Extending the de Bruijn dual grid method to arbitrary dimensions.

Joshua Colclough
Supervisor: Dr Felix Flicker, Cardiff University

Abstract

The de Bruijn dual grid method of generating rhombic tilings of two-dimensional space is implemented in Python. The method is then extended to three dimensions to generate structures such as icosahedral quasicrystals. It is then further generalised to generate arbitrary structures of n-dimensional space where the structures generated have rhombuses as faces.

Contents

1 Introduction.

1.1 Background.

Aperiodic order has been of interest to humanity for centuries. Some of the earliest examples of our interest in aperiodic patterns can be found in religious architecture dating from the 10th century [1]. Since then, aperiodic tilings have been a large part of recreational mathematics, with Sir Roger Penrose expanding upon Kepler's work [2] to discover Penrose Tilings; aperiodic tilings that exhibit 5-fold rotational symmetry. A section of Penrose tiling can be seen in Fig. 1. The Dutch mathematician Nicolaas Govert de Bruijn expanded further upon Penrose's work, discovering an algebraic method of generating the Penrose tiling, and other general aperiodic tilings of rotational symmetry in the plane [3], named the de Bruijn grid method.

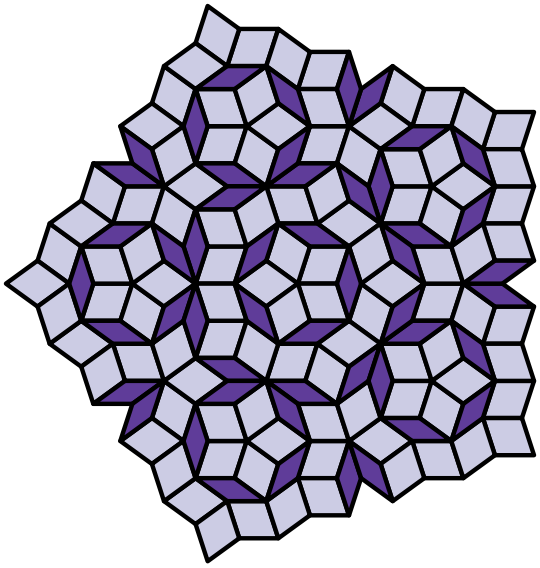


Figure 1: A section of Penrose tiling.

1.2 Motivation.

One particular area of interest relating to aperiodic structures is quasicrystals; ordered yet aperiodic structures found in some forms of matter. According to the crystallographic restriction theorem, crystals can only possess two, three, four and six-fold rotational symmetry, simply due to the requirement of discrete translational symmetry. This all got turned on its head with the discovery of these aperiodic structures, as these structures displayed no ordered translational symmetry and could not be described in terms of a single unit cell (as can be seen in the Penrose tiling in Fig. 1).
Furthermore, the discovery of materials that displayed structure with strong correlation to 5-fold (and other odd symmetries) symmetrical tilings when viewed down specific axes confirmed that the restrictive model of crystals had to be reconsidered.

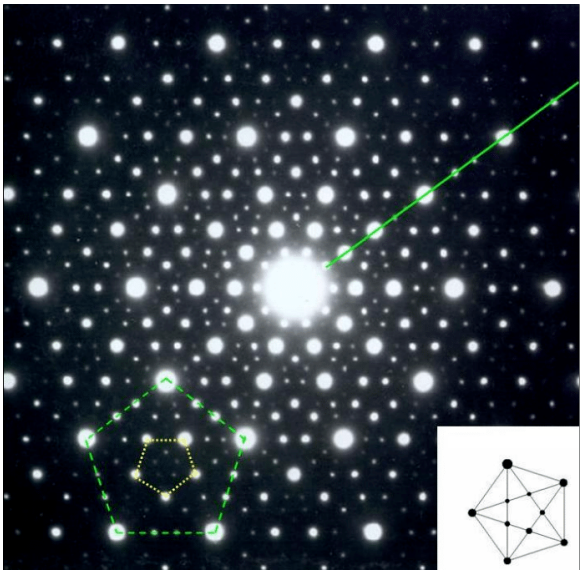


Figure 2: The electron diffraction pattern for an AlNiCo decagonal crystal. A 10-fold rotational symmetry can be seen. A Pythagorean pentagon is shown in the bottom right with a trace of one drawn on top of the electron diffraction pattern, showing the 10-fold symmetry. Image taken from [4].

It was evident that there was order in the material due to the clear Bragg diffraction patterns, meaning that these materials must be some new form of crystal as opposed to a disordered material.

An example of a material that has an abnormal diffraction pattern is an aluminium-manganese alloy Al_6Mg, which was found to have an electron diffraction pattern with 10-fold symmetry [5]. An example of a 10-fold electron diffraction pattern can be seen for an aluminium nickel cobalt decagonal crystal (AlNiCo) in Fig. 2.

An implementation for generating aperiodic tilings of the plane would therefore be useful to crys-

tallography but also many other disciplines. There are many methods of generating these aperiodic tilings. One of the more popular ones being the “inflation” method, whilst the method I detail in this report is the “de Bruijn dual grid” method.
Inflation takes the unique tiles (two for the Penrose tiling) used in the tiling, and breaks them down into smaller sections of the same primitive tiles. This step is called composition. By matching up the new inflated tiles with each other so that they form the correct primitive tile shapes you can form a new inflated tiling. By repeating this you can generate the tiling to any arbitrary size. However, due to the nature of this method you can only produce tilings of a single form; it is not a generic algorithm, it is limited to producing whatever tiling it has been designed for. You have to define a set of rules for each tiling . An example of inflated Penrose rhombs can be seen in Fig. 3.

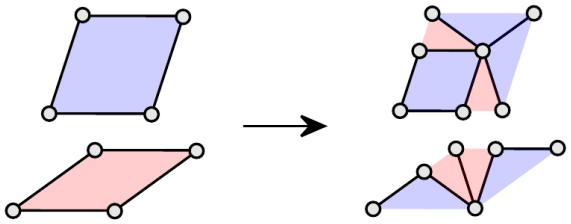


Figure 3: Diagram showing a single inflation step performed on Penrose rhombs. Image taken from [6].

Whilst the inflation method is fine for well known tilings such as the Penrose tiling and Ammann–Beenker (8-fold) tiling, it does not cover every generic case. The inflation method has to be pre-determined for each rotational symmetry. In contrast, the de Bruijn grid method can generate aperiodic tilings with arbitrary rotational symmetry with a single algorithm.

The aim of this project is to create a Python implementation of the de Bruijn dual grid method in arbitrary dimensions. The de Bruijn grid method is an elegant approach to generating aperiodic tilings, exploiting the fact that the tilings consist of sets of “Conway worms”;

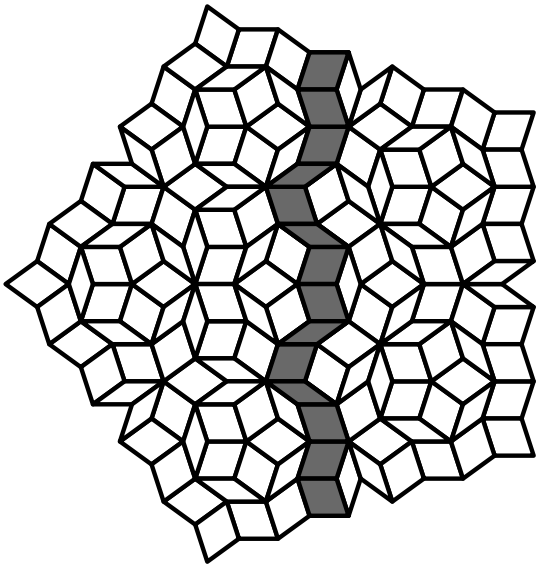


Figure 4: A Conway worm highlighted within a Penrose tiling. They are formed by matching parallel edges of neighbouring rhombuses. They continue infinitely in the plane [7].

lines of rhombi formed by matching parallel edges of the rhombi together. Fig. 4 shows a segment of a single Conway worm that forms along one of the lines used in the construction of a Penrose tiling. These Conway worms overlap to form the final tiling.

A brief summary of the de Bruijn dual grid method: It takes N unit vectors in the plane, and places sets of parallel lines perpendicularly to these basis vectors. Each empty space the grid (or ”multigrid“ as we will see later) formed by the superposition of the individual grids on top of one another directly corresponds to vertices in the final tiling. This tiling fills two dimensional space.

Quasicrystals are three-dimensional, whereas these aperiodic tilings are tilings of the two-dimensional plane. The de Bruijn grid method is very general, and so it should be expandable to three dimensions. The idea is that instead of using two-dimensional lines to form grids, planes are used, where some basis shape defines the directions at which these grid planes face. The spaces in between the multigrid will then directly correspond to vertices in the final structure. The algorithm could also be expanded to any arbitrary number of dimen-

sions, the use cases for which are slightly more abstract but do exist. For example, quasicrystals are known to exist up to and including four dimensions [8].

2 Methodology.

2.1 Prerequisites.

In the first part of the methodology I will detail N.G. de Bruijn's
two-dimensional dual grid method as detailed in his 1981 paper.
There will however be a key difference in approach: N.G. de Bruijn uses the complex number space to represent directions in
two-dimensional space. However, this will not suffice when expanding the method to higher dimensions. Therefore where there are complex numbers in de Bruijn's paper, I will be translating this into vector algebra.
An example of this is in the equation used to generate basis vectors: Eq. (2). This equation is represented as

$$\zeta = e^{\frac{2\pi i}{5}} \tag{1}$$

in de Bruijn's paper [3]. $\zeta^j, j \in \mathbb{Z}$ gives you basis vectors evenly spaced around a unit circle. Using Euler's equation gives you the real and imaginary components. Setting the real component as the x axis and the complex component as the y axis allows us to use vector algebra.

2.2 The de Bruijn dual grid method.

Choose N two-dimensional unit vectors to form the basis of the tiling. For the Penrose case, we chose five unit vectors that are evenly spaced around a unit circle [3].

$$\boldsymbol{e}_j = \cos\left(j\frac{2\pi}{S}\right)\hat{\mathbf{i}} + \sin\left(j\frac{2\pi}{S}\right)\hat{\mathbf{j}} \qquad j, S \in \mathbb{Z} \tag{2}$$

Where S is the order of rotational symmetry. In the Penrose case $S = 5$, and j takes the values $j = 0...(N-1)$, in this case $N = 5$ and so $j = \{0, 1, 2, 3, 4\}$ [3].
It will be seen later that every vertex in the tiling will be a sum of integer multiples of these basis vectors.

Once the basis has been chosen, a grid is placed normal to each of the basis vectors. A grid is defined as a set of equally spaced parallel lines, where each line within each grid is assigned an integer index K_j, where j is the integer corresponding to the basis vector $\boldsymbol{e}_j$ used in constructing the grid.

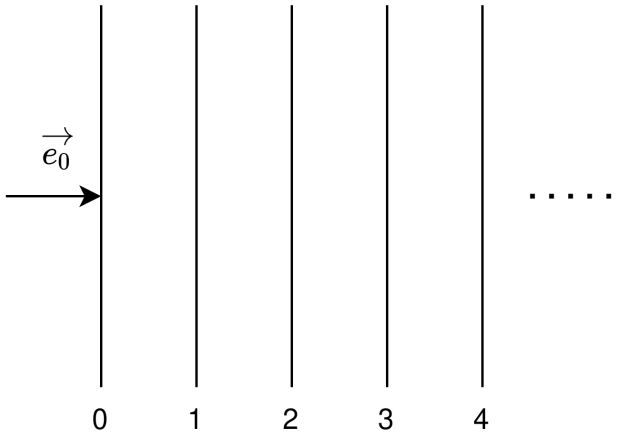


Figure 5: A grid along the basis vector $\mathbf{e_0}$, starting at an index of $K_0 = 0$.

A grid G_j can be defined formally as [9]

$$G_j = \{r \in \mathbb{R} \mid \boldsymbol{e}_j(r + \gamma_j) \in \mathbb{Z}\}, \quad j = 0, ..., (N-1) \tag{3}$$

where $\boldsymbol{e}_j$ is the basis vector for each grid as defined before in Eq. (2), and N is the number of basis vectors.
Once each of the grids has been constructed, they are superposed on top of each other. This forms a multigrid. In the case of the Penrose tiling, this forms a pentagrid [3].
The dual graph of the graph formed by the intersections between the grids gives the graph of the final tiling, where vertices are connected if they are separated by only a single grid line. A graph is simply vertices connected by edges. It does not depend on geometry it simply details which vertices are connected to each other. The vertices can be assigned arbitrary data. The dual of this graph in particular would be found by taking the multigrid, and placing vertices on each of the voids (or faces) formed by the multigrid. These vertices are then connected together whenever the void shares an

edge with a neighbouring void, to form a dual of the graph of intersections.

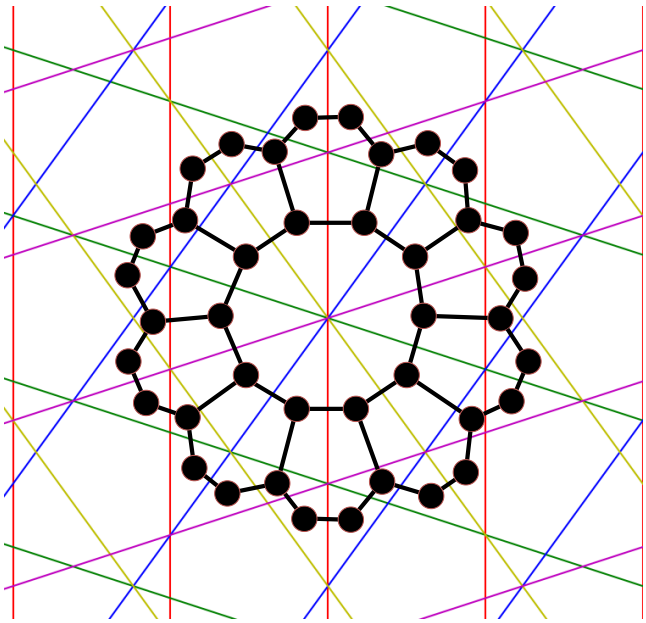


Figure 6: A Penrose pentagrid with no offsets, with the dual graph overlayed. Colour is used to differentiate between the grids; red = grid $j = 0$, and so on.

However, it can be noticed that the pentagrid shown in Fig. 6 would not produce a rhombic tiling. For example, in the centre of the diagram there is a loop of 10 vertices, which would result in a tile with 10 vertices, and further out are loops of 5 vertices which would in turn result in tiles with 5 vertices. The Penrose tiling consists of rhombi only and so this tiling would be an invalid Penrose tiling.
This is because this pentagrid arrangement is what de Bruijn refers to as a 'singular' pentagrid [3]. Meaning that some intersections within the pentagrid are formed by the meeting of more than 2 lines.
Rhombic tilings can be guaranteed if all of the intersections within the pentagrid occur between 2 lines only. This follows from the fact that these intersections will be guaranteed to give a loop of four vertices (there are four voids around each intersection).
In order to accomplish this, each grid is given some offset γ_j, where

$$0 < \gamma_j < 1. \tag{4}$$

This constraint is necessary as two grids (grid $j = 0$ and grid $j = 1$) shifted by γ_0 and $\gamma_1 = \gamma_0 + n$ respectively, where n is an integer, will be the exact same grids. However, the index of each line of grid $j = 1$ will be incremented by n. This will result in an identical tiling, but with every vertex shifted by $-n\boldsymbol{e_1}$. Any grid with an offset less than 0 is the same as a grid with an offset $1 - \gamma_j$, again just resulting in the same grid with the indices shifted [3].
Having an offset of more than 1 or less than 0 is therefore redundant. This may become more intuitive once the full method for 2D has been explained.
For generating a tiling centred on the axis of rotational symmetry, equal offsets are used. For the Penrose case (an arbitrary choice), this is

$$\boldsymbol{\gamma} = (0.2, 0.2, 0.2, 0.2, 0.2) \tag{5}$$

i.e. $\frac{1}{S}$ for every offset.

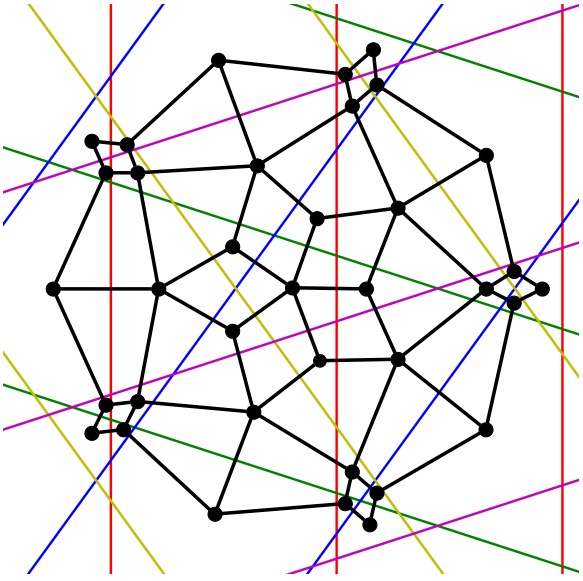


Figure 7: A Penrose pentagrid with equal offsets, with the dual graph overlayed.

As seen in Fig. 7, using offsets results in all loops in the dual graph consisting of 4 vertices, which will result in rhombic tiles in the final tiling.
This gives us the graph of the tiling. So there is one last step in order to draw the tiling, and that is getting the real position of the vertices

in the tiling so that we can use this as an embedding for our graph. This is where the indices of the individual lines become important. The index $\boldsymbol{K}$ of each vertex in the dual graph must be found, which is an N-dimensional vector of integers such that

$$\boldsymbol{K} = (K_0, K_1, K_2, \cdots, K_{(N-1)}) \tag{6}$$

where each K_j corresponds to the index of the line that the vertex resides beside within grid j.
K_0 is the index of the line within grid $j = 0$ that the intersection lies on/beside, K_1 is the index of the line within $j = 1$ that it lies on/beside, etc.

This index and the real location of the vertex are the two pieces of arbitrary data assigned to each vertex. One simple way of finding every vertex in the dual graph is by first finding the intersections, and then finding the four vertices around each intersection. This will result in vertices being found multiple times, however this is easily solved and it will also give the tile edges for free - each intersection corresponds to a tile with 4 vertices. Knowing what edges form which tiles will be useful when rendering the tiling.

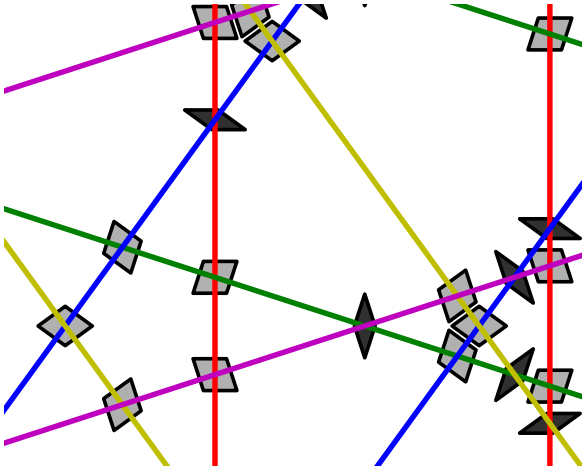


Figure 8: A Penrose pentagrid with the final tiles moved over their corresponding intersections, and scaled down. This illustrates how each rhombus is related to its parent intersection.

Fig. 8 shows a visual representation of how each rhombus of a tiling can be related to an intersection in the multigrid.

In finding the first index of the intersection, two of the grid indices are already known. These are simply the indices of the two grid lines that intersected. For this example, let's take these to be line 0 of grid 0 ($K_0 = 0$), and line 0 of grid 2 ($K_2 = 0$). This gives $\boldsymbol{K} = (0, K_1, 0, K_3, K_4)$. Finding the other three indices, K_1, K_3 and K_4, is straightforward given the position of the intersection $\boldsymbol{x}$, and takes the form [3]

$$K_j(\boldsymbol{x}) = \lceil \boldsymbol{e}_j \cdot \boldsymbol{x} - \gamma_j \rceil \tag{7}$$

where $\lceil$ and $\rceil$ is the ceiling function, which rounds a real number up to the nearest integer. This can be intuitively understood as breaking down the position into a sum of each of the basis vectors, whilst accounting for the grid offsets γ_j.
Now the index for the intersection is known, the indices of the neighbouring vertices must be found. The index of the intersection can be taken to be the index corresponding to the vertex with the lowest indices out of the group of 4. This is arbitrary; if instead the index of the intersection was chosen as the highest out of the 4, this would result in a net shift of the final tiling, but the structure would remain the same. It also follows de Bruijn's method detailed in the 1981 paper [3].

The neighbouring vertices are separated by the two intersecting grid lines. It follows that the index of each of the vertices differ by a value of 1 in each grid for the two lines that crossed. This is illustrated in Fig. 9. The final positions of each vertex of the tile can now be found as

$$\boldsymbol{r} = \sum_{j=0}^{N} K_j \boldsymbol{e}_j \tag{8}$$

where $\boldsymbol{e}_j$ is the j^{th} basis vector that was chosen at the start of the process.
This process needs to be repeated for every intersection in the tiling. The problem can be broken down into two parts: Find which grids need to be compared with each other, and then within those comparisons find every intersection between the two grids.

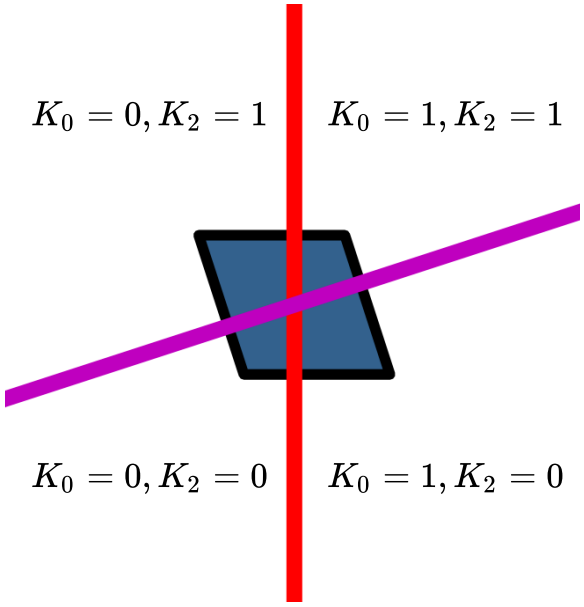


Figure 9: An intersection between grid $j = 0$ and grid $j = 2$, showing the difference in index of each surrounding vertex.

It is possible to get the vector equation of each line from the basis vector used to create the grid

$$(\boldsymbol{r} - \boldsymbol{P}_{K_j}) \cdot \boldsymbol{e}_j = 0 \tag{9}$$

where $\boldsymbol{r}$ is an arbitrary vector, and $\boldsymbol{P}_{K_j}$ is a point that lies on the line within grid j with index K_j. Note that Eq. (9) is just the vector equation of a plane in three dimensions, which also happens to work in two dimensions. This is because the basis vectors $\boldsymbol{e}_j$ are normal to the lines within a grid. This will become an integral part of expanding the algorithm to n dimensions.

Eq. (9) can be rearranged to

$$e_{jx}x + e_{jy}y = \boldsymbol{e}_j \cdot \boldsymbol{P}_{K_j} \tag{10}$$

where x and y are the components of the arbitrary vector $\boldsymbol{r}$, and e_{jx} is the x component of the basis vector, and similarly e_{jy} is the y component of the basis vector.
Note that $\boldsymbol{P}_{K_j}$ must be some real multiple of the basis vector $\boldsymbol{e}_j$, as each line within the grid lies normal to this basis vector. Each point will be spaced $\boldsymbol{e}_j$ apart, and every point will be shifted from the origin by $\gamma_j \boldsymbol{e}_j$, and so it follows that

$$\boldsymbol{P}_{K_j} = (K_j + \gamma_j)\boldsymbol{e}_j \tag{11}$$

where K_j is the index of a single line within the grid. Substituting this into Eq. (10) gives

$$e_{jx}x + e_{jy}y = K_j + \gamma_j \tag{12}$$

which is the Cartesian form of each line within grid j, where K_j is the index of each line within the grid (an integer variable that varies from $-\infty$ to ∞ for a tiling that tiles the plane infinitely).

Finding intersections between two lines from different grids can be made into a matrix equation where the first matrix is the matrix of coefficients (C_{ab}), i.e. the components of the basis/normal vectors of the two grids a and b we are comparing, packed by row into a square matrix. The second matrix is then the point of intersection between the two lines, $\boldsymbol{x}$.
The final matrix is the final part of the Cartesian form found in Eq. (12) packed into a column vector. Each element of this column vector is the position at which specific lines of index K_a and K_b lie along their corresponding normal vector. Putting this all together gives

$$\begin{pmatrix} e_{ax} & e_{ay} \\ e_{bx} & e_{by} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} K_a + \gamma_a \\ K_b + \gamma_b \end{pmatrix} \tag{13}$$

Where γ_a and γ_b are the offsets of grids a and b respectively. By multiplying both sides of the equation by the inverse of the coefficient matrix, we can obtain

$$\boldsymbol{x} = C_{ab}^{-1} \begin{pmatrix} K_a + \gamma_a \\ K_b + \gamma_b \end{pmatrix} \tag{14}$$

which gives the position of the intersection $\boldsymbol{x}$. For a pentagrid, we find the intersections between every line in set 0 with set 1, 2, 3 and 4, set 1 with set 2, 3, 4, etc. We would not compare set 1 with set 0 again since it has already been compared.

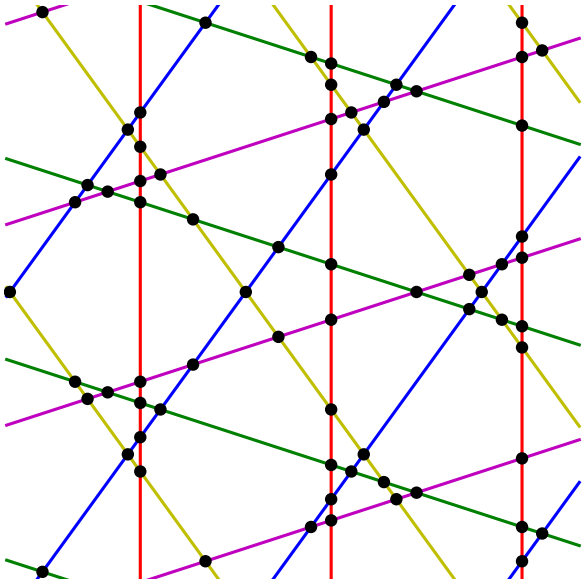


Figure 10: A plot of all intersections (shown in black) on a pentagrid with equal offsets.

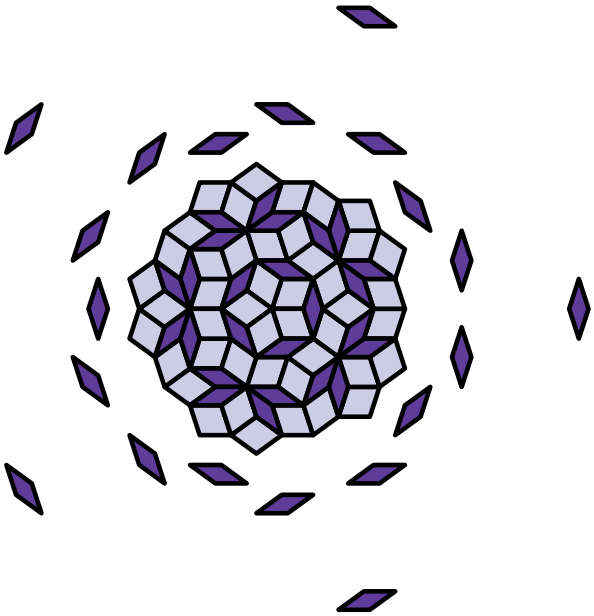


Figure 11: Raw output of the de Bruijn dual grid method, with grid indices $K_j = \{-1, 0, 1\}$ for each grid.

A plot of this algorithm applied to the pentagrid is shown in Fig. 10.

Now that all of the intersections have been found, the index of each of the four neighbouring voids are found as discussed earlier, these index values are converted into real space, and the final tiling can now be drawn as in Fig. 11.

There some more minor details to discuss in order to complete the full picture of the algorithm.

2.2.1 Generating a complete graph.

One caveat of the de Bruijn dual grid method is that the tiling produced will most likely not form a complete graph (there are a few special cases, i.e. $K_j = \{0\}$ and equal offsets for the pentagrid case). This can be easily seen in Fig. 11, as there are rhombuses that lie outside of the main body of tiles. When looking to make a graph of vertices, these vertices will not have any neighbouring vertices and so will form their own separate graph. This is not useful as most applications of this program will most likely rely on a complete graph.

This can be easily addressed by applying some form of filtering. One simple way of doing this is by choosing a shape (e.g circle/sphere, square/cube) to filter out, and removing all tiles that lie outside of this shape. As long as a suitable K_j range is picked (one that produces enough tiles to fill the shape chosen), then a complete graph can be built.
Another method of filtering can be done using the index of each vertex ($\boldsymbol{K}$), and filtering by the maximum and minimum line index allowed. This is the method used in Fig. 1 and Fig. 4.

Once all neighbouring vertices are known around each intersection, we now have a list of individual tiles. The tiles are then filtered as described in the last section. Tiles will share vertices by definition as they completely fill the two-dimensional space. In order to generate a graph (the more abstract mathematical form of the tiling), the duplicates have to be removed. This is simple to do and it is just a process of going through the list of vertices and comparing the index each vertex with the index of every other vertex, and if it matches one of the two must be removed from the list.

Now duplicate vertices have been removed, we need to know which vertices should connect to each other. Vertices should be connected when one of the index values in the vertex's index

$\boldsymbol{K}$ differs by 1 compared to the other vertex's index.
More formally, given a set of vertices V, where each element $\boldsymbol{V_i}$ is an index of a single vertex in the tiling (no particular order), the set of edges E of a given vertex with index $\boldsymbol{K}$ is

$$E_{\boldsymbol{K}} = \{i \in \mathbb{Z}^+ \quad | \quad |\boldsymbol{V_i} - \boldsymbol{K}| = 1\}. \tag{15}$$

2.2.2 Generating a tiling with no gaps.

In order to generate a complete tiling, all intersections must be found. This means that every line of a grid must be compared with every line of every other grid. However it must be noted that once grid 0 has been compared fully with grid 1, there is no need to compare grid 1 with grid 0. In the Penrose implementation this takes the form of comparing grids (0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4).
Within the comparison between two grids, we define some K_j range, for example
$K_j = \{-1, 0, 1\}$ used in generating Fig. 11, and compare every line with every other line, so (-1, -1), (-1, 0), (-1, 1), ... , (1, 0), (1, 1) in this particular case.

Once every intersection has been found within our K_j range, we have generated a correct chunk of tiling as shown in Fig. 11, without any gaps in the centre.

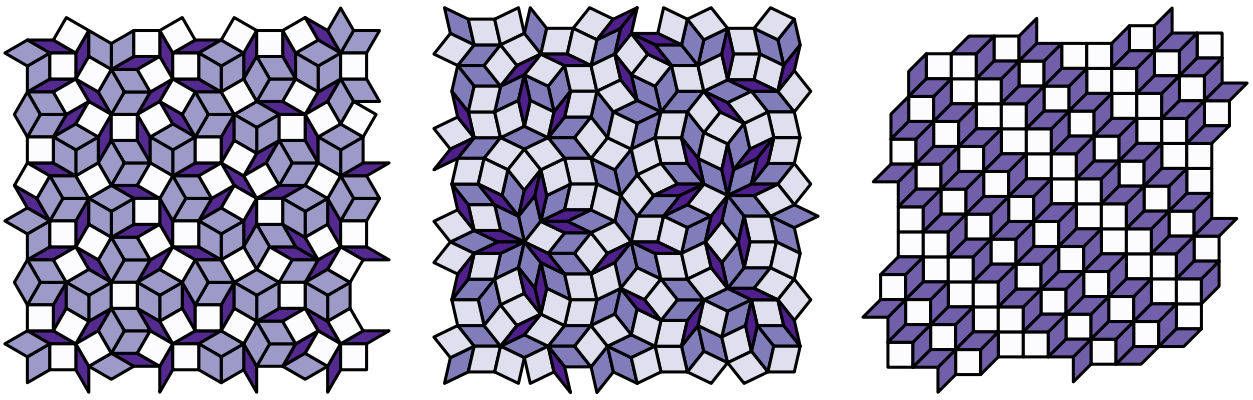
2.2.3 Special conditions for a Penrose tiling.

The de Bruijn dual grid method is very general, as shown in Fig. 12. Any basis vectors can be given, and the method will give you some tiling of the plane consisting of convex tiles with 4 vertices each. It is therefore up to the user of the algorithm to ensure that the basis vectors and grid offsets are correct for their desired tiling.
In the construction of the Penrose tiling, there are a couple of matching rules that must be obeyed to form the correct aperiodic structure (as discussed by de Bruijn)[3]. These are enforced by the de Bruijn dual grid method using

$$\sum_{j=0}^{N} \gamma_j = 0 \tag{16}$$

and Eq. (4).

It appears at first glance then that the offsets we used in Eq. (5) must be invalid, because the offsets sum to 1. However, notice that if we were to take the last offset (γ_4) and take away 1, we would get



a) 12-Fold

b) 7-Fold

c) Tiling generated with basis vectors of
$\boldsymbol{e}_0 = (0, 1)$,
$\boldsymbol{e}_1 = (1, 0)$,
$\boldsymbol{e}_2 = \frac{1}{\sqrt{2}}(1, 1)$

Figure 12: Various tilings generated with the de Bruijn dual grid method with varying basis vectors.

$$\boldsymbol{\gamma} = (0.2, 0.2, 0.2, 0.2, -0.8) \qquad (17)$$

which does sum to 0. This is also the exact same tiling as explained earlier in the method. Therefore this is a valid Penrose tiling.

2.2.4 Special conditions for tilings with even rotational symmetry.

For tilings that have even rotational symmetry, so for example the 8-Fold Ammann-Beenker tiling, if we were to use $N = S$ (same number of basis vectors as the order of rotational symmetry) in Eq. (2), it would result in duplicate grids going in opposing directions. This is a special case for even symmetry.

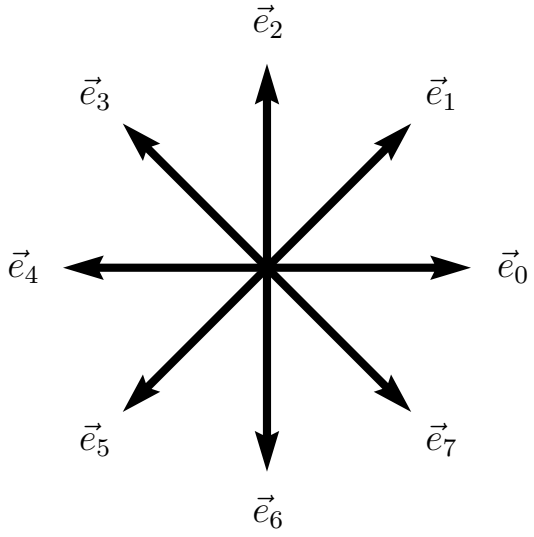


Figure 13: The basis vectors for the 8-Fold Ammann-Beenker tiling if $N = S$.

Fig. 13 shows what would happen if we were to generate basis vectors for $N = S$. The issue here is that we have basis vectors that directly oppose each other: $\boldsymbol{e}_0 = -\boldsymbol{e}_4$, $\boldsymbol{e}_1 = -\boldsymbol{e}_5$ and so on.
This means that the two grids corresponding to these two opposing basis vectors will never intersect, however they will intersect with the remaining grids. The result is duplicate tiles with slightly different positions.

A remedy for this problem is to remove any basis vectors that have another basis vector directly opposing them. The result is shown in Fig. 14.

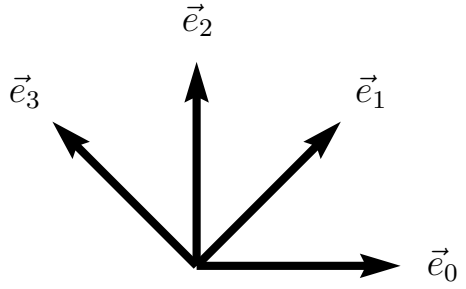


Figure 14: The basis vectors for the 8-Fold Ammann-Beenker tiling if $N = \frac{S}{2}$.

2.3 Expanding to three dimensions.

In order to expand the method, it's important to think of an abstract description of the method. The essence of the de Bruijn dual grid method can be explained as: Partitioning space into finite sections (voids), assigning each section a set of integers corresponding to the combination of partitions that formed the section, and converting this into a vertex position.

Applying this reasoning to the method in two dimensions: Partition the space into finite sections using lines, resulting in 4 voids around each line intersection, and then give each void an "index" $\boldsymbol{K}$ based on the positions of neighbouring lines. This is then converted into the final vertex positions on the plane.

It follows that the method in three dimensions must have the form: Partition the space into finite sections using **planes** (resulting in 8 voids around each point intersection between 3 planes), and then give each void an index $\boldsymbol{K}$ based on the positions of neighbouring planes. Each index is then converted into the final vertex positions in space.

It's best to start with one of the simplest cases, a cubic lattice. Cubic lattices can be found frequently in nature, a well-known one being table salt (sodium chloride) [10].
The basis vectors for a cubic lattice will be the unit vectors $\hat{\boldsymbol{i}}$, $\hat{\boldsymbol{j}}$ and $\hat{\boldsymbol{k}}$. This will form a lattice of a single unit cell where every face of the cell has internal angles of 90 degrees at its vertices. We now make our grids as in the two-dimensional

method, where instead of placing lines normal to each basis vector, we place planes that have a normal vector equal to each basis vector, and then each plane is defined by a point $\boldsymbol{P}_{\boldsymbol{K}_j}$ within each grid. See Fig. 15 for a render of the construction plane of index 0 within each set.
Note that Eq. (11) that gives us $\boldsymbol{P}_{\boldsymbol{K}_j}$ works for three dimensions as well as two dimensions, so this can be used in the three-dimensional method. Since $\boldsymbol{P}_{\boldsymbol{K}_j}$ is defined as a multiple of one of the basis vectors, this equation holds for any dimension, as it is dependent on the basis vector's dimensions.
Each offset γ_j will always be a scalar value.

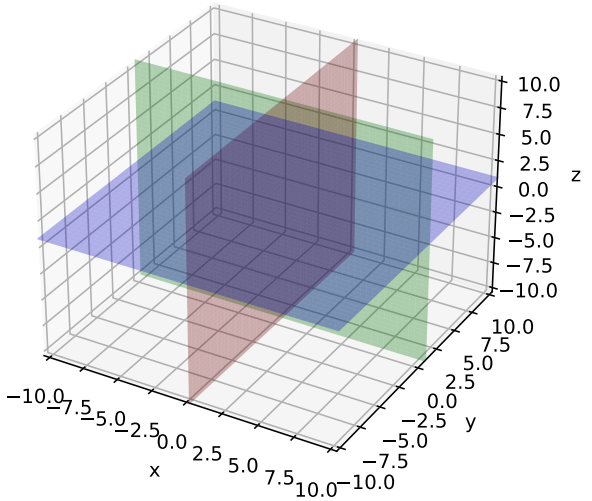


Figure 15: A plot of one plane out of each grid for use in constructing a cubic crystal. Random (arbitrary) offsets have been chosen for each grid.

A point intersection between 3 planes will form 8 voids around that intersection. Each void directly corresponds to a vertex in the final crystal. If we were to connect those voids like in Fig. 7, where we connect the voids if they are separated by a single plane, then we would end up with a rhombohedron. This is a three-dimensional cell, rather than the two-dimensional tile.
Just like in the two-dimensional algorithm, before we find the index of each void we must find the location of the intersection. The intersection between three planes can be found by simply expanding Eq. (13) to get

$$\begin{pmatrix} e_{ax} & e_{ay} & e_{az} \\ e_{bx} & e_{by} & e_{bz} \\ e_{cx} & e_{cy} & e_{cz} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} K_a + \gamma_a \\ K_b + \gamma_b \\ K_c + \gamma_c \end{pmatrix} \tag{18}$$

This gives the intersection between planes of index K_a, K_b and K_c of grids a, b and c respectively. Now doing the exact same steps as before (taking the inverse of the coefficient matrix to obtain the x, y and z values), we can get $\boldsymbol{x}$, the position of the intersection between these 3 planes.

For the cubic case, the index of each void can be found easily. This is because there are only 3 grids, and every intersection in the multigrid will be between these 3 grids.
However, if there were more than 3 basis vectors (where none is normal to any of the others) then any intersection will lie in a space between two of the planes in a separate grid. This is identical to the two-dimensional multigrid, where an intersection may lie between a number of other grids that were not involved in the intersection. It may help to refer to Fig. 10 in order to see that this is true.
This is where the equation for finding the index comes in: Eq. (7). This equation, similarly to Eq. (11), holds for three dimensions because it is a general vector equation. So it can be reused here.

Back to the cubic case; the intersections are now known as

$$\boldsymbol{x} = C_{abc}^{-1} \begin{pmatrix} K_a + \gamma_a \\ K_b + \gamma_b \\ K_c + \gamma_c \end{pmatrix} \tag{19}$$

where C_{abc} is the coefficient matrix of planes in grids a, b and c, and C_{abc}^{-1} is its matrix inverse. Now that the intersections are known, we find the index $\boldsymbol{K}$ of each neighbouring void.

A way to think about the problem is if we view the construction planes (i.e. the first two planes in each grid shown in Fig. 15) along one of the axes, as in Fig. 16.
It turns out that when viewed along one the basis vector of a grid involved in the intersection, it becomes a copy of the two-dimensional

problem. This means we can apply our method used in two dimensions twice; one for each side of the plane we are viewing along (speaking in terms of normal vector).

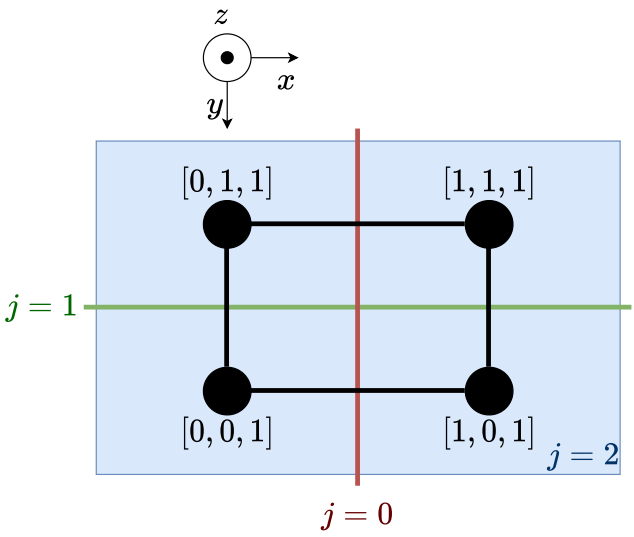


Figure 16: A view down the negative z axis of the previous figure (Fig. 15), showing vertices on one side of the plane in grid $j = 2$.

For the set of vertices shown in Fig. 16, the last index in $\boldsymbol{K}$ for each of the vertices in the figure is 1 because we are viewing down the negative z axis. So the voids on the side of plane, with index 0 in grid $j = 2$, that we are looking at will have an index of 1.
The indices of neighbouring vertices (to the intersection) then become

$$\begin{aligned} \boldsymbol{K}^0 &= [0,0,0] & \boldsymbol{K}^4 &= [0,0,1] \\ \boldsymbol{K}^1 &= [0,1,0] & \boldsymbol{K}^5 &= [0,1,1] \\ \boldsymbol{K}^2 &= [1,0,0] & \boldsymbol{K}^6 &= [1,0,1] \\ \boldsymbol{K}^3 &= [1,1,0] & \boldsymbol{K}^7 &= [1,1,1] \end{aligned} \tag{20}$$

Each index can now be converted into a real space vertex in the lattice. We can use Eq. (8) for this, as it is a general vector equation.
Our vertices then become

$$\begin{aligned} \boldsymbol{r}^0 &= 0, & \boldsymbol{r}^4 &= \hat{\boldsymbol{k}} \\ \boldsymbol{r}^1 &= \hat{\boldsymbol{j}}, & \boldsymbol{r}^5 &= \hat{\boldsymbol{j}} + \hat{\boldsymbol{k}} \\ \boldsymbol{r}^2 &= \hat{\boldsymbol{i}}, & \boldsymbol{r}^6 &= \hat{\boldsymbol{i}} + \hat{\boldsymbol{k}} \\ \boldsymbol{r}^3 &= \hat{\boldsymbol{i}} + \hat{\boldsymbol{j}}, & \boldsymbol{r}^7 &= \hat{\boldsymbol{i}} + \hat{\boldsymbol{j}} + \hat{\boldsymbol{k}} \end{aligned} \tag{21}$$

This gives us a cube given that we connect vertices that differ by an index of ± 1 in only one of the index elements. Repeating this for $K_j = \{-1, 0, 1\}$ gives us a section of cubic lattice as shown in Fig. 17. If we were to use an index range from $-\infty$ to ∞, i.e. $K_j = \mathbb{Z}$, then this would fill three dimensional space with cubes.

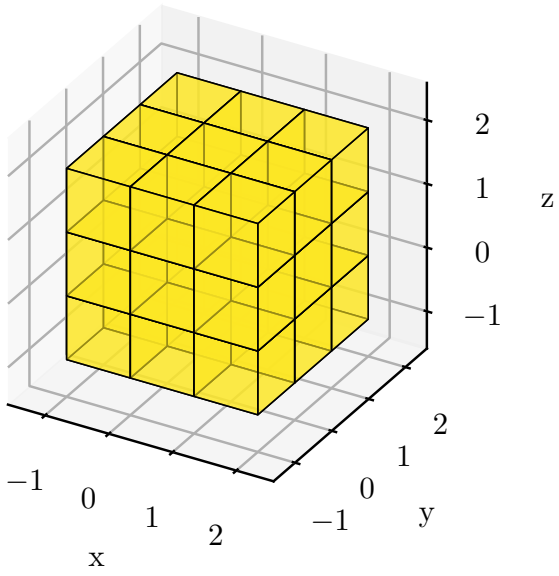


Figure 17: An orthographic projection of the cubic lattice in three dimensions with an index range of $K_j = \{-1, 0, 1\}$, with each cell drawn as a translucent cube.

2.3.1 Three-dimensional quasicrystal example.

An example of a three-dimensional quasicrystal is a icosahedral quasicrystal [11]. These are crystals with an aperiodic structure, just as with the Penrose tiling, and lack discrete translational symmetry.
Just as the Penrose tiling consisted of two cells (a thin and a thick rhombus), the icosahedral similarly consists of two forms of rhombohedra: Oblate and prolate rhombohedron [12].

It is formed by taking six basis vectors, five of which are arranged in a 5-Fold rotationally symmetric arrangement around the z axis, and the last is the unit vector in the z direction, $\hat{\boldsymbol{k}}$ [13]

$$\begin{aligned} \boldsymbol{e}_{j=0\ldots4} &= \frac{1}{\sqrt{5}}\left(2\cos\left(j\frac{2\pi}{5}\right), 2\sin\left(j\frac{2\pi}{5}\right), 1\right) \\ \boldsymbol{e}_5 &= (0,0,1) \end{aligned} \tag{22}$$

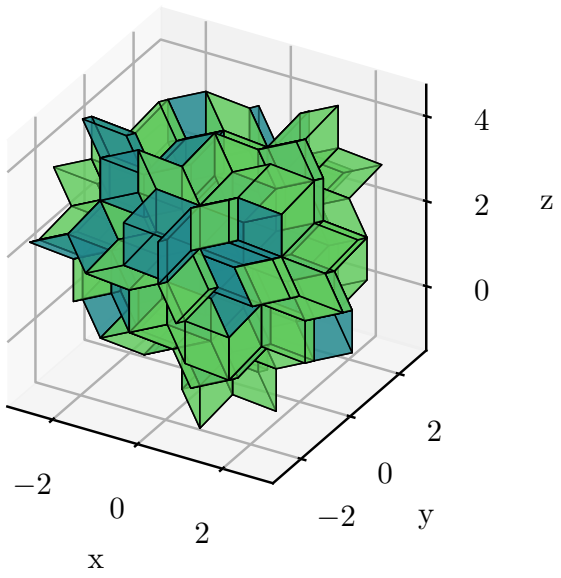


Figure 18: An orthographic projection of an extract of icosahedral quasicrystalline lattice in three dimensions. The colour of each cell is determined by it's volume (scalar triple product).

The basis vectors lie on the vertices of a regular icosahedron, if we take the origin as being in the centre of an icosahedron. However, just as in the even cases in Sec. 2.2.4, opposing basis vectors ($\boldsymbol{a} = -\boldsymbol{b}$) are ignored and so we do not use the full twelve vertices, only the six described by Eq. (22).
An orthographic projection of the icosahedral quasicrystal can be seen in Fig. 18.

2.4 Generalisation to arbitrary dimensions.

The algorithm can be split into segments. Finding the intersections, and then finding the index of each void neighbouring that intersection. From there it is trivial to convert each index to a final vertex position (Eq. (8)).
So what needs to be done is to find a general form of finding the intersections between construction grids (lines in two dimensions, planes in three dimensions), and also a general form of finding the index of each neighbouring void.

2.4.1 Finding intersections in arbitrary dimensions.

It may be useful to start from two dimensions and work our way up. So in two dimensions, we had exactly two lines meeting at a point. This is made certain by giving each grid an offset.

Then we move to three dimensions, where each point intersection must happen between three planes (there is an extra degree of freedom). Two planes must first meet at a line, and then a third plane is introduced and meets that line at a point.

Four dimensions is hard to visualise because we live in a three-dimensional (plus time) world, our brains are not evolved to think in four dimensions. However it is possible to imagine that perhaps rather than grids made of planes such as in three dimensions, in four dimensions we have grids made of three-dimensional planes that are evenly spaced, and these then intersect to form two dimensional planes, which then intersect to form lines, and finally four of these volumes meet at a point in four dimensions.

Whilst it is hard to conceptualise how the construction might work in four dimensions and above, the mathematics comes easily from what we saw in going from two dimensions to three. The coefficient matrix becomes a 4x4 matrix of the coefficients of each vector normal to each plane, and the column vector containing the position of each plane along its basis vector also expands. Generalised for n dimensions this becomes

$$\boldsymbol{x} = C^{-1}_{ab\ldots\alpha}\begin{pmatrix} K_0+\gamma_0 \\ K_1+\gamma_1 \\ \vdots \\ K_{n-1}+\gamma_{n-1} \end{pmatrix} \tag{23}$$

where n is the number of dimensions, $n \in \mathbb{Z}^+$, $n > 1$, and the coefficient matrix $C_{ab...\alpha}$ is given by

$$C_{ab...\alpha} = \begin{pmatrix} e_{0x} & e_{0y} & \cdots & e_{0\alpha} \\ e_{1x} & e_{1y} & \cdots & e_{1\alpha} \\ \vdots & \vdots & \ddots & \vdots \\ e_{(n-1)x} & e_{(n-1)y} & \cdots & e_{(n-1)\alpha} \end{pmatrix} \tag{24}$$

where α is the final coordinate in that dimension. So for example for two dimensions: $\alpha = y$, three dimensions: $\alpha = z$ and so on.

We can verify this for four dimensions by finding an intersection that we know using Eq. (23). A basic arrangement of planes could be as follows:

- A plane in the x direction with an offset of $\gamma_0 = 1$.
- A plane in the y direction with an offset of $\gamma_1 = 0$.
- A plane in the z direction with an offset of $\gamma_2 = 0$.
- A plane in the w direction (fourth dimension) with an offset of $\gamma_3 = 0$.

So the basis vectors are defined as

$$\begin{aligned} \boldsymbol{e}_0 &= (1,0,0,0) \\ \boldsymbol{e}_1 &= (0,1,0,0) \\ \boldsymbol{e}_2 &= (0,0,1,0) \\ \boldsymbol{e}_3 &= (0,0,0,1) \end{aligned} \tag{25}$$

It should be evident that the y, z and w planes must all meet at 0 for each of their coordinates in the y, z and w axis, as each of the planes in these directions meet at 0. The remaining x coordinate then must equal 1. So the intersection must occur at $\boldsymbol{x} = (1,0,0,0)$.
The index of each plane is 0. Substituting these basis vectors into Eq. (23) rearranged, we get:

$$\begin{pmatrix} e_{0x} & e_{0y} & e_{0z} & e_{0w} \\ e_{1x} & e_{1y} & e_{1z} & e_{1w} \\ e_{2x} & e_{2y} & e_{2z} & e_{2w} \\ e_{3x} & e_{3y} & e_{3z} & e_{3w} \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{pmatrix} \tag{26}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{27}$$

which is true as the coefficient matrix is the identity matrix for this simple example.

2.4.2 Finding the index of voids that neighbour intersections in arbitrary dimensions.

In de Bruijn's 1981 paper "Algebraic theory of Penrose's non-periodic tilings of the plane. I", de Bruijn details the mathematical form of finding the index of each neighbouring void around a two-dimensional intersection $\boldsymbol{x}_0$, between grids r and s, as [3]

$$\begin{aligned} (K_0(\boldsymbol{x}_0), ..., K_4(\boldsymbol{x}_0)) &+ \epsilon_1(\delta_{0r}, ..., \delta_{4r}) \\ &+ \epsilon_2(\delta_{0s}, ..., \delta_{4s}) \end{aligned} \tag{28}$$

by taking $(\epsilon_1, \epsilon_2) = (0,0), (0,1), (1,0), (1,1)$ respectively, and δ_{ij} is the Kronecker delta. Applying these pairs of ϵ in turn, we get the index of each vertex in a tile as shown in Fig. 9.

In order to generalise this, it is useful to look at the three-dimensional algorithm to see if there is a pattern that can be exploited.
In three dimensions, we have the two-dimensional scenario applied twice, as described earlier in the method and shown in Fig. 16. We are dealing with three planes intersecting at a point, so there are planes of three grids intersecting, and these grids involved with the intersection will be labelled q, r and s respectively. From comparing Eq. (20) to the neighbours obtained by applying de Bruijn's Eq. (28), the neighbours obtained could be written in a similar form as de Bruijn's equation

$$(K_0(\boldsymbol{x}_0), ..., K_{(N-1)}(\boldsymbol{x}_0)) + \epsilon_1(\delta_{0q}, ..., \delta_{(N-1)q}) + \epsilon_2(\delta_{0r}, ..., \delta_{(N-1)r}) + \epsilon_3(\delta_{0s}, ..., \delta_{(N-1)s}) \quad (29)$$

by taking

$$(\epsilon_1, \epsilon_2, \epsilon_3) = (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1) \quad (30)$$

respectively, and where $\boldsymbol{x}_0$ is the three-dimensional point intersection that these voids neighbour, and N is the number of basis vectors as usual.

It is apparent that the values of ϵ are directly corresponding to counting up from 0 in the binary number system. The binary number system is a numbering system of base 2, meaning that the first numeral in each binary number (starting from the right hand side of the number) corresponds to multiples of 2^0, the second numeral is multiples of 2^1, and so on. So counting up from 0 in binary (3 significant figs) would look like: 000, 001, 010, 011, 100, 101, 110, 111, and so on to higher significant figures.

In two dimensions, this would give us a maximum value of 11 in binary, which is 3 in decimal. Remembering to count 0 as a number this gives us four different vertices, giving a tile. In three dimensions, we get a maximum value of 111, which is 7 in decimal, so we have 8 vertices in total. This is the same as the the number of vertices of a rhombohedron which form the cells of the three dimensional lattices generated by the method.
The values of ϵ in four dimensions would then go up to 1111, which is 15 in decimal and so it would give 16 vertices. A 4-cube has 16 vertices - think of two cubes with corresponding vertices connected in the fourth dimension; this gives 16 vertices. This is analogous to how a three-dimensional cube is two squares with corresponding vertices connected in the third dimension. Hence the cell generated also has the number of vertices of a four-dimensional rhombohedron.

There are still a few minor details that need to be figured out for an implementation of the full method for arbitrary dimensions.

2.4.3 Generating a tiling with no gaps, in arbitrary dimensions.

In two dimensions, there are always two lines meeting at an intersection, therefore every pair of grids must be compared as discussed in Sec. 2.2.2. In three dimensions, we now have three planes meeting at a point, and so triplets of grids must be compared with each other. For example, in the cubic case we would compare grids (0, 1, 2) only (any other comparison would be redundant; there are only three grids).

It follows then that in four dimensions we would compare groups of four grids together, and more generally that in n dimensions we would compare groups of n grids together.

2.4.4 Generating a four-dimensional hypercubic lattice.

It is important to verify the algorithm works by generating a known crystal in a higher dimension. For three dimensions, a cubic lattice is generated using the basis vectors $\boldsymbol{U}_0$, $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ where each of these $\boldsymbol{U}_j$ vectors is a spatial vector in $n = 3$ dimensions, and where each basis vector is normalised, and all are orthogonal to one other. The result is stacked cubes in each direction.
It follows that in four dimensions, a 4-cubic lattice would be generated using the orthonormal basis vectors $\boldsymbol{U}_0$, $\boldsymbol{U}_1$, $\boldsymbol{U}_2$ and $\boldsymbol{U}_3$. The result should be stacked four-dimensional hypercubes (4-cubes) in every orthogonal direction.

To check that the index of neighbouring voids are being generated correctly, the special case of $K_j = \{0\}$ can be used, i.e. using a single intersection between grids to generate a single cell. In three dimensions this cell will be a cube whereas in four dimensions this will be a

4-cube. Checking that the shape generated is a 4-cube can be done by outputting the graph in Python, and checking that each vertex is connected by four edges. This is shown in Fig. 19.

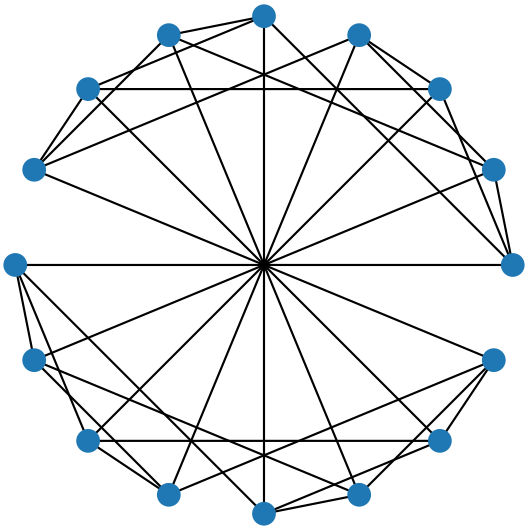


Figure 19: Graph produced by the Python program using the networkx package. This corresponds to a 4-cube.

Fig. 19 confirms that the shape being generated is in fact a 4-cube, as every vertex has four edges connected to it. Increasing the index range of each construction grid to $K_j = \{-1, 0, 1\}$ and rendering the connections in three dimensions (truncating the first three elements of each position vector and connecting edges) gives us Fig. 20.
As you can see in Fig. 20, we have the usual cubic lattice as expected, with a few extra connections between vertices. Note that a lot of the edges are drawn over each other multiple times depending on which vertices are connected in the extra fourth dimension.
A few of these extra connections in the fourth dimension can be seen clearly in those edges that do not lie on top of existing ones, i.e. edges that do not lie at 0 or 90 degrees in the yz plane of Fig. 20.

Fig. 21 shows the graph output of the same method applied in five dimensions for $K_j = \{0\}$. The graph directly corresponds to a 5-cube as each vertex is connected to another five vertices.

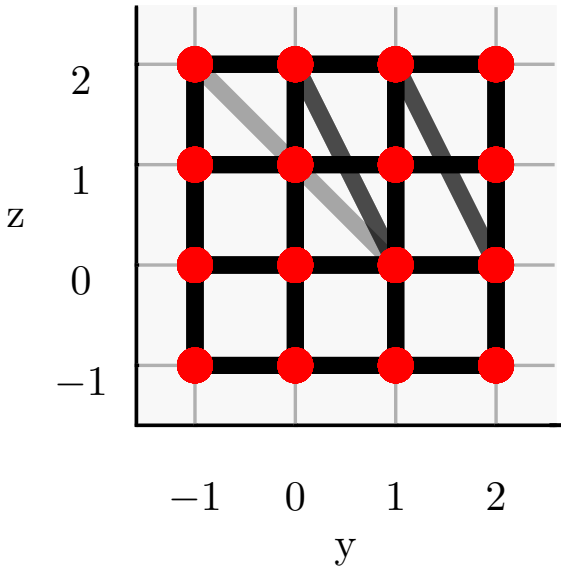


Figure 20: An orthographic render of a 4-cubic lattice viewed down the negative x axis. Each edge of the graph is rendered with an opacity value of 0.33. This means that when more than three edges overlap the line appears completely opaque.

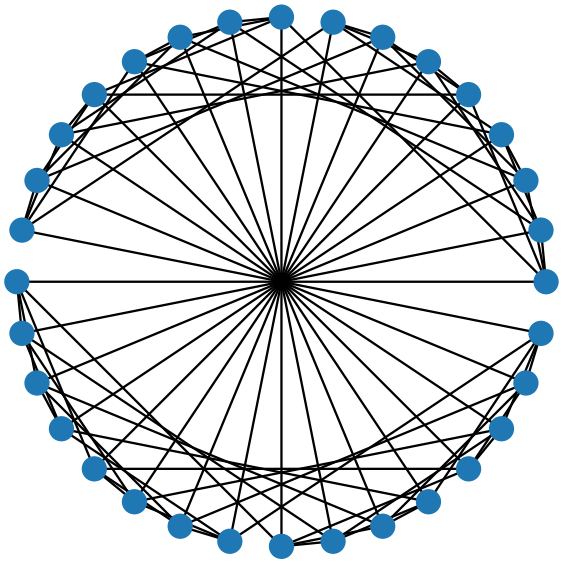


Figure 21: Graph produced by the Python program using the networkx package, given orthonormal basis vectors $\boldsymbol{U}_0$, $\boldsymbol{U}_1$, $\boldsymbol{U}_2$, $\boldsymbol{U}_3$ and $\boldsymbol{U}_4$. Using a K range of $\boldsymbol{K}_j = \{0\}$ gives a 5-cube.

3 Discussion.

3.1 Use cases in condensed matter research.

Whilst this method is very abstract in nature, there are quite a few real-world examples where this method can be used.
The program (the implementation of the method in Python) could be used to generate crystals that can be used in Python programs created by other researchers, an example potentially being tight binding models.

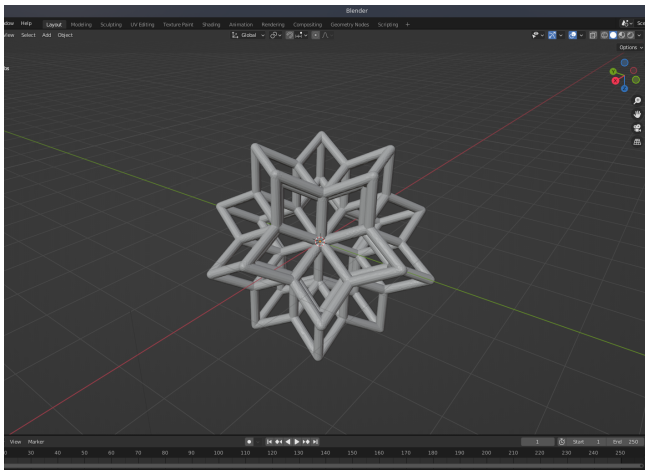


Figure 22: A screenshot of a section of icosahedral quasicrystal generated as an STL file, as viewed in the 3D modelling software Blender.

For example, it may be a good tool to test condensed matter theories that work on periodic structures, but have not yet been proven to work in aperiodic systems. They may only work due to the periodicity. Aperiodic crystals that this program generates would therefore be useful to use to numerically test theories on aperiodic crystals that have been proven to exist (and therefore the theories must be consistent with aperiodic structures not just periodic ones).

Another particular area this program may be useful in is in designing metamaterials. Metamaterials are manufactured materials formed of (usually) microscopic structures that are periodic most of the time. This program may allow the creation of aperiodic metamaterials. A team lead by Prof. Ronan McGrath at the University of Liverpool was particularly interested in the code and are now using the code to begin a large experimental programme looking at the mechanical properties of these structures. McGrath requested a feature that allowed you to save the structure as an STL file. An example of an STL file generated by the program can be seen in Fig. 22.

The mechanical properties of quasicrystals could potentially be quite interesting, because they have low slippage when compared to their periodic counterparts.
This is because of the aperiodic nature of the crystal. Periodic crystals have long range order and so have high slippage along certain directions. Therefore aperiodic crystals may have interesting mechanical properties that may be of use in industry.

Dr. Sam Ladak of Cardiff University has also expressed interest in using the code for 3D printing the structures for use in micromagnetic arrays and to create weakly-coupled Josephson junctions by coating the structures in superconducting material.

3.2 Future research.

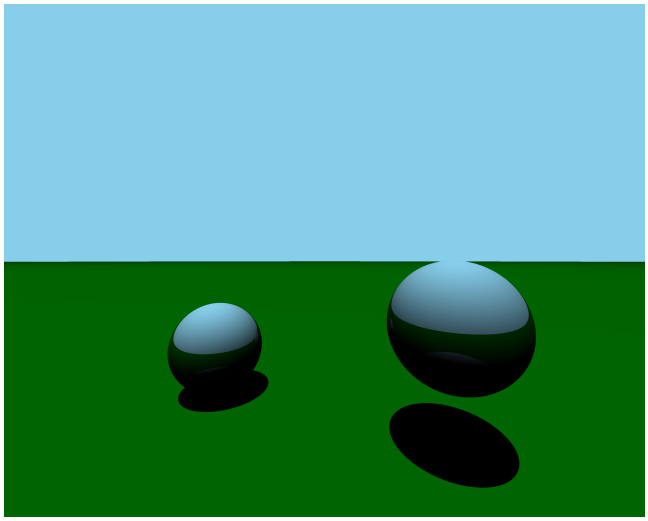


Figure 23: A raytraced scene from my 3D implementation, a scene of two reflective spheres and a light source.

Research that could be interesting in the future is some way of visualising the four dimensional (and higher) crystals, in a way in

which you do not lose too much information. A lot of information is lost when you truncate each n-dimensional positional vector to fit into a lower-dimensional projection, which is the method adopted to generate images such as Fig. 20.

One method that comes to mind is a four-dimensional implementation of raytracing – a computer graphics algorithm for rendering shapes which simulates light rays in three dimensions, which could potentially be abstracted to four dimensions. A three-dimensional implementation can be seen in Fig. 23.

Acknowledgements

I would like to give a huge thank you to Dr Felix Flicker for being an excellent supervisor for the past two semesters and for supporting my ideas.

References

[1] Raymond Tennant and Abu Dhabi. "Medieval Islamic architecture, quasicrystals, and Penrose and Girih tiles: questions from the classroom". In: *Symmetry: culture and science* 19.2-3 (2008), pp. 113–125.

[2] Johannes Kepler. *Harmonices mundi libri V*. 1619.

[3] N.G. de Bruijn. "Algebraic theory of Penrose's non-periodic tilings of the plane. I". In: *Indagationes Mathematicae (Proceedings)* 84.1 (1981), pp. 39–52. ISSN: 1385-7258.

[4] Enrique Macia. "Improving the Efficiency of Thermophotovoltaic Devices: Golden Ratio Based Designs". In: Dec. 2015, pp. 83–94. ISBN: 978-84-608-5438-8.

[5] Denis Gratias and Marianne Quiquandon. "Discovery of quasicrystals: The early days". In: *Comptes Rendus Physique* 20.7 (2019). La science en mouvement 2 : de 1940 aux premières années 1980 – Avancées en physique, pp. 803–816. ISSN: 1631-0705.

[6] Felix Flicker, Steven H. Simon, and S. A. Parameswaran. "Classical Dimers on Penrose Tilings". In: *Phys. Rev. X* 10 (1 Jan. 2020), p. 011005.

[7] Martin Gardner. *Penrose Tiles to Trapdoor Ciphers: And the Return of Dr Matrix.* Revised Edition. Cambridge University Press, July 1997.

[8] Veit Elser and Neil JA Sloane. "A highly symmetric four-dimensional quasicrystal". In: *Journal of Physics A: Mathematical and General* 20.18 (1987), p. 6161.

[9] Helen Au-Yang and Jacques H.H. Perk. "Quasicrystals—The impact of N.G. de Bruijn". In: *Indagationes Mathematicae* 24.4 (2013). In memory of N.G. (Dick) de Bruijn (1918–2012), pp. 996–1017. ISSN: 0019-3577.

[10] Charles F Davidson and Michael R Slabaugh. "Salt crystals-science behind the magic". In: *Journal of chemical education* 80.2 (2003), p. 155.

[11] Dov Levine and Paul Joseph Steinhardt. "Quasicrystals: A New Class of Ordered Structures". In: *Phys. Rev. Lett.* 53 (26 Dec. 1984), pp. 2477–2480.

[12] Dov Levine and Paul J Steinhardt. "Quasicrystals. I. Definition and structure". eng. In: *Physical review. B, Condensed matter* 34.2 (1986), pp. 596–616. ISSN: 0163-1829.

[13] Joshua ES Socolar and Paul J Steinhardt. "Quasicrystals. II. Unit-cell configurations". In: *Physical Review B* 34.2 (1986), p. 617.