The Single Helix

R. Robert Hentzel
Scott Williams
Iowa State University
Ames, IA 50011
topquark@iastate.edu
williams@ttd.teradyne.com

Advisor: Stephen J. Willson

Introduction

We present an iterative algorithm that finds all points of intersection between a finite, infinitely thin helix and an infinite plane, ordered along the helix. The algorithm has constant space complexity and has time complexity proportional to the number of intersections and the logarithm of the desired precision.

Assumptions

- The helix is of finite length.
- The helix is infinitely thin.
- The plane is infinite in extent.
- The helix has constant radius.
- No more than one helix needs to be considered simultaneously; if multiple helices must be modeled, the routine may be run sequentially on each.

Input and Output

Input to the program consists of:

- the endpoints of the central axis of the helix,
- one point which is on the helix,
- the winding number of the helix,
- the handedness of the helix,

- the specification of a plane, and
- the desired precision of the solutions.

The *winding number* of a helix is the number of times that a point traveling on the helix makes a complete circle while its projection on the helix axis advances by one unit. In other words, it is the number of coils of the helix contained in one distance unit parallel to its axis. **Figure 1** illustrates a helix with a winding number of 2.0.

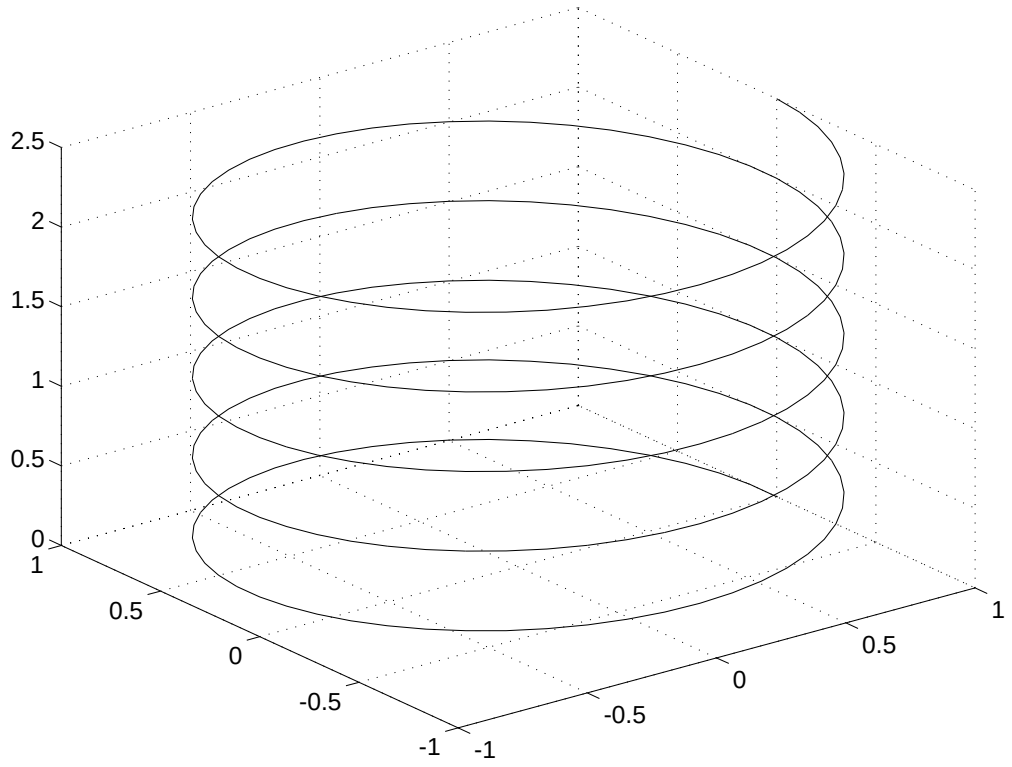

Figure 1. A helix with winding number of 2.0 (note the axes!).

Output consists of an ordered list of intersections, giving for each the distance along the axis of the helix, the coordinates of the intersection point, and the distance from the helix point to the plane (which will be less than the prescribed precision.

Helicoidal Normal Form

We will do all of our computations with geometric figures (helices and planes) which have been put into *helicoidal normal form* (HNF), described below. Doing so takes advantage of the symmetries of helices and of the infiniteness of planes to simplify the resulting calculations and increase the speed of calculating intersection points. The transformation is invertible,

so the coordinates of intersection points obtained may be transformed back into the original system.

Definition of Helicoidal Normal Form

In HNF, helices and planes have the following properties:

- The axis extends from $(0, 0, 0)$ to $(0, 0, 2\pi\eta L)$, where L is the length of the original helix and η is the original winding number of the helix.
- The point $(1, 0, 0)$ is on the helix; that is, the coordinate system is oriented so that the lowest point on the helix is on the x-axis and the radius is unity.
- The winding number of the helix is $1/2\pi$.
- The plane's normal vector is normalized to unit length.
- The helix is right-handed.
- The z-component of the plane's normal vector is nonnegative.

Thus, a normalized helix may be parameterized as

$$r(t) = 1, \qquad \theta(t) = t, \qquad z(t) = t$$

in radial coordinates or as

$$x(t) = \cos t, \qquad y(t) = \sin t, \qquad z(t) = t$$

in Cartesian coordinates. The equation of a plane is given by

$$ax + by + cz + d = 0,$$

with the triple (a, b, c) representing a unit normal vector, so that

$$a^2 + b^2 + c^2 = 1 \qquad \text{and} \qquad c > 0,$$

as per the definition of the helicoidal normal form.

Transformation to Helicoidal Normal Form

The helix and plane are specified by the user with the following data:

- the endpoints of the symmetry axis of the helix:
$$\vec{x}_0 = (x_0, y_0, z_0), \qquad \vec{x}_1 = (x_1, y_1, z_1);$$
- any point on the helix, $\vec{p} = (x_p, y_p, z_p)$;

- the winding number of the helix, $\eta > 0$;
- any (a, b, c, d) quadruplet representing the plane, so that the normal vector $\vec{n}$ is (a, b, c); and
- the handedness of the helix.

The transformation to HNF consists of seven steps:

1. translation of one axis endpoint to the origin,
2. rotation of coordinates to bring the other endpoint to the z-axis,
3. rotation of coordinates to eliminate any initial phase of the helix,
4. space inversion to ensure right-handedness of helix,
5. scaling of coordinates to normalize the helix radius,
6. scaling of coordinates to normalize the helix winding number, and
7. normalizing the normal vector of the plane.

The details of each step follow.

Translation of one axis endpoint to the origin

We translate the coordinate system to bring x_0 (one end of the helix's symmetry axis) to $(0, 0, 0)$:

$$\begin{aligned} \vec{x}_0 &\leftarrow \vec{x}_0 - \vec{x}_0 = \vec{0} \\ \vec{x}_1 &\leftarrow \vec{x}_1 - \vec{x}_0 \\ \vec{p} &\leftarrow \vec{p} - \vec{x}_0. \end{aligned}$$

We can find the effect on the representation of the plane as follows. If $\vec{x}$ is originally on the plane, then

$$\vec{n} \cdot \vec{x} + d = 0,$$

so the translated point $\vec{x} - \vec{x}_0$ satisfies the condition

$$\vec{n} \cdot (\vec{x} - \vec{x}_0) + (d + \vec{n} \cdot \vec{x}_0) = 0.$$

Accordingly, we transform

$$d \leftarrow d + \vec{n} \cdot \vec{x}_0.$$

Rotation of coordinates to bring the other endpoint to the z-axis

Taking

$$\theta = \arctan(x_1/y_1), \qquad \phi = \arctan\left(\frac{\sqrt{x_1^2 + y_1^2}}{z_1}\right),$$

we can construct the rotation matrix

$$R = R_{yz}R_{xy} = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

which can be applied to bring x_1 to coincide with $(0, 0, L)$, where L is the original length of the helix, given by

$$L = \sqrt{\vec{x}_1 \cdot \vec{x}_1}.$$

That is, we transform

$$\vec{x}_0 \leftarrow R\vec{x}_0 = \vec{0}, \qquad \vec{x}_1 \leftarrow R\vec{x}_1 = (0, 0, L), \qquad \vec{p} \leftarrow R\vec{p}.$$

We can find the effect on the representation of the plane as follows. If $\vec{x}$ were originally on the plane, then

$$\vec{n} \cdot \vec{x} + d = 0.$$

Now the rotated point $R\vec{x}$ satisfies the condition

$$R\vec{n} \cdot R\vec{x} + d = 0,$$

so we make the transformation

$$\vec{n} \leftarrow R\vec{n}.$$

Rotation of coordinates to eliminate any initial phase

We now look at the point p on the helix and note its parametric representation as

$$r_p = \sqrt{x_p^2 + y_p^2}, \qquad \theta_p = \arctan(y_p/x_p), \qquad z_p = z_p.$$

Thus, in going downward along the helix's axis from $z = z_p$ to $z = 0$ (the bottom end), we will go through ηz_p rotations, bringing the initial angular position of the helix (θ_0) to

$$\theta_0 = \theta_p - 2\pi\eta z_p.$$

We wish this to coincide with the x-axis, so we apply an additional counterclockwise rotation of $-\theta_0$ about the z-axis to $\vec{x}_1, \vec{x}_2, \vec{p}$ and the representation of the plane, using the same techniques as in the previous section.

Scaling of coordinates to normalize helix radius

We can now parameterize our helix as

$$x(t) = r\cos t, \qquad y(t) = r\sin t, \qquad z(t) = t,$$

with $r = r_p = \sqrt{x_p^2 + y_p^2}$. We note that if any point $\vec{x}(t)$ coincides with the plane, then

$$a \cdot r\cos t + b \cdot r\sin t + ct + d = 0;$$

so the radius-normalized point $(\cos t, \sin t, t)$ satisfies

$$(ar)\cos t + (br)\sin t + ct + d = 0,$$

and we can make the transformation

$$a \leftarrow ar, \qquad b \leftarrow br, \qquad r \leftarrow 1.$$

Space inversion to ensure right-handedness of helix

If the helix is originally left-handed, it can be made right-handed by effecting a spatial inversion of the coordinate system about the xz-plane. This is accomplished by negating the y-coordinate of the plane's normal vector:

$$b \leftarrow -b, \qquad \text{handedness} \leftarrow \text{right}.$$

Scaling of coordinates to normalize helix winding number

The final parameter to normalize is the helix winding number. We do this by forcing the helix to advance one rotation per 2π advance along its axis. To compensate, we scale our helix.

If the helix originally advanced η turns per unit axis advance and had a length of L, then the same number of turns will be made in a length of $2\pi\eta L$ with a winding number of $1/2\pi$. Thus:

$$L \leftarrow 2\pi\eta L, \qquad \eta \leftarrow \frac{1}{2\pi}.$$

We can find the effect on the representation of the plane as follows. If a point $\vec{x} = (x, y, z)$ is originally on the plane, then

$$ax + by + cz + d = 0.$$

Then, since the transformed point $(x, y, \eta z)$ fulfills the condition

$$ax + by + \frac{c}{2\pi\eta}(2\pi\eta z) + d = 0,$$

we can make the transformation:

$$c \leftarrow \frac{c}{2\pi\eta}.$$

Normalizing the normal vector of the plane

We make the transformations

$$a \leftarrow \frac{a}{\sqrt{a^2+b^2+c^2}}, \qquad b \leftarrow \frac{b}{\sqrt{a^2+b^2+c^2}},$$
$$c \leftarrow \frac{c}{\sqrt{a^2+b^2+c^2}}, \qquad d \leftarrow \frac{d}{\sqrt{a^2+b^2+c^2}}.$$

If $c < 0$, we negate each component of the normal vector (but not d).

Locating Intersections

Transcendental Lemma

We need solutions of the transcendental equation

$$f(t) = a\cos t + b\sin t + c = 0$$

subject to the restriction that $a^2 + b^2 + c^2 = 1$. We observe that

$$\begin{aligned}
&\sqrt{a^2+b^2}\sin\left(t+\arctan\frac{b}{a}\right)\\
&\quad= \sqrt{a^2+b^2}\left[(\sin t)\left(\cos\arctan\frac{b}{a}\right)+(\cos t)\left(\sin\arctan\frac{b}{a}\right)\right]\\
&\quad= \sqrt{a^2+b^2}\,(\sin t)\left(\frac{a}{\sqrt{a^2+b^2}}\right)+\sqrt{a^2+b^2}\,(\cos t)\left(\frac{b}{\sqrt{a^2+b^2}}\right)\\
&\quad= a\sin t + b\sin t.
\end{aligned}$$

So, we attack the original problem as:

$$\begin{aligned}
\sqrt{a^2+b^2}\sin\left(t+\arctan\frac{b}{a}\right) &= -c,\\
\arcsin\left(\sin\left(t+\arctan\frac{b}{a}\right)\right) &= \arcsin\left(\frac{-c}{\sqrt{a^2+b^2}}\right),\\
t &= \arcsin\left(\frac{-c}{\sqrt{1-c^2}}\right)-\arctan\left(\frac{b}{a}\right).
\end{aligned}$$

We note:

- If $c > \sqrt{2}/2$, then the initial $\arcsin$ will have an argument larger than unity and will be nonexistent.
- The $\arcsin$ function will yield two distinct values in $[0, 2\pi)$, both of which must be checked.
- If t is a solution of $f(t) = 0$, then $t + n2\pi$ must also be a solution for integers n.

Difference Function

The phrase "the (helix) point t" will be taken to mean the helix point parameterized by t, namely $(\cos t, \sin t, t)$ for $t \in [0, L]$. Since all points $\vec{x}$ on the plane satisfy

$$ax + by + cz + d = 0,$$

we see that finding the points of intersection between the helix and the plane amounts to finding t such that

$$f(t) = a\cos(t) + b\sin(t) + ct + d = 0$$

for $t \in [0, L]$.

We name f our *difference function* since it provides a measure of the distance between the helix and the plane for a given t. In fact, $f(t)$ is the perpendicular distance from the helix point t to the plane. We will eventually seek intersections by attempting to minimize the absolute value of $f(t)$.

Slope of the Difference Function

The slope of the difference function at the helix point t is given by:

$$f'(t) = \left.\frac{df(x)}{dx}\right|_{x \leftarrow t} = -a\sin t + b\cos t + c$$

We note:

- For $c > \sqrt{a^2 + b^2}$ (which is equivalent to $c > \sqrt{2}/2$), the slope is everywhere positive, so the difference function is monotonically increasing.
- The form of the slope function is that studied in the **Transcendental Lemma** above, so we know that we can find zeros of the slope function, and hence local extrema of the difference function, analytically. Here the two values for the $\arcsin$ correspond to a local maximum and a local minimum of the distance function.

Upper and Lower Bounds

It is easy to find bounds on t that contain all of the possible intersection points. At an intersection point, we have

$$a\cos t + b\sin t + ct + d = 0,$$

so

$$t = \frac{-a\cos t - b\sin t - d}{c}.$$

Simple calculus, plus using $c = \sqrt{a^2 + b^2}$, $|\cos t| \leq 1$, and $|\sin t| \leq 1$, shows that

$$-\sqrt{a^2 + b^2} \leq -a\cos t - b\sin t \leq \sqrt{a^2 + b^2},$$

which provides bounds on t:

$$\ell_b \equiv \frac{-\sqrt{a^2 + b^2} - d}{c} \leq t \leq \frac{\sqrt{a^2 + b^2} - d}{c} \equiv u_b.$$

All intersection points t must lie within these bounds.

If $c = 0$ (the discontinuity in the above formulae), the plane runs exactly parallel to the axis of the helix and would intersect an infinite helix either infinitely often (inside the radius) or never (outside the radius). In this case, ℓ_b may be taken to be $-\infty$ and u_b to be $+\infty$, since both values will be truncated by the finiteness of the helix, as mentioned in the next subsection.

Intervals and Subintervals

We define the *search interval lower bound* ℓ_B to be the larger of 0 (one end of the helix) and ℓ_b. We define the *search interval upper bound* u_B to be the smaller of L (the other end) and u_b. All intersections must lie in the *search interval* $[\ell_B, u_B]$.

We divide the interval $[\ell_B, u_B]$ into subintervals, broken by the local extrema, both maxima and minima. Each subinterval, except the leftmost and rightmost, is bounded by adjacent local extrema. The leftmost and rightmost are bounded on their "inner" side by a local extremum and on their "outer" side by either ℓ_B or u_B. We can do this because we can apply the **Transcendental Lemma** to the slope function to calculate two local extrema, one minimum and one maximum. Since the spacings of minima and maxima are both 2π, we can now locate all local extrema within the search interval by repeated addition/subtraction of 2π from the original maximum and minimum (see **Figure 2**).

It may be the case, if $c > \sqrt{2}/2$, that there are no local extrema. Then there is only one subinterval, $[\ell_B, u_B]$, which contains the only intersection. (There can be only one intersection, since f is monotonically increasing in this case.)

Searching the Subintervals

We now consider each subinterval separately. We let ℓ_t and u_t represent the initial left and right endpoints of the subinterval. We evaluate $f(\ell_t)$ and $f(u_t)$ and compare the signs. If the signs agree, then there *cannot* be an intersection in the subinterval; if there were, there would have to be a local extremum *within* the subinterval, but all local extrema fall on subinterval boundaries by construction. However, if the signs are different, there *must*

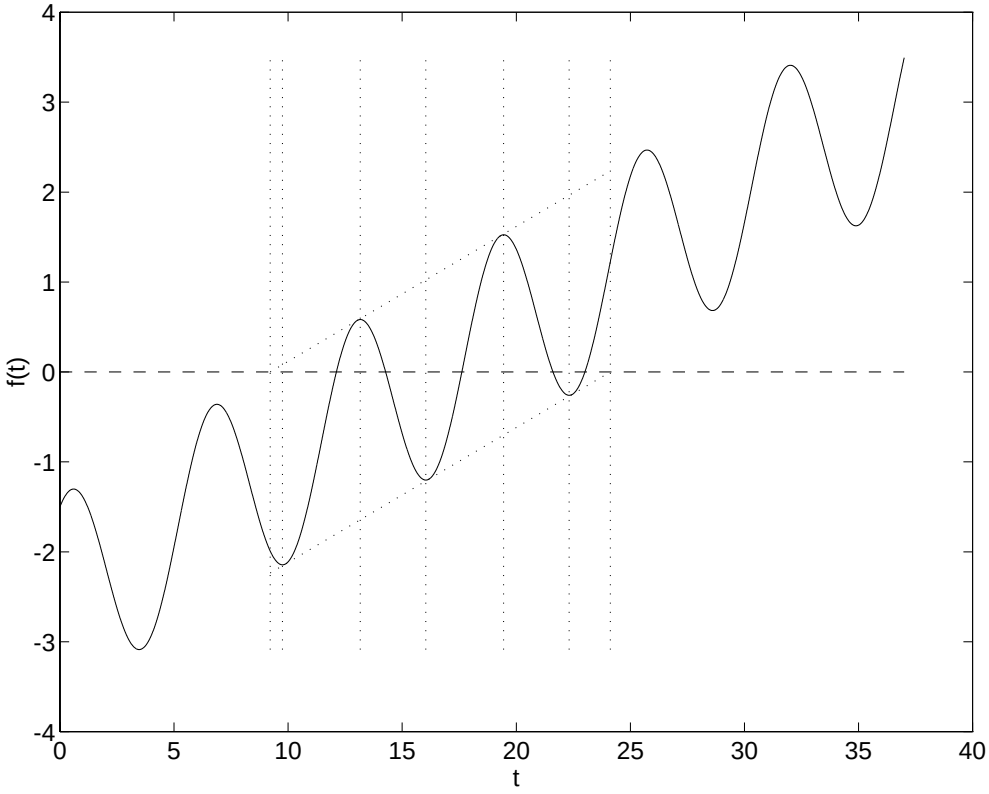

Figure 2. Difference function $f(t)$ with search interval limits shown as the outermost dotted vertical lines, zero shown as a dashed horizontal line, and subinterval boundaries shown as dotted vertical lines. Note the small(er) subintervals at the ends.

be an intersection point, by application of the Intermediate Value Theorem to the continuous function f, which is positive at one endpoint and negative at the other.

If endpoints show that no intersection can exist (i.e., both have the same sign), then the subinterval is immediately discarded (this can occur only with the leftmost and rightmost subintervals). Otherwise, we begin a binary search for the intersection point, provided that the endpoint itself is not a point of intersection within the desired precision.

We evaluate f at the midpoint of the subinterval $(\ell_t + u_t)/2$ and compare its sign with the endpoints. By the Intermediate Value Theorem, the intersection must lie between the midpoint and whichever endpoint differs in sign from it (they cannot *both* differ, since they have different signs). We can therefore define a new subinterval equal to either the left or right half of $[\ell_t, u_t]$ as appropriate and repeat this process. Eventually, evaluation of f at the midpoint must yield a number within the prescribed accuracy from zero. At this point, the binary search terminates, having located the intersection with sufficient precision.

We note that we apply the binary search *only* in subintervals that are guaranteed to contain an intersection by virtue of the Intermediate Value

Theorem.

When intersections are found, the proper inverse transformations are performed to return the point to its original coordinate system, and it is displayed.

After completing a subinterval, processing proceeds with the next and terminates following the rightmost. Because of the left-to-right processing of subintervals, the intersections are generated *in order* of increasing t along the helix axis.

Space and Time Complexity

Space Complexity

In our implementation, each subinterval is calculated and searched subsequently, obviating the need to store large arrays of points found or of edges of subintervals. Thus, the storage space needed is independent of the input. For our implementation, it is approximately 300 bytes.

Time Complexity

The time to put the helix and plane into HNF is independent of the input and, in the current implementation, involves approximately 118 floating-point multiplies/divides and 36 floating-point trigonometric functions and square roots. Floating-point adds, subtracts, and comparisons take negligible time relative to multiplication, division, square roots, and trigonometric function calls.

The time per intersection is 40 floating-point multiplications/divisions and 8 complicated functions, plus 4 multiplications and 3 complicated functions per iteration required. The largest that a subinterval may be is 2π, so to divide this by halving into a slice as small as the desired accuracy ϵ requires $\log_2(2\pi/\epsilon)$ steps. Thus, for each intersection, $40 + 4\log_2(2\pi/\epsilon)$ multiplications and $8 + 3\log_2(2\pi/\epsilon)$ complicated functions are required. For an accuracy of one part in 10^6, this amounts to 131 multiplications and 76 complicated functions per intersection. These are absolute bounds.

There are sufficiently few operations that on a DECstation 5000/240, the processing can be carried out in a tiny fraction of a second for as many as 200 intersection points (we did not test larger numbers).

We believe that this is sufficiently fast to be used in modeling large numbers of helices or as part of a real-time rendering engine.

Algorithm Analysis

Strengths

- The algorithm requires little memory.
- The algorithm is guaranteed to find all intersection points within the desired precision, unless this is finer than the machine's floating-point precision.
- Intersections are generated quickly enough for real-time applications.
- The algorithm requires time linear in the number of intersection points and logarithmic in the desired precision. Hence, an order-of-magnitude increase in accuracy costs only 12 multiplications and 9 complicated functions per intersection.

Weaknesses

- The algorithm does not take previous intersections into account when searching a subinterval; there is evidence to think that doing so could provide a moderate increase in speed, perhaps 15%.
- The number of multiplications necessary per intersection can be cut by 18, by combining the three separate rotation matrices into a single one.
- The algorithm currently weights each endpoint equally when choosing a new subinterval during the binary search. Using a linear interpolation that weighted each endpoint by the value of f evaluated at that endpoint could increase speed significantly, because of the smoothness and near-linearity of f.