On the Segmented Helix Formed by Stacking Similar Objects

Robert L. Read *

Founder, Public Invention, an educational non-profit.

January 30, 2019

Abstract

In nature, helical structures arise when identical structural subunits combine sequentially, the orientational and translational relation between each unit and its predecessor remaining constant. A helical structure is thus generated by the repeated action of a screw transformation acting on a subunit.[1]

Contents

1 Introduction **2**

2 A Warm-up: Two Dimensions **2**
2.1 The Inscribed Figure 3

3 The Segmented Helix **3**
3.1 The Interior of a Shelix 5

4 The Intrinsic Properties of Repeated Units **5**
4.1 The 4-point Algorithm (Kahn's Method) 6
4.2 The Face Normal Method 7

5 Checks and Explorations **10**

6 The up-vector **10**

7 Applying to The Boerdijk-Coxeter Tetrahelix **11**

8 Implications **12**

9 Applied to Periodic Regular Simplex Chains **13**

*read.robert@gmail.com

10 Relating to the Screw Transform **13**

11 Future Work **13**

12 References that need to be studied or reviewed **14**
12.1 To Be Done 15

13 Acknowledgements **15**

A Mathematica Code **16**

1 Introduction

During the Public Invention Mathathon of 2018[2], software was created to view chains of regular tetrahedra joined face-to-face in chains. The participants noticed that whenever the rules for which face to add the next tetrahedron to were periodic, the resulting chain was always a helix.

Although unknown to us at that time, we now call Lord's Observation:

> In nature, helical structures arise when identical structural subunits combine sequentially, the orientational and translational relation between each unit and its predecessor remaining constant.[1]

The purpose of this paper is to provide mathematical tools and software for studying the segmented helices generated in this way.

The fundamental method of finding the properties of a segmented helix from two segments on the helix, which we call Kahn's method[3], is explained and implemented in Mathematica and an interactive, 3D rendering website written in JavaScript which allows both calculation and interactive play and study, with an aim to allow a helix to be designed from an object or an object designed to produce a particular helix. The method is re-utilized allowing the specification of joint normals and the joint twist, properties intrinsic to a single object and the joining rule. The relationship between these values are explored graphically. Finally, we use these tools to produce a table of all possible segmented helices generated by vertex-matched face-to-face joinings of the Platonic solids.

2 A Warm-up: Two Dimensions

Considering the problem in two dimensions may be a valuable introduction. Suppose that we consider a polygon that that as two edges, called A and B, and that we define the length L of the polygon as the distance between the midpoints of these edges. Suppose that we are only allowed to join these polygons by aligning A of one polygon to B of another polygon, with their midpoints coincident. Let us further assume that we disallow inversions of the polygon. Let us imagine that we have a countable number of polygons P_i indexed from 0. Then what shapes can we make by chaining these polygons together?

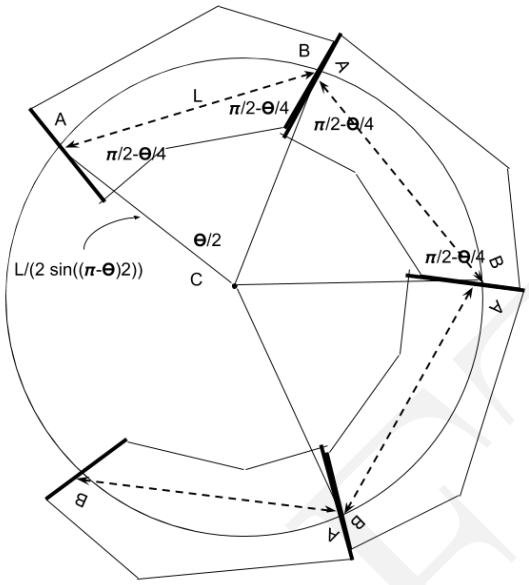

Figure 1: A 2D Analog of a Helix Generated by Repeated Subunits

Each joint J_i between polygons P_i and P_{i+1} will place the axes of at the same angle, θ, since our polygons do not change shape. Let us define θ to be positive if we move anti-clockwise from P_i to P_{i+1} and negative if we move clockwise. If $\theta = 0$, the joints will be collinear.

If $\theta \neq 0$, it seems they polygon joints will always lie on a circle. A proof of this is that each polygon has associated with it an isoceles triangle A, B, C, where $\angle CBA = \angle CAB = \theta/2$, and $\angle ACB = (\pi - \theta)$. AC and BC are not necessarily aligned with an edge of the polygon. The length AB is L, and the lengths AC and BC are $(L/2)/\sin(\pi - \theta)/2$. In any chain of polygons, these triangles all meet at point C, and there all joints are on the circle centered at C with radius $\frac{L}{2\sin\frac{\theta}{2}}$.

2.1 The Inscribed Figure

TODO: Bring the reasoning from [4] here to define the inscribed figure.

3 The Segmented Helix

An analogous, though far more complicated, result holds in three dimensions.

In this section we consider segmented helix, or a helix evaluated only at regular points.

Following the Wikipedia artilce `https://en.wikipedia.org/wiki/Helix`, we set up a helix parametrically.

$$\begin{aligned} P_x(t) &= r \sin t \\ P_y(t) &= r \cos t \\ P_z(t) &= bt \end{aligned}$$

Such a helix has a radius of r and slope (if $r \neq 0$) of b/r,. Note that a helix may be degenerate in two ways. If $r = 0$, these equations become a line. If $b = 0$, these equations describe a circle in the xy-plane. If $r = 0$ and $b = 0$, the figure is a point.

Such helixes are continuous, but we are investigating stacks of discrete objects. We in fact wish to derive the parameters for a continuous helix from such discrete objects which constrain discrete points, so we wish to study a helix evaluated at integral points. We call such an object a *segmented helix* or *shelix (SHELL-lix), shelices (SHELL-ih-sees.*. A segmented helix may be thought of as function that given an integer gives back a point in three space.

$$\begin{aligned} P_x(n) &= r \sin n\theta \\ P_y(n) &= r \cos n\theta \\ P_z(n) &= nd \end{aligned}$$

d is the distance or *travel* along the z-axis between adjacent joints, and θ is the rotation around the z-axis between adjacent points. r is the radius of the shelix. If we think of the shelix as describing a polyline in 3-space, we would like to investgiate the properties of that polyline.

- L is the distance between any two adjacent points.
- c is the length of a chord formed by the projection of the segment between two points into the xy-plane.
- ϕ is the angle in the plane containing two joints which is perpendicular to xy-plane (in other words the plane parallel to the z axis containing two points.)

These quantities are related:

$$c = 2r \sin \frac{\theta}{2} \tag{1}$$

$$L^2 = c^2 + d^2 \tag{2}$$

$$\arctan \frac{c}{d} = \phi \tag{3}$$

Now we are attempting to relate these properties to properties intrinsic to the joint or interface between two segments or objects in the shelix. If given an object, the length between the joint points L is easily measured.

TODO: Create a definition of the "sidedness" of shelix, in terms of the number of sides in one pitch, and relate this matehmatically to the other points.

3.1 The Interior of a Shelix

What is the radius a cylinder that would just fit inside a shelix without any of the segments of the shelix touching the interior of the cylindar? An approach to solving this problem from [4] may be useful. As shown in [4], if the rotation angle θ/π is rational, the interior may in fact be n-gon prism, otherwise it will be a circle.

4 The Intrinsic Properties of Repeated Units

If we have chains of repeated 3D units conjoined identically, they generate a helix. Although it may have been known earlier, we call this Lord's Observation:

Observation 1 (Lord's Observation)**.** *In nature, helical structures arise when identical structural subunits combine sequentially, the orientational and translational relation between each unit and its predecessor remaining constant.[1]*

Lord's Observation should perhaps be clarified that in fact identical objects conjoined via a rule produce chains of objects that are uniformly intersected by segmented helices (shelices) and that they may be degenerate in one of three ways that we might not strike us as a helix if we are not seeking them:

1. The segments may form a straight line.
2. The segments may be planar about a center, forming a polygon or ring.
3. The segments may form a planar saw-tooth pattern of indefinite extent.

TODO: Add figures for each of these cases, maybe in one figure.

The are two complementary ways of learning about such shelices. In one approach, we may have knowledge of the shelix, and wish to learn about the subunits and the rule with which the subunits are combined. For example, we may have microscopic objects such as proteins or atoms, and we know, from crystallography something about the positioning of these objects, without knowing ahead of time the angles at which these objects would combine in their natural environment. In this case, we use a method we call Kahn's method[3] for determining the radius, travel, and twist of the shelix (these terms will be defined precisely below.)

In the other approach, we may know *a priori* exactly the relevant properties of the objects and the rule which which they combine, and we seek to compactly describe the shelix they create. For example, a mathematician may consider a chaing of dodecahedra, or a woodworker may cut identical blocks of maple wood, which are to be glued together face-to-face. In these cases everything about the objects and the rules for conjoing is knownw before the first two objects are glued together. We call this the *face normal method*, because it can be simulated by joining two flat faces together with a specified twist, even if the objects in question do not actually have a physical face.

In both cases, we would like to understand how a change in a face normal or a twist affects the parameters of the shelix, and, conversely, we would like to be able to choose the construction of the subunits to acheive a particular shelix.

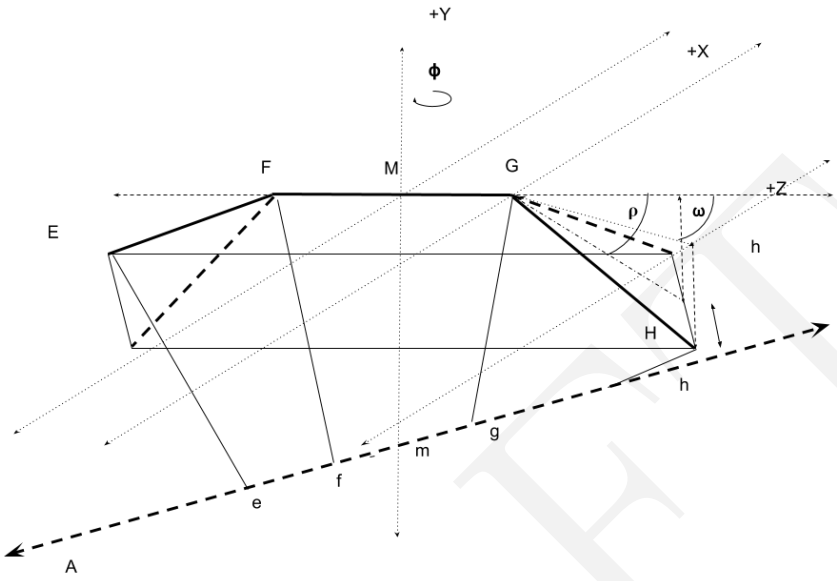

Figure 2: Three Members

In engineering, sometimes the term "special helix"[5] is use for helical curves on non-cylindrical surfaces. This paper use the term "helix" only in the sense of "cylindrical helix".

4.1 The 4-point Algorithm (Kahn's Method)

TODO: this figure is okay, but it should be modified to use A,B,C and D.

TODO: This requires explanation. Having related the face angles to positions of the points directly connected to an object on the z-axis with a joint at the orign, it now becomes more convenient to think of the object's midpoint as placed at the origin.

In the derivations below, we rely on certain facts about the segmented helix formed by the stack of objects:

- Without loss of generality, we may think of any member whoses faces and twist generate a non-degenerate helix as being "above" the axis of the helix. We furthermore choose to place the object in this figure so that $G_y = H_y$, that is, that the members are symmetrix about the z-axis.
- Every joint (E, F, G, H) is the same distance r from the axis of the helix.
- Every member is in the same angular relation to the axis of the helix.
- Since every member cuts across a cylinder around the axis, the midpoint of every memeber is the same distance from the axis which is general a little a less than

r. In particular the midpoint M whose closest point on the helix axis m is on the y-axis and $\overline{Mm} < \overline{Ff}$.

- The points (e, f, g, h) on the axis closest to the joints (E, F, G, H) are eqidistant about the axis and centered about the y-axis. In particular, $\overline{fm} = \overline{gm}$

From the obeservations that $\overline{Ff} = \overline{Gg}$, $\overline{fm} = \overline{mg}$, and we concluded that the helix axis is in a plane parallel to the XZ-plane, it intersects the y-axis, but in general is not parallel to the z-axis. (Note: this assertion requires a formal proof. I can't quite produce a short proof yet.)

From these observations we may state a fundamental fact:

Observation 2. *The angle bisectors of each joint are in general skew and are closest at the axis of helix.*

The conjecture allows us to use the standard linear algebra formula for computing the closest points of two skew lines to find two points on the axis of the helix. The distances between these points is d, and the distances between these points ad the joints is r.

Following the `https://en.wikipedia.org/wiki/Skew_lines`, we calculate:

$$\overline{X} = \overline{F} + \lambda(\overline{F} - \overline{M_{FH}} \tag{4}$$

$$\overline{Y} = \overline{G} + \lambda(\overline{F} - \overline{M_{FH}} \tag{5}$$

Note that in Figure 2 there is great room for confusion in terms of plane ω is actually measured against. The three triangles FfG, FmG, and FgG are all in general not co-planar, that is, they are all at slight angles to each other.

4.2 The Face Normal Method

If we are given a physical or mathematical object that stacks, the faces may be specified by a normal vector or by the angles of the face relative to the axis. The normal vector is a more linear-algebraic approach; the face angles may be more natural to chemists and mechanical engineers. Since the two methods are interchangable, it is largely a matter of convenience. However, we seek specifically to develop a formula which allows helices to be designed, and the face-angle approach seems more suited to the carpenter grasping a hand saw or the molecular biologist designing a molecule.

As shown in figures and , We place a joint point on each face, and call these points A and B. We define the axis of the object as $\overline{AB}$. Imagine the object placed on the z axis in a right-handed coordinate system, so that A is in the negative z direction and B is at the origin. Then the cut of the B face can be described by two angles. α is the angle in the XZ plane, and β is the angle in the YZ plane. In other words, if a box or cuboid were drawn with three edges aligned with the axis and at the origin, and the vector defining the face-normal defined the diagonal of the box or cuboid, α is the angle with the z-axis of the face diagonal in the XZ plane (or the projection of the body diagonal into that plane), and β is the angle of the z-axis with the YZ-plane.

The face angles for A are denoted α_A, β_A, and likewise the independent face angles for B are α_B, β_B.

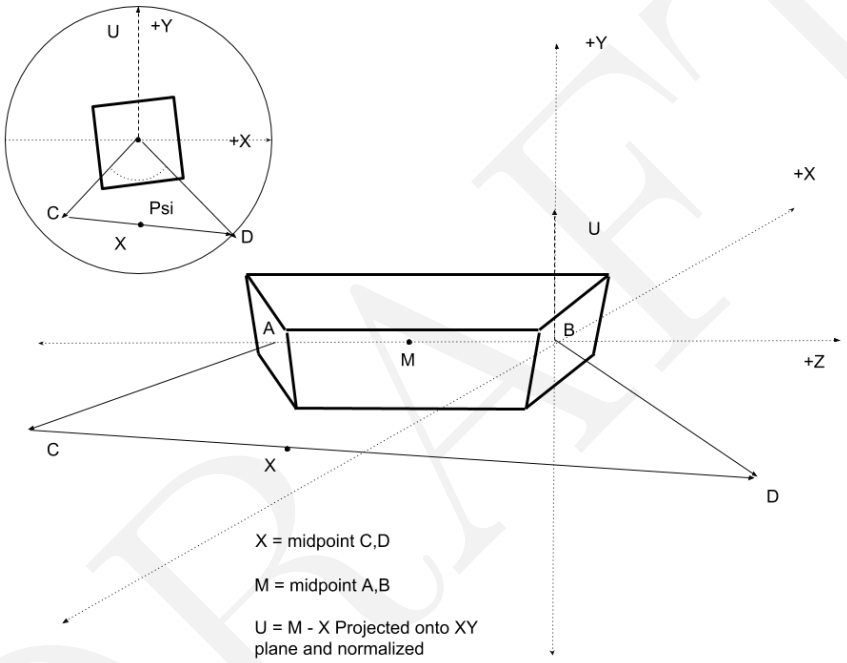

Figure 3: The rotatable prism of three objects

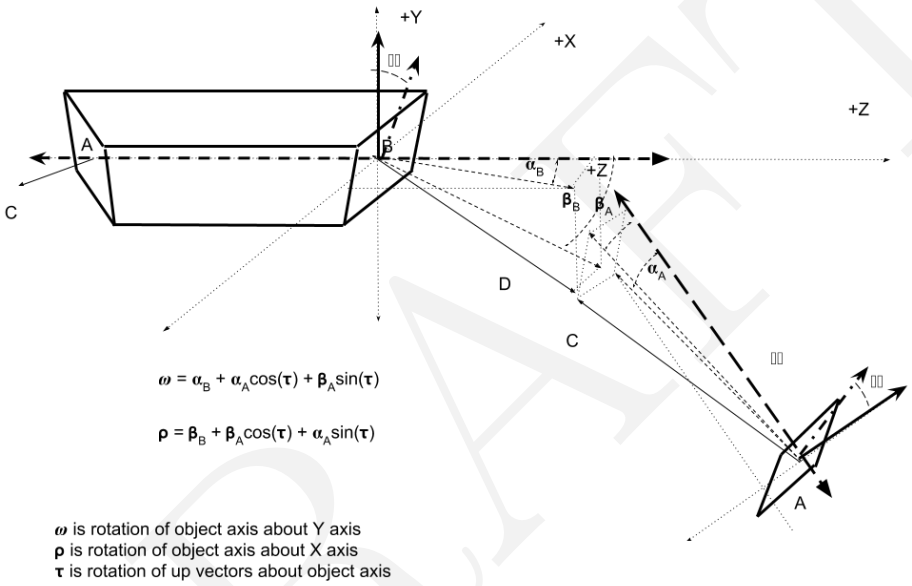

Figure 4: Joint Geometry

In the angle method, we start with these intrinsic properties of an object, and additionally the rule for how objects are laid face-to-face. That is, knowing the length between two joint points and a vector normal to the faces of the two joints, we almost have enough to determine the unique stacking of objects. The final piece is that we must know the *twist*. That is, when face A of a second objects is placed on face B of a first object so that they are flush (that is, their normals are in opposite directions), it remains the case that the second object can be rotated about the normals. To define the joining rule, we must attach an *up vector* to each object. Then a joining rule is "place the second object against the first, joint point conincident to joint point, and twist it so that its up vector differs by τ degrees from the up vector of the first object."

- An object with two identified faces, labeled F_A and F_b. These faces are in fact planes, with $\alpha_A, \beta_A, \alpha_B, \beta_B$, specifying the angular separation of the normal from the XZ and YZ planes. We assume that normals point out of the object rather than in. The angles in the method may be considered the angles of the face diagonals of the axis-aligned cuboid whose body digaonal is the vector normal.
- The length L of an object, measured from joint point A to joint point B.
- A joint twist τ defining the change in computed up-vector between objects, mesured at the joint face.

5 Checks and Explorations

In this section add graphs. Also, a check against the BC helix. Possibly software should be used to produce a 3D simulation of the issues.

6 The up-vector

From the normals, we can compute an *up vector* which is intrinsic to the object. We could think of the up vector as a fine mark made on each face in the same direction (if projected onto a plane perpendicular to the axis.) When face F_A of object $N+1$ is placed against F_B of object N, object $N+1$ is twisted until the mark on F_A angles τ away from the mark on object N, measured anticlockwise looking from A to B on object N.

To define the up vector, place the object so that joint B is at the origin and the axis is aligned with the z-axis, with the point A on the negative z-axis. The projection of the face normals form a minimum angle less than or equal to π radians. Rotate the object about the z-axis so that the projections of the face normals form equal angles with the negative y-axis. Then define the direction straight up the y axis to be the up vector. If we had a physical object, we might mark the face normals with an arrow to mark the up vectors.

If one of the face normals is aligned with object axis, the up-vector is the opposite of the other face normal. (If both face normals are aligned with the axis of the object, then shelix has zero radius and $d = L$, so we need do no further math.)

Mathematically, we will treat the up vector as unit-vector.

$$M = \frac{A + B}{2} \tag{6}$$

$$X = \frac{(A + F_a) + (B + F_b)}{2} \tag{7}$$

$$U = \frac{M - X}{\|M - X\|} \tag{8}$$

The up vector is not defined when the face normals point in precisely opposite directions, in which case $d = L$.

In the placement of the first object described above, it would be nice to know where the position of point B on the second object is. This can be obtained by computing the Euler angles of the face normal N_A at the origin, and then simply adding the negation of the Euler angles N_B, and using the resulting angles to compute a rotation $R(N_A, N_B)$ to be a applied to a vector of length L, and then performing the twist by $T(\tau)$. This process can be repeated at the point B_1 (though no longer at the axis) to generate four or more points. In fact the entire process of constructing the stack could be thought of a series of "rotate then translate" steps by the composition of the rotation matrixes RT applied to a vector $\overline{AB}$ and the up vector. However, it is a tiny bit of work to compute this given only the intrinsic properties of the object and joint.

TODO: Compute the vector angle between axes here. In this positioning, compute where point B of the second object will be.

Note: The twist is not the same as θ. When $\tau = 0$, $\theta = \alpha$, I think. Possible $\theta = \alpha +$ some function of τ.

Note: Possibly theta goes down as tau goes up.

Note: Considering the pitch of the segmented helix may be useful in relating variables.

Note: I now think we can think in terms of the skew lines (the joint lines), and I believe τ is directly the angle formed with the up vector, that the up-vector aims at the mid point of the axis between the intersection skew line points, and that angle bend α (or $(\pi - alpha)/2$) let's us determine the skew line as well. So we have the mid point, and we have another line based purely on τ and α, so that may allow θ to be computed more simply than the skew line intersection construction.

Todo: Try to do my 4-point solution with an Midpoint and upvector solution.

Todo: Try to prove that τ is really either the angle bisector angle or twice it.

7 Applying to The Boerdijk-Coxeter Tetrahelix

TODO: This is higly redundant.

The Boerdijk-Coxeter tetrahelix is a chain of conjoined regular tetrahedra which has been much studied[6, 7, 8, 4] and happens to have irrational measures, making it an ideal test case for our algorithms. Because the face-normals can be calculated and the positions of the elemenbts of the BC helix directly calculated, we can use it to test our algorithms, and in fact these algorithms give the correct result.

However, it should be cautioned that the helix which Coxter identified[6] goes through every node of every tetrahedron. Constructing the helix that goes through only "rail" nodes allows the tetrahelix to be modified[4]. Howewver, the shelices defined in this paper do neither; rather, it is most natural to imagine them moving through the centroid of face of a tetrahedron. The rotation of a segment thus matches the BC Helix ($\arctan -3/2$), but the radius of the generating shelix in the paper would be smaller than those shelices that intersect the nodes.

In light of Lord's ovservation and the Shelix algorithm, we can now reconsider the BC Helix, and in fact a variety of segmented helices which should perhaps becalled *Platonic delices* or if you prefer *Platonic shelixes*.

This complementary view is to think of the BC Helix not as the helix that intersects the vertices of the tetrahedron as Coxeter did[6], nor a single rail as may be valuable to engneers[4], but rather as a helix through the center point of the faces of the tetrahedron. This is a shelix of very small radius compared to the other two approaches, but it has the advantage that it is far more general. For example, it is clearly defined if one used truncated tetrahedra.

More generally, the same approach gives the formula for the shelix created by placing dodecahedra or icosahedra or octahedra face-to-face in a regular pattern. The table of parameters of such delices is provided below.

Note this also makes clear that in these cases we must also specify the *twist*, even if we insist on perfect face-to-face matching. Thinking of it this way, there are actually 3 tetrahedral delices, depending on which twist is chosen (keeping the faces matching). In the case of the tetrahedron, this creates the clockwise BC Helix, the anti-clockwise BC Helix, and the not-quite-close tetrahedral torus.

In the case of the icosahedron, there are in fact many possibilities, as one need not choose the precisely opposite face as the joining face, and one may choose up to three twists.

All of this is a consequence of Lord's observation that *any* repeated transformation produces a shelix.

Unfortunately, the complexity of these formualae exceed the author's comprehension. However, we may check these formuale by graphing them against comprehensible examples. Obvious examples are extreme solutions, where α and θ are 0 or $\pi/2$, for example. We also have the particular non-trivial example of the Boerdick-Coxeter tetrahelix, formed by regular tetrahedra, which has been studied enough to have a known pitch.

8 Implications

One of the implcations of having a formulaic understanding of the math is that it may be possible to design helixes of any radius and pitch by designing periodic (possibly scalene) segments. Combined with slight irregularites, this means that you have a basis of design molecular helices out of "atoms" which correspond to our objects.

This would mean that if you wanted to build a brace of length exactly 3 meters with bars of exactly 1/2 meter you would be able to come as close to this as mathematically possible.

9 Applied to Periodic Regular Simplex Chains

It is now clear that in fact ANY repeated rule applied to chains of tetrahedra, or any other object, is simply producing a larger repeated subunit, and that Hahn's method and Face Normal method may be applied to the larger subunit.

In some circumstances we may wish to design a shelix based on "molecules" comprised of "atoms", rather than using fully versatile objects. For example, it might be convenient for us to recombine pre-fabricated regular tetrahedra or some other unit, rather than constructing units having arbitrary face angles. If our construction method allow, we might ask what can be accomplished with a set of "atoms" and a variation in the twist τ[9].

Corollary 1. *Every regular simplex chain formed by a periodic generator has a helical structure.*

10 Relating to the Screw Transform

TODO: Lord has described these operations in terms of screw transforms. In a sense this is the opposite or what we want to do, but it should be possible to interrelate the two mechanisms. Ideally would would have an algorithm that operates on Screw Transforms and gives the shelix, and be able to produce a Screw Transform from a shelix.

11 Future Work

We propose that the math developed in this paper can be used to build an exhaustive table of the properties Platonic delices, that is, segmented helices constructed solely out of regular Platonic solids. Such tetrahelices, icosahelices, octahelices and dodechelices have been mentioned in a number of papers[9, 10, 11], but not exhaustively studied in the purely helical form. Because in some cases Platonic delices may be found in nature or related to structures found in nature[12], it would be conveneient to have a table, and images, of all such Platonic delices for reference.

Note: Must read this: `https://www.researchgate.net/profile/Peter_Kahn/publication/220667044_Defining_the_axis_of_a_helix/links/5b86ab1e299bf1d5a730ff2e/Defining-the-axis-of-a-helix.pdf`[3].

Note this extremely important observation: "Since V1 and V2 are both perpendicular ot the axis, their corss product will have the direction of the axis". (Here V1 and V2 are angle bisectors.) This should make my work much simpler! To some extent this suggest that this work is not as original as I had thought.

Note further that Equations 7 and 8 of this paper give BETTER equations for radius r and the distance d than what I have so far given.

Note: Here is an example of a question asked on Math Stack Exchange which is essentially answered by this paper:

`https://math.stackexchange.com/questions/878051/why-does-a-3d-line-of-segments-with-constar`
`878079#878079`

The answerer in fact predicts the linear bisection method which I have outlined.

Note: This must be studied immediately:
https://math.stackexchange.com/questions/1041780/how-to-prove-the-bisector-vector-of-the-ar
1042231#1042231
This assumes helical to begin with so not of much use.
Note: Must read this: https://www.researchgate.net/profile/Peter_Kahn/
publication/220667044_Defining_the_axis_of_a_helix/links/5b86ab1e299bf1d5a730ff2e/
Defining-the-axis-of-a-helix.pdf[3].
https://www.win.tue.nl/~wstomv/publications/mathmitering-final.pdf https:
//www.clinbiomech.com/article/S0268-0033(98)00080-1/abstract https://gist.
github.com/peteristhegreat/3b76d5169d7b9fc1e333 https://www.sciencedirect.
com/science/article/pii/S0022309303008573 https://www.sciencedirect.com/
science/article/pii/S0022309307005583
This reference is EXTREMELY IMPORTANT https://link.springer.com/article/
10.1023/A:1015863923728
This may be worth reading: https://link.springer.com/article/10.1007/PL00011063
Some discussion of "screw transformations" http://dergipark.gov.tr/download/
article-file/56483
CRITICAL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=56653
This is a valuable reference http://www.12000.org/my_notes/screw_axis/index.
htm
This paper considers only simple unbranched chains of objects. In the future it might be valuable apply similar software and mathematics to branching systems such as Lindemayer systems[13].
Chains of tetrahedra are interesting because they are structurally strong; however, branched and rejoined structures are equally interesting. The current work might assiste in the design of trusses which begin at a point, trifurcate into chains, and eventually rejoin, create spindle-enveloped spaceframes which would resist buckling.

12 References that need to be studied or reviewed

NOTE: This is a discussion of representing joint angles, it is not obvious how valuable it is: https://www.clinbiomech.com/article/S0268-0033(98)00080-1/abstract
NOTE: This needs to be read and understood, it is not clear how related it is: https://ieeexplore.ieee.org/document/56653
This is a long, expensive book, but it may be quite relevant[14]: https://books.
google.com/books?hl=en&lr=&id=1LZlSZ7ORrQC&oi=fnd&pg=PP1&ots=0hSEwJvlUB&
sig=xNG9UWv_H1OXHwaOiOBJN7TW6xA#v=onepage&q&f=false
Note: There is another long, deep book that needs to be obtained and studied[15].
https://books.google.com/books?hl=en&lr=&id=FHPlDWvz1bEC&oi=fnd&pg=PP1&ots=
TsOnodavEZ&sig=HO86UUVlqRVWGqY-Tv02nb7x7NA#v=onepage&q&f=false
Note: This work uses the term "segmented" helix, perhaps that is a known term and I need to switch to it:
https://www.researchgate.net/publication/236066626_Segmented_helical_
structures_formed_by_ABC_star_copolymers_in_nanopores
This is a discussion of segmened coils in a protein structure:

https://www.sciencedirect.com/science/article/pii/0022283688903701

12.1 To Be Done

Need to understand what an "alpha coil" protein structure is. (Done: An alpha coil is one of the most common and studied protein forms, roughly providing distance in the protein, consisting of a helix formed by amino acids, which tend to be cross bonded.

Need to answer the Math stack exchange question.

Need to understand possibility of further simplifying specification of object.

Need to clean up Mathematica code, and make possibly use lstlisting environment for highlighting.

13 Acknowledgements

Thanks to Prof. Eric Lord for his direct communication and Mr. Robert Gatliff for his assistance.

The enthusiasm of the participants of the 2018 Public Invention Mathathon intitiated this work.

References

[1] Eric A Lord. Helical structures: the geometry of protein helices and nanotubes. *Structural Chemistry*, 13(3-4):305–314, 2002.

[2] Robert L. Read. The story fo a public, cooperative mathathon, December 2019. [Online; posted 12-December-2019].

[3] Peter C Kahn. Defining the axis of a helix. *Computers & chemistry*, 13(3):185–189, 1989.

[4] Robert Read. Transforming optimal tetrahelices between the boerdijk-coxeter helix and a planar-faced tetrahelix. *Journal of Mechanisms and Robotics*, 2018.

[5] Lizhi Gu, Peng Lei, and Qi Hong. Research on discrete mathematical model of special helical surface. In *Green Communications and Networks*, pages 793–800. Springer, 2012.

[6] HSM Coxeter et al. The simplicial helix and the equation tan(n θ) = n tan(θ). *Canad. Math. Bull*, 28(4):385–393, 1985.

[7] Garrett Sadler, Fang Fang, Julio Kovacs, and Klee Irwin. Periodic modification of the Boerdijk-Coxeter helix (tetrahelix). *arXiv preprint arXiv:1302.1174*, 2013.

[8] R.B. Fuller and E.J. Applewhite. *Synergetics: explorations in the geometry of thinking*. Macmillan, 1982.

[9] Michael Elgersma and Stan Wagon. The quadrahelix: A nearly perfect loop of tetrahedra. *arXiv preprint arXiv:1610.00280*, 2016.

[10] Hassan Babiker and Stanisław Janeczko. *Combinatorial cycles of tetrahedral chains*. IM PAN, 2012.

[11] EA Lord and S Ranganathan. Sphere packing, helices and the polytope {3, 3, 5}. *The European Physical Journal D-Atomic, Molecular, Optical and Plasma Physics*, 15(3):335–343, 2001.

[12] EA Lord and S Ranganathan. The γ-brass structure and the boerdijk–coxeter helix. *Journal of non-crystalline solids*, 334:121–125, 2004.

[13] Przemyslaw Prusinkiewicz and James Hanan. *Lindenmayer systems, fractals, and plants*, volume 79. Springer Science & Business Media, 2013.

[14] Stephen Hyde, Z Blum, T Landh, S Lidin, BW Ninham, S Andersson, and K Larsson. *The language of shape: the role of curvature in condensed matter: physics, chemistry and biology*. Elsevier, 1996.

[15] Jean-François Sadoc and Rémy Mosseri. *Geometrical frustration*. Cambridge University Press, 2006.

A Mathematica Code

```
The following math in Mathematica may be useful.

Note: Mathematica has build in Vector Angle function that can be used for alpha!!

(* c = 2 r Sin[theta/2] *)

Chord[r_,theta_] := 2 r Sin[theta/2]
 (* 1 == c^2 + d^2 *)
DZ[r_,theta_] := Sqrt[1 - Chord[r,theta]]
(* Note, this may be a problem --- Chord lenght can be diameter (2r), but I am treating it as
(* Some radiuses are not possible for some thetas with a length of one *)
(* Maximum radius as a function of theta *)
Mxr[theta_] := MaxValue[1 == 2 r Sin[theta/2],{r}]

Mxr[theta_] := 1/ (2  Sin[theta/2])


 Note: I should go ahead and make these functions....

(* P0 = {0, r, 0} *)
(* P1 = {r Sin[theta], r Cos[theta], d} *)
(* P2 = {r Sin[2 theta], r Cos[2 theta], 2 d} *)
(* P3 = { r Sin[3 theta], r Cos[3 theta], 3 d} *)
(* S0 = P0 - P1 *)
(* S1 = P1 - P2 *)
(* S2 = P2 - P3 *)
P[n_,r_,theta_] := { r Sin[n theta], r Cos[n theta], n DZ[r,theta]}
```

```
S[n_,r_,theta_] := P[n+1,r,theta] - P[n,r,theta]

(* Define Ps to be P symmetric; the points P0 and P1 are
symmetric in the XY plane. *)
Ps[n_,r_,theta_] :=
With[{nh = (n - (1/2))},
  { r Sin[nh theta], r Cos[nh theta], nh DZ[r,theta]}]


AlphaC[r_,theta_] := VectorAngle[S[0,r,theta],S[1,r,theta]]


f[r_, theta_] :=
 ArcCos[-1 + (r - r Cos[theta]) (-r Cos[theta] + r Cos[2 theta]) +
   4 r^2 Sin[theta/2]^2 -
   r Sin[theta] (-r Sin[theta] + r Sin[2 theta])]

Plot3D[f[r, theta]/ Degree, {r, 0, 2}, {theta, -Pi/6, Pi/6},
 AxesLabel -> Automatic]


Plot3D[AlphaC[r, theta]/ Degree, {r,0,N[Mxr[Pi]]}, {theta, 0, Pi},
 AxesLabel -> Automatic]

(*

Note: Given that we can compute $\alpha$ from $r,\theta$ in this way, one interesting thing to
is simply what is the minimum or maximum r (or theta) that matches an alpha. The maximum r is
largest helix that matches, the minimum is the smallest. But with $\psi$ we will know more.
*)

PlaneNormal[a_, b_] := Normalize[Cross[a, b]]
(* Maybe psi is just theta? *)

Psi[pv0_, pv1_, v_] :=
With[{n = PlaneNormal[pv0, pv1]}, ArcSin[(n.v) /(Norm[v] Norm[n])]]

(* This is computing the angle between planes
PsiC[rx_,thetax_] :=
With[{r = rx,theta = thetax}, Evaluate[Psi[S[0,r,theta],S[1,r,theta],S[2,r,theta]]]]


Plot3D[PsiC[r, theta]/ Degree, {r, 0, N[Mxr[Pi/2]]}, {theta, 0, Pi/2},
 AxesLabel -> Automatic]

(*
```

```
Note: A strategy for computation is: given alpha, find maximum or minimum radius.
Then slow change radius until \psi if find.  However, Mathematic has optimization
built in, so we may be able to use an energy function.
*)

Enrg[alpha_,psi_,r_,theta_] := (AlphaC[r,theta] - alpha)^2 + (PsiC[r,theta] - psi)^2


(* These seem to work, but they produce rules, instead of values, what up with that? *)

Rx[alpha_,psi_] := r /. FindMinimum[Enrg[alpha,psi,r,t],{r,t}][[2]][[1]]
Tx[alpha_,psi_] := t /.  FindMinimum[Enrg[alpha,psi,r,t],{r,t}][[2]][[2]]

(* This should return 2! *)
(* Is Psi the VectorAngle of the Normals of each ABC joint? *)
Rx[AlphaC[2, Pi/20], PsiC[2, Pi/20]]
FindMinimum[Enrg[AlphaC[2, Pi/20], PsiC[2, Pi/20],r,t],{r,t}]

Unprotect[$PerformanceGoal]
$PerformanceGoal = ``Speed''

Plot3D[Rx[a,p]/Degree, {a,0,Pi/10},{p,0,Pi/10}]

(* Now that we can in theory find Rx, Tx as a function of alpha and psi,
It is important that we have a check function (or two.)
The best check would be to use a reconstruction based on transformations
of alpha and psi. I need to think about how this should be done with a
RollPitchYaw Matrix, and in what order. First you by \alpha, then you rotate by \psi.
I think we are doing ``Pitch, Roll, Yaw''. First we Pitch by \alpha, then
we Roll into the plane of the joint, then we yaw by \psi.
In fact what we are trying to do is to fly a little airplane in a helical course.
``Pitch down by alpha'', ``Yaw by psi'', ``roll so that we are perpendicular
to the the plane of the last joint''.  Maybe we never need to the roll component.
I want intrinsic angle rotation (by the definition.) We do seem to have a mobile
frame of reference.
So in fact I want EulerMatrices, not RollPitchYaw matrices.

Euluer[{\alpha,\beta,\gamma},{a,b,c}], where a,b,c = 1,2,3, let us specify those.
So we need to define how our mobile frame works. We will want $y$ to always point
``out'' of the helix, ``z'' to point along the motion'', and $x$ to be
tangential to the helix.

If we start with the intrinsic frame aligned with the extrinsic frame, repeated
transofrmations will produce a helix, but not about the $z$ axis.
We have so far described our mostion as ``rotate about y'' by $\alpha$, then
rotation about $x by $\psi$. Then our $z$ axis would be pointed in the
```

```
write direction, we would translate  by $z$, in the intrinsic frame of refrence,
and repeat.

Note, mathematica seems to have a way to render a helix, this would be very useful to me:
*)

ListPointPlot3D[
Table[Table[{t, Cos[t + s Pi/2], Sin[t + s Pi/2]}, {t, 0,
      5 Pi, .2}], {s, 4}], BoxRatios -> Automatic]

Graphics3D[
GeometricTransformation[{Hue[#/Pi], Sphere[{5, 0, 0}, 1]},
  EulerMatrix[{#, Pi/2, #}]] & /@ Range[0, 2 Pi, Pi/16]]

(* My claim is that we can somehow combine an EulerTransformation with a translation
to end up with a helix.

We can call GeometricTransoforamtion on an Euler matrix, can we pass a vector
as an object? Can we create an object of length L which is a vector
and then create a transformation?
*)

Graphics3D[Arrow[{{1, 1, 1}, {1, -1, 2}}], Axes -> True,
  AxesLabel -> {"X", "Y", "Z"}, ImageSize -> Large]

Graphics3D[
GeometricTransformation[{Hue[#/Pi],Arrow[{{1, 1, 1}, {1, -1, 2}}] },
Composition @@ {
     TranslationTransform[{1, 1, 1}],
      EulerMatrix[{0, Pi/2, #}]
}] & /@ Range[0, 2 Pi, Pi/16]]


 EulerMatrix[{Pi/15, Pi/20, 0}] [1,1,1]


 myt = TranslationTransform[{0, 0, 1}] EulerMatrix[{Pi/15, Pi/20, 0}]


 Graphics3D[{
  Opacity[1]
  , Red
  , Arrow[{{0, 0, 0}, {1, 0, 0}}]
  , Green
  , Arrow[{{0, 0, 0}, {0, 1, 0}}]
  , Blue
```

```
 , Arrow[{{0, 0, 0}, {0, 0, 1}}]
 , Opacity[0.2]
 , GeometricTransformation[Cuboid[-{1, 1, 1}/4, {1, 1, 1}/4],
  Composition @@ {
    RotationTransform[Pi/4, {0, 0, 1}]
    , TranslationTransform[{1, 1, 1}]
    }
  ]
}]

Graphics3D[{
 Opacity[1]
 , Red
 , Arrow[{{0, 0, 0}, {1, 0, 0}}]
 , Green
 , Arrow[{{0, 0, 0}, {0, 1, 0}}]
 , Blue
 , Arrow[{{0, 0, 0}, {0, 0, 1}}]
 , Opacity[0.2]
 , GeometricTransformation[Cuboid[-{1, 1, 1}/4, {1, 1, 1}/4],
   Composition @@ {
      RotationTransform[Pi/15,{0,1,2}],
    , TranslationTransform[{1, 1, 1}]
    }
  ]  & /@ Range[0, 5, 1]
}]


Graphics3D[
  GeometricTransformation[{Hue[#/Pi],Arrow[{{1, 1, 1}, {1, -1, 2}}] },
      Composition @@ {
      RotationTransform[Pi/15,{0,1,2}],
    , TranslationTransform[{1, 1, 1}]
    }
]]


Composition @@ {
    RotationTransform[Pi/4, {0, 0, 1}]
    , TranslationTransform[{1, 1, 1}]
}

Graphics3D[

 (GeometricTransformation[Cuboid[-{1, 1, 1}/4, {1, 1, 1}/4],
  Composition @@ {RotationTransform[Pi/4, {0, 0, #}],
```

```
    TranslationTransform[{1, 1, #}]}]) & Range[3]

 ]

double[x_] := 2 x

Nest[double,#,4] & Range[3]

(* Now, what we really want to do here, is  construct
a transform out of \alpha, \pi, ending up with the arrows head to tail
*)

Clear[ComposeN]
ComposeN[0] = ScalingTransform[{1,1,1}]
ComposeN[1] = Composition @@ {
  RotationTransform[ Pi/6, {0, 1, 0}],
  Composition @@ {
    TranslationTransform[{1, 1, 1}],
    RotationTransform[ Pi/6, {0, 0, 1}]
  }
}

ComposeN[n_] :=  Composition[ComposeN[Floor[n/2]],ComposeN[Ceiling[n/2]]]


 myarrows =
 Table[
   GeometricTransformation[{
     Arrow[{{0, 0, 0}, {0, 0, 2}}],
     Arrow[{{0, 0, 0}, {0,1/2, 0}}]},
     ComposeN[i]
   ],{i,1,10}
   ]

 Graphics3D[myarrows]


 (* I think to do what I want, I need to do my own transformationns. *)
 (* An object is a vector and an upvector.
 The 0th object is a z-aligned vector starting at the origin, with a y upvector.
 The n+1th object is:
 A) Take the nth object (AB), find the head B of the vector.
 B) Create a vector of length N pointing in the same direction as the nth object.
 C) Translate it along the nth object to the head.
 D) Relative to
 E) Rotate it in the AB coordinate frame in the Y direction by Alpha.
```

```
 F) Rotate it in the X direction by Psi.
 G) Apply these same arguments to the up Vector.

Note that Mathematica has powerful RotationMatrix functions built in to which
 we can use the Nth vectors as input, to rotate in a plane. So in this sense
 we may actually be able to accomplish this.
*)

(* I'm going to try to use F to mean the nth object. *)
(* We will make the first to points computed from the Helix *)
F[0,alpha_,psi_,init_] := init

(* We are forced to pass the initial object in as a starting point *)
F[n_,alpha_,psi_,init_] :=
  With[{Prev = F[n-1,alpha,psi,init]},
      With[{
          A = Part[Prev,1], B = Part[Prev,2], U = Part[Prev,3]},
        With[{BA = B-A, UA = U-A},
        (* now we want to contruct the parts of P_n *)
        Module[{NA,NB,NU,RA,RP,NV},
          NA = B;
          V = BA;
          (* This is in theory spanned by BA, UA *)
          RA = RotationMatrix[alpha,{BA,UA}];
          RP = RotationMatrix[psi,BA];
          NB = RP.RA.V;
          NU = RP.RA.UA;
          NB = NB + B;
          NU = NU + B;
          {NA,NB,NU}
    ]
  ]]]]


  (* F requires an initial object (of three points, A, B Up.) These
  are essentially P[0],P[1], and P[0] + [0,1,0]. *)

  FTest[n_,alpha_,psi_,r_,theta_] :=
  With[{P0 = P[0,r,theta]},
    Part[F[n,alpha,psi,{P0,P[1,r,theta],P0+{0,1,0}}
      ],1]
    - P[n,r,theta]]

  FTest[1,Pi/8,0,1,0]

  FStratTest[k_,alpha_,psi_] :=
```

```
  With[
    {r = Rx[alpha,psi], t = Tx[alpha,psi], P0 = {0,0,0} },
    With[{P0 = P[0,r,t], P1 = P[1,r,t]},
      With[{init = {P0,P1,P0+{0,1,0}}},
    Print["alpha, psi, r, t ",alpha,", ",psi,", ",r,", ",t 180 / Pi];
    For[i = 0, i < k, i++,
      Print["point ",i];
      Print["P ",P[i,r,t]];
      Print["F ",Part[F[i,r,t,init],1]]
      ]
    ]
  ]]

  (* Pretty sure this is showing the r is way to small *)
  FStratTest[4, 10 Degree,10 Degree]


  (* now attempting to compute the transformation matrix
  that transforms {A,B,U},{L,N_A,N_B, Tau} into the next A,B,U.
  M = W . R . T , where W is the twist, R is the rotation,
  and T is the translation. We can test this by using a synthetic shelix to generate points an
  *)

  (* I first need to build and test a simple Skew test.
  I want a function that yields the closest points on two skew lines,
  defined by vectors. *)

  Skew[p1_,d1_,p2_,d2_] := (* p1(2) is a point on line with direction d1(2),
  result is a vector containg C1 and C2, nearest points on those lines.
  from: https://en.wikipedia.org/wiki/Skew_lines *)
  With[{d12 = Cross[d1,d2],
      d21 = Cross[d2,d1]},
(*    Print[N[d1]]
    Print[N[d2]]
    Print[N[d12]]
    Print[N[d21]]
    Print[123123123]
    Print[N[Cross[d2,d12]]]
    Print[444444444]
    Print[N[Cross[d1,d21]]]
    Print[666666666]*)
  With[{n2 = Cross[d2,d12],
      n1 = Cross[d1,d21]},
(*    Print[555555555555]
    Print[N[n1]]
    Print[N[n2]] *)
```

```
    With[{C1 = p1 + (((p2-p1) . n2)/ (d1 . n2)) d1,
        C2 = p2 + (((p1-p2) . n1)/ (d2 . n1)) d2},
(*    Print[N[C1]]
    Print[N[C2]]
    Print[999999999999999] *)
    (* For unknown perforance reasons, I have to force evaluation to a number here *)
      {N[C1],N[C2]}]]]

 TestSkew[] :=
 With[{xp1 = {2,0,0}, xd1 = {1,0,0},
     yp1 = {1,5,1}, yd1 = {0,2,0}},
   Skew[xp1,xd1,yp1,yd1]]

 (* Now we could try a 3-point test. We take the
 first 3 points of our shelix and then compute the mid-points,
 and try to find r by treating these as skew lines with
 closes intersections. *)

 M[n_,r_,theta_] := (* midpoint of Pn-1, Pn+1 *)
 (P[n-1,r,theta]+P[n+1,r,theta])/2

 AngelBi[n_,r_,theta_] := (* unit angle bisector *)
Normalize[M[n,r,theta] - P[n,r,theta]]

(* This does not return the radius, it returns d, the travel *)
(* If we made this a function of 4 points instead, then
this would be independent of the current defintion of our
test shelix.*)

(* This seems to work. So if we can compute 4 points
by virtue of the transform, we can probably compute d (z travel).
However, the two points should let us go further. Since
they lie on the axis plane, we can compute theta by projecting
onto the plane normal to this line. Or just translate
the points along this angle and then call VectorAngle.


*)
ComputeAxisPointsFrom4[ps_] := (* Input is a vector of four points *)
With[{P0 = Part[ps,1],
    P1 = Part[ps,2],
    P2 = Part[ps,3],
    P3 = Part[ps,4]},
With[{M1 = (P0+P2)/2,
    M2 = (P1+P3)/2},
  With[{B1 = Normalize[M1-P1],
```

```
      B2 = Normalize[M2-P2]},
    With[{Sk = Skew[P1,B1,P2,B2]},
      With[{C1 = Part[Sk,1],
            C2 = Part[Sk,2]},
        Print[N[VectorAngle[P1 - C1, P2 -C2]]]
        Sk
    ]]]]]

 (* I think this is supposed to return D *)
TestAxisPoint[k_,r_,theta_] :=
   With[{P0 = P[k,r,theta],
       P1 = P[k+1,r,theta],
       P2 = P[k+2,r,theta],
       P3 = P[k+3,r,theta]},
     With[{CS = ComputeAxisPointsFrom4[{P0,P1,P2,P3}]},
      Norm[Part[CS,1] - Part[CS,2]]]]

   (* IMPORTANT: This seems to be my most successful approach, the only one that is clear. *)
   (* This function returns DZ[r,theta], which seems important. *)
   (* This is the distance along the shelix axis of two points representing angle
   bisetctors. *)
  Test3PointTheory[k_,r_,theta_] :=
  With[{D0 = N[AngelBi[k,r,theta]],
      D1 = N[AngelBi[k+1,r,theta]]},
    Print[D0]
    Print[D1]
    Print[357]
    With[{CS = Skew[P[k,r,theta],D0,P[k+1,r,theta],D1]},
      Print[CS]
      Print[3333333333333]
      Norm[Part[CS,1] - Part[CS,2]]
  ]]

  (* Now beginning simple work on face diagonals *)

  Zf[w_,r_] := 1/Sqrt[1 + Tan[r]^2 + Tan[w]^2]
  Xf[w_,r_] :=
  With[{z = Zf[w,r]},
    z Tan[w]]
  Yf[w_,r_] :=
  With[{z = Zf[w,r]},
    z Tan[r]]

  TestXYZ[w_,r_] :=
  With[{ z = Zf[w,r], y = Yf[w,r], x = Xf[w,r]},
    Print N[Norm[{x,y,z}]]
```

```
    ]


  DfromOmega[L_,w_] :=
  With[{ t = Tan[w]},
    (L t) /(2Sqrt[(t^2)/4 + 1])]

  (* One way to test is to generate a synthetic shelix,
  compute rho, omega from it, and then check that this
  formular for the D from omega against this synthetic verison.
  This requires the computation of rho,omega, from r,theta *)

  (* Returns pair rho, omega (rotation about X, rotation about Y) *)
  (* This is currently wrong becasue it relies on the P0_x = P1_x,
  but I am not calculating it that way !!! *)
  (* I could compute these points and compute a rotation about Y I suppose.)
RhoOmegaFromRTheta[r_,theta_] :=
 With[{P0 = Ps[0,r,theta],
       P1 = Ps[0+1,r,theta],
       P2 = Ps[0+2,r,theta]},
   With[{a = ArcTan[Part[P1,1]/Part[P1,3]]},
     With[{rt = RotationTransform[-a,{0,1,0}]},
(*       Print[444]
       Print[N[a] 180 / Pi]
       Print[N[P1]]
       Print[N[rt[P1]]]
       Print[5555555555]
       Print[N[rt[P2]]]
       *)
     With[{v = rt[P2] - rt[P1]},
(*       Print[N[v]]  *)
       With[{x = Part[v,1],
             y = Part[v,2],
             z = Part[v,3]},
         {ArcTan[z,y],ArcTan[z,x]}
         ]]]]]


 Len[r_,theta_] := Norm[Ps[0,r,theta]-Ps[1,r,theta]]

 (* This is a check, we need to be able to compute rho and omega
 and recover the 3rd point from it. Okay, this now works. *)
 HCheck[r_,theta_] :=
 With[{ ro = RhoOmegaFromRTheta[r,theta],
   L = Len[r,theta]},
   With[{ rho = Part[ro,1], omega = Part[ro,2]},
```

```
    With[{ q = QFromRhoOmega[L,rho,omega]},
    {q Tan[omega], q Tan[rho], q}]]]


DfromRTheta[r_,theta_] :=
With[{ro = RhoOmegaFromRTheta[r,theta],
     L = Len[r,theta]},
  DfromOmega[L,Part[ro,2]]
  ]


DFromLQW[L_,q_,w_] :=
L Sin[ArcTan[Tan[w]/((L/q) - 2)]]

ChiFromLQW[L_,q_,w_] :=
With[{A = q Tan[w] /2,
    B = (L/2) - q},
  Print[N[A]]
  Print[N[B]]
ArcTan[A/B]]

TestDFromLQW[r_,theta_] :=
 With[{ ro = RhoOmegaFromRTheta[r,theta],
  L = Len[r,theta]},
  With[{ rho = Part[ro,1], omega = Part[ro,2]},
    With[{ q = QFromRhoOmega[L,rho,omega]},
      Print[N[L]]
      Print[N[q]]
      DFromLQW[L,q,omega]]]]

 (* I believe I can compute rho, omega from r,theta. I am now attempting to
 compute d from rho, omega, L, based on the projection diagrem. To debug this,
 I need to compute the points F, G, H, and check the projection diagram.
 Since I have already computed the points and a the rotation about Y which
 brings them to Z axis, this would seem to be straightforward. *)



FGH[r_,theta_] :=
With[{
    ro = RhoOmegaFromRTheta[r,theta],
    L = Len[r,theta],
    P0 = Ps[0,r,theta],
    P1 = Ps[0+1,r,theta],
    P2 = Ps[0+2,r,theta]},
  With[{ a = ArcTan[Part[P1,1]/Part[P1,3]],
```

```
        rho = Part[ro,1],
        omega = Part[ro,2]},
      With[{
          rt = RotationTransform[-a,{0,1,0}],
          q = QFromRhoOmega[L,rho,omega]},
      With[{ x = q Tan[omega],
          y = q Tan[rho],
          z = q },
        With[{ F = {0,0,-(L/2)},
            G = {0,0,(L/2)},
            H = {x,y,(L/2)+z},
          c = {0, Part[rt[P0],2], 0}},
          { {F,G,H},
            {rt[P0]-c,rt[P1]-c,rt[P2]-c}
            }
]]]]]

(* To complete the current approach, I need to take several steps: *)
(* Modify the code to compute $d$ from 4 arbitrary points.
Test that this matches the current results by using a helical test pattern. *)
(* Generate 4 points based on Rho,Omega, by using symmetry and the approach above. *)


(* This routine generates 4 symmetric points on the Z axis matching the input values. *)
(* To test this is sensible, I really should be able to invert it---that is,
To find an r,theta parametrization that matches the L,rho,omega parametrization.
Of course, in a ense, that is what we are trying to do entirely---so why not back out with my
computations? *)
Generate4Points[L_,rho_,omega_] :=
With[{
    B = {0,0,-L/2},
    C = {0,0,L/2},
    q = QFromRhoOmega[L,rho,omega]
  },
  With[{ x = q Tan[omega],
     y = q Tan[rho]},
    With[{
      A = {-x,y,-(L/2 + q)},
      D = {x,y,(L/2 + q)}},
      {A,B,C,D}
]]]


Generate4PointsRTheta[r_,theta_] :=
{Ps[-1,r,theta],Ps[0,r,theta],Ps[1,r,theta],Ps[2,r,theta]}
```

```
(* Compute d from 4 points *)
Bisector[X_,O_,Y_] :=
Normalize[O - (X+Y)/2]

(* In typical cases Skew is hanging below.
I don't even know a mechanism by which that should be possible. *)
ComputeDR[A_,B_,C_,D_] :=
 With[{Bb = Bisector[A,B,C],
     Cb = Bisector[B,C,D]},
   With[{CS = Skew[B,Bb,C,Cb]},
     {Norm[Part[CS,1] - Part[CS,2]],
      Norm[Part[CS,1] - B]}
 ]]

 ComputeD[A_,B_,C_,D_] :=
 Part[ComputeDR[A,B,C,D],1]


 TestComputeD[r_,theta_] :=
 With[{
     ro = RhoOmegaFromRTheta[r,theta],
     L = Len[r,theta]
   },
   With[{
       rho = Part[ro,1],
       omega = Part[ro,2]
       },
   With[{
       abcd = Generate4Points[L,rho,omega]
       },
   With[{
       A = Part[abcd,1],
       B = Part[abcd,2],
       C = Part[abcd,3],
       D = Part[abcd,4]
     },
(*     Print[N[abcd]] *)
     ComputeD[A,B,C,D] - N[DZ[r,theta]]]]]]

  TestComputeD0[r_,theta_,abcd_] :=
 With[{
     ro = RhoOmegaFromRTheta[r,theta],
     L = Len[r,theta]
   },
   With[{
       rho = Part[ro,1],
```

```
      omega = Part[ro,2]
      },
  With[{
      A = Part[abcd,1],
      B = Part[abcd,2],
      C = Part[abcd,3],
      D = Part[abcd,4]
    },
    ComputeD[A,B,C,D]]]]

TestComputeDagainstHelix[] :=
With[{abcd = Generate4PointsRTheta[2, Pi/10]},
  With[{
      A = Part[abcd,1],
      B = Part[abcd,2],
      C = Part[abcd,3],
      D = Part[abcd,4]
    },
    ComputeD[A,B,C,D] - DZ[2,Pi/10]]]

TestComputeDagainstROmega[rho_,omega_] :=
With[{abcd = Generate4Points[1, rho, omega]},
  With[{
      A = Part[abcd,1],
      B = Part[abcd,2],
      C = Part[abcd,3],
      D = Part[abcd,4]
    },
 ComputeD[A,B,C,D]]]



(* Full test computation: Given rho,omega, compute r, theta,
and see that it matches.
1) Starting with r, theta, generate rho, omega, L.
2) from rho, omega, L, generate 4 points
3) Compute D from this.
5) Use basic formulae to compute r,theta.
*)

(* Given the cord length, how do I compute R? *)
ChordFromLD[L_,D_] := Sqrt[L^2 - D^2]
ThetaFromRC[R_,C_] :=
2 ArcSin[C/(2R)]
```

```
  (* Return basic helix params: r, theta, d, c *)
GetHelixParams[L_,rho_,omega_] :=
     With[{abcd = Generate4Points[L, rho, omega]},
        With[{
            A = Part[abcd,1],
            B = Part[abcd,2],
            C = Part[abcd,3],
            D = Part[abcd,4]
          },
          With[{dr = ComputeDR[A,B,C,D]},
            With[{di = Part[dr,1],
                r = Part[dr,2]},
              With[{
                  c = ChordFromLD[L,di]},
                With[{ theta2 = ThetaFromRC[r,c]},
                  {r,theta2,di,c}
                  ]]]]]]

  TestComputeDagainstROmega[rad_,theta_] :=
  With[{ L = Len[rad,theta],
      ro = RhoOmegaFromRTheta[rad,theta]},
    On[Assert];
    With[{
        rho = Part[ro,1],
        omega = Part[ro,2]},
      With[{ hp = GetHelixParams[L,rho,omega]},
        With[{ r = Part[hp,1],
            theta2 = Part[hp,2]},
          Print[theta2 180 / Pi]
          Assert[Abs[theta2 - theta] < 0.00001];
          Assert[Abs[r - rad] < 0.00001];
          Off[Assert];
          ]]]]


  (* One way to try to solve this is to fold into
  one function, then use symmetries and zeroes there
  to simplify.)

  (* This is an intermediate point with the bisector operations
  unfolded *)
  UnifiedComp0[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      C = {0,0,L/2},
      q = QFromRhoOmega[L,rho,omega]
```

```
  },
  With[{ x = q Tan[omega],
      y = q Tan[rho]},
    With[{
        A = {-x,y,-(L/2 + q)},
        D = {x,y,(L/2 + q)},
        Bb = {x/2,-y/2,(q-L)/2},
        Cb = {-x/2,-y/2,(L-q)/2}},
      With[{CS = Skew[B,Bb,C,Cb]},
          With[{
          dr = {Norm[Part[CS,1] - Part[CS,2]],
          Norm[Part[CS,1] - B]}},
            With[{di = Part[dr,1],
                r = Part[dr,2]},
              With[{
                  c = ChordFromLD[L,di]},
                With[{ theta2 = ThetaFromRC[r,c]},
                  {r,theta2,di,c}
    ]]]]]]]


(* Now I attempt to unfold the Skew *)
UnifiedComp1[L_,rho_,omega_] :=
With[{
    B = {0,0,-L/2},
    C = {0,0,L/2},
    q = QFromRhoOmega[L,rho,omega]
  },
  With[{ x = q Tan[omega],
      y = q Tan[rho]},
    With[{
        A = {-x,y,-(L/2 + q)},
        D = {x,y,(L/2 + q)},
        Bb = {x/2,-y/2,(q-L)/2},
        Cb = {-x/2,-y/2,(L-q)/2}},
      With[{BbXCb = N[Cross[Bb,Cb]],
          CbXBb = N[Cross[Cb,Bb]]},
        With[{n2 = N[Cross[Cb,BbXCb]],
            n1 = N[Cross[Bb,CbXBb]],
            Bbz = Part[Bb,3],
            Cbz = Part[Cb,3]},
          (* Note: C-B exist only in z dimesion! *)
       With[{CmBz = Part[C-B,3],
           BmCz = Part[B-C,3],
           n2z = Part[n2,3],
           n1z = Part[n1,3]
```

```
          },
(*          Print[555555555]
          Print[(C-B) . n2]
          Print[(CmBz)  n2z]
          Print[B-C]*)
(*          With[{CmBz = L,
                BmCz = -L
           },
           *)
           (* Note: this dot product can be replaced with a
           z multiple, since the other terms are zero *)
(*             With[{C1 = B + (((C-B) . n2)/ (Bb . n2)) Bb,
                C2 = C + (((B-C) . n1)/ (Cb . n1)) Cb},*)
             (* But that allows us to cancel out... *)
             With[{C1 = B + ((CmBz  n2z)/ (Bb . n2)) Bb,
                 C2 = C + ((BmCz  n1z)/ (Cb . n1)) Cb},
(*             With[{C1 = B + (L / Bbz ) Bb,
                    C2 = C + (-L / Cbz) Cb}, *)
(* But that can be furher simplied... *)
(*              With[{C1 = B + (2 L / (q-L) ) Bb,
                C2 = C + (2 L / (q-L)) Cb}, *)
                  (* And then decomposed.... *)
 (*                 With[{S = 2 L / (q - L)},
                    With[{C1 = B + S Bb,
                        C2 = C + S Cb},
                      *)
                     With[{ CS =
                       {N[C1],N[C2]}},
                     With[{
                        dr = {Norm[Part[CS,1] - Part[CS,2]],
                          Norm[Part[CS,1] - B]}},
                       With[{di = Part[dr,1],
                           r = Part[dr,2]},
                         With[{
                             c = ChordFromLD[L,di]},
                           With[{ theta2 = ThetaFromRC[r,c]},
                             {r,theta2,di,c}
        ]]]]]]]]]]]]

N[UnifiedComp1[1, Pi/10, Pi/30]]
```

```
N[UnifiedComp2[1, Pi/10, Pi/30]]


 UnifiedComp2[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      C = {0,0,L/2},
      q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho]},
      With[{
          A = {-x,y,-(L/2 + q)},
          D = {x,y,(L/2 + q)},
          Bb = {x/2,-y/2,(q-L)/2},
          Cb = {-x/2,-y/2,-(q-L)/2}},
        With[{BbXCb = Cross[Bb,Cb],
            CbXBb = Cross[Cb,Bb]},
          With[{n2 =   Cross[Cb,BbXCb],
              n1 =  Cross[Bb,CbXBb]
            },
            With[{
             n2z = Part[n2,3],
             n1z = Part[n1,3]
              },
              With[{C1 = B + ((L  n2z)/ (Bb . n2)) Bb,
                  C2 = C + ((-L  n1z)/ (Cb . n1)) Cb},
                         With[{
                             dr = {Norm[C1 - C2],
                               Norm[C1 - B]}},
                           With[{di = Part[dr,1],
                             r = Part[dr,2]},
                           With[{
                               c = ChordFromLD[L,di]},
                             With[{ theta2 = ThetaFromRC[r,c]},
                               {r,theta2,di,c}
          ]]]]]]]]]]


  UnifiedComp3[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      C = {0,0,L/2},
      q = QFromRhoOmega[L,rho,omega]
```

```
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho]},
      With[{
          A = {-x,y,-(L/2 + q)},
          D = {x,y,(L/2 + q)},
          (* Apparently we can take out the fact of 1/2 (1/8) here *)
          Bb = {x,-y,(q-L)},
          Cb = {-x,-y,-(q-L)}},
        With[{BbXCb = Cross[Bb,Cb],
            CbXBb = Cross[Cb,Bb]},
          With[{n2 =   Cross[Cb,BbXCb],
              n1 =   Cross[Bb,CbXBb]
            },
            With[{
             n2z = Part[n2,3],
             n1z = Part[n1,3]
              },
              With[{Bn2 = n2z / (Bb . n2),
                    Cn1 = n1z / (Cb . n1)},
              With[{C1 = B + (L  Bn2) Bb,
                  C2 = C + (-L  Cn1) Cb},
                          With[{
                              dr = {Norm[C1 - C2],
                                Norm[C1 - B]}},
                            With[{di = Part[dr,1],
                                r = Part[dr,2]},
                              With[{
                                  c = ChordFromLD[L,di]},
                                With[{ theta2 = ThetaFromRC[r,c]},
                                  {r,theta2,di,c}
 ]]]]]]]]]]]]


N[UnifiedComp3[1, Pi/10, Pi/30]]

 UnifiedComp4[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      C = {0,0,L/2},
      q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho]},
      With[{
          A = {-x,y,-(L/2 + q)},
```

```
          D = {x,y,(L/2 + q)},
          (* Apparently we can take out the fact of 1/2 (1/8) here *)
          Bb = {x,-y,(q-L)},
          Cb = {-x,-y,-(q-L)}},
        With[{BbXCb = Cross[Bb,Cb],
            CbXBb = Cross[Cb,Bb]},
          With[{n2 =   Cross[Cb,BbXCb]
(*              n1 =   Cross[Bb,CbXBb] *)
            },
            With[{
             n2z = Part[n2,3]
(*             n1z = Part[n1,3] *)
             },
             With[{Bn2 = n2z / (Bb . n2)
               (*  Cn1 = n1z / (Cb . n1) *)
               },
             With[{C1 = B + (L  Bn2) Bb
(*                 C2 = C + (-L  Cn1) Cb *)
               },
               (* A Key point -- C1.y = C2.y, and C2.x = - C2.x, and C2.z = - C2.z *)
               (* This should simplify the norming operation below! *)
               (* I this case the norm depends only on x and z:
               With[{A = {a, b, c},
         B = {-a, b, -c}},
                 Norm[A - B]] => 2Sqrt[a^2 + c^2] *)
               (* So, in the first place, don't compute C2! *)
                        With[{
                            di = 2Sqrt[Part[C1,1]^2 + Part[C1,3]^2],
                            r =   Norm[(L  Bn2) Bb]},
                            With[{
                              c = ChordFromLD[L,di]},
                             With[{ theta2 = ThetaFromRC[r,c]},
                               {r,theta2,di,c}
  ]]]]]]]]]]



N[UnifiedComp4[1, Pi/10, Pi/30]]



 QFromRhoOmega[L_,rho_,omega_] :=
 L/Sqrt[1+Tan[rho]^2+Tan[omega]^2]

  ChordFromLD[L_,D_] := Sqrt[L^2 - D^2]
```

```
 ThetaFromRC[R_,C_] :=
 2 ArcSin[C/(2R)]

UnifiedComp5[L_,rho_,omega_] :=
 With[{
     B = {0,0,-L/2},
     q = QFromRhoOmega[L,rho,omega]
   },
   With[{ x = q Tan[omega],
       y = q Tan[rho],
     u = q - L},
     With[{
         A = {-x,y,-(L/2 + q)},
         D = {x,y,(L/2 + q)},
         (* We have removed a factor of 1/2 (1/8) here *)
         Bb = {x,-y,u},
         Cb = {-x,-y,-u}},
       With[{n2 =   Cross[Cb,Cross[Bb,Cb]]
         },
           With[{
            n2z = Part[n2,3]
             },
             With[{Bn2 = n2z / (Bb . n2)
               },
               With[{LBn2 = L Bn2},
             With[{C1 = B + LBn2 Bb
               },
               With[{
                   di = 2Sqrt[Part[C1,1]^2 + Part[C1,3]^2],
                   r =   Abs[LBn2] Norm[ Bb]},
                 With[{
                     c = ChordFromLD[L,di]},
                   With[{ theta2 = ThetaFromRC[r,c]},
                     {r,theta2,di,c}
 ]]]]]]]]]]


  (* Note, possibly this would be better done with C,D, or A,B *)
  (* Also, this neither takes nor returns tau, so it must be making an assumption *)
  (* How do we compute tau from this? tau should be computed from the points C1, D
  As teh change in the upvector. *)
UnifiedCompTwoPoints[L_,B_,D_] :=
With[{q = D[[3]] - L/2,
      x = D[[1]],
      y = D[[2]]},
```

```
    With[{u = q - L},
      With[{Bb = {x,-y,u},
          Cb = {-x,-y,-u}},
        With[{n2 =   Cross[Cb,Cross[Bb,Cb]]
          },
            With[{
             n2z = Part[n2,3]
              },
              With[{Bn2 = n2z / (Bb . n2)
                },
                With[{LBn2 = L Bn2},
              With[{C1 = B + LBn2 Bb
                },
                With[{
                    di = 2Sqrt[Part[C1,1]^2 + Part[C1,3]^2],
                    r =   Abs[LBn2] Norm[ Bb]},
                  With[{
                      c = ChordFromLD[L,di]},
                    With[{ theta2 = ThetaFromRC[r,c]},
                      {r,theta2,di,c}
  ]]]]]]]]]]


(* This does not seem to take tau into effect. Unified Comp needs to return tau *)
UnifiedComp7[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho],
      u = q - L},
      With[{
          D = {x,y,(L/2 + q)}
      },
      UnifiedCompTwoPoints[L,B,D]]]]


  N[UnifiedComp5[1, Pi/10, Pi/30]]
  N[UnifiedComp7[1, Pi/10, Pi/30]]


Plot3D[Part[UnifiedComp5[1,rho,omega],3],{rho,0,Pi/4},{omega,0,Pi/4},AxesLabel->{rho, omega}]

Plot3D[Part[UnifiedComp5[1,rho,omega],1],{rho,0,Pi/2},{omega,0,Pi/30},AxesLabel->{rho, omega}]
```

```
Plot[Part[UnifiedComp5[1,rho,0],1],{rho,0,Pi/2},AxesLabel->{rho}]

Plot[Part[UnifiedComp5[1,Pi/30,omega],1],{omega,0,Pi/2},AxesLabel->{omega}]



(* In this section, I am purely attempting to work out the
coordinates of the two face-to-face tetrahedra in order to
compute rho, omega in this platonic case. *)

(* This is all wrong because the centroid to centroid distance is actually 1/3 *)
(* These are values for a regular tetrahedron, I think. *)
(* These numbers are wrong, I think... *)
alphaA = 0;
alphaB = 0;
betaA = ArcTan[Sqrt[2]/2];
betaB = ArcTan[Sqrt[2]/2];


tau = 2 Pi / 3;

(* I now believe these are wrong, and the approach is probably
wrong. Instead we need to use Rodrigues' rule, rotate by tau,
and get the resulting unit vector back and apply simple trig
to it to get omega and rho. Note this also raises the
question that it might be better to consider the face angles
as unit vectors in the first place, but you still have to
rotate them. However, possibly unfolding the definition of rho and
omega as unit vectors will let us come closer to a formula.
We will, however, write this as rotating a vector of length L,
becasue that way a valuable point will be produced for our
compuation later. *)

(* I may not need this with built in Mathematica stuff. *)
vRot[k_,v_,theta_] := v Cos[theta] + Cross[k,v] Sin[theta]
+ k (k . v)(1 - Cos[theta])

vRotM[k_,theta_] :=
With[{ x = Part[k,1],
    y = Part[k,2],
    z = Part[k,3]},
  With[{C = {
        {0, -z, y},
        {z, 0, -x},
        {-y,x, 0}
    }},
```

```
    Print[C // MatrixForm];
    Print[(1 - Cos[theta])C^2 // MatrixForm];
    Print[IdentityMatrix[3] + (1 - Cos[theta])C^2 // MatrixForm];
    Print[IdentityMatrix[3] + (Sin[theta] C)+ (1 - Cos[theta])C^2 // MatrixForm];
    IdentityMatrix[3] + (Sin[theta] C) + (1 - Cos[theta]) C^2]]

vRotAux[k_,v_,theta_] :=
vRotM[k,theta].v

(* NOTE: This is incorrect, the vector v must be rotated into
k. I is the opposite of k (if symmetric) but must be rotated. *)
testvRot[tau_] :=
With[{alphaA = 0,
    alphaB = 0,
    betaA = -ArcTan[Sqrt[2]/2],
    betaB = ArcTan[Sqrt[2]/2],
    L = 1
  },
  With[{k = Normalize[{Sin[alphaB],Sin[betaB],Cos[betaB]}]},
    With[{
      v = Normalize[-L {Sin[alphaA],Sin[betaA],Cos[betaA]} + k]
    },
    Print[k];
    Print[Norm[k]];
    Print[v];
    Print[Norm[v]];
    With[{ rt = RotationTransform[tau,k]},
      With[{
          (*  vr = vRot[k,v,tau] *)
          vr = rt[v]
        },
      Print[N[vr]];
      Print[Norm[vr]];
      Print[N[Normalize[vr]]];
      With[{ x = Part[vr,1],
          y = Part[vr,2],
          z = Part[vr,3]},
        With[{ ro =
            (* Warning: Mathematic uses x,y order here, Javascript other *)
            { ArcTan[z,y],ArcTan[z,x]}},
          Print[N[ro 180 / Pi]];
          ro]]]]]]]
```

```
N[ArcTan[Sqrt[2]/2]]


(* Starting new work here. This is an attempt to compute
the point A (and thereby D) from NB and NC (normal vectors)
and tau (and L). This is based on my new understand that
we can compute a vector to be rotated.

I hope to test this with both regular regular objects,
but in the end with the regular tetrahedron, for which
we have a known solution (via Javascript work.)

Variables:

NB = Vector normal of the face containing B
NC = Vector normal of the face containg C
L = distance between B and C
tau = the twist

PA = the projection of BA onto the vector NB
R = The up-pointing rejection of BA onto NB. (perpendicular)
delta = angle between NC and the axis
gamma = angle between the XZ plane and the NB vector.

V0 = Vector to be rotated

AfromParams[v0,taux,NB] -- a function to compute A

V0fromLNB[L,NB,delta]

TestAWith225s[] = test AfromParams with faces cut and 22.5 degrees

*)

AngleWithXZ[v_] :=
VectorAngle[{v[[1]],0,v[[3]]},v]

(* I now believe this should be done with a rotation.
V0 is a rotation of NB vector by delta in the pure
up direction (that is, in the ZY plane about {1,0,0} *)
V0fromLNB[L_,NB_,NC_,delta_] :=
With[{k = Normalize[NB],
    rt = RotationTransform[-delta,{1,0,0}]},
  With[{ v0 = L Normalize[rt[NB]]},
    v0]
  ]
```

```
ADirFromParam[L_,v0_,tau_,NB_] :=
With[{k = Normalize[NB]},
  With[{ rt = RotationTransform[tau,k]},
    L  Normalize[rt[v0]]]]

ComputeBalancingRotation[NB_,NC_] :=
(* our stategy will be to project on the XY plane,
and then compute the midpoint vector *)
With[{ NBN = Normalize[NB],
    NCN = Normalize[NC]},
  With[{Bp = {NBN[[1]],NBN[[2]]},
      Cp = {NCN[[1]],NCN[[2]]}},
    Print[N[{Bp,Cp}]];
    With[{ ba = ArcTan[Bp[[1]],Bp[[2]]],
           ca = ArcTan[Cp[[1]],Cp[[2]]]},
    Print[N[ba] 180 / Pi];
    Print[N[ca] 180 / Pi];
    Print[N[(ba+ca)/2] 180 / Pi];
    With[{m = Bp+Cp},
      With[{mn = m / 2},
        Print[N[ArcTan[mn[[1]],mn[[2]]]] 180 / Pi];
        (* We want to return the angle that makes the negative of this vector point up *)
        Print[N[mn]];
        With[{theta = -(ba+ca)/2 + -Pi/2 },
          Print[70707070];
          Print[N[theta] 180 / Pi];
          With[{
              (*             rt = RotationTransform[{{mn[[1]],mn[[2]],0},{0,-1,0}}] *)
              rt = RotationTransform[-theta,{0,0,-1}]
            },
            With[{ Br = rt[NB],
              Cr = rt[NC]},
            Print[N[{theta,Br,Cr}]];
            {theta,Br,Cr}
    ]]]]]]]]

TestComputeBalancingRotation[] :=
With[{C = {1,1/2,1},
    B = {-1/2,-1,-1}},
  With[{res = ComputeBalancingRotation[B,C]},
    Print[res];
    With[{ theta = res[[1]],
        Br = res[[2]],
        Cr = res[[3]]},
      With[{Bp = Normalize[{Br[[1]],Br[[2]]}],
```

```
          Cp = Normalize[{Cr[[1]],Cr[[2]]}]},
        Print[8888888];
        Print[N[Bp]];
        Print[N[Cp]];
          Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[Bp[[2]] < 0];
        Assert[Cp[[2]] < 0];
        res
  ]]]]

TestComputeBalancingRotation2[] :=
With[{C = {1,3/4,1},
    B = {1,0,-1}},
  With[{res = ComputeBalancingRotation[B,C]},
    Print[N[res]];
    With[{ theta = res[[1]],
        Br = res[[2]],
        Cr = res[[3]]},
      With[{Bp = Normalize[{Br[[1]],Br[[2]]}],
          Cp = Normalize[{Cr[[1]],Cr[[2]]}]},
         Print[8888888];
        Print[N[Bp]];
        Print[N[Cp]];
        Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[Bp[[2]] < 0];
        Assert[Cp[[2]] < 0];
        res
  ]]]]

(* Warning: I may not have the sign of this correct. *)
AfromLtauNBNC[L_,tau_,NBunb_,NCunb_] :=
With[{ res = ComputeBalancingRotation[NBunb,NCunb]},
  With[{NB = res[[2]],
    NC = res[[3]]},
    With[{delta = VectorAngle[{0,0,1},NC]},
      With[{v0 = V0fromLNB[L,NB,NC,delta]},
        Assert[N[Norm[v0]] == L];
        With[{ Ad = ADirFromParam[L,v0,tau,NB],
            B = {0,0,-L/2}},
          Assert[Abs[Norm[Ad] -L ] < 0.00001];
          Ad + B]]]]]

(* For testing, I create a block of size 1
with NB and NC based on half of a 45 degree angle. *)
```

```
(* I need to write a function that takes two face normals,
and computes the balancing of them, returning two new normals
that are balanced around the up vector. Then I need
to test that the same point is computed from both directions
even when we have unbalanced normals. *)

TestAfromLtauNBNC[tau_] :=
With[{
    L0 = 1,
    NB0 = {0,-Sin[Pi/8],-Cos[Pi/8]},
    NC0 = {0,-Sin[Pi/8],Cos[Pi/8]},
    tetAngle = ArcTan[Sqrt[2]/2]},
  With[{
      NB1 = {0,-Sin[tetAngle],-Cos[tetAngle]},
      NC1 = {0,-Sin[tetAngle],Cos[tetAngle]},
      B = {0,0,-L/2}
    },
    With[{
        AA = AfromLtauNBNC[L0,tau,NB1,NC1]
      },
      AA
  ]]]

(* I need to write a test that sends the face normals
into UnifiedComp. Probably it would be better
to separate out a version of UnifiedComp that takes
points (the rho/omega version can implement that. *)


On[Assert];

TestRegularTets[] :=
With[{
    L0 = 1/3,
    tetAngle = ArcTan[Sqrt[2]/2]},
  With[{
    NB1 = {0,-Sin[tetAngle],-Cos[tetAngle]},
    NC1 = {0,-Sin[tetAngle],Cos[tetAngle]},
    tau = 2 Pi /3
    },
    With[{
        A = AfromLtauNBNC[L0,tau,NB1,NC1]},
      Print[N[A]];
      With[{ B = {0,0,-L0/2},
          D = {-A[[1]],A[[2]],-A[[3]]}
        },
```

```
        With[{ res = UnifiedCompTwoPoints[L0,B,D],
            theta = ArcCos[-2/3]},
          Assert[Abs[res[[2]] - theta] < 0.00001];
]]]]]

TestRegularTets2[] :=
With[{
    L0 = 1,
    tetAngle = ArcTan[Sqrt[2]/2],
    (* Rotation by an aribitraty amount should give us the same number. *)
    testRot = Pi/17
  },
  With[{ rt = RotationTransform[testRot,{0,0,1}]},
    With[{
        NB1 = rt[{0,-Sin[tetAngle],-Cos[tetAngle]}],
        NC1 = rt[{0,-Sin[tetAngle],Cos[tetAngle]}],
        tau = 2 Pi /3
      },
      With[{
          A = AfromLtauNBNC[L0,tau,NB1,NC1]},
        With[{ B = {0,0,-L0/2},
            D = {-A[[1]],A[[2]],-A[[3]]}
          },
          With[{ res = UnifiedCompTwoPoints[L0,B,D],
              theta = ArcCos[-2/3]},
            Assert[Abs[res[[2]] - theta] < 0.00001];
]]]]]]
```