A few words about the Cellular Automaton
VERY RAW VERSION

Robin Glattauer

September 4, 2011

Abstract

I documented my code with doxygen, but of course this a good way to document code and not to document an idea. So for getting the overall picture of it, I wrote this short introduction. It should serve as a quick guide, telling what a cellular automaton is, why I use it and how I designed my classes.

1 What is a Cellular Automaton

It is a set of discrete entities, which can have discrete states and change their states depending on their environment. Let's give an example for this, because the last sentence is horrid: the maybe best example for it are cells (tada). Which kinda is the basic idea behind it. Cells are discrete because there is always one cell or two or seven cells, but never 2.3 cells. And we assign them discrete states like: "living" or "dead" or others.

The idea now is, that with every timestep the situation evolves. So when a cell for example finds that there is space to expand, it might create new cells around itself. And when there's no space, it might just do nothing. And when it's surrounded by dead cells it might think "oh, what the heck" and go for suicide too.

So how the cells are evolving, what their states are and so on, depends completely on the implementation. A word said here: No, my Cellular Automaton is not capable of doing that. It is designed for pattern recognition in track search.

2 The Cellular Automaton for Pattern Recognition

So far it sounds like the Cellular Automaton would be an interesting tool to simulate cells or population growth, but that's not, what I'm using it for.

What is a pattern? As hard as it is to define "pattern" exactly, what we might say is, that a pattern follows some rules: like the pattern of tiles on a chess board will always follow the rule, that there is a black tile next to a white tile. And a cellular automaton has rules too: there are the rules how to change the state. So if one implements rules in a cellular automaton that resemble the rules of a pattern one wants to find and then lets cells following this pattern survive and others die, at the end we are left with living cells forming the pattern.

2.1 Track Search

What I want to do is Track Search in the forward direction. So essentially I have a lot of spacepoints stemming from a track and need to find out how they are connected. I have 7 layers of discs (all normal to the z axis) in the forward direction I can use. And there is a magnetic field in z direction, so all the charged particles, that cause the hits will run on helix tracks. As a helix is well defined with 3 points, I could gather all triplets of possible points and make helixes and then try to pick up other points from other layers. Usually I would do this using a Kalman Filter, but this means a lot of use of a maybe time consuming Kalman Filter at an early stage.

So what is done instead is, that we try to find connections, that follow patterns, that are created hy a helix track. Those connections (I call them segments) are found by using a Cellular Automaton on all possible connections.

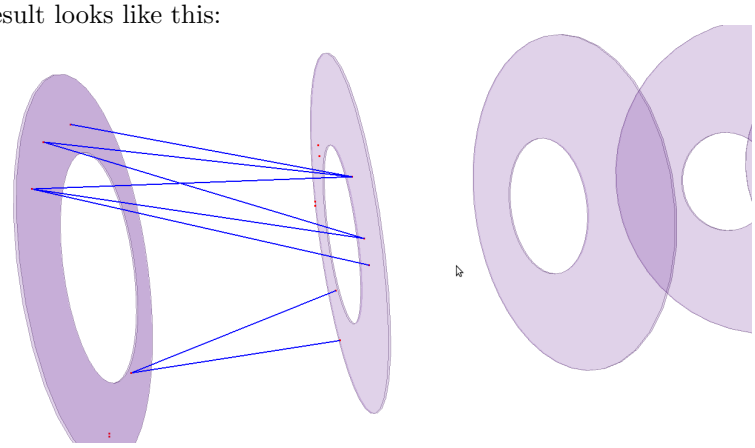
Let's see how this works:

At first we just have all the points on all the layers. If we now just take all triplets we straightly drive down the road of combinatorical disaster. So we have to start sorting out.

We start building segments that contain 2 hits. Somehow logical, I call them 2-segments. We only connect two hits, if we think they could belong to a good track (which for example means: not too curly). Our first criteria here is: the distance between the hits divided through their z-distance should be below some reasonable value like 1.1.

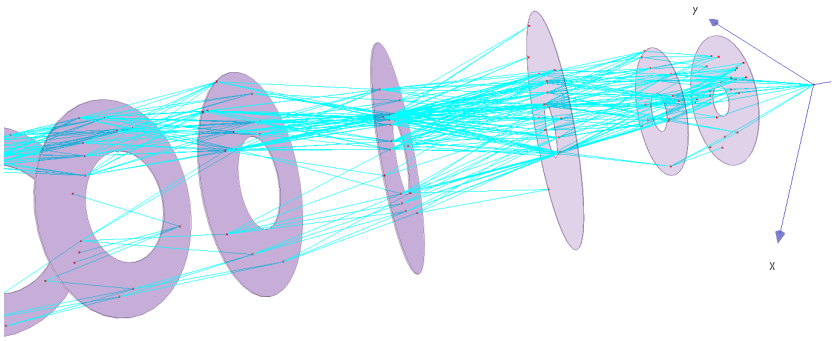
$$\frac{R}{\Delta z} < 1.1 \tag{1}$$

The result looks like this:



A little statement here: In reality it looks of course a bit more dense with segments than here. All the pictures I used are some, that help illustrate the procedure, and not realistic ones. Realistic ones would have much more noise and therefore much more segments.

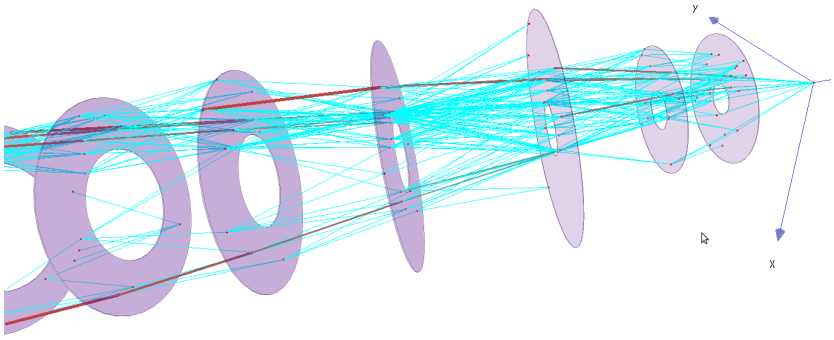
After for all layers the 2-segments are created it looks like this:



An important thing is, that all those 2-segments get connected. So a segment will always know its children (connected segments on the inner side) and its parents (connected segments on the outer side). So at the end, once we got rid of all wrong segments and wrong connections, we can start at a segment which has no parents and move inwards, creating tracks with all children until we reach the innermost layer.

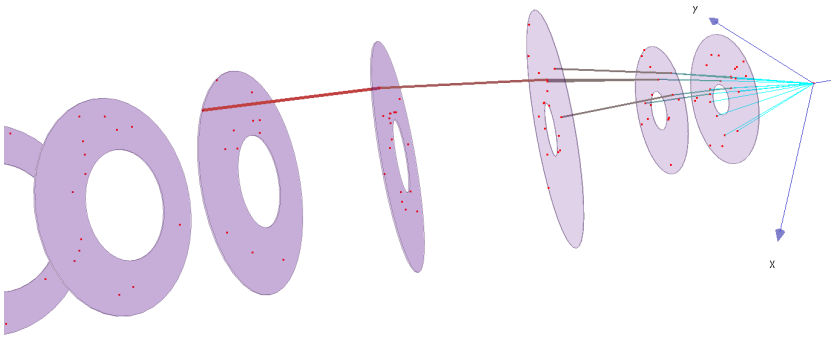
Now we can start the Cellular Automaton. What it does is this: He checks every segment, if it has a neighbor. A neighbor is a childsegment, that has the same state as the segment itself and fullfills some additional criteria. I will come to those criteria soon, as they are what defines our track. At the beginning every segment has state 0. So of course they all have the same state. If a segment now has a neighbor, so at least one of his children fullfills the criteria for being one, the segments state will be raised by 1 at the end of the round. And then this is repeated. Again and again until no more changes in states happen. The reason behind doing this is simple: segments need to be connected to other segments in a good way which again are connected to... and so on to get a high state. Having a neighbor alone will get my state raised only once. Cause after this, if the neighbor doesn't have a neighbor itself, I will stay at this state. Because as was mentioned: a neighbor only is, what has the same state. TODO: insert a picture of this, so it's easier to understand.

After doing the automaton, it looks like this (the higher the state the redder and the fatter the lines get):



In the next step we clean up and erase all segments, that don't have the

highest possible state in their layer:



So now comes the point: what are the criteria. Here are the things that I demand from 2 2-segments to be compatible with each other:

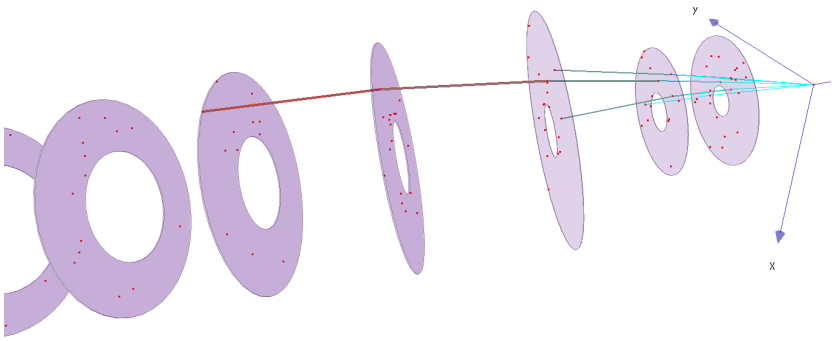
- The angle between them (i.e. between the straight lines connecting them) should not be to big
- If the 3 hits form the 2 2-segments are taken and we calcualte a circle in the xy - plane from it, this circle should pass close to the xy- origin
- From the radius of the circle we can approximate the transversal momentum of the particle and demand a minimum p_T.

So now we are done with criteria using 3 points. We are therefore at the end of the 2-segments. The next step is to build longer ones: 3-segments, which contain of course 3 hits each.

And again we perform the Cellular Automaton on them using other criteria:

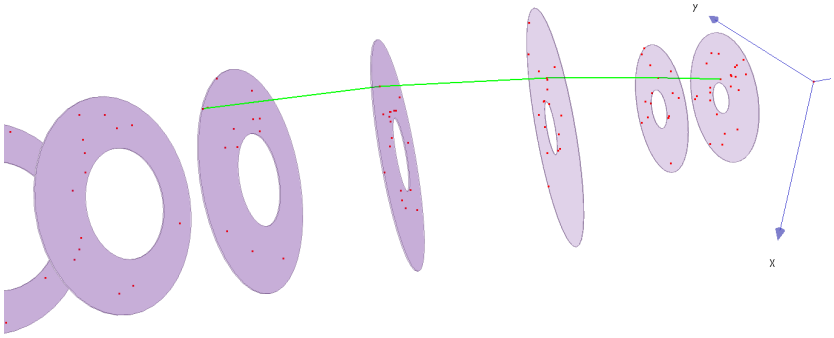
- No zigzagging is allowed
- A 3-segment consists of 2 2-segments. And between them there is an angle in the xy plane. This angle should not change a lot
- The radius should not change much.
- The angle from the center of the cricle to two of the hits should be proportional to the z distance of the hits.
- From the 3 hits of a 3-segment we can extrapolate to the z value of a fourth hit. The 2D-distance of this hit (belonging to the other 3-segment) is not allowed to be too high.

And then we clean again as well and get this:

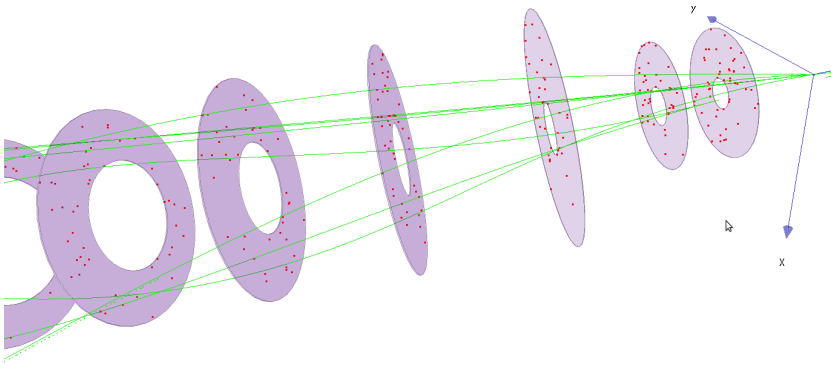


We could now go higher and build 4-segments and so on. As long as we have criteria we can use, we may do this.

The final step is then to collect all the track candidates. Because helix fitting starts to make sense from 4 hits on, we only collect tracks with more than 4 hits. Here, we found exactly one trackcandidate:



Now let's have a look at the true tracks and see, if we found them:

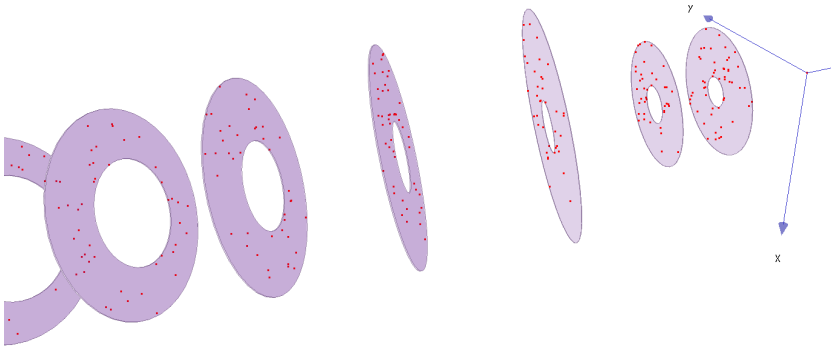


Well there he is, our reconstructed track. And some others... Where do they come from? They all didn't hit the inner layers for geometrical reasons, so this means we also have to include the possiblity to jump directly from the origin to layer 2,3 or 4. And not only that, we should as well allow the skipping of single

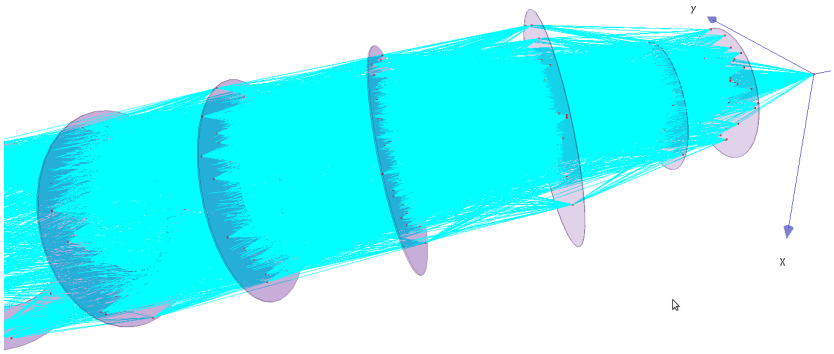
layers, as the hardware might not always work perfectly.

So let's repeat the procedure with also connections from layer 2 to layer 0 and we might as well add some background, let's say 500 additional hits (to around 100 real hits in both directions, but we only have a look at backwards now).

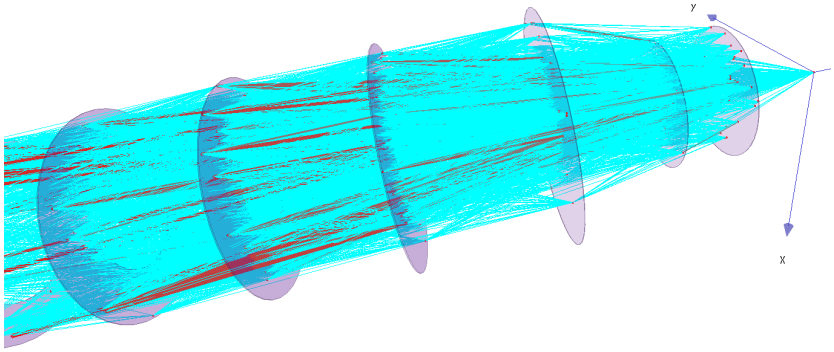
1.) The hits:



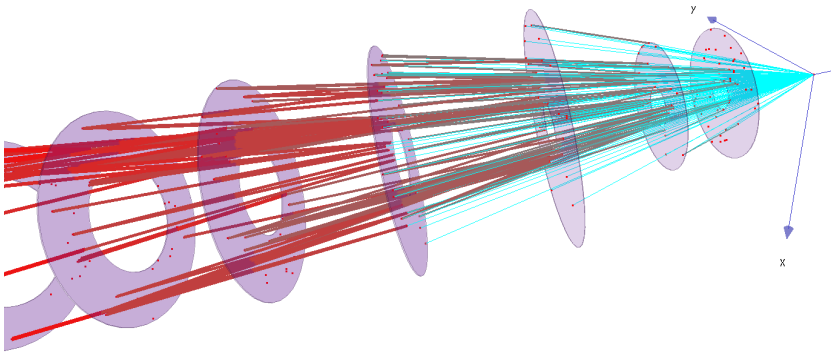
2.) The 2-segments:



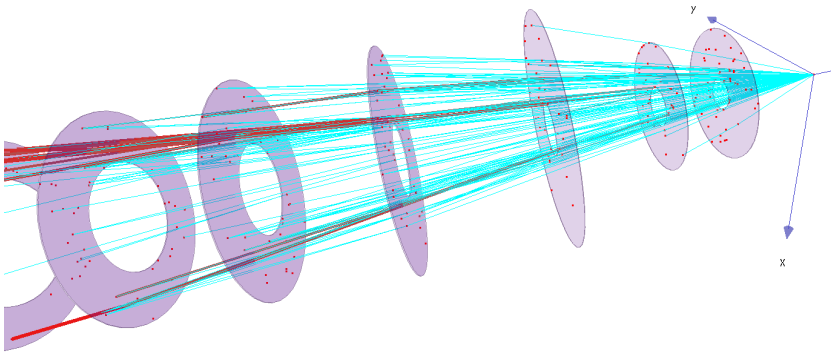
3.) Doing the Cellular Automaton:



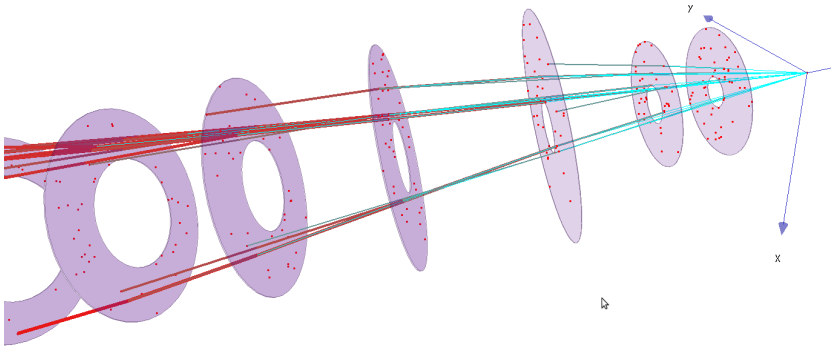
4.) Cleaning:



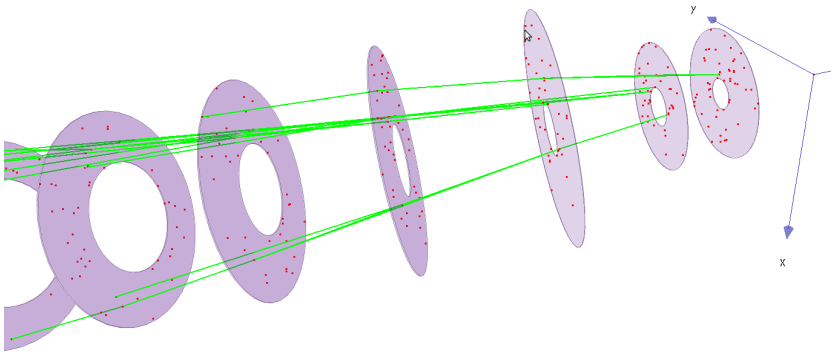
5.) Building 3-segments and doing the Automaton with them:



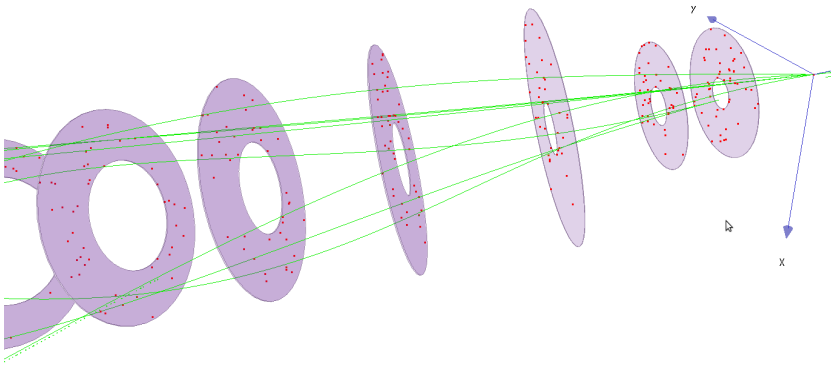
6.) Cleaning again (this time the 3-segments):



7.) Collecting the tracks:



Let's compare once more with the real ones:



We see that we found most of them. They might look a bit different, because the true tracks are displayed as the helix corresponding to the monte carlo particle. That's also the reason, we can't see, why those two tracks in the middle weren't found: multiple scattering! Those two tracks have kinks in them. So we shall not forget, that tracks with kinks are likely to be left out in this reconstruction so far. Not only this, but we also assumed, that the tracks origin close to the point of interaction, which is true for most detected particles, but not for all. A pair prodution from a photon decaying between layer 3 and 4 for example would be missed too. With those remarks about the limits of the criteria we used so far I close this little introduction.

Thanks!