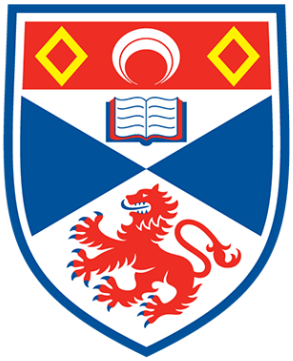
Topological Origami: Material Properties from Geometry and Combinatorics

Joseph Edwards



University of
St Andrews

supervised by
Dr. Louis Theran

August 2021

Abstract

Through the analysis of infinitesimal flexes and stresses, we explore the rigidity and flexibility of infinite screw-periodic bar-joint frameworks.

1 Introduction

In 1766, Leonhard Euler made the claim that "a closed spatial figure allows no changes, as long as it is not ripped apart". In other words, this is saying that a polyhedron made of solid faces and flexible joints cannot be deformed whilst maintaining face shape and connectivity. It was this conjecture that ignited the study of rigidity theory.

In the years to come, many steps were taken on the journey to prove or disprove this conjecture of Euler. In 1813, Cauchy proved his eponymous rigidity theorem that states all *convex* polyhedra are rigid and, in 1974, Gluck improved on this, by proving that almost all polyhedra are rigid [3]. Despite results that may have suggested Euler was correct, in 1977, Connelly found a counterexample [1].

During this process of ultimately disproving Euler, we cemented the foundations of modern rigidity theory. By bringing mathematical rigour and purity into a field which was historically used in the manufacturing industry, we have unlocked new techniques and algorithms for generating large rigid frameworks [5] that have real world applications.

In this essay, we focus on a special type of bar-joint framework, namely infinite screw-periodic frameworks. By analysing the stresses and flexes that exist within these structures, we seek methods and algorithms that can be used to categorise the different constructions of rigid frameworks.

2 Preliminaries and Definitions

Before we can address screw-periodic frameworks, we must first present some background definitions and theorems from geometry and graph theory.

2.1 Classical Frameworks

In rigidity theory, we want to study idealised structures embedded in d-dimensional space, consisting of rigid bars that are free to rotate about universal joints. This idea is encapsulated in the following definitions.

Definition 2.1.1 (Configuration)**.** Let $d \in \mathbb{Z}^+$ be a dimension. A *configuration* on n points in dimension d is an ordered tuple

$$\boldsymbol{p} = (p_1, \ldots, p_n)$$

of n points such that $p_i \in \mathbb{R}^d$ for all i.

Definition 2.1.2 (Framework)**.** Let $d \in \mathbb{Z}^+$ be a dimension and $G = (V, E)$ be a simple graph with n vertices. Then a *framework* in dimension d is the pair $(G, \boldsymbol{p})$, where $\boldsymbol{p}$ is a configuration on the n vertices of G in dimension d.

2.2 Infinitesimal Rigidity

As one might expect, we are very much interested in the rigidity of these structures. We can analyse this by considering the possible configurations of a framework that preserve the lengths of the edges, however the non-linearity of this problem makes calculations quite difficult. As a result, for the purpose of this essay, we will be focusing on the linearisation of this problem - that of *infinitesimal rigidity*.

Definition 2.2.1 (Infinitesimal flex)**.** Let $(G, \boldsymbol{p})$ be a framework in dimension d with n vertices. Then a configuration $\boldsymbol{v}$ of n vectors in $\mathbb{R}^d$ is an *infinitesimal flex* if

$$\langle p_j - p_i, v_j - v_i \rangle = 0 \quad \text{for all } ij \in E.$$

An infinitesimal flex is *trivial* if

$$\langle p_j - p_i, v_j - v_i \rangle = 0 \quad \text{for all } i \neq j \in V.$$

Notice that the second statement is much stronger, as it means that we must also preserve non-edge lengths when flexing the framework.

Definition 2.2.2 (Infinitesimally rigid)**.** Let $(G, \boldsymbol{p})$ be a framework in dimension d with $n \geqslant d+1$ vertices. Then $(G, \boldsymbol{p})$ is *infinitesimally rigid* if every infinitesimal flex is trivial.

For frameworks $(G, \boldsymbol{p})$ with fewer than $d + 1$ vertices, we define them to be infinitesimally rigid if they are complete and p is affinely independent.

Definition 2.2.3 (Rigidity Matrix)**.** Let $(G, \boldsymbol{p})$ be a framework in dimension d with n vertices and m edges. The *rigidity matrix* $\mathcal{R}(\boldsymbol{p})$ of $(G, \boldsymbol{p})$ is the $m \times dn$ matrix of the system

$$\langle p_j - p_i, v_j - v_i \rangle = 0 \quad \text{for all } ij \in E.$$

In other words, the rigidity matrix is the $m \times dn$ matrix with columns that correspond to vertices p_i of $(G, \boldsymbol{p})$, and rows that correspond to edges $p_i - p_j$ of $(G, \boldsymbol{p})$. We observe that the nullspace of this matrix gives the vector space of infinitesimal flexes.

In order to begin to familiarise ourselves with these definitions, we proceed by considering the following example.

Example 2.2.4. Consider the 4-cycle graph C_4. We choose to embed this framework as a square in $\mathbb{R}^2$ with the configuration

$$\boldsymbol{p} = \{p_1 = (0, 0),\ p_2 = (0, 1),\ p_3 = (1, 1),\ p_4 = (1, 0)\},$$

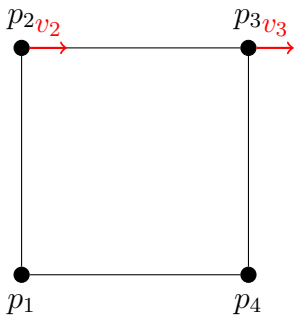


Figure 1: A square in $\mathbb{R}^2$ with an infinitesimal flex.

as in Figure 1. Then by assigning the vertices the non-trivial infinitesimal flex

$$\boldsymbol{v} = \{v_1 = (0,0),\ v_2 = (1,0),\ v_3 = (1,0),\ v_4 = (0,0)\},$$

we see that this framework is not infinitesimally rigid - say infinitesimally flexible. For completeness, we also construct the symbolic and numeric rigidity matrices $\mathcal{R}(\boldsymbol{p})$:

$$\begin{pmatrix} p_1 - p_2 & p_2 - p_1 & 0 & 0 \\ 0 & p_2 - p_3 & p_3 - p_2 & 0 \\ 0 & 0 & p_3 - p_4 & p_4 - p_3 \\ p_1 - p_4 & 0 & 0 & p_4 - p_3 \end{pmatrix}$$

and

$$\begin{array}{c} \overbrace{}^{p_1}\ \overbrace{}^{p_2}\ \overbrace{}^{p_3}\ \overbrace{}^{p_4} \\ \begin{pmatrix} 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

2.3 Rigid Constructions

So far, we have seen one way in which we can take a framework and begin to classify its rigidity. Therefore, a reasonable question to ask is "which other frameworks share this property of infinitesimal rigidity"? One approach to this question would be to consider every graph with every configuration, and find its non-trivial flexes. However, since neither you nor I have access to an uncountably infinite amount of time, this is perhaps a bit ambitious. Instead, we look at ways in which we can transform one framework into another. For our purposes, we consider the vertex split as described by Whiteley in [6].

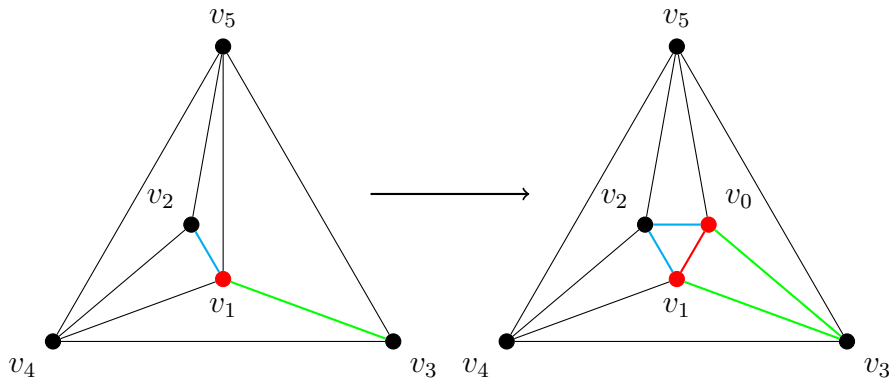


Figure 2: Vertex split on two edges in dimension 3, where $N_1 = \{v_2, v_3\}$, $N_2 = \{v_4\}$ and $N_3 = \{v_5\}$.

Definition 2.3.1 (Vertex split)**.** Let $G = (V, E)$ be a graph, and $v \in V$ have neighbours $N = \{u_1, \dots u_k\}$. Partition N into three sets N_1, N_2, N_3, specifically where $N_1 = \{u_1, u_2\}$. A *vertex split* on v in dimension 3 adds the vertex v', removes the edges $\{vu_i \mid u_i \in N_3\}$, and adds the edges $\{v'u_j \mid u_j \in N_1 \cup N_3\}$.

We can think of this vertex split as the inverse operation to contracting along the shared edge of two triangles in the graph. For an example, see Figure 2.

To know when a new framework is infinitesimally rigid or not, we look to the following theorem.

Theorem 2.3.2 (Whiteley [6])**.** *Given an infinitesimally rigid framework $(G, \boldsymbol{p})$ in dimension 3 with vertex v, the new framework $(G', \boldsymbol{p}')$ formed by a vertex split on v is infinitesimally rigid for almost all placements $p'_{v'}$ of the new vertex v'.*

2.4 Screw-Periodic Frameworks

We have seen above how we can use classical frameworks and infinitesimal flexes to learn about the rigidity of finite structures. However, it is sometimes interesting to analyse infinite examples, and it is this that we shall focus on for the remainder of the essay. For our definitions of screw-periodic frameworks, we follow the constructions described by McInerney et al. in [4].

Definition 2.4.1 (Periodic Graph)**.** Let $G = (V, E)$ be a finite graph. Then $\tilde{G} = (\tilde{V}, \tilde{E})$ is a *periodic graph* if there exists $\Lambda < \mathrm{Aut}(G)$ such that Λ is isomorphic to $\mathbb{Z}^2$.

We may think of a periodic graph as a *quotient* graph G, and a set of equivalence classes of vertices, such that each equivalent vertex is related by a *marking* $\boldsymbol{n} \in \mathbb{Z}^2$.

In order to go from a periodic graph to a screw-periodic framework, we must define the transformations, also known as screw motions, that determine the vector positions of the configuration. These screw motions consist of two rotations, S_1 and S_2, and two translations, l_1 and l_2, that satisfy

$$S_1 S_2 = S_2 S_1 \tag{1}$$

$$l_1 + S_1 l_2 = l_2 + S_2 l_1. \tag{2}$$

Vertices of the quotient graph are placed in general position, and equivalent vertices are rotated and translated according to their marking $\boldsymbol{n}$. From this, we get that the $\boldsymbol{n} = (n_1, n_2)$ copy of the identity vertex placed at p_i, say $p_i(\boldsymbol{n})$, has the location given by summing the rotations and transformation required to get to it:

$$p_i(\boldsymbol{n}) = \sum_{i=0}^{n_1-1} S_1^i l_1 + S_1^{n_2} \sum_{j=0}^{n_2-1} S_2^j l_2 + S_1^{n_1} S_2^{n_2} p_i. \tag{3}$$

We observe that constraint 1 ensures that both rotations share a common axis, and that constraint 2 ensures that we arrive at the same vector, irrespective of the order in which we apply S_1 and l_1 or S_2 and l_2.

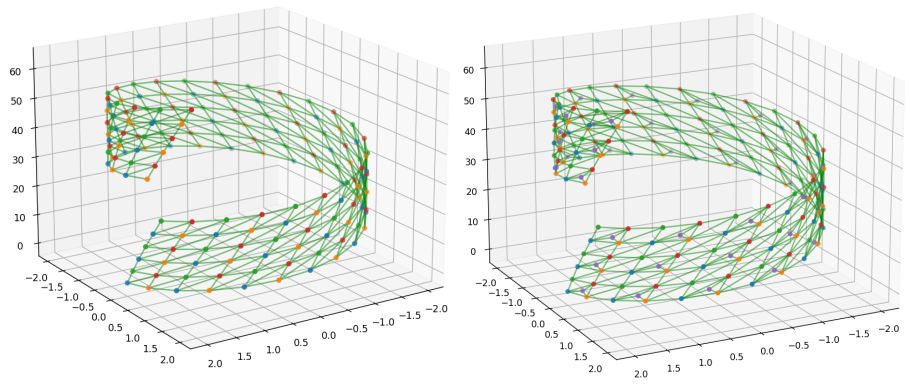
With the idea above in mind, we finally consider a vertex split on a screw-periodic framework. Heuristically, this type of vertex split is achieved by taking a single cell in the screw-periodic framework, applying the vertex split operation as described in Definition 2.3.1, and then having that split 'percolate through' every other equivalent cell in the framework. A more robust definition can be defined using the quotient graph, though that presents unnecessary complications that we need not consider for our purposes. For an example of screw-periodic a framework, see Figure 3.

3 Results

In this section, having gained a thorough appreciation for the study of classical bar-joint frameworks, we look to prove a new result that pertains specifically to the generation of rigid screw-periodic frameworks.

Proposition 3.0.1. *Let $(\tilde{G}, \tilde{\boldsymbol{p}})$ be a screw-periodic framework modelled on a triangulation of the torus, with quotient framework $(G, \boldsymbol{p})$ in general position. Then $(\tilde{G}, \tilde{\boldsymbol{p}})$ is infinitesimally rigid.*

The proof of this proposition requires two steps. Firstly, we require a geometric argument that shows vertex splitting preserves infinitesimal rigidity. Secondly, we require a combinatorial argument that shows every screw-periodic graph on the



(a) Framework before split. (b) Framework after split.

Figure 3: An example of a vertex split on a screw-periodic framework, with the new vertex in purple, generated using [2].

triangulation can be generated by performing a series of vertex splits on a graph from a finite collection of bases.

For the purpose of this essay, we first focus on the proof of step one, and then discuss the code that will be used to prove the second step.

3.1 Vertex Splits

Theorem 3.1.1. *Let $(\tilde{G}, \tilde{\boldsymbol{p}})$ be a screw-periodic framework modelled on a triangulation of the torus, with quotient framework $(G, \boldsymbol{p})$ in general position. Then the framework $(\tilde{G}', \tilde{\boldsymbol{p}}')$ formed by a vertex split on two non-parallel bars of the quotient graph is rigid for almost all positions of the added vertex v'.*

We take a step back to state an intermediate result that will ultimately be used to prove the above theorem.

Lemma 3.1.2. *Let $(\tilde{G}, \tilde{\boldsymbol{p}})$ be a screw-periodic framework modelled on a triangulation of the torus, with quotient framework $(G, \boldsymbol{p})$ in general position. Suppose that this framework has a stress space of dimension k. Then the framework $(\tilde{G}', \tilde{\boldsymbol{p}}')$ formed by a vertex split on two non-parallel bars has a stress space of dimension at most k for almost all positions of the added vertex v'.*

Proof. Suppose that edges of the graph on which we split are vu_1 and vu_2, and suppose further that the $T(p_i, \boldsymbol{n})$ is defined to be the transformation given in Equation 3. Then we say that v has position p_v, u_1 has position $T(p_1, \boldsymbol{n}_1)$ and u_2

has position $T(p_2, \boldsymbol{n}_2)$. Therefore, the edge vectors are given by $p_v - T(p_1, \boldsymbol{n}_1)$ and $p_v - T(p_2, \boldsymbol{n}_2)$.

By our hypothesis, the two edges are non-parallel. Therefore, there exists no non-trivial linear combination of the two that sum to zero, and thus these two edges cannot independently support a non-trivial stress. While the dimension of our stress space is greater than 0, by the above, there exist some edge with a non-zero stress coefficient. By removing this edge, we reduce the stress space by one. Repeating this process k times, we are left with a stress-free framework. With this, we proceed by following a proof very similar to Whiteley's proof of Theorem 2.3.2 as presented in [6].

Suppose that we choose to add our split vertex v' at $p_{v'}$. To remove the issue of a zero edge vector, take the edge to be in the direction $d \neq \mathbf{0}$, and not in the plane defined by $p_v - T(p_1, \boldsymbol{n}_1)$ and $p_v - T(p_2, \boldsymbol{n}_2)$ (this does not strictly constitute as a framework due to the 'fake' edge, but it defines a rigidity matrix and we shall refer to it as a pseudo-framework). Assume this pseudo-framework has a self stress ω. By setting $\omega'_{v1} = \omega_{v1} + \omega_{v'1}$, $\omega'_{v2} = \omega_{v2} + \omega_{v'2}$ and $\omega'_{vj} = \omega_{v'j}$ for $j > 2$, we define a stress ω' on the original framework. However, since the original framework was stress free, $\omega'_{ij} = 0$ for all i, j. Thus, by summing around v and ignoring terms with zero coefficients, we have

$$\omega_{v1}(p_v - T(p_1, \boldsymbol{n}_1)) + \omega_{v2}(p_v - T(p_2, \boldsymbol{n}_2)) + \omega_{vv'}d = \mathbf{0}.$$

Since d is not in the plane defined by the two edge vectors, we necessarily have $\omega_{v1} = \omega_{v2} = \omega_{vv'} = 0$. By a similar argument summing around v', we get $\omega_{v'1} = \omega_{v'2} = 0$. Therefore, we conclude that our stress ω is trivial. A small change on the position of v' along the direction of d creates a true framework that preserves the independence in the rigidity matrix, thus showing that the vertex split reduced framework has stress space of dimension 0.

If any of the edges we removed earlier were incident to v, we are free to choose whether to re-add them incident either to v or v'. By also re-adding the remainder of the k edges, the dimension of the stress space goes up by at most k. □

With the above result giving us an upper bound on the dimension of the stress space, we seek a lower bound.

Proof of Theorem 3.1.1. By the Index Theorem, any generically placed triangulated surface modelled on the torus is infinitesimally rigid if and only if it has a stress space of dimension two, and that there exists no triangulated surface with stress space less than two. By the above theorem, when performing a vertex split on an infinitesimally rigid triangulated surface, the dimension of the stress space is less than or equal to two.

Combining these two inequalities gives that the dimension of the framework formed after performing a vertex split on a generically placed triangulated surface is equal to two, and is therefore infinitesimally rigid. □

4 Code

As discussed at the beginning of Section 3, the second part of proving Proposition 3.0.1 will require computational assistance. In this section, we discuss the Python package `Rigidity` [2] that was developed during this project for that purpose.

`Rigidity` is a linear algebra-type package intended to model both classical and screw-periodic frameworks. Once a model has been created, we can perform functions that return the non-trivial flex spaces, generate the rigidity matrix, perform vertex splits, draw and manipulate the framework in 3 dimensional space, and perform other symbolic and numerical analyses.

4.1 `rigidity.py`

The main module in the `Rigidity` package is `rigidity.py`. It is in this module where we define the three classes that are used to model our frameworks - `Framework`, `AffineTransform` and `ScrewFramework`.

4.1.1 The `Framework` Class

This is the class used to model classical frameworks. When initialised, this class takes two parameters - a graph and a configuration. We implement a graph as a list of length 2, where the first element is a list of vertices, and the second element is a dictionary that represents edges of the graph. For the configuration, we take advantage of `numpy`'s `array` class to create a $d \times n$ matrix to describe the n vectors of the configuration. With these specified, we now define some functions that either perform computations on the framework, or help us visualise the framework.

The first function that we highlight is the `draw` function. Provided that the given framework is in either dimension two or three, the `draw` function uses `matplotlib.pyplot` to render the framework as a series of line graphs for edges, and a scatter plot for the vertices.

Although relatively simple to implement, this function is extremely powerful due to the nature of the object it returns. If `Rigidity` has been imported into a Jupyter Notebook, when the `draw` function is called alongside the magic `%matplotlib widget` statement, the framework is displayed and can be rotated and scaled, allowing the user to see the finer details of the framework from all

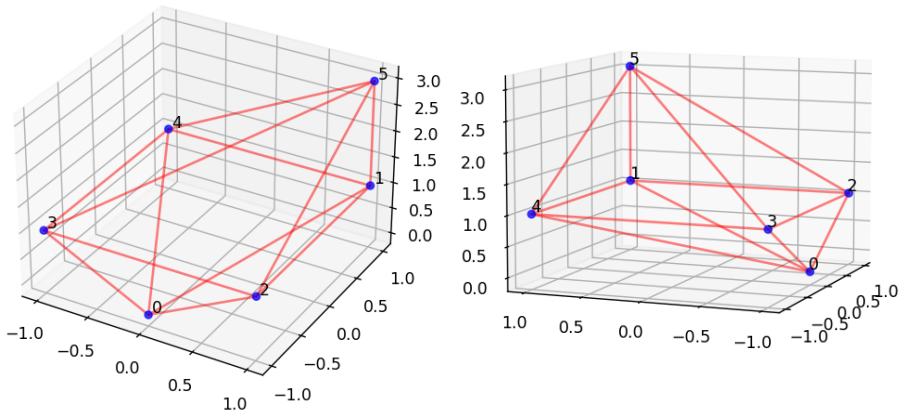


Figure 4: A 3 dimensional framework viewed from different points of view, generated using [2].

directions. For an example of the types of framework this function can display, see Figure 4.

The next function we will highlight is the `nonTrivialFlex` function. When called on a `Framework` object, perhaps unsurprisingly, this function returns an array that represents an orthonormal basis of the vector space of non-trivial flexes that exist on the framework. In order to achieve this we first use `scipy.linalg` to calculate the nullspace of the rigidity matrix, giving the space of infinitesimal flexes on the framework. Next, we generate the space of *trivial* infinitesimal flexes (translations and rotations) that exist on the framework. We then calculate the projection matrix of this space, and project the flexes away from the trivial flexes, leaving a set of non-trivial flexes. Finally, we calculate an orthonormal basis of the span of this set, and return it.

This function has a lot of utility in itself, as we can use the existence or non-existence of infinitesimal flexes to categorise the rigidity of a framework. However, in the spirit of *seeing things makes them easier to understand*, the `drawFlex` function was created to visualise these flexes. When called on a framework with an array to represent the vectors of a flex, this function draws a `quiver` plot that adds arrows to the vertices of a framework. For an example, see Figure 5.

The final function from this class that we highlight is the `vertexSplit` function. As the name suggests, given a framework, a vertex, and a partition of the set of neighbours of that vertex, this function returns the framework obtained by performing a vertex split as outlined in Definition 2.3.1. As our previous theorem of Whiteley tells us, vertex splits are an excellent way of generating more infinitesimally rigid frameworks. Therefore, the inclusion of this function allows for the

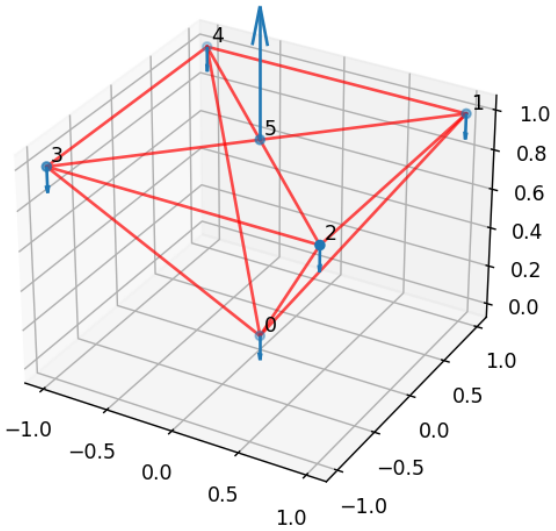


Figure 5: A flattened octahedron drawn with a non-trivial infinitesimal flex, generated using [2].

rapid computation of many infinitesimally rigid examples.

4.1.2 The `AffineTransform` Class

Compared to the `Framework` class, `AffineTransform` is much smaller. As described in the preliminaries, in order to define a screw-periodic framework, we must first define screw motions - this is the role of the `AffineTransform` class.

Two parameters are required when initialising an object from this class; a square skew-symmetric matrix and a translation vector. With these specified, a callable object is created. These objects are called on flat `numpy` arrays, with an optional power parameter that states how many times the transformation should be applied. If the given square skew-symmetric matrix is invertible, negative powers may be supplied to represent the inverse of the screw motion.

With the screw motions modelled, we may now create our screw-periodic frameworks.

4.1.3 The `ScrewFramework` Class

The `ScrewFramework` class, much like the `Framework` class, was designed to model, display and analyse a type of framework, namely screw-periodic frameworks. Unlike classical frameworks, however, the screw-periodic framework is infinite. Therefore, through the necessity of having a finite amount of storage, several differences arise between the two implementations.

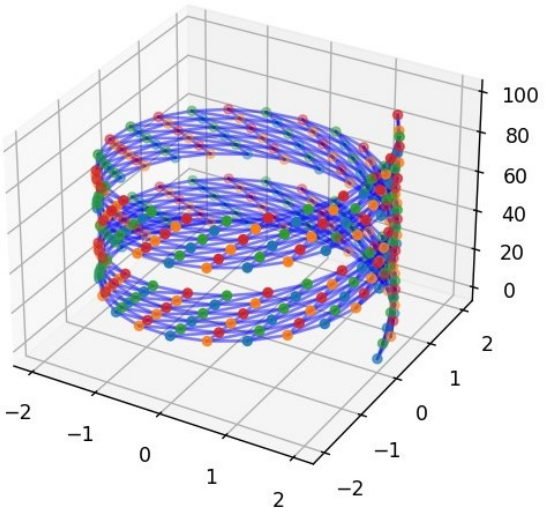


Figure 6: A finite segment of a screw-periodic framework, generated using [2].

The first difference relates to the implementation of the graph. Rather than storing a copy of the graph, we instead store a copy of the quotient graph. For the equivalence classes of vertices, we create a list of integers as the representatives. For the edges, we cannot simply provide a start and an end vertex - we must also state which copy of the vertices we are interested in. As such, our implementation of the edges is a dictionary, where the keys are our vertex representatives, and the values are lists of 3-tuples, where the first element is the vertex at the end of the edge, and the second and third index represent an element of $\mathbb{Z}^2$ for the marking.

The second difference is that of the `draw` function. In addition to the optional arguments for `matplotlib`, we must also pass an element of $\mathbb{Z}^2$ to represent which portion of the infinite framework to display. With this information, our `AffineTransform` objects are called on the vectors of the quotient configuration to calculate the necessary vertex locations, as in equation 3, and the framework is drawn. For an example of the graphics this function can create, see Figure 6.

5 Acknowledgements

Firstly, I want to thoroughly thank my supervisor, Dr Louis Theran, for his patience and support during my journey exploring rigidity theory. Without his help in defining the project, the frequent whiteboard sessions, the help in preparing the material for this document and the great number of hours he has dedicated to me, the completion of this project would have been all but impossible.

Secondly, I would like to thank my parents and peers that have supported me through this process. It is fair to assume that many of them are not fluent in the language that is degree level maths, let alone rigidity theory, but they have tirelessly listened to me talk about my project, and even put in the effort to try and support me with the maths. For this, I am extremely grateful.

Finally, I would like to thank Lord Laidlaw and the wider Laidlaw foundation for funding this research project and granting me this incredible opportunity to develop as a leader and a scholar. I have learnt many lessons about myself during the first year of my scholarship, and I can't wait to apply them in the future. So once again, thank you all.

References

[1] Robert Connelly. "A counterexample to the rigidity conjecture for polyhedra". eng. In: *Publications Mathématiques de l'IHÉS* 47 (1977), pp. 333–338. URL: http://eudml.org/doc/103949.

[2] Joseph Edwards and Louis Theran. *Rigidity: A Package for Studying Rigidity Theory*. Aug. 2021. URL: https://github.com/theran/laidlaw-origami.

[3] Herman Gluck. "Almost all simply connected closed surfaces are rigid". In: *Geometric Topology*. Ed. by Leslie Curtis Glaser and Thomas Benjamin Rushing. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975, pp. 225–239. ISBN: 978-3-540-37412-1.

[4] James McInerney et al. "Hidden symmetries generate rigid folding mechanisms in periodic origami". In: *Proceedings of the National Academy of Sciences* 117.48 (Nov. 2020), pp. 30252–30259. ISSN: 1091-6490. DOI: 10.1073/pnas.2005089117. URL: http://dx.doi.org/10.1073/pnas.2005089117.

[5] Tiong-Seng Tay and Walter Whiteley. "Generating Isostatic Frameworks". In: *Structural Topology* 11 (Jan. 1985), pp. 27–33.

[6] Walter Whiteley. "Vertex Splitting in Isostatic Frameworks". In: *Structural Topology* 16 (Jan. 1990).