Refinement II - Modern Developments

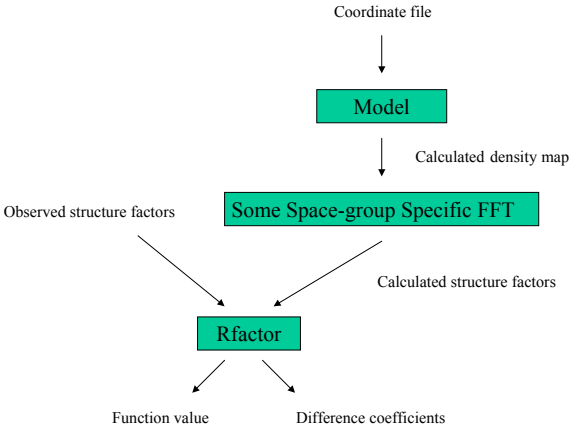
Dale E. Tronrud
Howard Hughes Medical Institute
University of Oregon

TNT Refinement Package

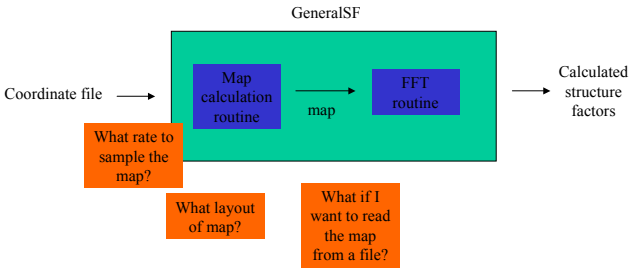
- Designed and partially implemented by Lynn Ten Eyck in the late 1970's
- I have been working on it since 1981.
- We started distributing copies to other labs around 1984.
- It is used as the refinement engine in Buster/TNT from Gerard Bricogne's group.

Relevant Design Aspects

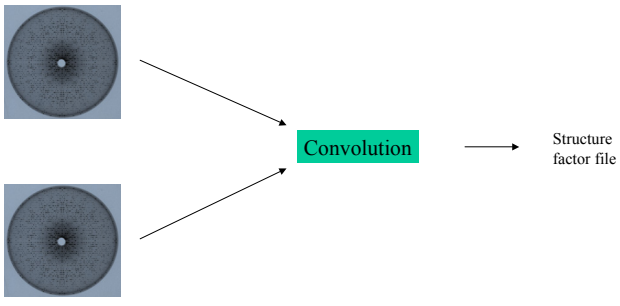
- The first popular reciprocal space refinement program was PROLSQ from Konnert and Hendrickson.
- This was a monolithic program.
  - To explore your own ideas of refinement you had to understand and modify their code.
- Lynn went the opposite way: a collection of programs coordinated by a scripting language.
  - One advantage of this design is that individual programs can be replaced with better implementations without worrying about the rest of the programs.
  - Another advantage is that one has access to the data stream between TNT programs, which allows one to add enhancements to without modifying or even trying to understand our programs.



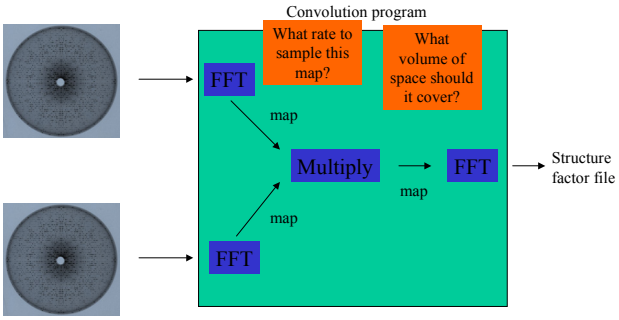
So, we write a single program for structure factor calculation



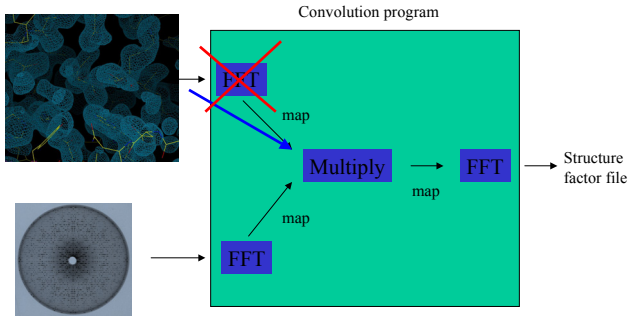
A more complicated example of the same problem



A more complicated example of the same problem



A more complicated example of the same problem



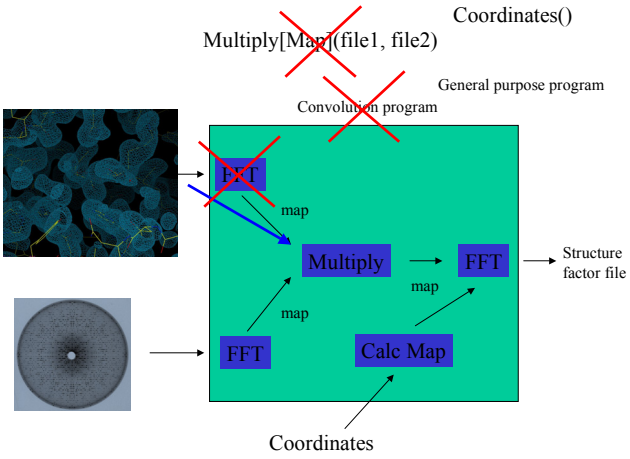
The Solution

- Change the programming model.
  - Instead of programming serial operations, go to an event driven model.
- The user isn't asking the program to do all these steps. The program should begin with the user's request.
  - I would like a structure factor file which contains the results of multiplying in real space the maps which correspond to the data in these two files.
- The program should formalize this request and pass it to a routine (class, object, module, function, subroutine, whatever) whose job it is to fulfill requests.
- The "request satisfier" will unfold the top layer of the request, and compose new requests for the data it needs.
- It then recursively calls itself to acquire the data it needs.

- I would like a structure factor file which contains the results of multiplying in real space the maps which correspond to the data in these two files.
  - (While the program knows at this point if each file is a map, or HKL's, the "request satistier" does not.)
- The program rephrases this request as: Give me the set of structure factors that result from this calculation:
  - Multiply[Map](File1, File2)
- The request is passed to the "request satisfier"
- The "request satisfier" sees that structure factors are requested but the script returns a map, so it passes the request to the structure factor calculating FFT for processing.
- The FFT builds a request for a map based on its needs.
  - It needs the map to cover an asymmetric unit of the unit cell.
  - It needs the sampling rate to be at lease twice the resolution of the map.
  - It doesn't know the space group or the resolution so its request must be **vague**.
  - The new request is passed to another incarnation of the "request satisfier".
- This "satisfier" sees a request for a map to be derived from a operator that creates a map.
  - It fires up the code that implements Multiply[MAP] and passes it the request.

- The map multiplier knows that it needs two maps to multiply.
  - It generates a request for its first argument.
    - It knows that it needs an asymmetric unit filled with density. The sampling rate is more of a problem.
      - The resolution limit will be higher than the arguments. My code assumes that the two maps will have the same resolution so I double the sampling rate of the requests maps. This map should be oversampled by a factor of four.
    - The script for this request is simply "File1".
  - It sends its request for its first argument to yet another incarnation of the "request satisfier".
- This incarnation sees that it has a request for a map but the source is a structure factor file. It calls of the map calculating FFT and passes on the request.
- The FFT knows that it needs structure factors and creates a request for them and passes that request, along with the script "File1" to yet another incarnation of the "request satisfier".
- The satisfier sees that it is being requested to return structure factors from a structure factor file, so it does so. Finally the space group and resolution limit is known.
- The FFT now knows the space group and resolution limit so it can decide on a sampling rate for the map and and its layout.
- It calculates the desired map and returns it to the satisfier who returns it to the map multiplier.

- The map multiplier receives the map for its first argument and builds a request for the second.
  - The second map must have the same sampling rate as the first because the grid points must line up. The layout must also be the same. This request is more specific.
- The second argument request is passed down the line and a map is returned.
- This map may not fulfill the needs of the request. The suppliers of data try their best to match a request but sometimes it is impossible.
  - The map multiplier must check that maps it receives are suitable for its calculation and adjust accordingly.
- The space group of the product map may differ from the arguments. The true space group must be passed back to the requestor.
- The map multiplier multiplies the maps and return the result to the structure factor calculating FFT.
- The FFT now learns the space group, resolution, and sampling rate of its map.
  - It must verify that it can work with this result.
  - If so, it does the FFT and returns the structure factors.
- The user finally gets the structure factors they wanted!



A Other Examples

- Apply_Mask(File, Envelope(Triangle(Truncate(File))))
  - This script applies Wang-style solvent flattening to the map, or structure factors from a file.
- Multiply[SF](File, Conjg[SF](File))
  - Calculates Patterson coefficients, a Patterson Map, or Patterson peaks depending on what type is requested.

Problems

- This entire concept is **illegal** in Fortran 77!
  - Recursion is forbidden.
- Designing the ability to make sufficiently vague requests is an ongoing task.
  - For example, when asking for an asymmetric unit of peaks you need a map that has a little extra around each edge of the asymmetric unit.
- Some decisions are hard without more knowledge.
  - For example, when zero-truncating a density map the resolution of the resulting map will likely be greater than the unmodified map. But by how much?
    - It depends on how many grid points you zero.

What next?

- Currently all the structure factor and map calculations in TNT are done with this scheme.
- I would like to expand this to other data types, such as atomic shifts, gradients and curvatures. With this I could restructure the package to simply answer the request "I would like a better model given this model".
- **Reality check**:
- I am reaching the limits of what I can do in this language. I am thinking in object oriented terms but my tools are the classic "stone knives and bear skins". It's probably time for a complete rewrite.