3D Interpolation Using Co-Helicity

Olivier St-Martin Cormier
ID:260234106
Email: olivier@cim.mcgill.ca

Center for Intelligent Machines
McGill University

April 25, 2010

Abstract

While conventional methods to infer structure from a point cloud provide great results with man made structures, other, more organic forms can be very hard to describe. This is particularly the case with fiber-like formations such as trees, which do not usually contain long straight line segments. The current project aims to represent trees as a set of helical curve segments. The concept of cohelicity is used to infer cures joining points in the data set. Relaxation labeling is then used to maximize support between neighboring points.

1 Literature Review

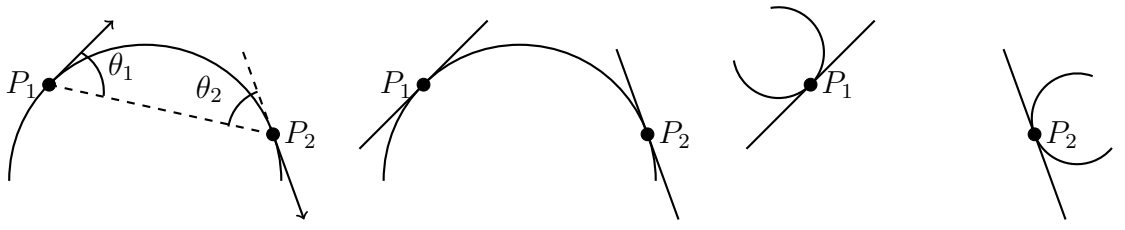
This section presents a brief summary of the papers on which the current project is based on. More emphasis is given to the parts that are directly related to the project. Only the first two sources will be summarized as most of the concepts from the [Savadjiev, Zucker and Siddiqi][3] paper will not be implemented in the current project.

1.1 Trace inference, curvature consistency, and curve detection[1]

This first paper covers three distinct topics, as its title indicates. The first, trace inference, presents the problems related to loss of information during the 2D projection and discretization stages of image acquisition. The authors then discusses the importance of not only inferring information about traces, but also about the tangents and the curvature of these traces. It is mentioned that once these parameters are known, a continuous curve can be synthesized. The text then describes how to extract tangent estimates using a set of convolutions. This first topic, while being very important in most cases, does not have a big influence in the current project.

The second topic is curvature consistency. It details three concepts used to determine whether sets of points belong to a single curve. Those are cocircularity, curvature classes, and lateral maxima.

Cocircularity between a pair of points is used to determine if there exists a circle satisfying the following two conditions: it passes through both points, and it is tangent to the orientation estimates at both points. To check for cocircularity between two points, a line is first drawn to join them and the angles between this line and the orientation estimates are computed. If the magnitude of both angles is the same, the points are said to be cocircular. This concept is shown in figure 1 (a), where points 1 and 2 are cocircular because $|\theta_1| = |\theta_2|$. If curvature information is know at each points, it is possible to determine whether they belong to the same curve or not, even if they appear to be cocircular. Figures 1 (b) and 1 (c) show how a pair of points with the same "spacial configuration" that could easily be cocircular (b) can belong to two distinct curves (c).



(a) Cocircular (b) Cocircular and compatible curvature classes (c) Cocircular but incompatible curvature classes

Figure 1: Distinction between cocircularity and curvature classes

The last presented concept related to curve inference is the use of lateral maxima. It allows to determine the exact sub-pixel position of a curve, thus decreasing the effects of discretization.

The final topic covered by this paper is relaxation labeling. Each pixel in the image has a set of labels consisting mostly of orientation estimates. The goal of relaxation labeling is to adjust the weights of these labels in order for each label to be consistent with labels from neighboring nodes.

The [Parent and Zucker][1] paper presents the basis for curve inference in two dimensions, but as will be seen later, some of these concepts intuitively extend in the third dimension. As the current project aims to detect three dimensional space curves, it will rely on the information from this paper.

1.2 3D Curve Inference for Diffusion MRI Regularization[2]

This second paper extends the cocircularity concept by demonstrating a method to obtain a similar information in three dimensions. Instead of trying to find a circle passing through two points on a plane and tangent to two orientation estimates, one tries to find a helix that passes through three points and tangent to three orientation estimates. A mathematical description of an helix is then presented. An algorithm to determine cohelicity is then detailed. Once three points and their respective tangents are known to be cohelical, the parameters of the helix passing through them can be recovered and the helix can be redrawn. The next part of the text shows that by simply replacing cocircularity by cohelicity, the same relaxation labeling process as described in the [Parent and Zucker][1] paper can be applied to get a set of consistent labels.

This text is quite short, but it addresses a clear problem and goes straight to the point. Most of the concepts contained in this paper will be described in more details in later sections of this report. This is because this paper is the basis for this project.

2 Helix

Helices are smooth 3D curves that lie on the surface of a cylinder and whose tangent at any point has a constant angle with the main axis of the cylinder. A helix is determined by two main parameters: the radius of the cylinder on which it resides, and the distance between two loops, called the pitch and denoted by c. *[distance between two loops = height travelled*

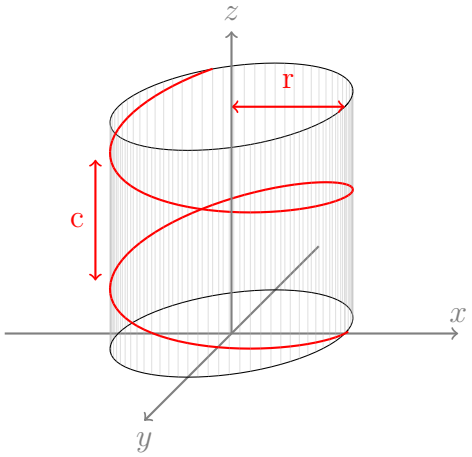


Figure 2: Helix of radius r and pitch c drawn from $\theta_0 = 0$ to $\theta_1 \approx 3.3\pi$

between i and $i + 2\pi$, where i is any real value] A sample helix and its parameters is shown in figure 2. The neat thing about helices is that if one was to project each point of an helix onto a plane perpendicular to the main axis of the helix, a circle would be obtained. This fact will be used in the algorithm to determine cohelicity that will be shown in a later section. The general equation of an helix having the z axis as its main axis is given in equation 1.

$$H(t) = \Big(x(\theta), y(x(\theta), z(x(\theta)\Big) = \Big(r\cos(\theta), r\sin(\theta), c\theta\Big) \tag{1}$$

To determine cohelicity, we will also need the equation of the tangent to the helix. The derivation is shown in equations 6 through 10 and the final result is shown in equation 5.

$$\begin{aligned}
T(t) &= \frac{H'(t)}{\|H'(t)\|} && (2)\\
&= \frac{(-r\sin(\theta), r\cos(\theta), c)}{([-r\sin(\theta)]^2 + [r\cos(\theta)]^2 + c^2)^{1/2}} && (3)\\
&= \frac{(-r\sin(\theta), r\cos(\theta), c)}{(r^2[\sin^2(\theta) + \cos^2(\theta)] + c^2)^{1/2}} && (4)\\
&= \frac{1}{\sqrt{r^2 + c^2}} \cdot \Big(-r\sin(\theta), r\cos(\theta), c\Big) && (5)
\end{aligned}$$

Even if it is not strictly required, the derivation of the normal and the equation of the normal to an helix are shown in equations 6 through 10 for completeness.

$$\begin{aligned}
N(t) &= \frac{T'(t)}{\|T'(t)\|} && (6)\\
&= \frac{(-r\cos(\theta), -r\sin(\theta), 0)}{([-r\cos(\theta)]^2 + [-r\sin(\theta)]^2 + 0^2)^{1/2}} && (7)\\
&= \frac{(-r\cos(\theta), -r\sin(\theta), 0)}{(r^2[\sin^2(\theta) + \cos^2(\theta)])^{1/2}} && (8)\\
&= \frac{(-r\cos(\theta), r\sin(\theta), 0)}{(r^2)^{1/2}} && (9)\\
&= \Big(-\cos(\theta), -\sin(\theta), 0\Big) && (10)
\end{aligned}$$

3 Cohelicity Detection Algorithm

This section will detail the steps required to determine cohelicity as presented in algorithm 1 of [2]. Some steps were slightly modified to adapt the method to the current problem, but the overall process remains very similar. Throughout this section, P_n, v_n

and θ_n for $n \in \{1, 2, 3\}$ will be used to denote the three points, the three tangent vectors, and the angle of the three points respectively. The presentation of this algorithm might seem lengthy, but it represents the biggest part of the project.

3.1 Checking for Collinearity

Collinearity is a special case of cohelicity. It can be represented either by an helix having a zero radius, in which case the cylinder on which the points of the helix lie would be a line, or it can be represented by an helix with an infinite radius, in which case the helix is locally perceived as a straight line. For three points and their respective orientation estimates to be collinear, two conditions need to be checked.

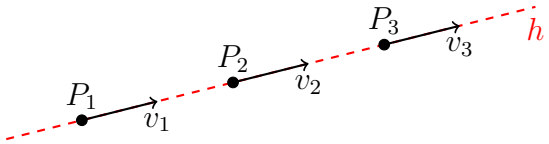


Figure 3: Collinearity between three points

First, as shown in figure 3, the three points need to lie on a single line. The second condition, which can also be seen in figure 3, the angle between the line formed by the three points and the three orientation estimates needs to be zero. Checking for collinearity during the first step prevents trying to compute a possibly infinite radius at a later step, which could create problems *[could also result in division by 0 in some cases]*. It also reduces the amount of computations required.

3.2 Find Main Axis of Putative Helix

As was mentioned earlier, helices have the property that their tangents have a constant angle with their main axis. Thus, the difference between two of them is as shown in equation 13, where $i, j \in \{1, 2, 3\}$ so that $i \neq j$ and points p_i and p_j are distinct, that is, $\theta_i \neq \theta_j$.

$$
\begin{aligned}
v_i - v_j &= \frac{1}{\sqrt{r^2 + c^2}} \cdot \left[\Big(-r\sin(\theta_i), r\cos(\theta_i), c \Big) - \cdot \Big(-r\sin(\theta_j), r\cos(\theta_j), c \Big) \right] && (11) \\
&= \frac{1}{\sqrt{r^2 + c^2}} \cdot \Big(-r\sin(\theta_i) + r\sin(\theta_j), r\cos(\theta_i) - r\cos(\theta_j), c - c \Big) && (12) \\
&= \frac{r}{\sqrt{r^2 + c^2}} \cdot \Big(\sin(\theta_j) - \sin(\theta_i), \cos(\theta_i) - \cos(\theta_j), 0 \Big) && (13)
\end{aligned}
$$

From which we see that the component in the z-direction is zero. Because of this, the cross product of two such differences, as seen in equation 14, is solely in the direction

of the main axis. Note that in equation 14, it is assumed that $i, j, k \in \{1, 2, 3\}$ so that $i \neq j \neq k$ and points p_i, p_j, and p_k are distinct, that is, $\theta_i \neq \theta_j \neq \theta_k$.

$$(v_i - v_j) \times (v_j - v_k) = \frac{r}{\sqrt{r^2 + c^2}} \cdot \Big(0, 0, z\Big) \tag{14}$$

Where z is non-zero and has the form shown in 15. A nicer way to write equation 15 is to set $\hat{\theta}_i = 0$ and assume $\hat{\theta}_j = \theta_j - \theta_i$ and $\hat{\theta}_k = \theta_k - \theta_i$. The reformulated expression is shown in 16. It shows more clearly that $z \neq 0$ if $\theta_i \neq \theta_j \neq \theta_k$. Further simplifications are possible, but not required.

$$z = \Big(\sin(\theta_j) - \sin(\theta_i)\Big) \cdot \Big(\cos(\theta_j) - \cos(\theta_k)\Big) - \Big(\cos(\theta_i) - \cos(\theta_j)\Big) \cdot \Big(\sin(\theta_k) - \sin(\theta_j)\Big) \tag{15}$$

$$z = -\sin(\theta_j)\cos(\theta_k - \sin(\theta_k) + \sin(\theta_j) + \cos(\theta_j)\sin(\theta_k) \tag{16}$$

From this, we see that the main axis of an helix can be found by computing the cross product of the vectors joining the tangent vectors. Only the case of an axis centered around the z axis has been shown, but the method works for helices centered around arbitrary vectors. This can be seen by simply applying a combination of vector rotation and vector translation to the standard helix equation in order to recenter the curve around any normal vector located at any position.

3.3 Find Two Dimensional Projections

Finding the projections of three dimensional vectors onto a two dimensional surface is what allows us to use some of the concepts developed in the [Parent and Zucker][1] paper. To do so, we need the points from the helix to project to a circle and for this reason, the surface onto which the projection is done needs to be perpendicular to the main axis of the helix, this plane will be denoted as π.

The procedure to project a vector onto a surface defined solely by a normal is to first project the vector (denoted as $\overrightarrow{v}$) onto the normal (denoted as $\overrightarrow{n}$) with the use of equation 17, in which $\|\cdot\|$ represents the Euclidean norm. Figure 4 shows graphically how an helix projects to a circle on a plane π.

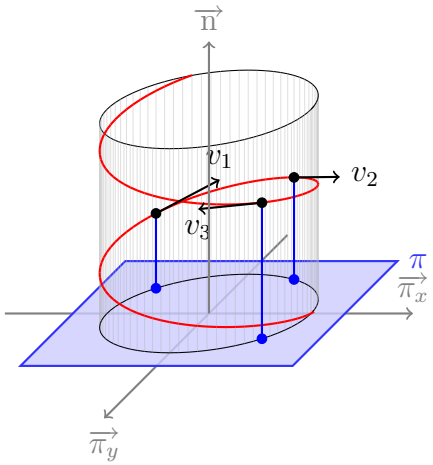


Figure 4: Vector projection onto plane $\pi \perp \overrightarrow{n}$

$$proj_{\overrightarrow{n}}(\overrightarrow{v}) = \frac{\langle \overrightarrow{v}, \overrightarrow{n} \rangle}{\| \overrightarrow{n} \|^2} \cdot \overrightarrow{n} \tag{17}$$

Once this is done, The projection on π is found by taking the difference between the original vector and the projection found previously. The method is summarized in equation 18.

$$proj_{\pi}(\overrightarrow{v}) = \overrightarrow{v} - proj_{\overrightarrow{n}}(\overrightarrow{v}) = \overrightarrow{v} - \frac{\langle \overrightarrow{v}, \overrightarrow{n} \rangle}{\| \overrightarrow{n} \|^2} \cdot \overrightarrow{n} \tag{18}$$

3.4 Find Radius of Putative Helix

Once the three points have been projected to a plane, the problem of finding the radius of the circumcirlce, aptly called the circumradius, of the triangle formed by the three points is a well-posed two dimensional problem that mathematicians have been finding solutions to for centuries. The circumcircle of a polygon is defined as the circle that passes through all of the vertices of the polygon. This circle exists and is unique for all triangles. A diagram showing the concept of a circumcircle is presented in figure 5. The most widely used formula to compute the circumradius is shown in equation 19, in which a, b, and c are the lengths of the three sides of the triangle that can be easily found by the Pythagorean theorem.

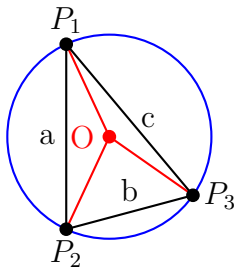


Figure 5: Representation of a triangle (black), its circumcircle (blue) and the circumradii (red)

$$R = \frac{a \cdot b \cdot c}{4 \cdot A_{\triangle}} \tag{19}$$

In order to use equation 19, one first needs to know $A_{\triangle}$, the area of the triangle. Again, there are multiple ways to find this value, but a nice way to do it is to use Heron's formula, shown in equation 20, where s is the semiperimeter (half the perimeter) of the triangle, as shown in equation 21.

$$A_{\triangle} = \sqrt{s(s-a)(s-b)(s-c)} \tag{20}$$

$$s = \frac{a+b+c}{2} \tag{21}$$

Using these equations to find the circumradius is efficient because it only requires knowledge of the Euclidean length of the sides of the triangles, and so no trigonometric functions are required.

3.5 Find Pitch of Putative Helix

The first step to finding the pitch of the helix is to find the angle between points 1 and 2. To find this, we can use the definition of vector dot product shown in equation 22, where a and b are vectors, and θ is the angle between them.

$$\langle a, b \rangle = |a| \cdot |b| \cdot \cos(\theta) \tag{22}$$

Equation 23 shows how to find the angle between two points.

$$\theta_2 - \theta_1 = \arccos\left(\langle \frac{proj_\pi(\overrightarrow{v_1})}{\|proj_\pi(\overrightarrow{v_1})\|}, \frac{proj_\pi(\overrightarrow{v_2})}{\|proj_\pi(\overrightarrow{v_2})\|} \rangle\right) \tag{23}$$

Now, to find an expression for the pitch, we will first expand the dot product of the tangents at point 1 and 2. This can be seen in equation 24.

$$\begin{aligned} \langle \overrightarrow{v_1}, \overrightarrow{v_2} \rangle &= \frac{1}{r^2 + c^2} \cdot \left(r^2 \cdot \sin(\theta_1) \cdot \sin(\theta_2) + r^2 \cdot \cos(\theta_1) \cdot \cos(\theta_2) + c^2 \right) \\ &= \frac{r^2 \cdot \cos(\theta_2 - \theta_1) + c^2}{r^2 + c^2} \end{aligned} \tag{24}$$

Isolating c in equation 24 yields an expression that can be used to find the pitch, which is shown in equation 25.

$$c = \sqrt{\frac{r^2 \cdot (\langle \overrightarrow{v_1}, \overrightarrow{v_2} \rangle - \cos(\theta_2 - \theta_1))}{1 - \langle \overrightarrow{v_1}, \overrightarrow{v_2} \rangle}} \tag{25}$$

3.6 Check that Orientation Estimates are Tangent to the Putative Helix

Once all of the parameters of the helix are known, it is trivial to find the expected value of the tangent at the three points being considered by using equation 5, derived earlier. We can then verify that the three orientation estimates are indeed tangent to the Putative Helix.

If they are indeed tangent to the helix, we have found that the three points are cohellical, and we have also found all of the parameters of the helix. If it is not the case, or if any of the previous steps failed, then the three points and their respective orientation estimates are not cohellical. With this information, it should be possible to draw the helix, if needed.

4 Relaxation Labeling

At each point, a set of labels are present, with each label representing an orientation estimate. Each of these labels has an associated "confidence" value $p_i(\lambda)$, that is, a weighting factor to represent the importance of the label with respect to other labels at the same point. The values of these "confidence" values are normalized so that their sum equals one. As the input data contains only one orientation estimate at each point, there is only one label at each node, and thus the relaxation labeling problem is greatly simplified. As we are working with cohellicity instead of cocircularity, the support function found in the [Parent and Zucker][1] paper need to be changed a bit. According to the [Savadjiev, Campbell, Pike and Siddiqi][2] paper, the support function

must be changed to that shown in equation 11, where $p_n(\lambda^{(m)})$ is the probability of label $\lambda^{(m)}$ at point n and $r_{ijk}(\lambda, \lambda', \lambda'')$ is the cohellical compatibility between the three labels at the three points.

$$s_i(\lambda) = \sum_{j,\lambda'} \sum_{k,\lambda'} r_{ijk}(\lambda, \lambda', \lambda'') p_j(\lambda') p_k(\lambda'') \tag{26}$$

5 Input data

The input data for this project is a point cloud representing a tree. This data set was obtained by scanning a tree with a LIDAR scanner. Data from multiple scan angles were merged together to get a 360 degrees view of the tree. The data was also preprocessed using a technique explained in the [Harrison][4] paper in order to obtain a Frenet-Serret frame at each point, that is, a tangent, normal and binormal vector is defined at each point. As the data set contains 30947 distinct points, the amount of memory required to load the data is quite manageable. These steps are not part of the implementation of the project, but the program assumes that all of this information is present in the input data. An initial plotting of the input data, presented in figure 6 reveals a clear tree-like structure. The most important thing that can be noticed is that a large part of the point cloud represents the ground plane. From this, we can expect that quite a few of the points on the ground should be detected as collinear by the algorithm presented earlier.

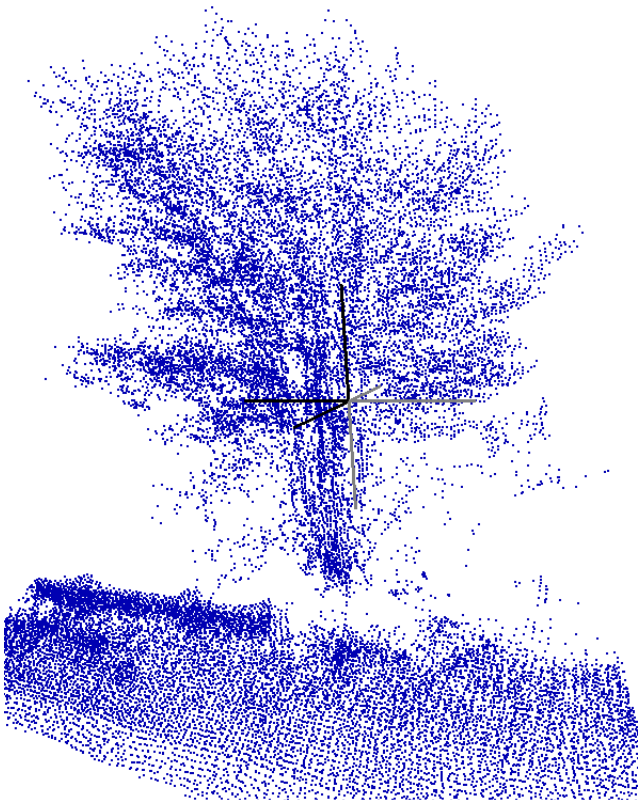


Figure 6: Plot of the input data

If one has no prior knowledge of the settings in which the data was captured, one of the following possibilities should be expected:

- The tree was scanned during the winter season – The tree has no leaves and thus the helicoidal curves are representation of the branches. The result should resemble a classic tree structure, and branching should be detectable.

- The tree was scanned during the summer season – The tree has leaves, and thus most of the points are positioned on the surface of leaves instead of branches. The

result of this case would probably be a a set of helices enclosing a 3D region of space.

- The tree was scanned during either the spring or autumn season – The tree has some leaves, this is a combination of the above two scenarios.

6 Implementation

Efficiency was a key factor in the implementation choices. The [Savadjiev, Campbell, Pike and Siddiqi][2] paper suggests precomputing the possible cohelical arrangements in a circular neighborhood, and then use this precomputed data as a look-up table to determine cohellicity. As the goal of the project was not to run the algorithm in real-time, but rather to simply implement it, the decision was made not to implement such a look-up table.

The goal was still to minimize run-time, but to compute everything on-line. For this reason, the program was written in the c language, as it is very low-level and thus allows to control the computer's resources more efficiently than possible with numerical programming languages such as the one presented by matlab.

To speed up run-time, whenever multiple actions need to be performed (ie, running the algorithm on multiple point triplets), the program forks into different threads to capitalize on the multi-core architecture of modern computers. The number of threads can be set at compile time, this allows to adjust the program for different types of processors.

It was chosen that when checking for cohellicity support, only the points within a certain neighborhood around the initial point would be considered. The neighborhood size that was chosen is a sphere of radius one. This is an arbitrary choice that greatly depends on the scale of the data. In the present case, the whole tree fits in a cube of side six, so considering a sphere of radius one is quite sufficient.

6.1 Display

[Figures for this part are shown on a separate page at the end]

In order to display results, the OpenGL graphic engine was chosen. This choice is mostly due to the fact that it is one of the simplest three dimensional graphic library to use. Also, it provides programmers with a comprehensive documentation library to ease development. All of the figure that will be presented in this section are shown twice, as viewed from two different vantage points. Showing two different representations of each scene helps to perceive the third dimension. To help visualizing the coordinate system, three colored line segments are drawn. The red segment represent the x axis, the green represents the y axis and the blue represents the z axis. *[Please note that the green axis is hard to show as it is located on the green normal plane.]*

To make sure that the algorithm was producing good results, it was tested on a set of sample helices. Also, each step of the algorithm was initially tested individually. It

can be seen from figure 7 that the normal vector (shown in pink/purple [0xFF00FF]) and the normal plane (shown in green) of a sample helix can be correctly computed.

Next, figure 8 shows the projection of the sample helix onto the normal plane (normal plane not shown). The blue points are the initial data points and the blue lines represent the initial tangents estimates. The solid red lines and points represent the projections and the dashed red lines are simply lines connecting the original points and vectors to their projection. These lines are drawn to easily visualize the orthogonal nature of the projections with respect to the normal plane.

Figure 9 shows that once the parameters of the helix are known, it is possible to draw a smooth helical curve passing through the three points. As there is no smooth curve plotting method supplied by the OpenGL library, one must draw a strip of line segments to simulate smoothness. While this is not a problem when drawing a single helix as in figure 9, it quickly becomes too expensive to draw smooth curves when a few thousand helices need to be drawn. This is why the final program draws straight lines between cohelical points in the final version of the program.

7 Results

Running the algorithm without relaxation labelling on the input data gives a somewhat messy result, as can be seen in figure 10(a). If one was to start thresholding to show only helices with a length greater than a certain amount, one can see that large helices are found on the ground plane. This makes quite a bit of sense considering that most of the points associated with the ground are collinear. This thresholding is shown in figure 10(b).

By using relaxation labelling, we expect to see a less noisy output as the helices formed by orientation estimates with a small support should disappear. This, however, was not the case. As seen in figure 11, the benefits of relaxation labeling are minimal, if existent. Inspecting figure 11(a) closely reveals that the results are somewhat worse than the ones in figure 10(a). Figure 11 was generated after running only a couple of iterations of relaxation labeling, but as the results have not improved, one cannot expect much better results after more iterations. There is no clear reason why relaxation labeling did not work as it should, it might be due to an implementation problem, but multiple reimplementation seemed to yield similar results.

8 Conclusion

Overall, the results were satisfying. It seems that when the tree was scanned, there were leaves on it, but not so much as to prevent the detection of branches, which can be clearly seen in figure 10.

References

1 Pierre Parent and Steven W. Zucker, *Trace inference, curvature consistency, and curve detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence 1989

2 Peter Savadjiev and Jennifer S. W. Campbell and G. Bruce Pike and Kaleem Siddiqi, *3D Curve Inference for Diffusion MRI Regularization.* 2005

3 Peter Savadjiev and Steven W. Zucker and Kaleem Siddiqi, *On the Differential Geometry of 3D Flow Patterns: Generalized Helicoids and Diffusion MRI Analysis.* International Conference on Computer Vision (ICCV) 2007

4 John Harrison, *Efficient Initial Segmentation of Point-Cloud Data.* Unpublished 2009

Figures

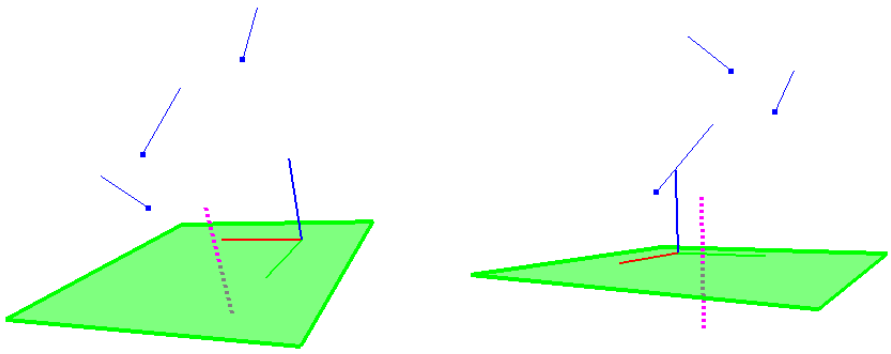


Figure 7: Normal vector and normal plane to three points and respective orientation estimates

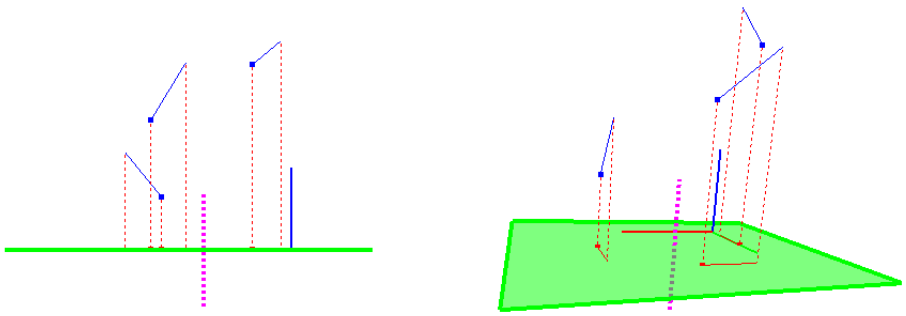


Figure 8: Projection of points and vectors onto normal plane

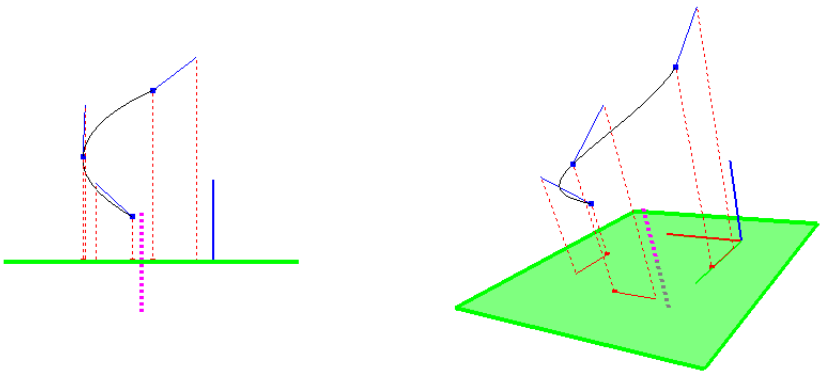
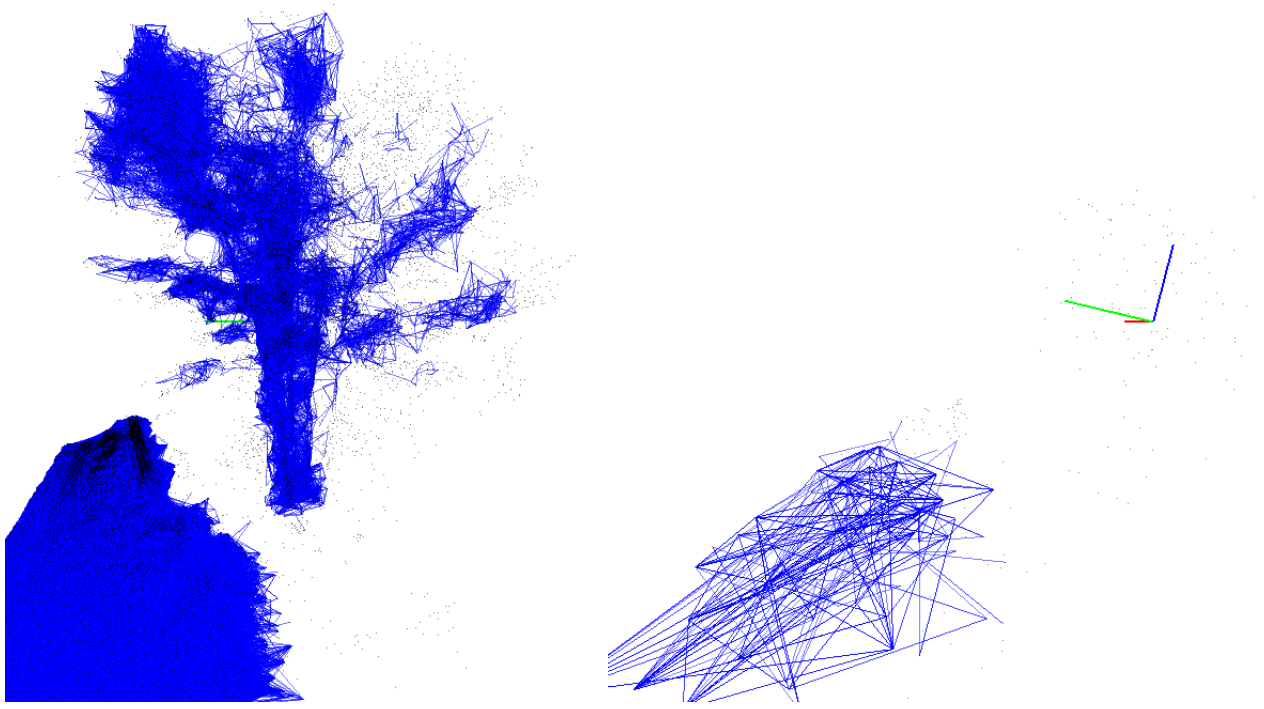


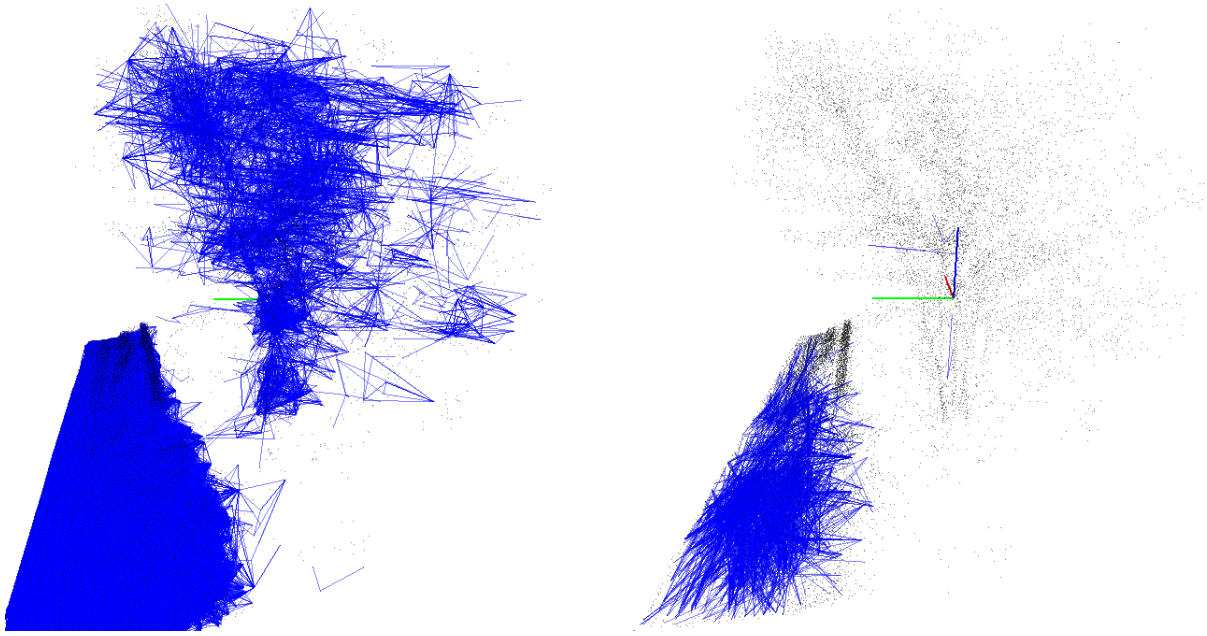
Figure 9: Redrawing of the helix



(a) Initial Helices

(b) Helices lying on the ground plane

Figure 10: Results without relaxation labelling



(a) Initial Helices (b) Helices lying on the ground plane

Figure 11: Results with relaxation labelling