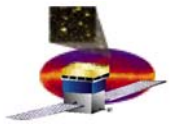
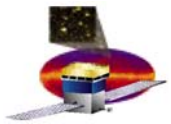


TkrRecon
Code and Documentation Review

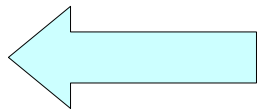
GLAST Software Analysis Group
Wednesday, September 4, 2002

The TkrRecon Group
Tracy Usher, Leon Rochester (SLAC)
Bill Atwood, Brian Baughman (SCIPP/UCSC)
Johann Cohen-Tanugi, Michael Kuss (Pisa)



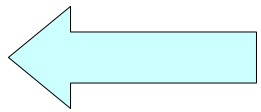
Outline

- Introduction and
- Basic Architecture



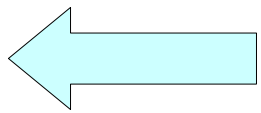
Tracy

- Services
- Clustering



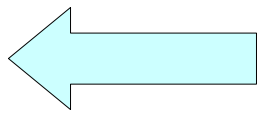
Leon

- Track Finding / Pattern Recognition
- Track Fitting



Bill

- Vertex Finding and Fitting

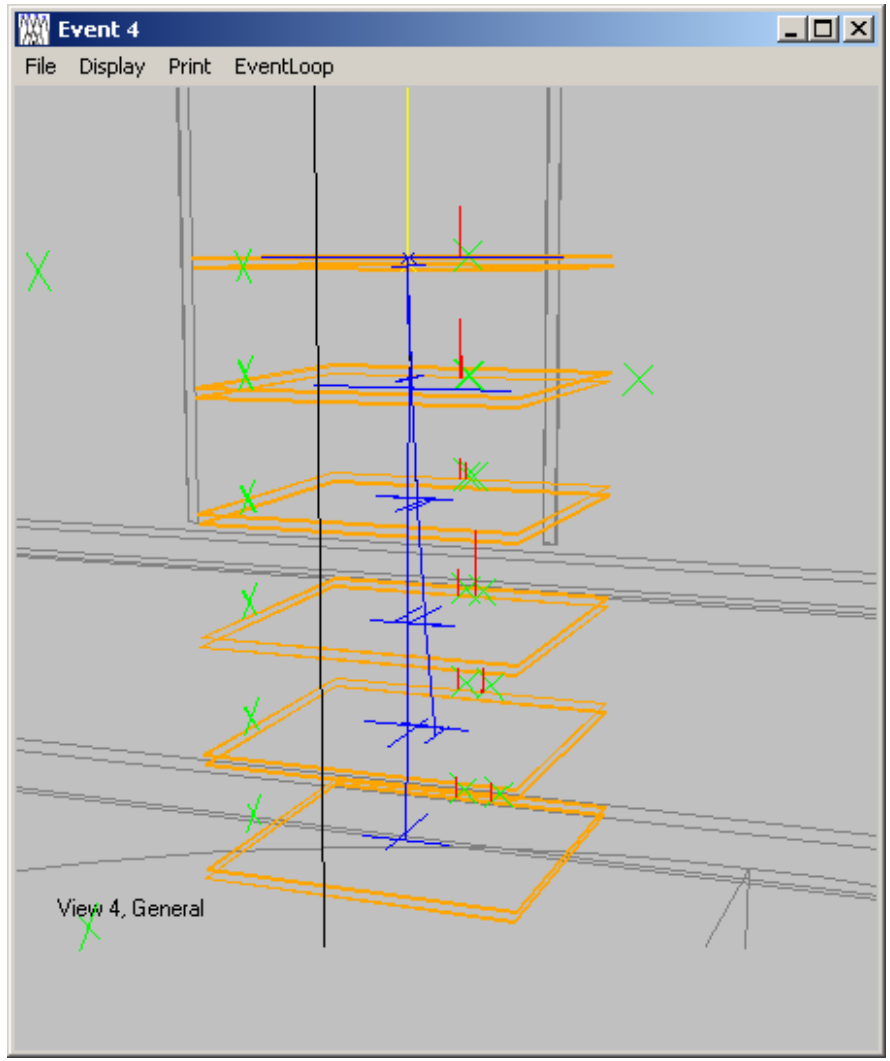


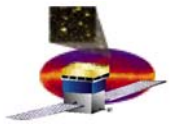
Johann

- Summary

Simplified Overview of the TkrRecon Task

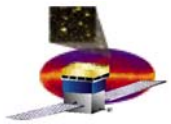
- **Reconstruction Goals:**
  - Determine the direction of incident gamma rays converting within the tracker
  - Aid in the rejection of Cosmic Ray and other backgrounds
  - Help the Cal energy reconstruction
    - Pointing to cluster centroids
    - Energy estimate/loss in the Tracker
- **Some Tracking Issues:**
  - Want to reconstruct Gammas across a wide energy range, from less than 30 MeV to greater than 100 GeV.
    - Gamma signatures will be energy dependent
  - Energy not well measured, especially for individual tracks.
  - The charged particles encounter a significant amount of material as they traverse the tracker. The reconstruction will need to be sensitive to these interactions.
  - Silicon strip measurements in x or y projections only, no stereo projections for mating individual tracks.





A Brief History of GLAST Track Reconstruction

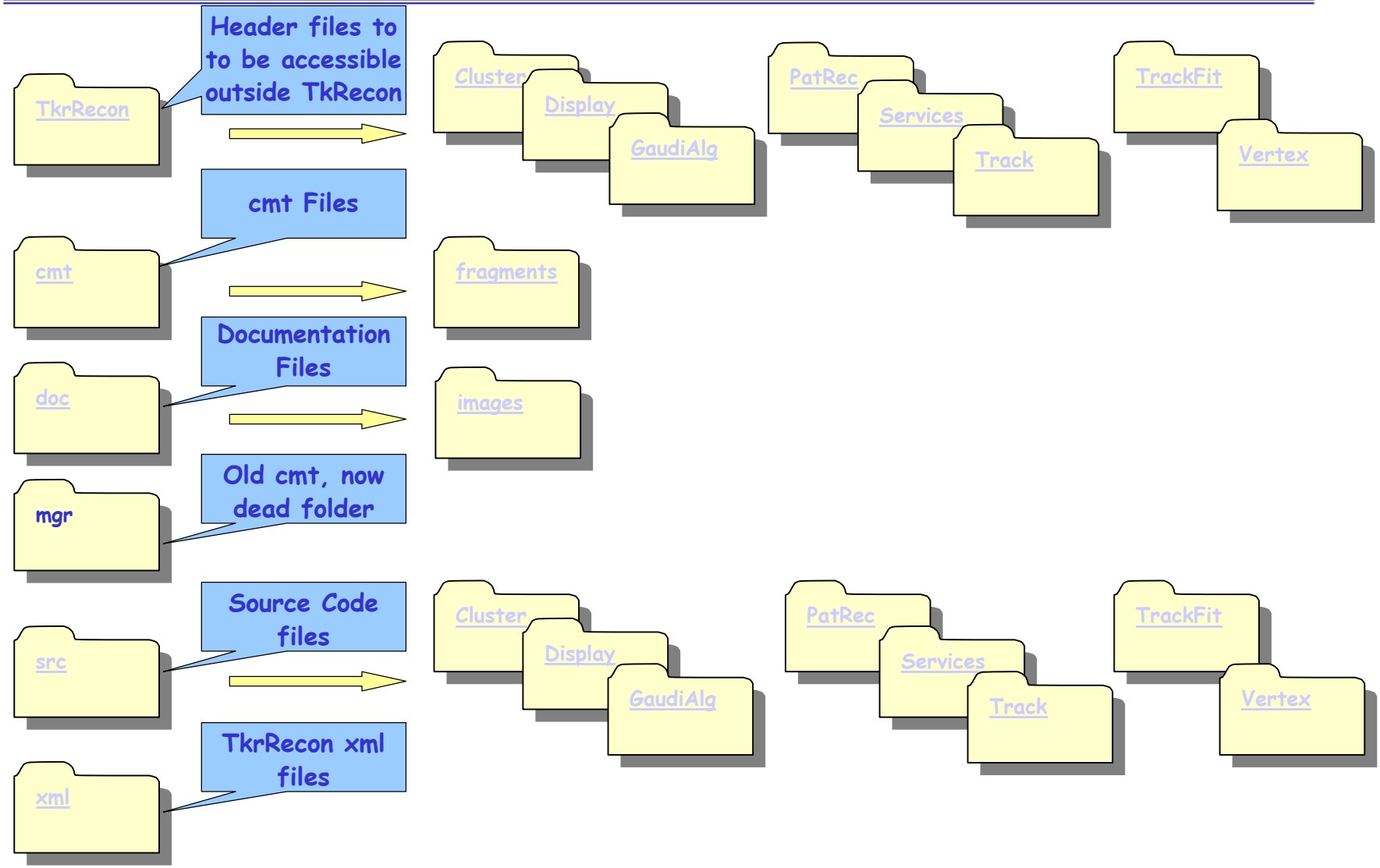
- **First Generation (Bill Atwood SLAC)**
  - Original version for initial studies of GLAST LAT ('92-'94)
  - Served as basis for the original proposals
- **Second Generation (Jose Hernando UCSC)**
  - Several modifications to the original version
    - Incorporate Kalman Filter
    - Modifications made to Pattern Recognition
  - Used for the "AO Response" (still in cvs as AoRecon)
  - Transformed and imported into the Centella framework (a Gaudi-like framework)
    - Used in analyses of the Test Beam data
    - Specific modifications made to solve the single tower problem of the BTEM
  - Initial import into the original Gaudi framework around February, 2001
    - Labeled "TkrRecon"
- **Third Generation (SLAC/UCSC/Italy groups now involved)**
  - Completed initial port of code to the Gaudi framework, used for PDR analysis
  - Implemented the first round of a new code architecture:
    - Use separate Gaudi Algorithms to control each step of the tracker reconstruction
    - Improved modularity of different components – goal of "plug and play"
    - Changed reconstruction strategy from finding 2-D gammas to finding 3-D tracks and then vertexing
      - Significant rewrite of previous code
    - Increasing use of Gaudi features – Algorithms, Services, Tools, TDS, PDS, etc.
    - Continuing to move forward on making Geometry, constants, etc., file driven and not hardwired into code
    - Etc.
- **Point: Reconstruction has continually evolved over nearly a decade!!**

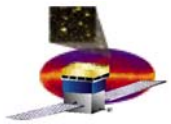


Basic Goals and Organization

- **Tracker Reconstruction inherited from Centella very complex**
  - Basically, a single module performing all tasks
  - Complex code relationships with unexpected consequences for small changes
  - Extremely difficult to understand and debug
- **Basic goals for new code organization**
  - Interchangeability
    - For example, provide mechanism to easily swap the "Combo" pattern recognition algorithm for Neural Net
  - Reduce complexity by breaking into smaller well defined tasks
    - Easier to understand each piece separately
    - Allows more people to be involved
  - Improve the long term maintainability
    - Smaller pieces easier to understand for future code maintainers
    - Documentation to aid future code maintainers
- **Reorganize TkrRecon cvs section to make finding the code easier**
  - Each major task (e.g. clustering) a subfolder from main level
  - Each specific implementation of that task into another subfolder
  - Services, Utilities, etc., into subfolders of main level

TkrRecon CVS Organizaton

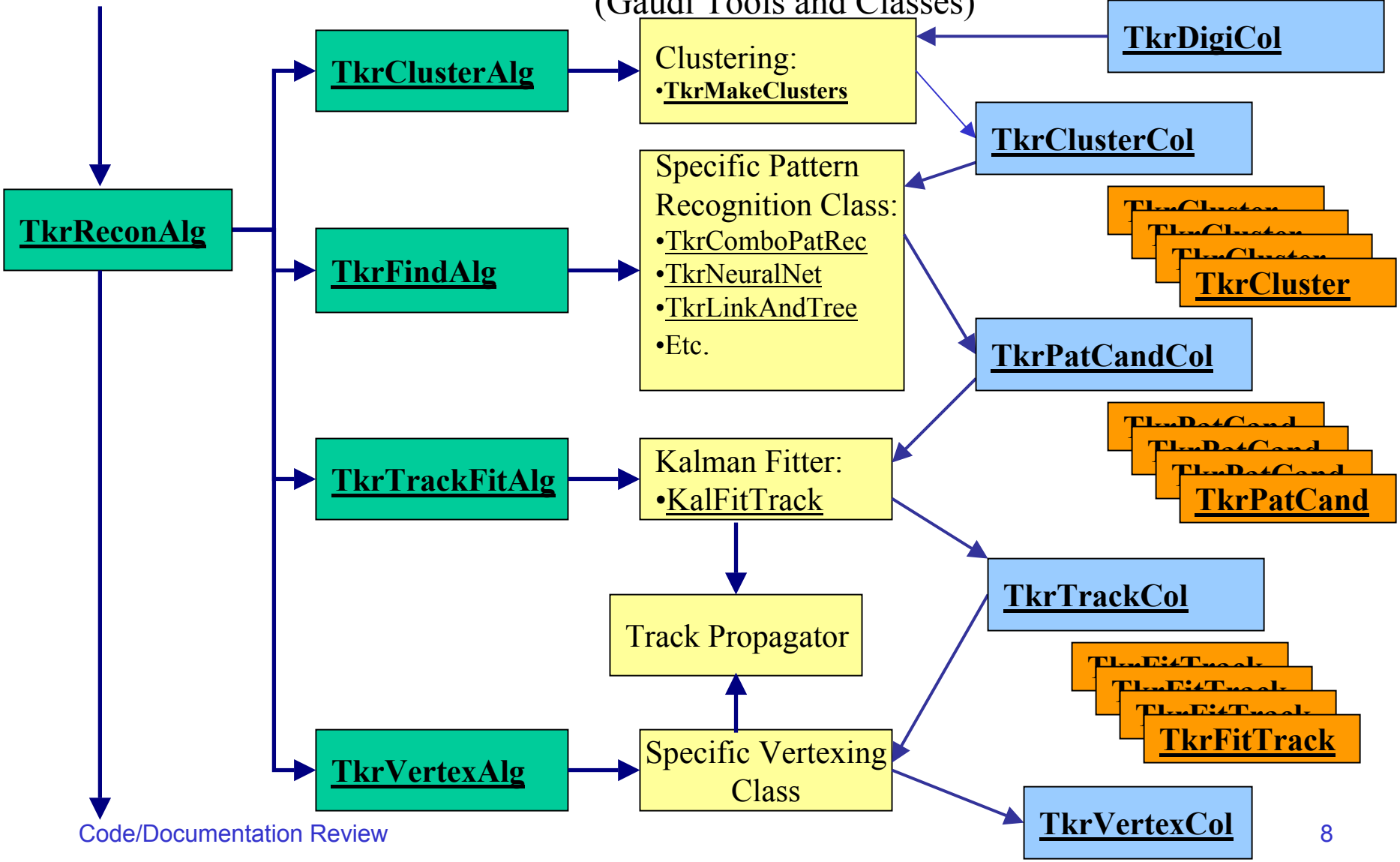


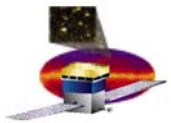


Overview of the Architecture

- **The basic strategy of the Tracker Reconstruction:**
  - Find and Fit all possible tracks in 3D
  - Find Gammas by "vertexing" the found tracks
- **Organize the main tasks into independent "Algorithms" which build upon the work of each previous step**
  - Clustering of hit strips ⇐ Clustering Algorithm
  - Track Finding ⇐ Pat Rec Algorithm
  - Track Fitting ⇐ Track Fitting Algorithm
  - Vertex Finding and Fitting ⇐ Vertexing Algorithm
  - Above implemented as "SubAlgorithms" of a main driving algorithm
- **Use "Tools" to interface to the specific implementation of an "Algorithm"**
  - A particular reconstruction method is implemented as a Gaudi Tool
  - For a particular algorithm, tools are accessed through an abstract interface.
  - Tools can be changed at initialization or "on the fly" during execution
  - Operate on the TDS output of the previous stage
  - Ouput results to the "standard" TkrRecon TDS objects
- **Use "Services" to provide necessary information**
  - Geometry, Reconstruction Constants, Calibration, etc.
- **"Extra" stuff**
  - Display routines, Utility routines, etc.

Tracker Reconstruction Diagram

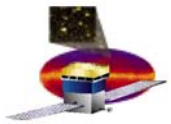




Current Status of Tracker Reconstruction

- First pass of new organization now in place
  - Geometry Service driven by xml files via detModel
    - Full flight currently only working version
    - Code should support BTEM and BFEM when those geometries become available
  - All (most?) constants can now be modified in job options file
    - Accessible to code in TkrControl singleton object
    - Initialized by the initialization service
  - Tasks driven by algorithms and tools
  - Have shown interchangeability of different reconstruction algorithms
    - Currently have three pattern recognition algorithms
      - "Combo" Pat Rec (See Bill's talk later)
      - Neural Net
      - Link and Tree (only a test version – not ready for general use)
  - Code documentation in place and nearly complete!

- Working version for more general use:
  - Current tag TkrRecon v6r5p3
  - Expect new tag in a few days – for more general use in Gleam
  - Default reconstruction will use the "Combinatoric" reconstruction (see Bill's talk)
    - Expect this to be the main recon for release at end of month

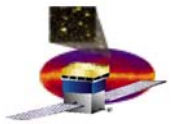


TkrRecon Services

There are three+ services in the TkrRecon package

- TkrInitSvc
  - Provides control constants for tracker reconstruction
- TkrGeometrySvc
  - Provides local copies of the geometry constants and access to necessary calculations
- TkrBadStripsSvc
  - Interfaces to bad-strips database, and provides access to the hits

+ TkrGeometryVisitor
  - Toy for now… illustrates how a visitor works

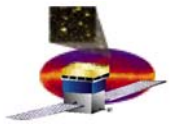


TkrInitSvc

This service creates the singleton TkrControl, and initializes it with a set of constants. The constants are properties of the service, so they can be modified in the jobOptions file.

The constants are control parameters for pattern recognition and track fitting.

Having the TkrControl singleton, rather than just using the service itself, avoids the business of passing a pointer to all the classes that use the control parameters, but of course, they have to get the pointer explicitly. The main reason to use it is that it's cool…



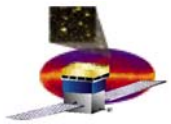
TkrGeometrySvc

The geometry service keeps local copies of various geometric quantities, which it obtains by asking GlastDetSvc for them, using getNumericConstByName(). These are then easily accessible to all the classes in TkrRecon.

It has 3 other methods:

- getStripPosition() returns the global position of a strip, given the parameters of the plane. Most of the work is done by GlastDetSvc, which actually passes the job on to SiPlaneGeometry. Since all the other geometric calculations for the tracker are done in SiPlaneGeometry, it might make sense to move this one there too.
- layerToTray()
- TrayToLayer()

The last two go from (layer, view) to (tray, botTop) and back. This information is only implicit in the detModel, so the calculation is just coded in. Again, there's no particular reason to put the calculation here; it might be more useful in GlastDetSvc.



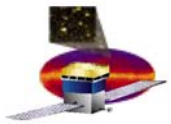
TkrBadStripsSvc

The bad-strips service interfaces to the database of bad strips (currently a text file, ultimately a file under control of the Calibration meta-database). It reads the data into a local array of lists of strips, with each strip marked bad.

Due to an excellent suggestion by Joanne, the strip number is now held in the lower-order 12 bits of an int, and the tag in the next 12 bits, starting in the right-most bit, but this is hidden by the access methods. One benefit of this is that a tagged strip is not a legal strip number, and an untagged strip looks just like a normal strip.

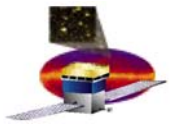
The service provides methods for access and manipulation of the lists and strips. For example, asking whether a strip is tagged as bad, accessing the strip number and tag field, returning a version of the strip for sorting, performing a sort on a list of strips, etc.

Currently, the only client of this service is TkrMakeClusters. But it's likely to be useful in pattern recognition: for example, if there's a missing hit in a plane, it would be nice to know the nearest dead strip to the extrapolated track.



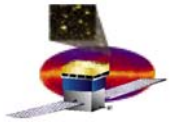
TkrGeometryVisitor

This is just a little exercise to remind myself how to code a geometry visitor. It's not actually called by TkrRecon, but maybe something like it should be with some simple test to demonstrate that the geometry is functioning.

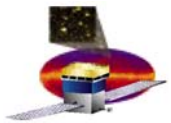


Clustering

- **Clustering Algorithm**
  - For each digi, TkrMakeClusters goes through all the strips, grouping them into contiguous clusters, separated by gaps. A gap is either:
    - An unhit strip. (This assumes that the strips are very efficient.)
    - The edge of a ladder. (The space between ladders is ~ 2 mm.)
- **A complication: bad strips**
  - If there is a dead strip in the middle of a cluster, the straight-forward algorithm would generate two clusters. Also, it would generate clusters composed of only hot strips.
  - So, if the TkrBadStripsSvc exists, for each digi, any bad strips for that plane are merged with the hit strips in the data. These are tagged so that they can be identified. Then, after a cluster is found, we can ask if it is good. A good cluster has:
    - at least one hit strip in it. This is determined by comparing the number of bad strips in a cluster with the first and last strip number.
    - (*NEW!*) No more than 10 strips. This should probably be no more than 10 bad strips, since even very wide clusters have some information, and it should be up to the client to make the decision.
  - Clustering works with or without the TkrBadStripsSvc.
- **Pointers to the clusters are added to TkrClusterCol, as a simple vector, and also as an array of pointers, dimensioned by layer and view.**



Other Presentations go in here



Summary

- TkrRecon has been successfully converted to a new architecture
  - Broken into smaller modules
  - Makes use of Gaudi Algorithms, Tools and Services to accomplish tasks
- New version can easily accommodate different reconstruction methods
  - New method implemented as a new Gaudi Tool
- New version has already undergone its first major revision
  - Converted from 2D to 3D
  - Now finding and fitting tracks, then finding and reconstructing vertices
- Have completed two rounds of code documentation
  - Beginning to get it right!
- Nearly ready for end of September code release:
  - "Combo" recon (current default) is ready now
  - Neural Net should be available soon (already works but needs testing?)
  - Other versions of Pattern Recognition could be available quickly
- Ready to start detailed performance studies!!