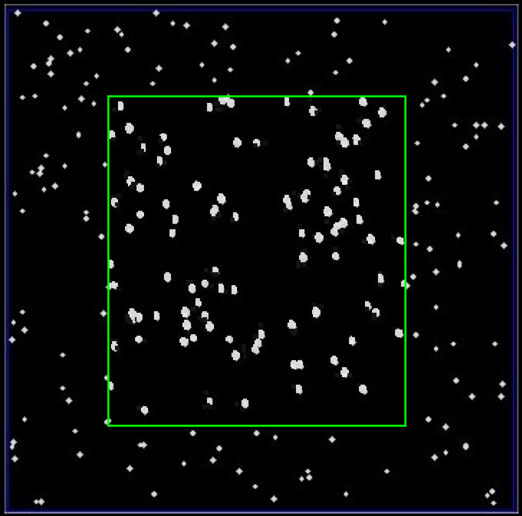
You try





Find this “field” on this “sky”.

Robust Matching

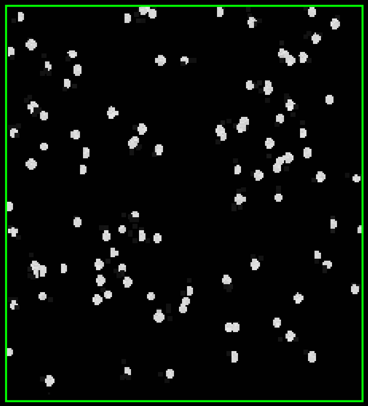


- We need to do some sort of robust matching of the test image to any proposed location on the sky.
- Intuitively, we need to ask: *"Is there an alignment of the test image and the catalogue so that (almost*) every catalogue star in the field of view of the test image lies (almost*) exactly on top of an observed star?"*

[*The details depend on the rate of distractors/dropouts.]

Solving the search problem

- Even if we can succeed in finding a good robust matching algorithm, there is still a huge search problem.
- Which proposed location should we match to?

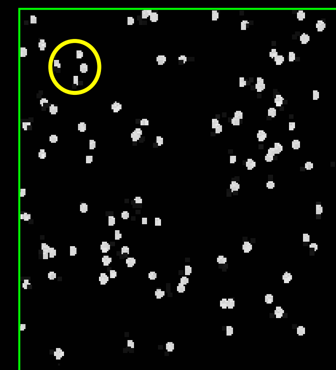


- ~~Exhaustive search?~~

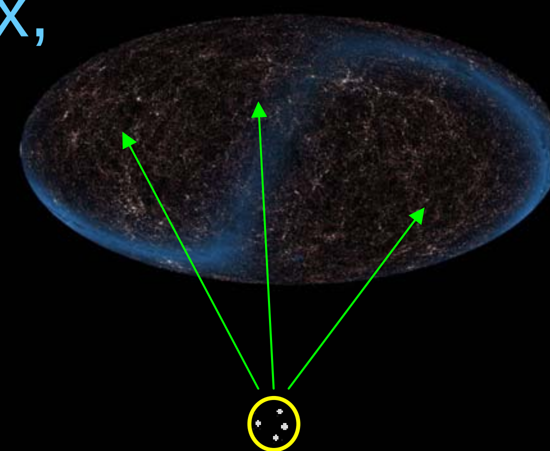
too expensive!

The Sky is Big™

(Inverted) Index of Features



- To solve this problem, we will employ the classic idea of an "inverted index".
- We define a set of "features" for any particular view of the sky (image).
- Then we make an (inverted) index, telling us which views on the sky exhibit certain (combinations of) feature values.

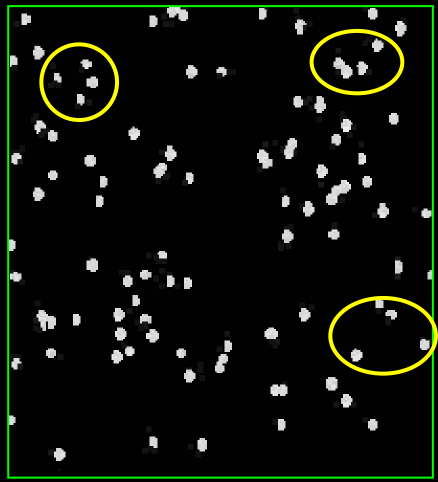


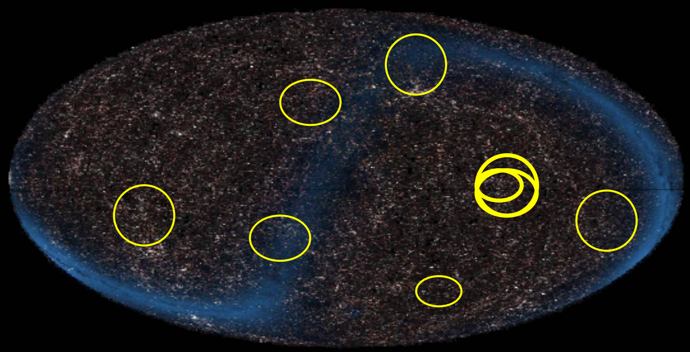
- This is like the question:
  Which web pages contain
  the words "machine learning"?

roweis@cs.toronto.edu

Matching a test image

- When we see a new test image, we compute which features are present, and use our inverted index to look up which possible views from the catalogue also have those feature values.
- Each feature generates a candidate list in this way, and by intersecting the lists we can zero in on the true matching view.





The features in our inverted index act as “hash codes” for locations on the sky.

roweis@cs.toronto.edu

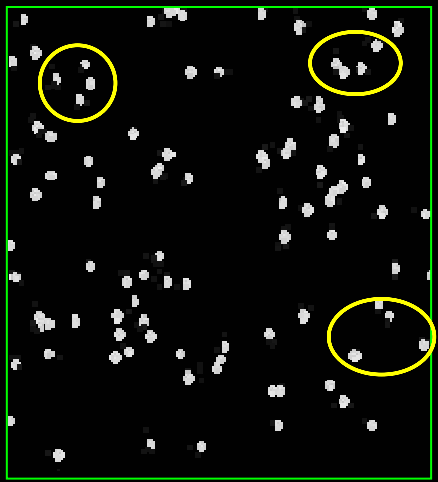
Caching Computation

- The idea of an inverted index is that is pushes the computation from search time back to index construction time.
- We actually do perform an exhaustive search of sorts, but it happens during the building of the inverted index and not at search time, so queries can still be fast.
- There are millions of patches of the scale of a test image on the sky (plus rotation), so we need to extract about 30 bits.

Robust Features for Geometric Hashing

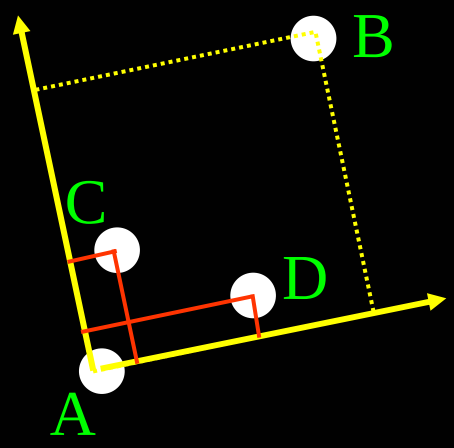
- In simple search domains like text, the inverted index idea can be applied directly.
- However, in our star matching task, the features we chose must be invariant to scale, rotation and translation.
- They must also be robust to small positional noise.
- Finally, there is the additional problem of distractor & dropout stars.

The features we use are the relative positions of nearby quadruples of stars.



Quads as Robust Features

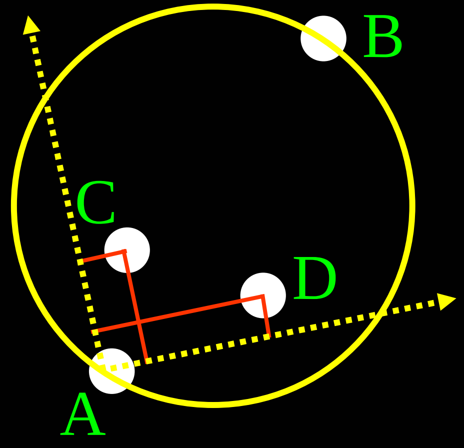
- We encode the relative positions of nearby quadruples of stars (ABCD) using a coordinate system defined by the most widely separated pair (AB).
- Within this coordinate system, the positions of the remaining two stars form a 4-dimensional code for the shape of the quad.
- Swapping AB or CD does not change the shape but it does “reflect” the code, so there is some degeneracy.



 roweis@cs.toronto.edu

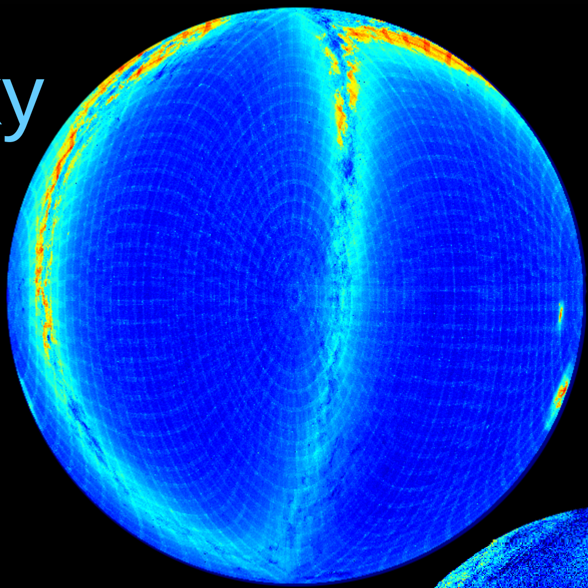
Quads as Robust Features

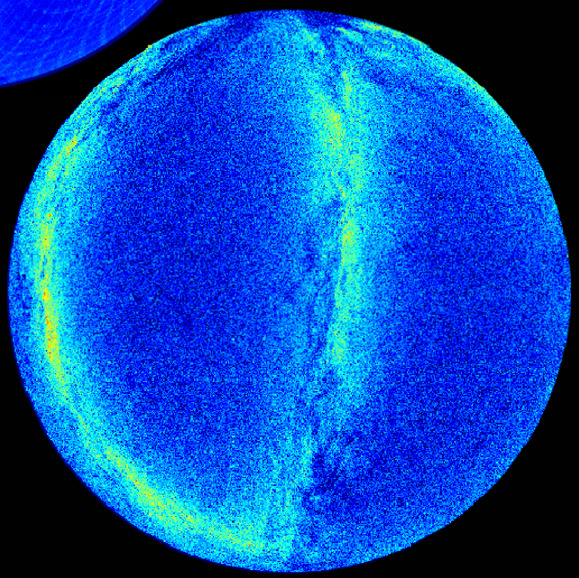
- This geometric hash code is invariant to scale, translation and rotation.
- It also has the property that if stars are uniformly distributed in space, codes are uniformly distributed in 4D.
- We compute codes for most nearby quadruples of stars, but not all; we require C&D to lie in the unit circle with diameter AB.



Catalogues: USNO-B 1.0 + TYCHO-2

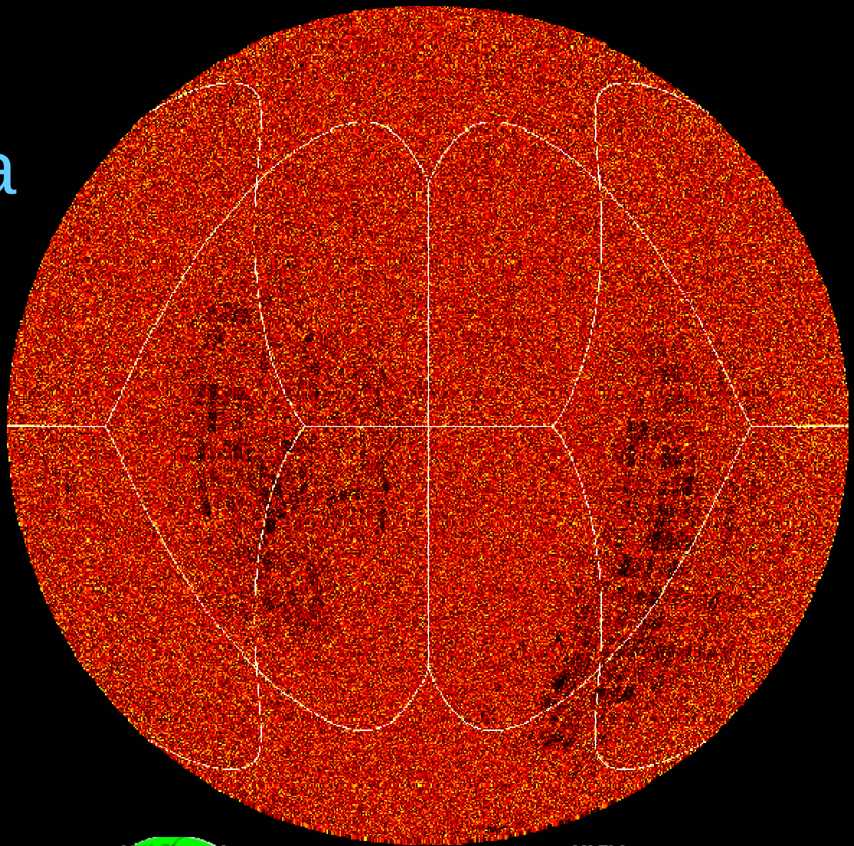
- USNO-B is an all-sky catalogue compiled from scans of old Schmidt plates. Contains about 10^9 objects, both stars and galaxies.
- TYCHO-2 is a tiny subset of 2.5M brightest stars.

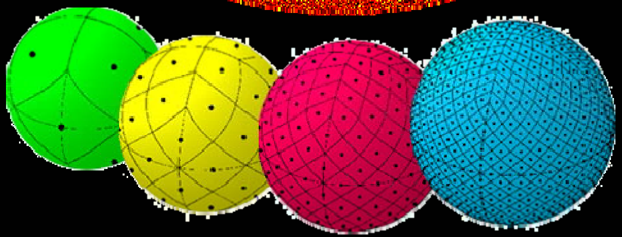




Making a uniform catalogue

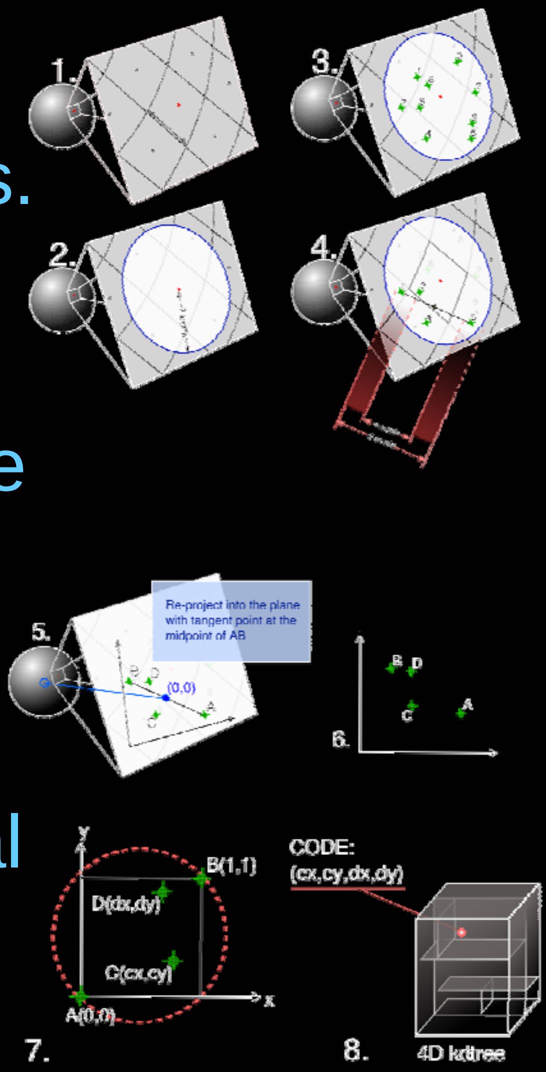
- Starting with USNO+ TYCHO we "cut" to get a spatially uniform set of the ~150M brightest stars & galaxies.
- We do this by laying down a fine "healpix" grid and taking the brightest K unique objects in each pixel.

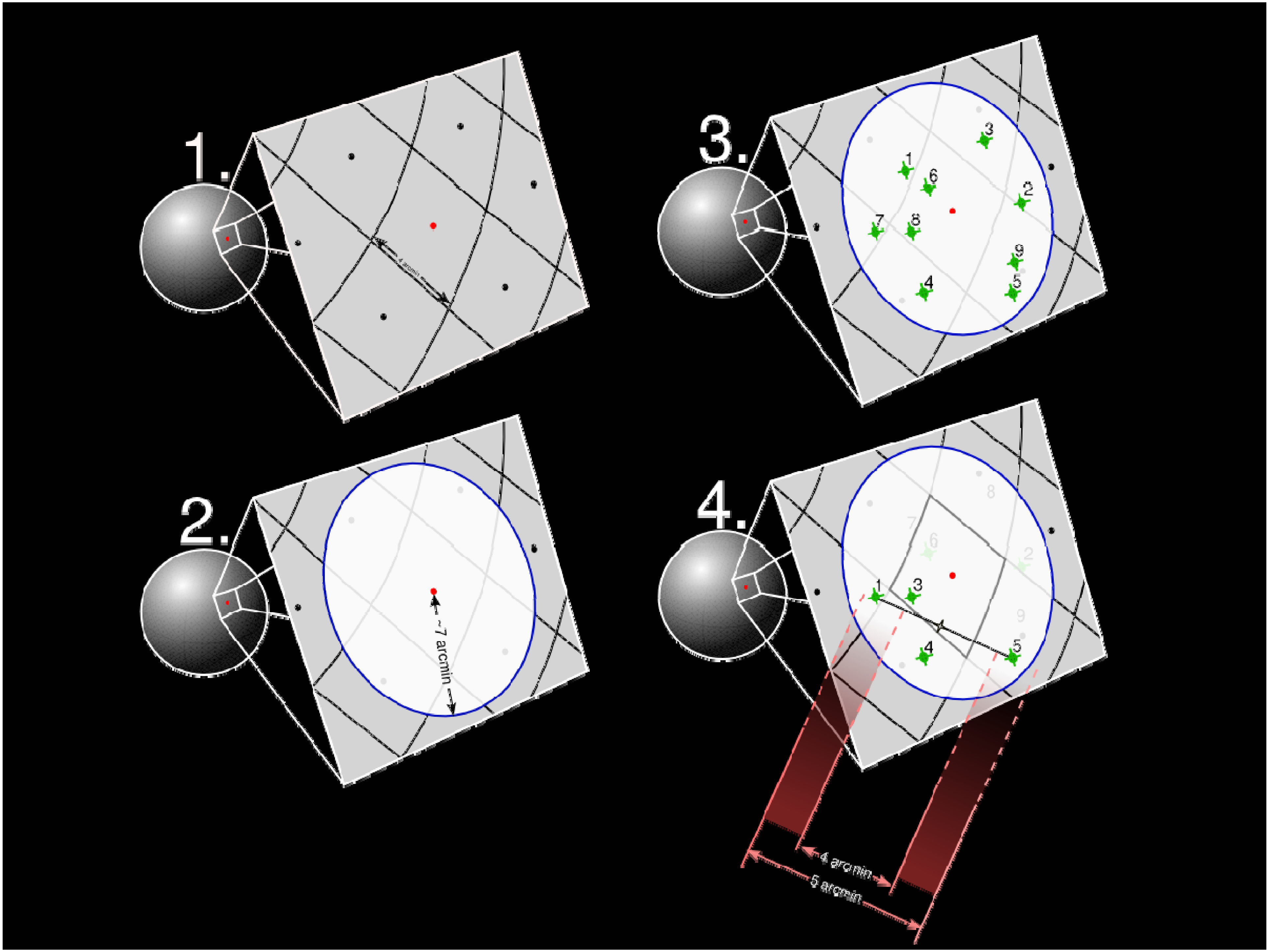




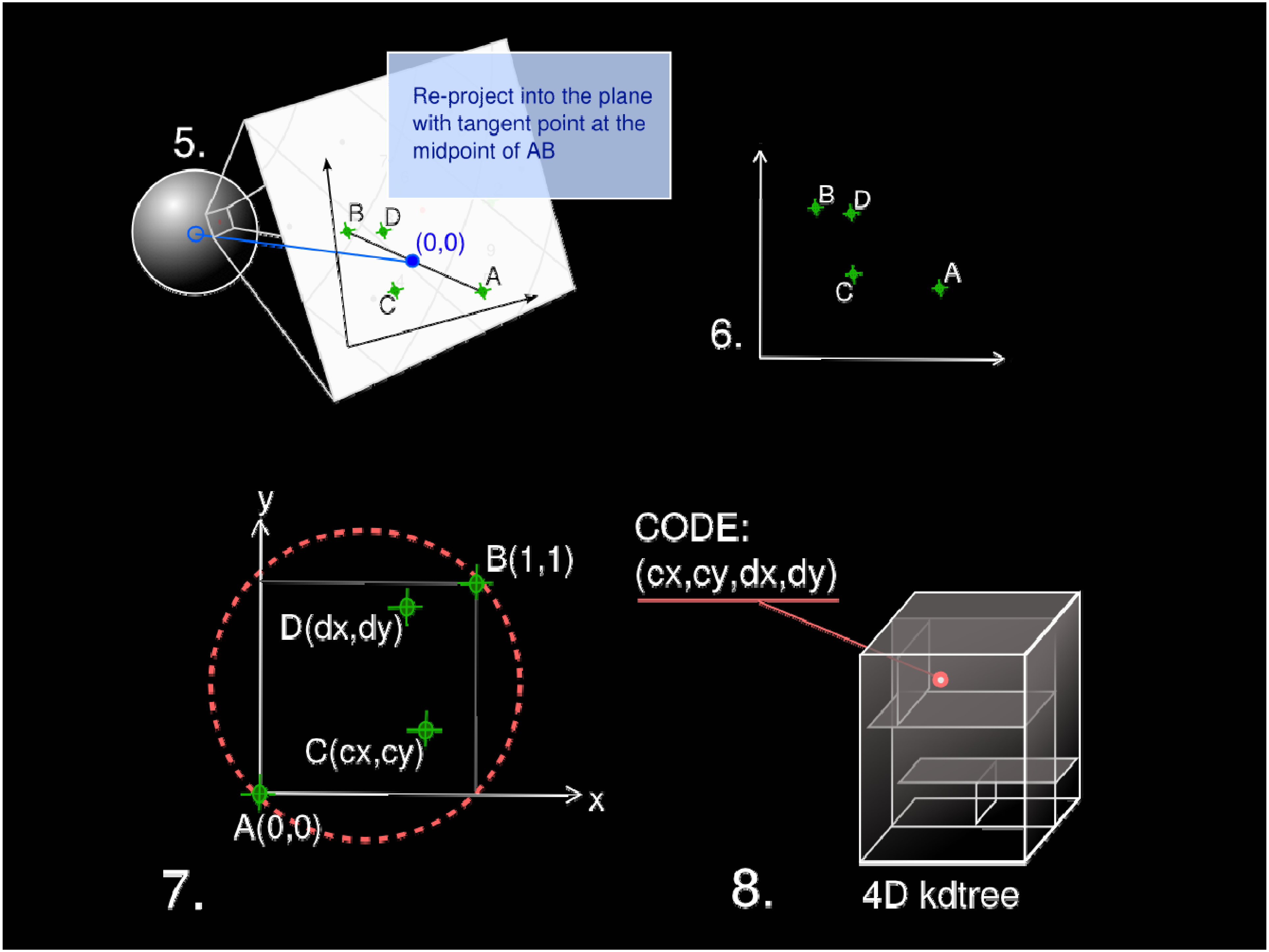
Building the index

- Start with the catalogue; build a kdtree on the 3D object positions.
- Place a fine healpix grid on the sky. Within each pixel, identify a valid quad whose size is near the target scale for the index.
- Compute 4D codes for those quads; enter them into another kdtree remembering their original locations. This is the index.



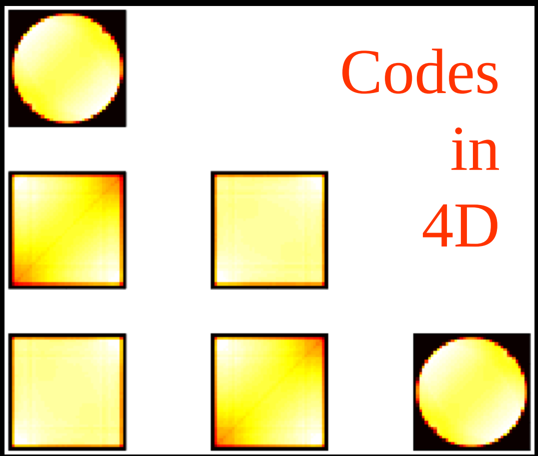


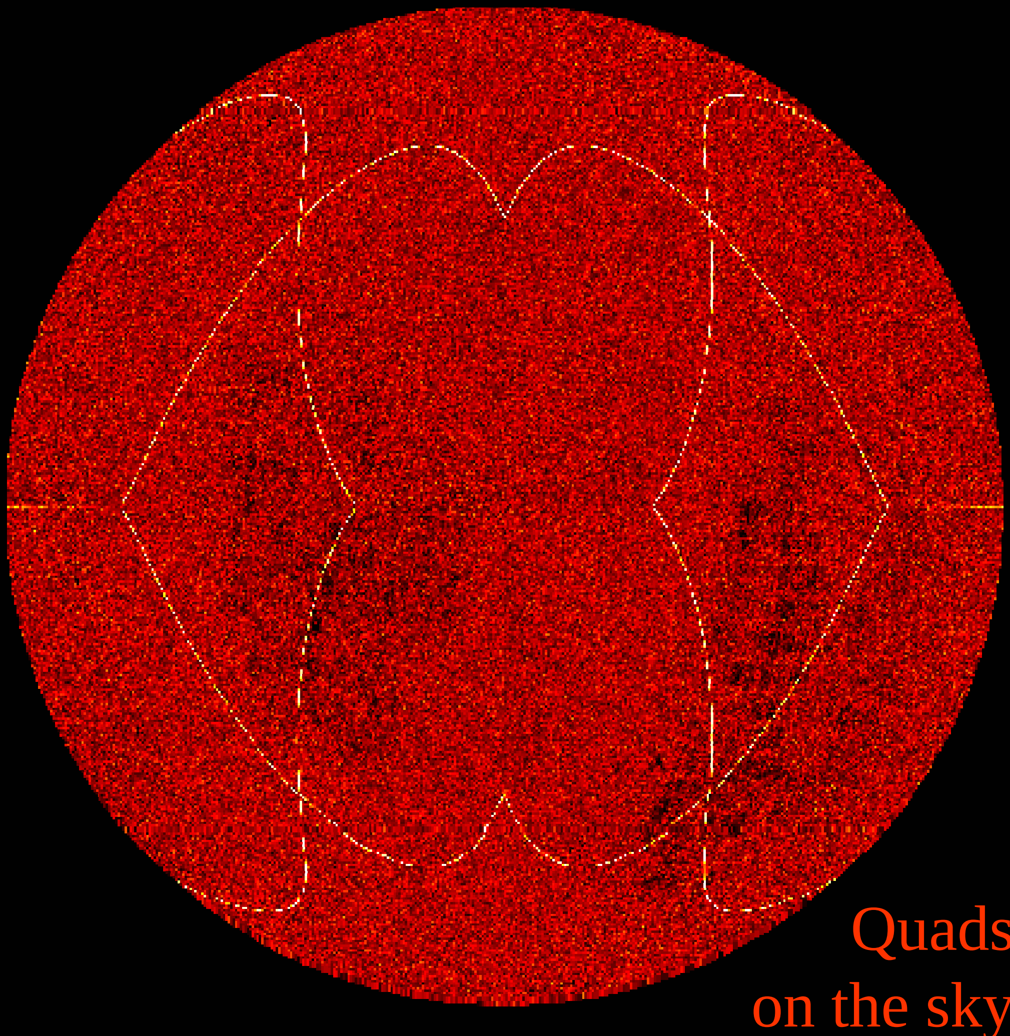
1.
2.
3.
4.
~7 arcmin
4 arcmin
5 arcmin


5.
Re-project into the plane with tangent point at the midpoint of AB
B
D
(0,0)
A
C
6.
B
D
C
A
y
B(1,1)
D(dx,dy)
C(cx,cy)
A(0,0)
x
7.
CODE:
(cx,cy,dx,dy)
8.
4D kdtree

A Typical Final Index

- 144M stars (6 quads/star)
- 205M quads (4-5 arcmin)
- 12 healpixes





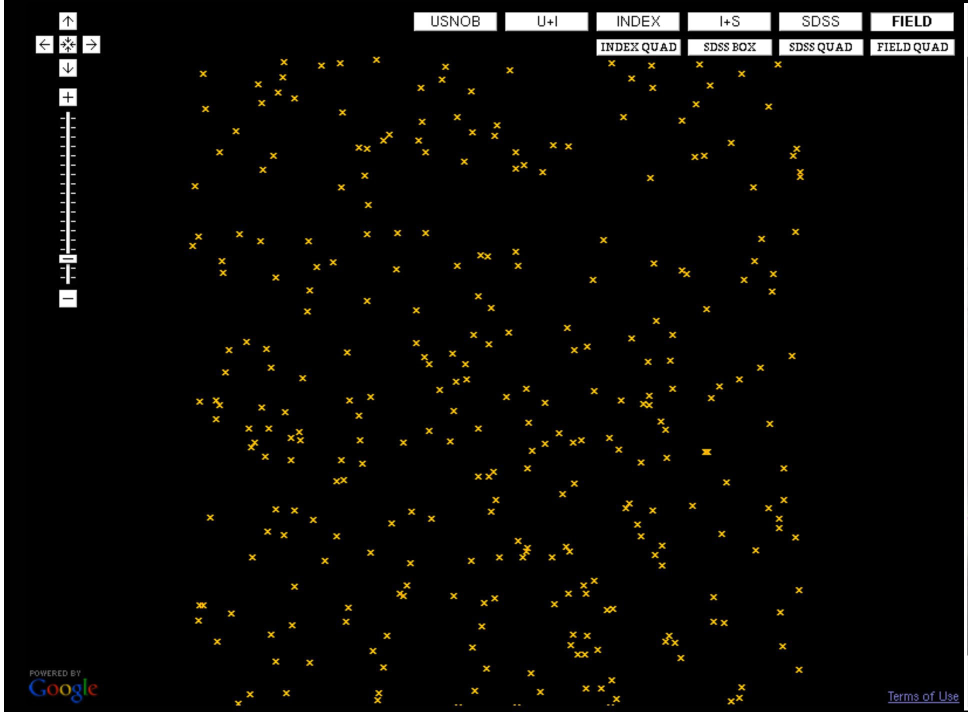
roweis@cs.toronto.edu

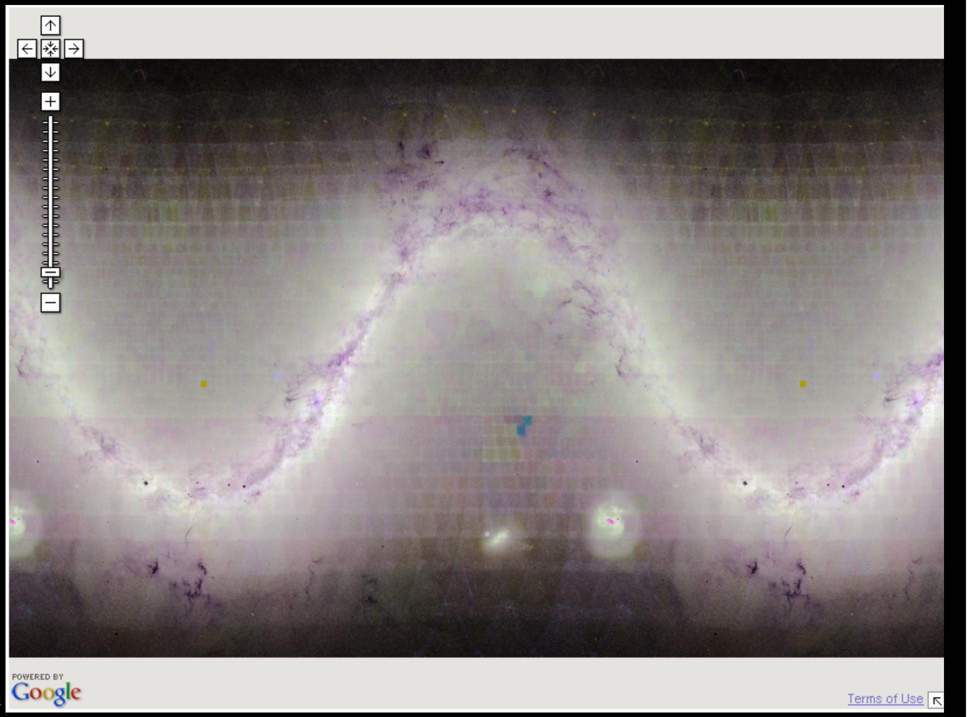
Solving a new test image

- Identify objects (stars+galaxies) in the image bitmap and create a list of their 2D positions.
- Cycle through all possible valid* quads (brightest first) and compute their corresponding codes.
- Look up the codes in the code KD-tree to find matches within some tolerance; this stage incurs some false positive and false negative matches.
- Each code match returns a candidate position & rotation on the sky. As soon as 2 quads agree on a candidate, we proceed to verify that candidate against all objects in the image.

roweis@cs.toronto.edu

A Real Example from SDSS

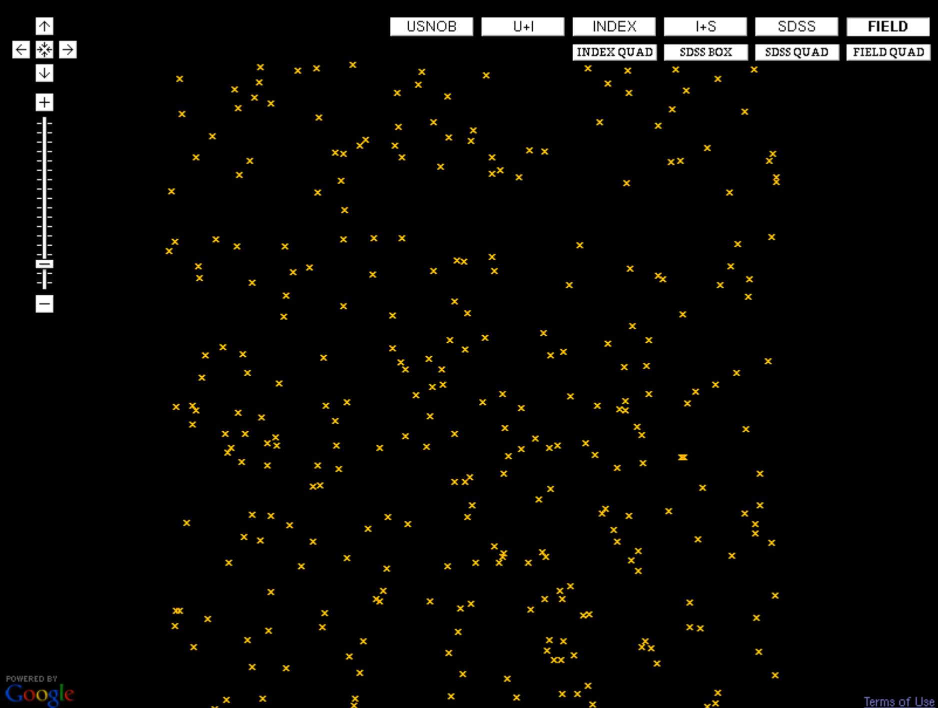


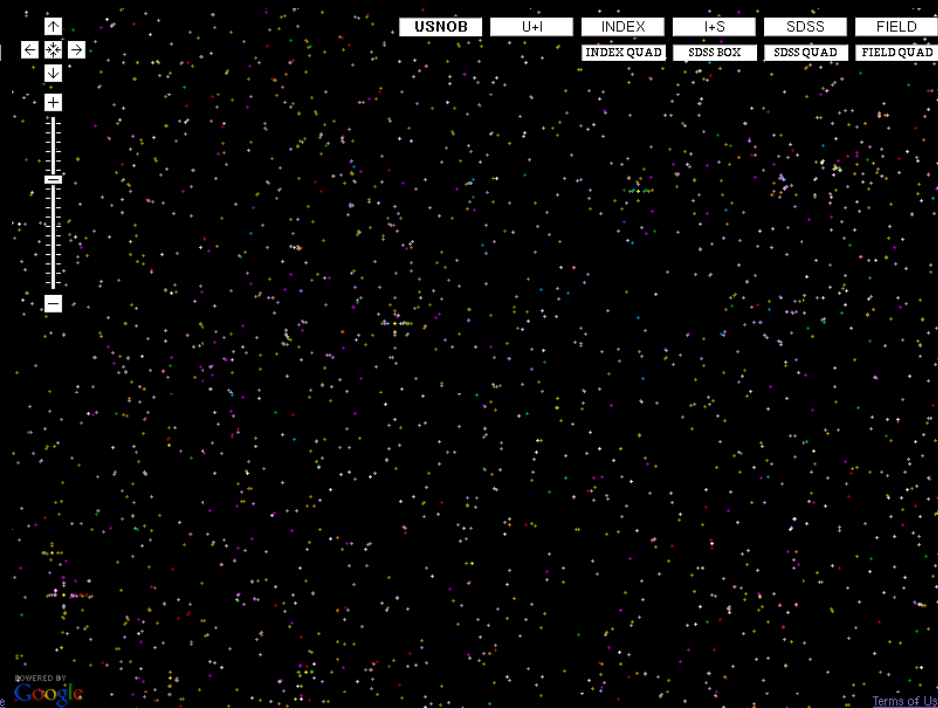


Query image
(after object detection).

An all-sky catalogue.

A Real Example from SDSS

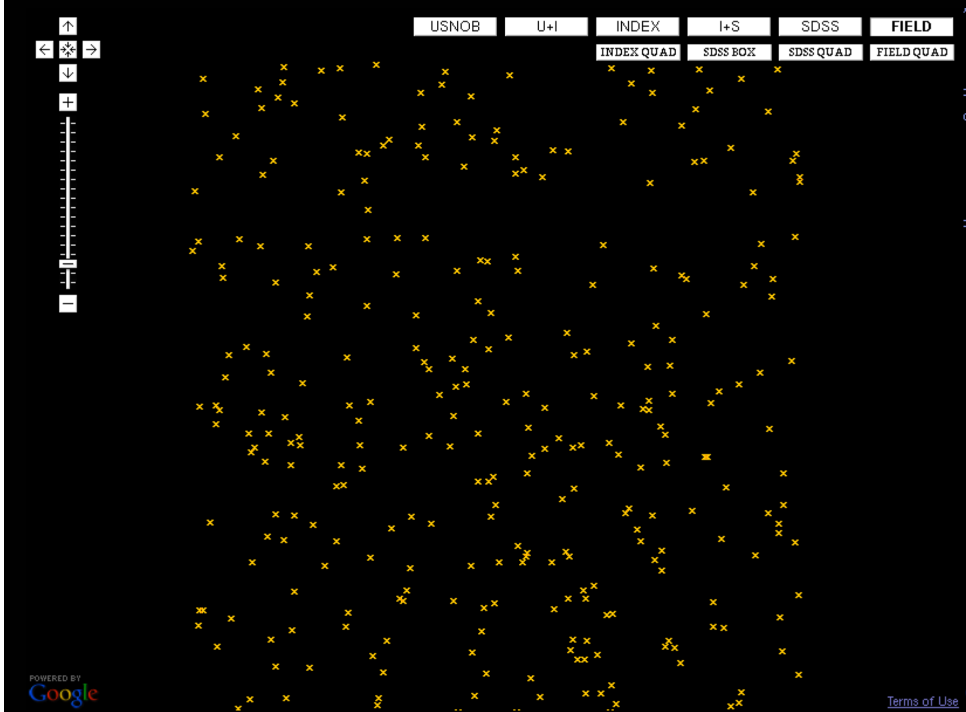


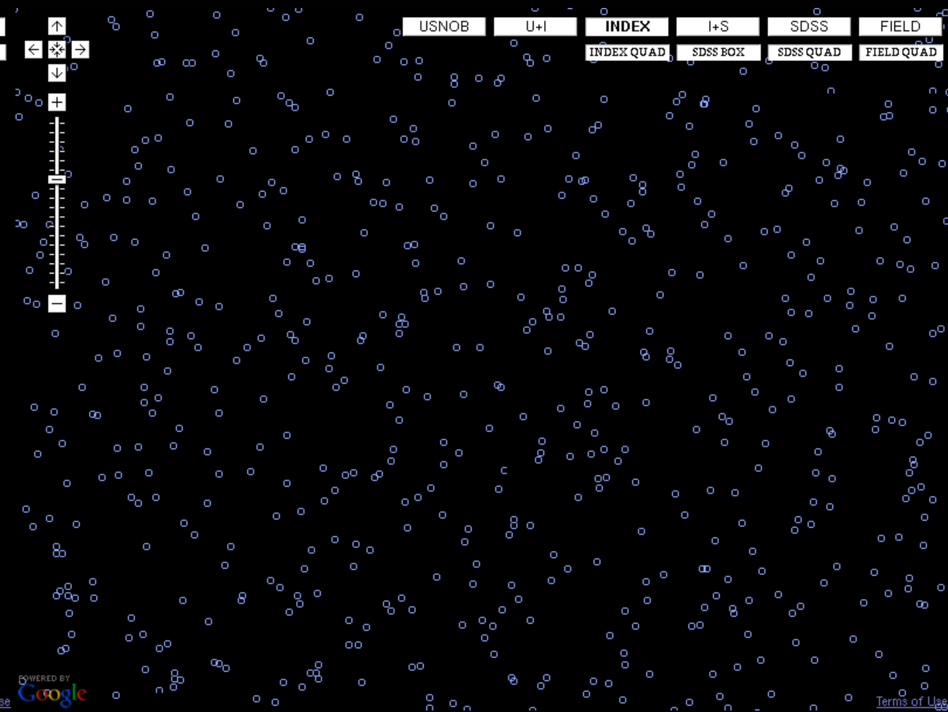


Query image
(after object detection).

Zoomed in by a
factor of ~ 1 million.

A Real Example from SDSS

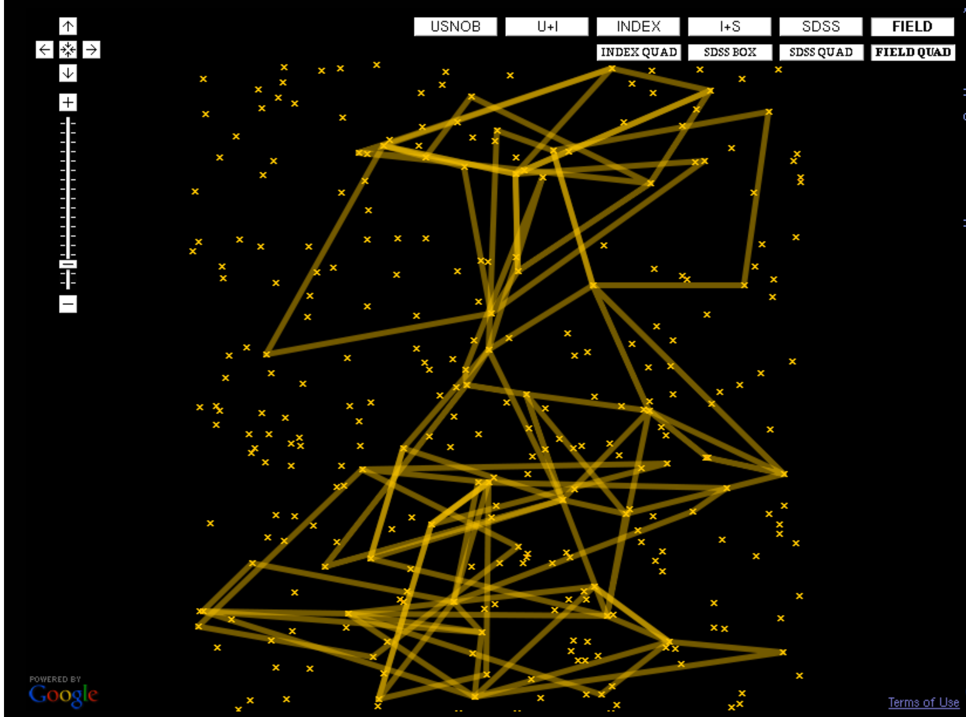


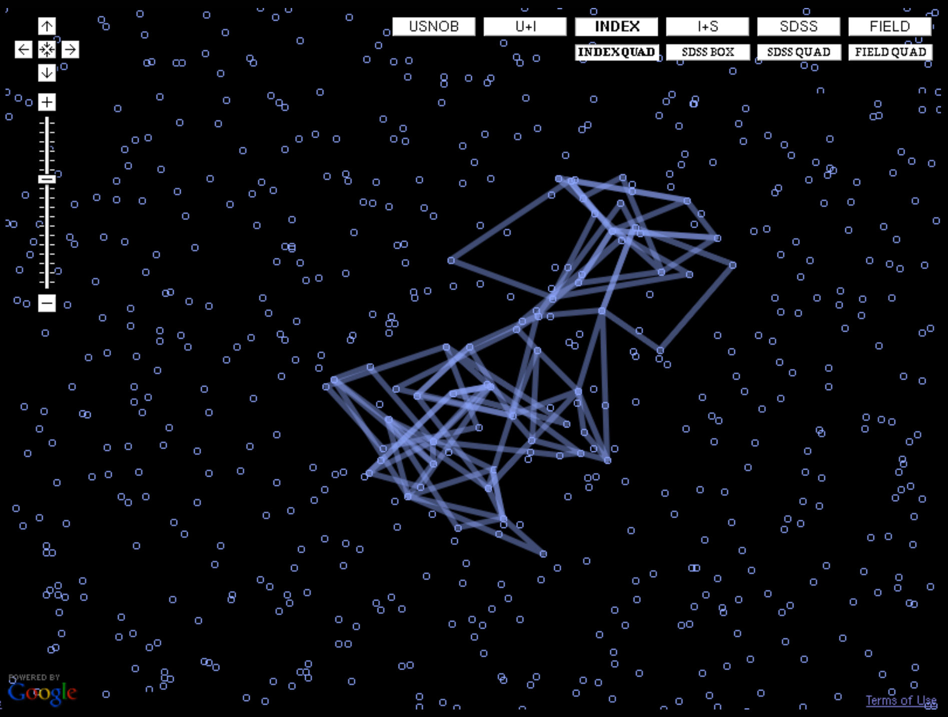


Query image
(after object detection).

The objects in our index.

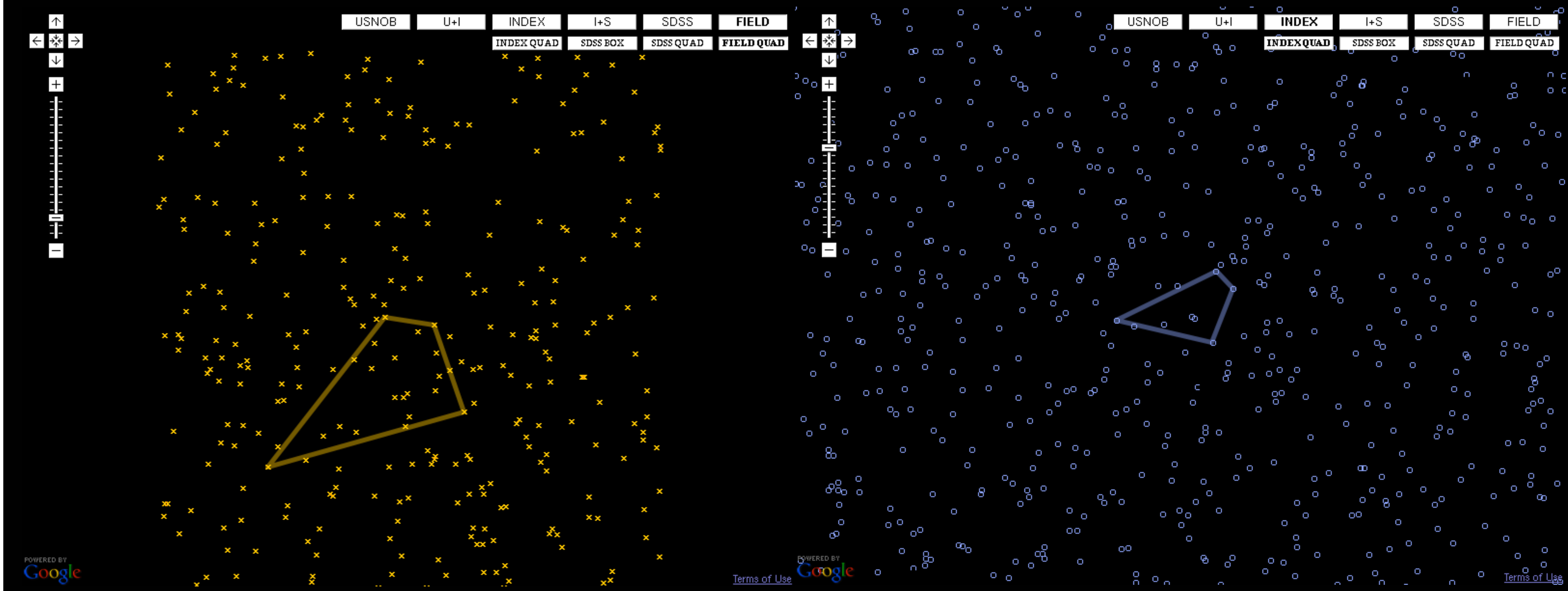
A Real Example from SDSS





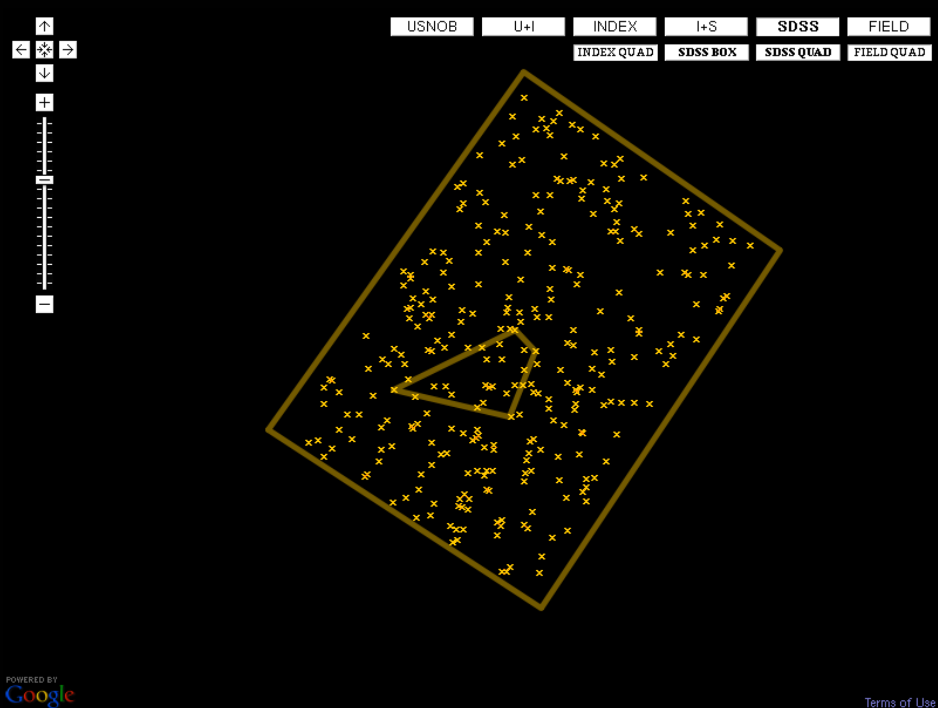
All the quads in our index which are present in the query image.

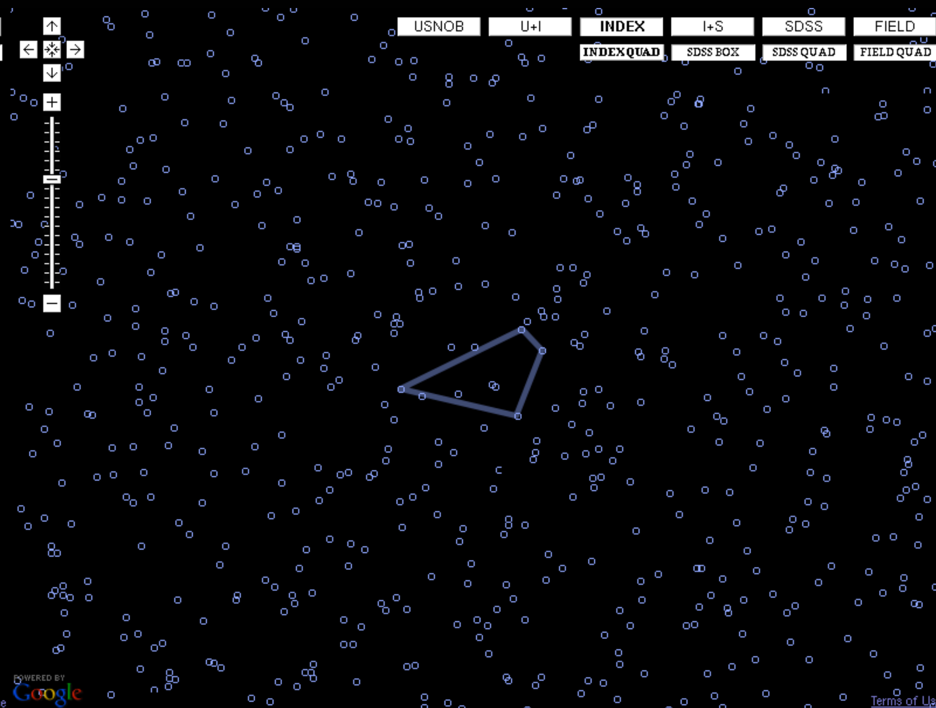
A Real Example from SDSS



A single quad which we happened to try.

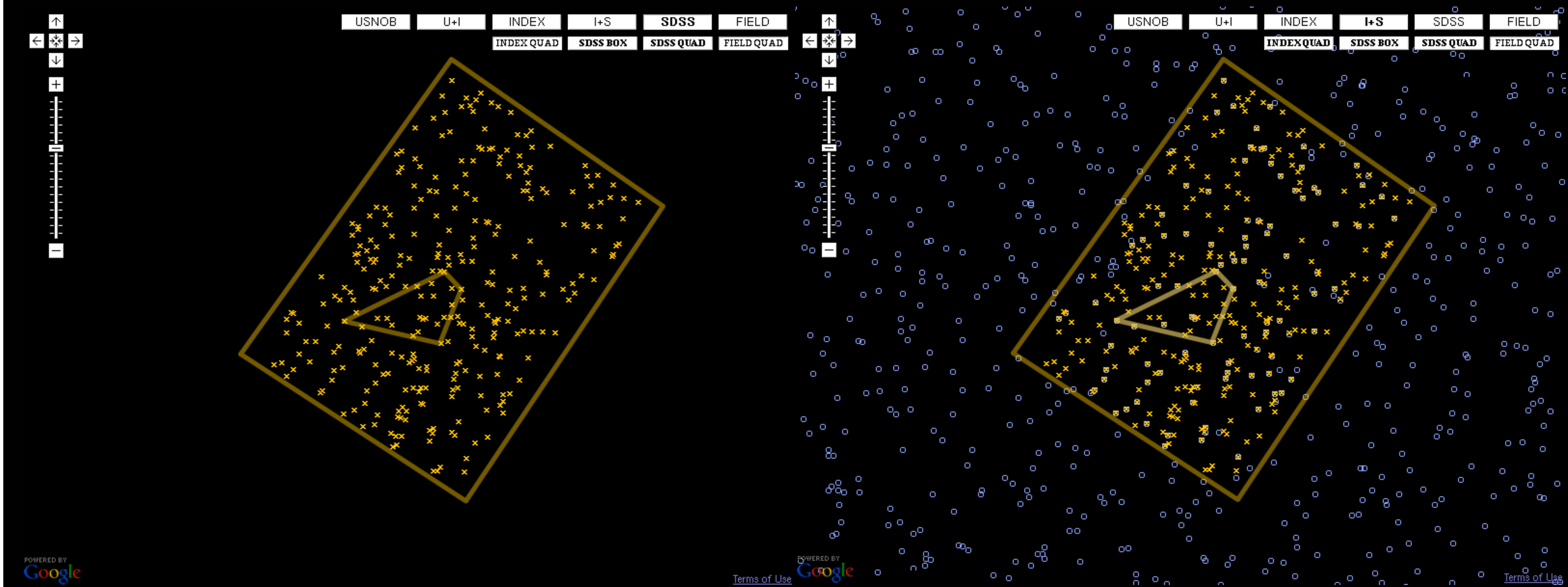
A Real Example from SDSS





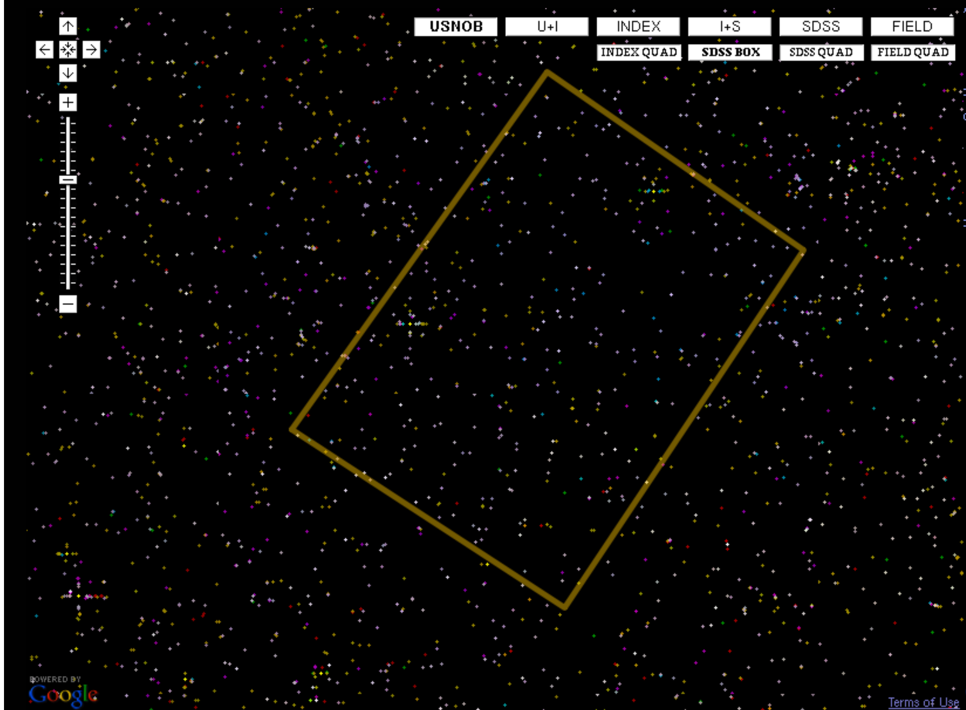
The query image scaled, translated & rotated as specified by the quad.

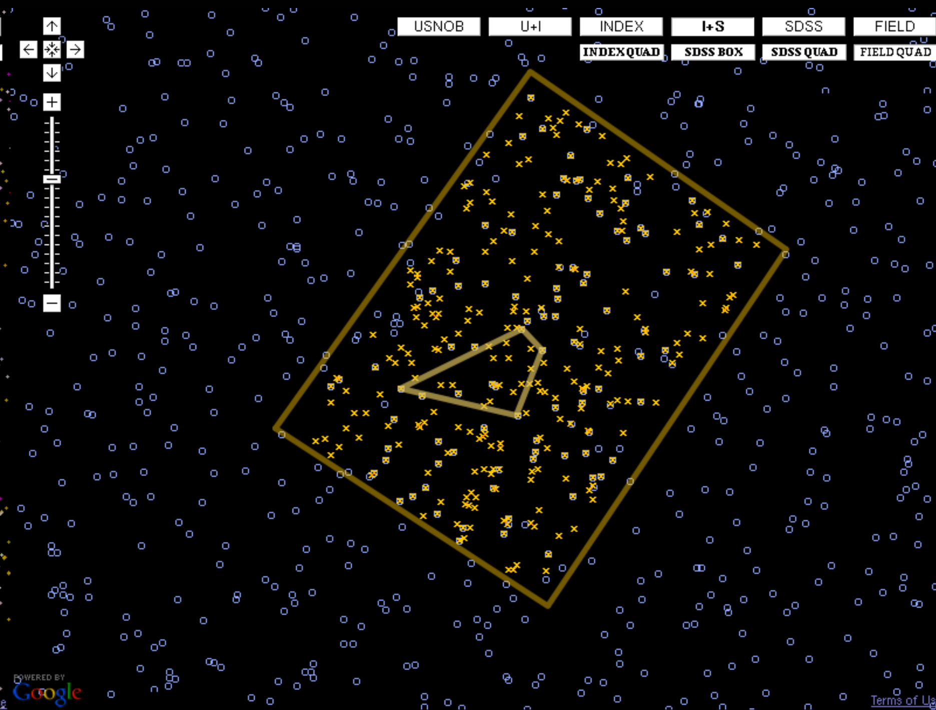
A Real Example from SDSS



The proposed match, on which we run verification.

A Real Example from SDSS



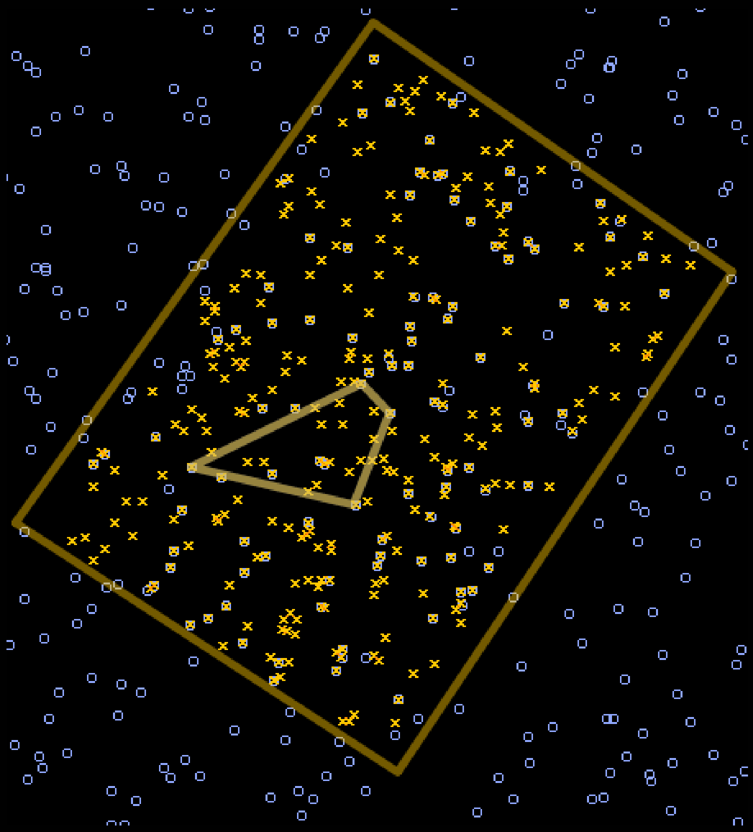


The verified answer, overlaid on the original catalogue.

The proposed match, on which we run verification.

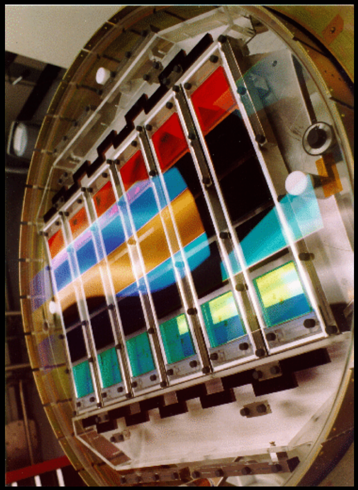
Final Verification

- After hash code matching, we are left with a list of candidate views that >1 codes agree on.
- If this list is empty, the search has failed.
- If this list is non-empty, we do a slower positional verification on each candidate to see if it really is the correct position in the catalogue.



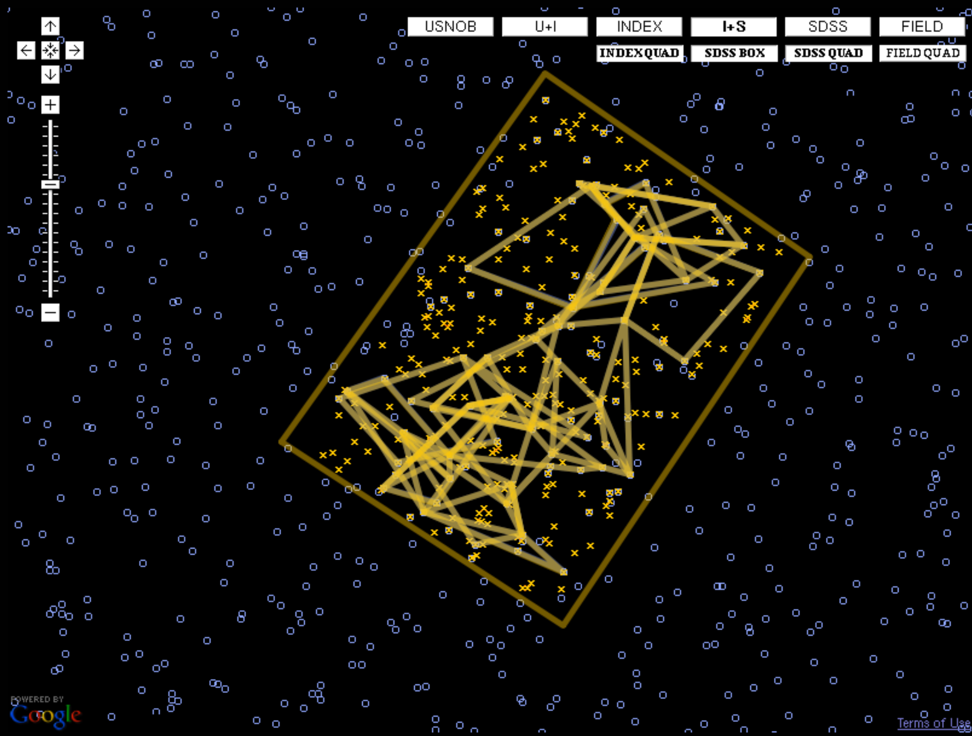
Preliminary Results: SDSS

- The Sloan Digital Sky Survey (SDSS) is an all-sky, multi-band survey which includes targeted spectroscopy of interesting objects.
- The telescope is located at Apache Point Observatory.
- Fields are 14x9arcmin corresponding to 2048x1361 pixels.



Preliminary Results: SDSS

- 336,554 fields science grade+
- 0 false positives
- 99.84% solved 530 unsolved
- 99.27% solve w/ 60 brightest objs



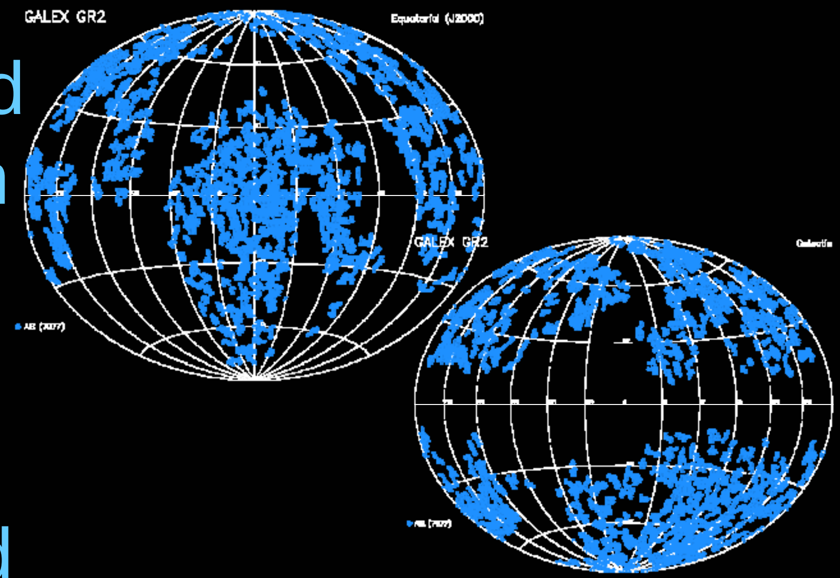
Magnitudes used only to decide search order.

Assume known pixel scale (for speedup of solving only.)

roweis@cs.toronto.edu

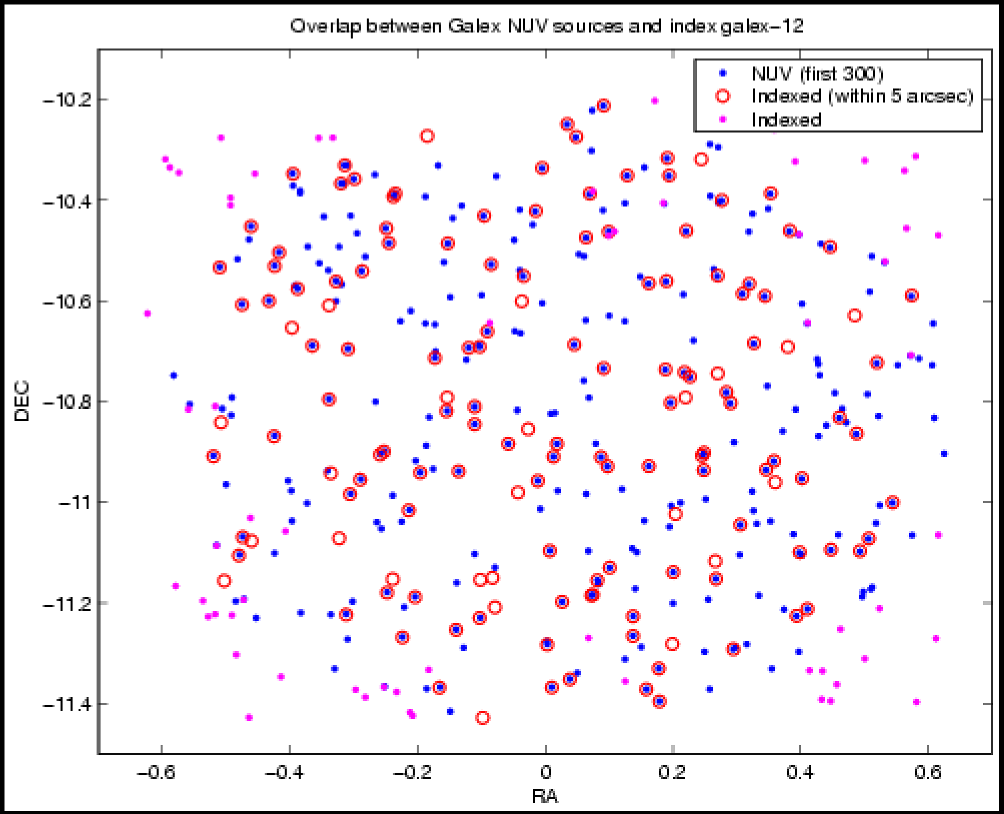
Preliminary Results: GALEX

- GALEX is a space-based telescope, seeing only in the ultraviolet.
- It was launched in April 2003 by Caltech&NASA and is just about finished collecting data now.
- It takes huge (80 arcmin) circular fields with 5arcsec resolution and spectra of all objects.



Preliminary Results: GALEX

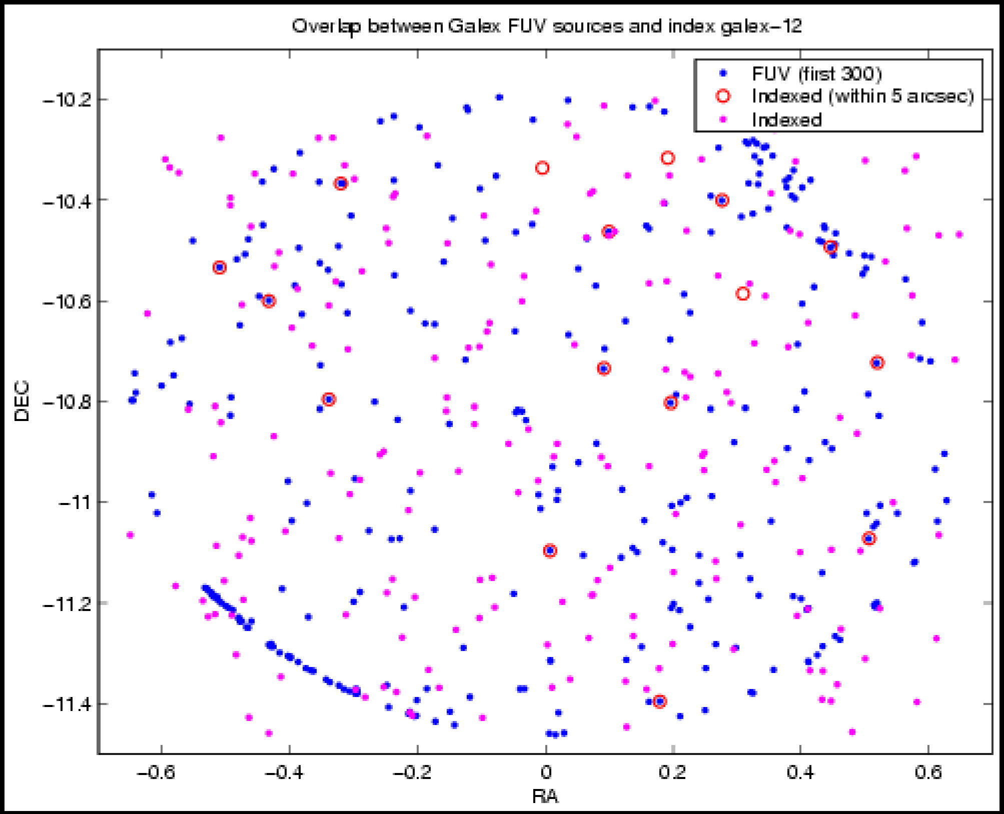
- GALEX NUV fields can be solved easily using an index built from bright blue USNO stars.



 roweis@cs.toronto.edu

Preliminary Results: GALEX

- GALEX FUV fields are much harder to solve using USNO as a source catalogue.

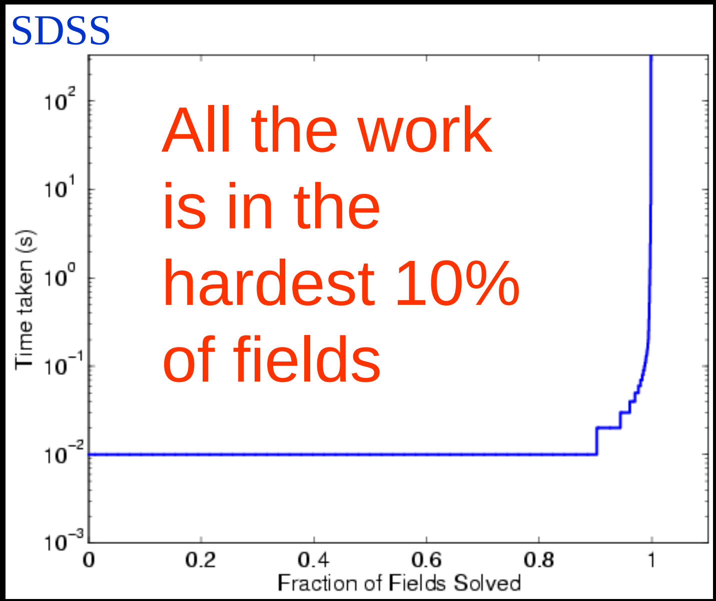


Frequency band(s) of the test images must have some substantial overlap with those of the catalogue.

roweis@cs.toronto.edu

Speed/Memory/Disk

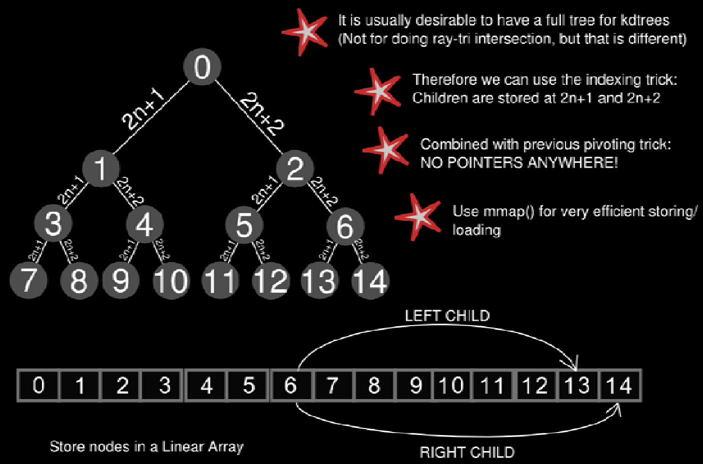
- Indexing takes ~12 hours, uses ~ 2 GB of memory and ~100 GB of disk.
- Solving a test image almost always takes <<1sec (not including object detection).
- Solving many fields is done by coarse parallelization on about 100 shared CPUs.



Reduces computation time from ~ 4months to overnight.

Algorithms & Data Structures

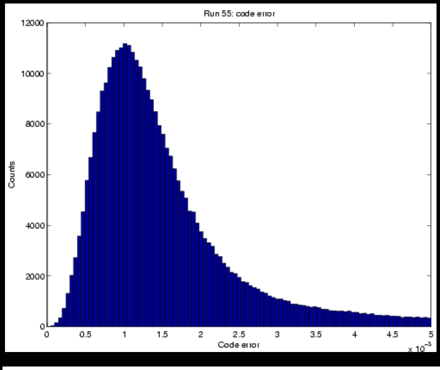
- Implementations are all in-core.
- Written in C & Python.
- Parallelization is at the script level, which has many aggregation & storage advantages.
- We make extensive use of mem-mapped files, some fancy AVL lists and a cool new "pointerless" KD-tree implementation. [Mierle & Lang]

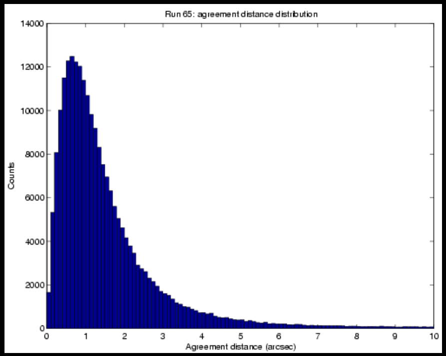


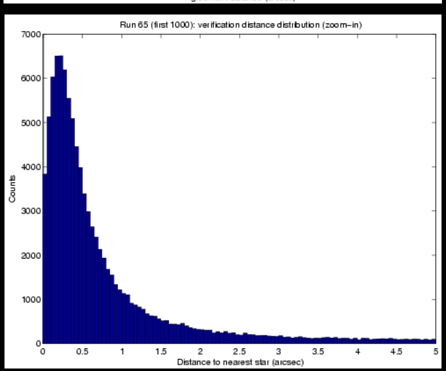
roweis@cs.toronto.edu

Setting the System Parameters

- There are several system parameters to tune, including range search sizes in code-space, agreement and verification tolerances on the sky, etc.
- Our approach has been to tune these by examining histograms of what happened across a large number of test cases where we know the ground truth.



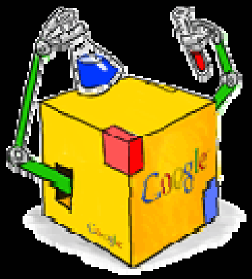




roweis@cs.toronto.edu

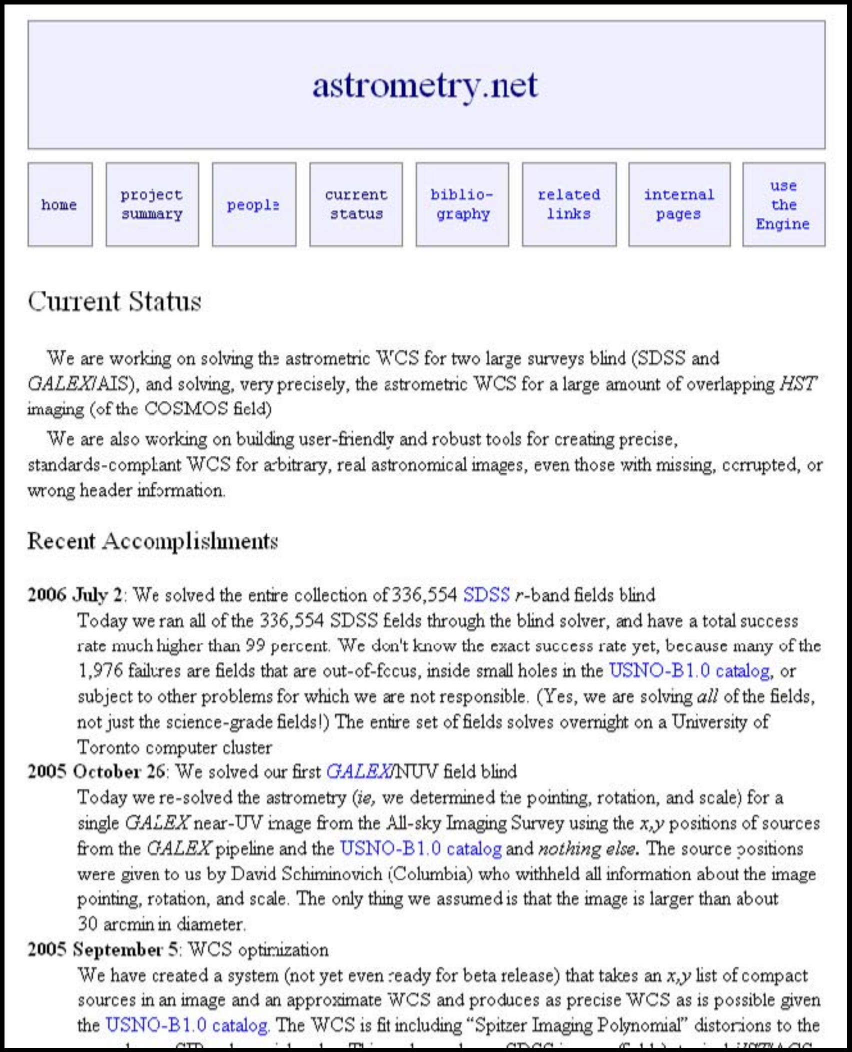
Googlers should love this!

- Massive indexing & pattern recognition.
- Coarsely parallel storage/processing.
- Cool algorithms & data structures.
- Organizes the sky's information and makes it searchable.



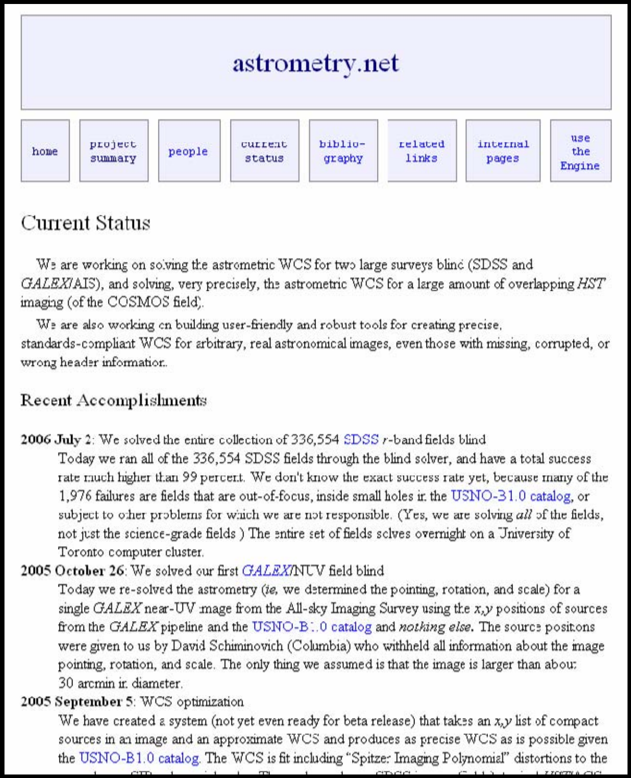
astrometry.net

- The project has a website, which should go “live” in a few weeks.
- It will allow any user to recover (or verify) the positional information in their image headers, label specific stars, automatically link into other surveys and more.



astrometry.net

- In the future, we plan to solve a wide range of images or image sets, using a variety of indexes.
- We also hope to insert the system into the observing pipeline of telescopes, debug standard catalogues, learn about individual instruments and facilitate “collaborative observing” tools.



astrometry.net

home | project summary | people | current status | biblio-graphy | related links | internal pages | use the Engine

Current Status

We are working on solving the astrometric WCS for two large surveys blind (SDSS and *GALEX*/AIS), and solving, very precisely, the astrometric WCS for a large amount of overlapping *HST* imaging (of the COSMOS field).

We are also working on building user-friendly and robust tools for creating precise, standards-compliant WCS for arbitrary, real astronomical images, even those with missing, corrupted, or wrong header information.

Recent Accomplishments

2006 July 2: We solved the entire collection of 336,554 SDSS *r*-band fields blind

Today we ran all of the 336,554 SDSS fields through the blind solver, and have a total success rate much higher than 99 percent. We don't know the exact success rate yet, because many of the 1,976 failures are fields that are out-of-focus, inside small holes in the USNO-B1.0 catalog, or subject to other problems for which we are not responsible. (Yes, we are solving *all* of the fields, not just the science-grade fields.) The entire set of fields solves overnight on a University of Toronto computer cluster.

2005 October 26: We solved our first *GALEX*/NUV field blind

Today we re-solved the astrometry (*ie*, we determined the pointing, rotation, and scale) for a single *GALEX* near-UV image from the All-sky Imaging Survey using the *x,y* positions of sources from the *GALEX* pipeline and the USNO-B1.0 catalog and *nothing else*. The source positions were given to us by David Schiminovich (Columbia) who withheld all information about the image pointing, rotation, and scale. The only thing we assumed is that the image is larger than about 30 arcmin in diameter.

2005 September 5: WCS optimization

We have created a system (not yet even ready for beta release) that takes an *x,y* list of compact sources in an image and an approximate WCS and produces as precise WCS as is possible given the USNO-B1.0 catalog. The WCS is fit including “Spitzer Imaging Polynomial” distortions to the

astrometry.net

- We are releasing all our code.
  email code@astrometry.net if you want to be a beta tester.
- We are putting the engine on the web.
  email hogg@astrometry.net if you want to be a beta tester.
- Our internal trac pages are public.
  Check out trac.astrometry.net if you want to see all the gory details.

Related Efforts

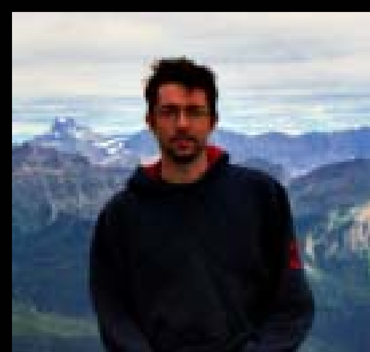
- automatch – John Thorstensen, Dartmouth
- Pinpoint – Robert Denny, DC-3
- TheSky/CCDSoft – Software Bisque
- Charon – Project Pluto
- imwcs (wcstools) – Doug Mink, Harvard CFA
- wcsfixer – IRAF-NVO@NOAO
- wcs correction service – NVO@U.Pitt

The Core Team

Sam Roweis

David Hogg

The real talent!



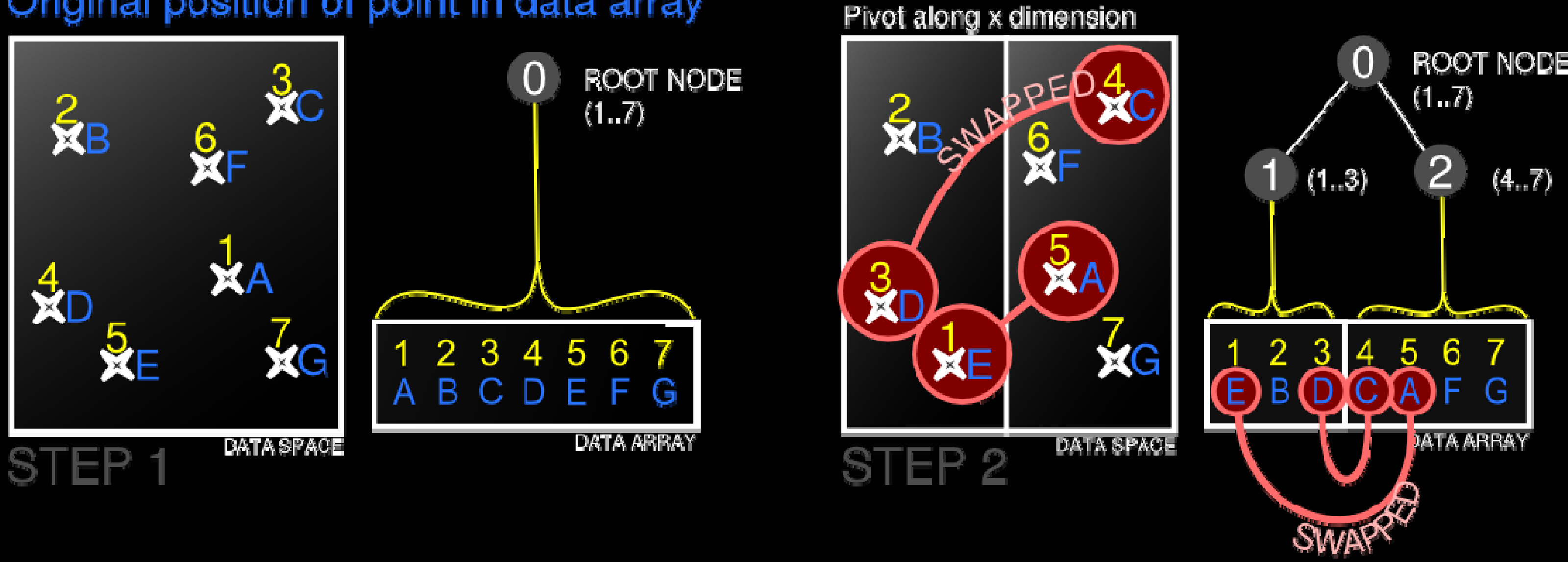
Dustin Lang

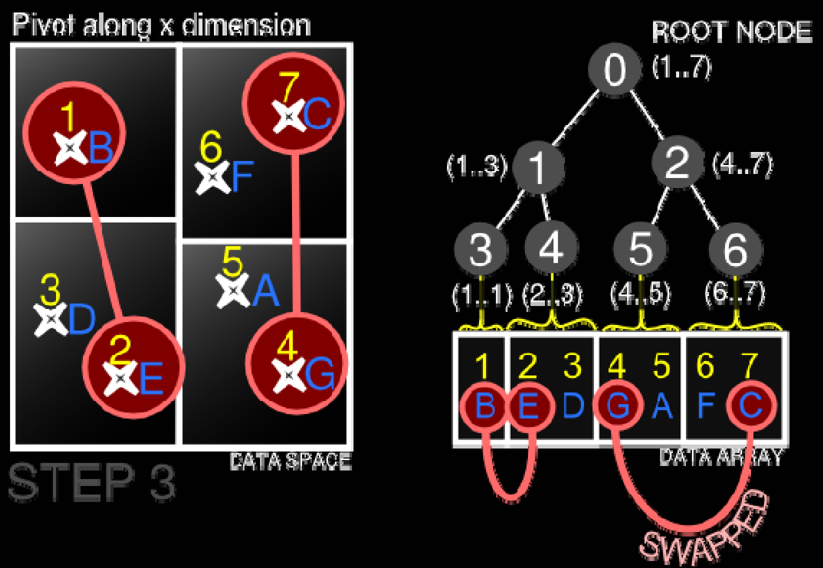
Keir Mierle

Michael Blanton

Pointer-Free KD-Trees

Position of point in data array
Original position of point in data array

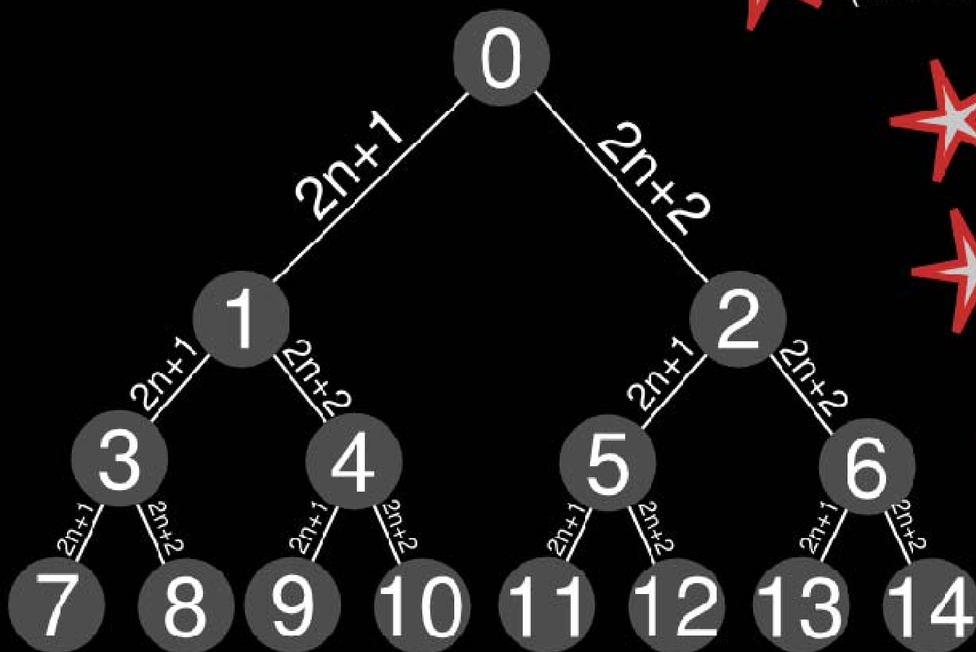


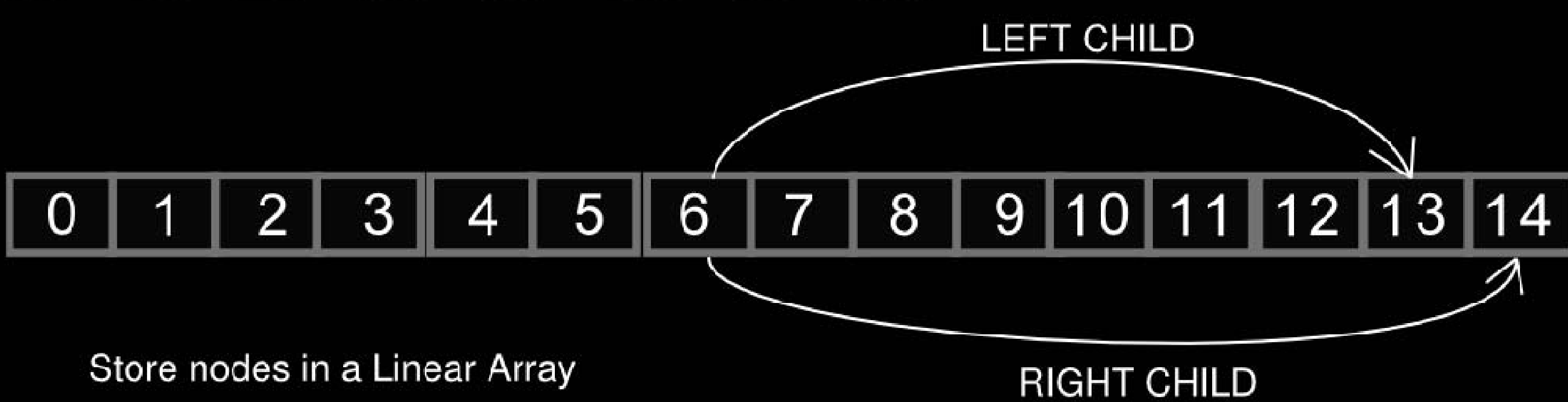


The nice thing about building a kdtree this way is that at the end of step three, all data points within a node are stored contiguously in the data array. This is very similar to quicksort.

Pointer-Free KD-Trees

- It is usually desirable to have a full tree for kdtrees (Not for doing ray-tri intersection, but that is different)
- Therefore we can use the indexing trick: Children are stored at 2n+1 and 2n+2
- Combined with previous pivoting trick: NO POINTERS ANYWHERE!
- Use mmap() for very efficient storing/ loading





Store nodes in a Linear Array