Tracking in emulsion

Johan Uiterwijk

April 3, 1998

Abstract

This note explains the track finding algorithm to be applied to digitised emulsion images in the approach adopted for the CERN scanning strategy. The tracking program is written in C++ using Object Oriented technology and one of the main components is a 3-dimensional binary tree that allows very rapid searching for grains in any random volume. First results concerning speed and efficiency of the first steps of the algorithm will be reported.

1 Introduction

In the analysis of the CHORUS emulsions, as it is envisaged in the CERN scanning group (described in the contribution by Bart Van de Vyver), an important part is full tracking of a region around a possible vertex. This requires a program capable of finding all tracks in the tomographic images taken from the emulsion. After track finding the tracking program must analyse the topology and classify what happens to the scan back track that disappears in the region analysed. The scan back track can either continue, end in a vertex or have a kink.

There exists already programs, e.g. SySal, that do the same as we are trying to do here. We tried to construct a flexible and robust algorithm that is isotropic in space and that could be easily modified using Object Oriented technology (C++ implementation). Part of the track finding, the segment matching, is based on the track finding algorithm used in the CHARM II experiment. This algorithm has proven to be able to handle distorted tracks (mult. scat.) and find kinks and vertices.

2 Tracking environment

The environment in which the tracks are located is governed by the following numbers. There are 16 to about 50 images, taken as input for the tracking program. Each image contains after digital filtering anything from a 100 to 3000 grains. Since grains are recognised in the filtering step, the filtering algorithm must be very efficient and adaptable to changing emulsion quality. The filtering algorithm we plan to use is described in a note by Piero Zucchelli [1].

A grain is a developed crystal in the emulsion with typical dimensions of about 1 μm. Its spatial coordinates can be determined to about 0.4 μm [1] in the transverse directions

[1] This is the intrinsic resolution of the emulsion, not the resolution of the stage and microscope ($\approx$ 0.15 μm).

inside a layer (x,y)[2]. The uncertainty in the z-position is much larger and is mainly determined by two factors. The depth of field of the optical system used to scan through the emulsion ($1\dots3\,\mu$m) and the shrinkage of the emulsion during development.

Tracks are characterised by long range correlations between grain positions. The emulsion used in CHORUS gives about 30 grains per $100\,\mu$m track length. A track will be a straight line with possible distortions (introduced during development, to first order parabolic), multiple scattering and kinks. Large angle kinks are not a problem because they are found as two independent track segments. Small kinks however could be combined into one track and are then missed, but some methods exist to recover these small kinks [2].

3 The D-dimensional binary tree

In most tracking algorithms one has to look up if there is a hit in a volume predicted by a previous combination of hits. With $O(10^5)$ grains on 50 layers it would take very long to scan all grains to check if they are close to a prediction. However, this searching time can be greatly reduced by sorting the grains and performing a binary search. One would like to sort the grains according to their position which is a 3-dimensional parameter space. The concept of a binary search tree was therefore extended into D dimensions. Instead of splitting a line segment into half at each step in 1 dimension, a cube is split into $2^3 = 8$ sub-cubes for 3 dimensions. In C++ this can be implemented using recursive algorithms and the main tool in the tracking program is a template D-dimensional binary tree class, which behaves according to the STL conventions [3, 4].

The principle of placing points into 1-, 2- and 3-dimensional trees is depicted in Fig. 1. Whenever a certain subvolume (half, quarter, 1/8 sub-cube) is already occupied it will be split into another level of subvolumes and the previous object is moved to its new subvolume, before the object that is inserted is placed into its subvolume (note that if the two objects are close together this process can be repeated many times, as is indicated in Fig. 1a).

Extracting all objects within a certain volume can be done very quickly. At the top of the tree each subvolume is checked to see if it contains (part of) the requested volume. If so, the subvolumes for that volume are checked, until one reaches the bottom of the tree, where the objects are stored. The object is checked to lie within the given boundaries and inserted in the found objects list. The time it takes to traverse the tree is $O(2^D \log N)$ with N the number of objects and D the dimension of the parameter space. For a linear search the time would be $O(N)$. A test showed that the binary search with 150000 grains is about a factor 2000 faster than a linear search. However, the time it takes to build the tree is equal to about 20 linear searches, so using the tree pays off only if more than 20 searches have to be done.

[2]The emulsion coordinate frame used here differs from the CHORAL frame. The coordinates in a plate are called x,y and the positive z-axis is normal to the plates in the beam direction.

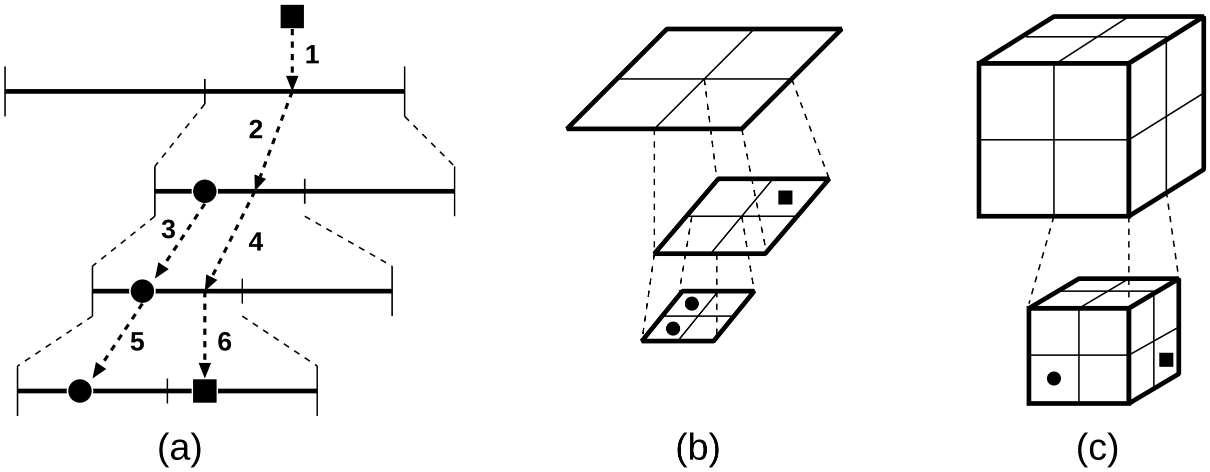

Figure 1: Sorting points using 1-, 2- and 3-dimensional trees. When inserting a new object (black square) a decision into which subvolume the object goes is made at each level of the tree. If a subvolume is already occupied (black dots) that volume is split and both the new and the current object are moved to their respective subvolumes. The numbers in figure (a) indicate the steps taken in the algorithm.

4 General tracking

The steps in the algorithm for the general tracking (all angles) are described below. After filtering the images the pixels surviving a threshold cut are clustered and the center of gravity of each cluster is stored in a 3-dimensional tree. Next, for all grains vectors (links) are created to all grains in a small cube (currently 15 μm) with the start grain in the center of the bottom surface (acceptance in forward direction only).

Now a new loop is started over all grains. For each grain, all links that are accepted by a track segment search initial criterion (400 mrad) are followed up. All links from the endpoint of the previous link are checked against an angular acceptance cone along the initial direction. If links are accepted this procedure is repeated recursively, The acceptance cone depends on the accuracy (number of previous links) of the prediction. The growing of a segment stops if no more links are accepted. When returning from the recursive calls of the growing method, a choice is made between the possible multiple segments at that stage. The choice is based on the quadratic distance of all grains to their respective prediction and the number of grains in the segment. In other words, straighter and longer segments are preferred. When returning to the top level a final acceptance criterion is applied, currently a minimum number of grains is required.

All the above mentioned criteria are implemented with a virtual `criterion` class with methods for initial, growing and final acceptance. Because this class is virtual, other criteria can be coded without the need to change any part of the program and different criteria can be selected at run time. Also the growth algorithm is contained in its own `segmentManager` class which allows to change the growth algorithm without changing the criteria or other parts of the program as long as the input and output of the class are unchanged.

The last step in the algorithm is the combination of segments into tracks, vertices and kinks. This part is currently still under construction. For each segment its 3D position with respect to the other segments is calculated. There are only a limited number of possibilities which can be described by the following parameters:

- Distance of closest approach, this parameter determines if the lines "cross" anywhere in space.
- Difference vector between end points of the segments. This parameter determines if the segments cross each other internally, are connected or are far away from each other.
- Angular direction difference. The cross-product of the two direction vectors of the segments tells if the segments are pointing in the same direction or meet each other under an angle.

The first part of the algorithm was tested on images taken using the manual scanning program on SS. **All** 7 tracks found manually in 5 views were also found by the tracking program. The 16 images in each view contained on average about 4000 grains which resulted in about 13000 links in total. The time it took to build the tree and generate all links is $\approx 70 + 540 = 610\,\mathrm{ms}$[3]. The segment finding needs about 900 ms to find all segments. The total time of about 1.5 seconds is twice as fast as the Nagoya FTS in general scanning. On average 9 to 10 grains are found to belong to the track. This relatively low number is due to the fact that the images on the top and bottom do not contain any grains anymore, because the images were not spaced optimally in the thin emulsion sheets. Looking manually at these tracks, also not more than 11 grains are found. Results for bulk are not yet known because no filtered images with known tracks are available yet.

Since the remaining number of segments is small, the time it will take to combine them into tracks and vertices will be comparatively small. For the SS images each track is found as one segment with loose angular acceptances in the algorithm. For more restrictive acceptances the track is often split into two consecutive segments.

5 Prediction scanning

If one is only interested if a track with a certain angle is present in an emulsion view then the track angle information can be put to good use to speed up the tracking. This "Track Selector" like algorithm can be made very fast in software using 2-dimensional trees. Instead of varying shifts over the acceptance region and making a sum of all pixels, in the algorithm tried here the whole acceptance region is checked in one go, speeding up the calculation because the 25 ... 35 summing trials are skipped.

The grains for each image are stored in a 2D-tree. Running over all grains, starting on the top layer, the position of that grain (origin) and the prediction plus acceptance define an area on the next layer. The tree is used to find all grains in that area. If there are grains present in that area, a counter indicating the number of grains in the right direction is incremented with the counter value of the origin. If there is no grain then the origin is shifted up to that layer with the shifts in x and y given by the predicted angle.

[3] All times given are for rsch03 (powerPC), the windows NT machines (Pentium II, 200 MHz) give the same performance.

This process is repeated for all layers, including the grains that are moved up to that layer. If the counter value is too low, indicating that the sum with the remaining layers can never reach the threshold anymore, the grain is discarded. After this step a low number (≈ 50) of grains, that crossed the threshold on the summation, will be left over on the bottom layers.

However, the grains that were added in the sum, do not necessarily lie on a straight line. It could be that the grains are "randomly" distributed within the acceptance area. Therefore a last step is needed (currently under construction) to check if the regions defined by the grains left over actually contain a straight track. At the moment the idea is to use an adapted version of the general scanning algorithm for that. But also an adapted TS summing algorithm in the small regions could be applied.

The expectation is that this algorithm can compete in speed with the FTS from Nagoya. Inserting the 4000 grains into the tree, performing the acceptance area lookups and promotion of grains takes about $45 + 45 = 90\,\mathrm{ms}$ for an acceptance of $25\,\mathrm{mrad} + 5\,\%$ of the prediction angle. The sum threshold was put at 11. Further investigations on efficiency, speed and track validation will be done by trying this algorithm in the automatic scanning program running the CERN microscopes on already scanned data.

6 Acknowledgements

This work could never have be done without the contributions and working atmosphere provided by many people. I want to thank Jaap for the many ideas, discussions and support when thinking about how to do tracking in Emulsion. Franca wrote the manual scanning program and did the manual scanning and image taking of the SS views. Bart and our Nagoya colleagues provided the necessary input on emulsion parameters and possible problems one could encounter. All the people in the CERN microscope group I have to thank for the nice discussions and the free exchange of ideas and general working spirit. The people who work on SySal have shown that track finding in the environment of the emulsion with large amounts of grains is possible.

References

[1] Piero Zucchelli, CHORUS internal note 97027 (13-02-98), "On microscope images, filtering and compression"

[2] K. Niwa, M. Miyanishi, Transparencies of the 11-13 March 1998 coll. meeting, "Run II ('96/'97) scanning", (http://chorus01.cern.ch/Internals/meetings/9803-CM/slides/sBniwa)

[3] Bjarne Stroustrup, Addison-Wesley (1997), ISBN 0-201-88954-4, "C++ programming language : third edition"

[4] David R. Musser, Atul Saini, Addison-Wesley (1996), ISBN 0-201-63398-1, "STL tutorial and reference guide : C++ programming with the Standard Template Library"