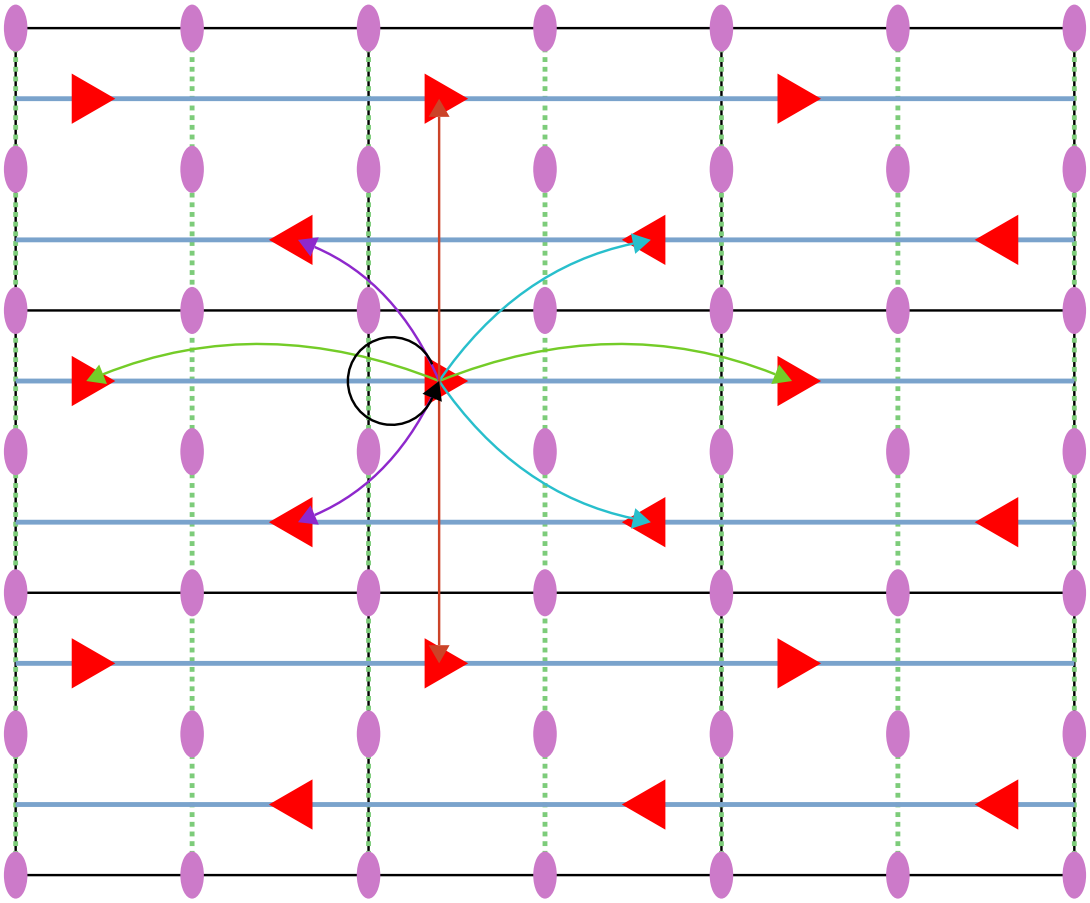
Symmetry Implementation for Pair Distribution Function

Max Krummenacher



Bachelor Thesis
02.07.2024
B.Sc. Material Science and Engineering, ETH Zürich
supervised by Arkadiy Simonov

Abstract

Pairs of symmetry related sites in crystals are important for many applications, for example the 3D-ΔPDF method, which is used to determine the structure of disordered crystals. In this work, a program to determine the multiplicity of such pairs was developed. This work explains the mathematics behind crystals, special sites in crystals and pairs of sites in crystals. Examples in 1, 2 and 3 dimensions are shown to explain the concepts and algorithms used to determine pair multiplicity and the structure of the program is described.

1. Introduction

1.1. Disordered Crystals

At finite temperatures, materials inherently posses some type of disorder. But, while gases and liquids posses no long-range order, in crystals the equilibrium positions of the atoms are arranged in periodic pattern. All crystals have disorder in the thermal motion of atoms around their equilibrium positions. [1] In Addition, in some crystals the formation of vacancies, dislocations and other faults or magnetic orientations might be a source of disorder. Disorder is especially pronounced in crystals which have degenerate energy states, i.e. crystals which have many states at similar energies.

A good example for disordered structures is the material class of Prussian blue analogues (materials with composition $M\left[M'(\mathrm{CN})_6\right]_x$ where M and M' are metal ions). Prussian blue analogues generally have an fcc structure. However, not all hexacyano metallate sites are occupied, as charge neutrality dictates the ratio x between metal ions and hexacyano metallates. In some Prussian blue analogues such as $\mathrm{Mn^{II}}\left[\mathrm{Co^{III}}(\mathrm{CN})_6\right]_{⅔}$ this fraction of vacancies is impossible to arrange in a way that follows the symmetries of the system. The symmetry of the crystal is thus broken. [2]

In conventional crystallography this is simplified by defining the average structure. Instead of providing precise information about the positions in the whole crystal, the average structure provides probabilistic information about the occupation of a position. In the example above, the probability of occupation of a hexacyano metallate is given by $x = ⅔$.

But by considering only average structure, a lot of information about a material is lost. In fact, these vacancies are not uniformly distributed throughout the crystal. For example, we might expect the vacancies to be more stable if they are arranged along certain direction or at certain distances. These correlations are essential as they influence the properties of the material. Such as porosity, diffusion coefficients, absorption coefficients and more, which find application ranging from medicine to the development of new batteries. [3]

1.2. Diffuse Scattering

X-ray diffraction is a common technique to determine the crystal structure as well as the lattice parameters of crystals. The technique uses the diffraction pattern generated by shining an X-ray beam on a single crystal. The pattern captured using this method reflects the symmetry of the crystal. Mathematically it can be expressed as the Fourier transform of the electron density of the crystal. The complex structure factor $F\left(\vec{h}\right)$ is thus:

$$F\left(\vec{h}\right) = \int_{\mathbb{R}^3} \rho(\vec{x}) \exp\left(2\pi i \vec{h} \cdot \vec{x}\right) d\vec{x} \tag{1}$$

However, only the magnitude $I\left(\vec{h}\right) = \left|F\left(\vec{h}\right)\right|^2 = F\left(\vec{h}\right)F^*\left(\vec{h}\right)$ of the structure factor can be measured in a diffraction experiment. For perfectly ordered crystals, the intensity measured is zero everywhere except at certain points, called Braggs peaks. From these diffraction peaks, the Laue group of the crys-

tal can be determined. Using iterative approaches such as charge flipping, the phase of the signal can be reconstructed and the average structure can be determined. [4]

1.2.1. 3D Pair Distribution Function

The Three-Dimensional Pair Distribution function (3D-PDF) is a function used to find correlations in structures. In the application for crystallography, it specifically refers to the autocorrelation of the electron density ρ.

$$\mathrm{PDF}(\vec{x}) = \int_{\mathbb{R}^3} \rho\left(\vec{\xi}\right)\rho\left(\vec{\xi} - \vec{x}\right)d\vec{\xi} \tag{2}$$

The PDF essentially shifts the electron density with respect to itself and then compares the shifted function to itself. A high value at point $\vec{x}$ means that the structure is highly correlated with a shift by $\vec{x}$. The PDF of the average structure is known as the Patterson function.

In diffraction experiments, the PDF is convenient as it is easily obtained as the inverse Fourier transform of the measured intensities. As such, no calculation of the phase factors is necessary.

$$\begin{aligned}\mathcal{F}^{-1}\left(I\left(\vec{h}\right)\right) &= \mathcal{F}^{-1}[\mathcal{F}(\rho(\vec{x}))\mathcal{F}^{*}(\rho(\vec{x}))] \\ &= \mathcal{F}^{-1}[\mathcal{F}(\rho(\vec{x}))\mathcal{F}(\rho(-\vec{x}))] \\ &= \rho(\vec{x}) * \rho(-\vec{x})\end{aligned} \tag{3}$$

The diffraction pattern from an experiment using a disordered crystal not only contains the Braggs peaks but a diffuse pattern in between the peaks. This diffuse part of the diffraction pattern contains information about the correlations described in the section on disordered crystals. If the average structure is known, the difference between the Patterson function and the real PDF calculated from the measured pattern can be formed. This function is called the Difference Pair Distribution Function ΔPDF.

The PDF and ΔPDF can be calculated from measurements from powder diffraction. This, however, leads to a loss of information because powder diffraction inherently averages the measurement radially.

The 3D-PDF needs to be measured from more complicated single crystal experiments using scanning techniques, but allows more information to be gained from the data analysis.

The 3D-ΔPDF can be used by programs like Yell to find the local correlations of structure. [5] This is where the main work of the program comes in. The program Yell needs to be given the multiplicity of pairs in a crystal to properly expand the PDF.

2. The Project

The program developed as part of this thesis covers the calculation of pair multiplicities. The next sections cover the mathematics behind space groups and how the program was implemented. To this end, affine and Euclidean spaces and transformations of them will be explained. Furthermore, this report contains the explanation of Wyckoff positions and pairs of sites. Additionally, examples of pairs in line groups, wallpaper groups and space groups are given and explained.

3. Mathematical Description of Crystals

In the following sections, the described spaces are assumed to be three-dimensional, but all concepts apply analogously to one and two dimensions. Higher dimensional spaces are possible, but exceed the scope of this work.

A Euclidean space is an affine space with a scalar product. In simple terms, an affine space is a vector space whose origin is forgotten, allowing translations to be part of a linear transformation. [6] The scalar product then allows for the calculation of distances and angles, producing a Euclidean space from an affine space.

An affine space consists of an associated point space A and an associated vectors space $\vec{A}$. These can be thought of as a collection of points and the group of translations acting on them. For vectors the notation $\boldsymbol{v} \in \vec{A}$, for a translation from point $X \in A$ to $Y \in A$ the notation $\overrightarrow{XY}$ used, this can also be thought of as map $\vec{\cdot} : A \times A \to \vec{A}$.

In the point space there is one special position O which is the point relative to which all coordinates are given. The coordinates of a point are then given in terms of a translation $\vec{v} \in \vec{A}$ away from the origin, for which we know how to work with coordinates from linear algebra. This vector is called the position vector. Thus, we can assign a coordinate triple to a point P based on the position vector $\overrightarrow{OP}$. Note that to specify a basis for an affine space, a basis for the associated vector space and an origin need to be specified.

In the context of crystallography, points are usually referred to as positions. This is reflected in the following section and the naming of types in the program.

A general linear transformation of a point X can then be described by a linear transformation of the position vector plus a translation.

Following the convention used by the International Table for Crystallography, the transformation will be denoted in the following way:

$$\mathcal{Q} = (\boldsymbol{Q}, \boldsymbol{q}) \tag{4}$$

$$\overrightarrow{O\widetilde{X}} = \overrightarrow{O\mathcal{Q}X} = \boldsymbol{Q}\overrightarrow{OX} + \boldsymbol{q} \tag{5}$$

Note that the inversion of $\mathcal{Q}$ is given by $\mathcal{Q}^{-1} = (\boldsymbol{Q}^{-1}, -\boldsymbol{Q}^{-1}\boldsymbol{q})$, the composition of two translations is given by $\mathcal{Q}\mathcal{P} = (\boldsymbol{QP}, \boldsymbol{Qp} + \boldsymbol{q})$ and the identity is given by $\mathcal{J} = (\boldsymbol{I}, \boldsymbol{o})$. Proofs for these equations, as well as associativity, are given in the appendix.

The scalar product is defined on the vector space with the usual axioms for the scalar product.

The coordinates of the affine space are called Cartesian, if the coefficients of the associated vector space are given in an orthonormal basis. In this case, the scalar product is the dot product.

If the coordinates are not Cartesian, the scalar product is given by:

$$\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \boldsymbol{v}^T \boldsymbol{G} \boldsymbol{w} \tag{6}$$

Where G is a symmetric 3 by 3 matrix known as the metric tensor. The norm is naturally induced by the scalar product $\|\boldsymbol{v}\| = \sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle}$ and distances between points P, Q can be measured by taking the norm of the translation vector between them $d(P, Q) = \left\|\overrightarrow{PQ}\right\|$.

3.1. Crystals Structures

In crystallography, a crystal structure is an ordered structure which fulfills three linearly independent translation symmetries $\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3 \in \mathbb{R}^3$.

The basis for the vector space can then be given in terms of these vectors. However, in crystallography the basis is sometimes given in terms of combinations of these vectors for an easier description of symmetries in the crystal structure. Since we are not concerned about the orientation of the space we can give the basis in terms of the length of the basis vectors a, b, c and the angles between them α, β, γ known as fractional coordinates.

The metric tensor G can be calculated from these values in the following way:

$$G = \begin{pmatrix} \boldsymbol{a}\cdot\boldsymbol{a} & \boldsymbol{a}\cdot\boldsymbol{b} & \boldsymbol{a}\cdot\boldsymbol{c} \\ \boldsymbol{b}\cdot\boldsymbol{a} & \boldsymbol{b}\cdot\boldsymbol{b} & \boldsymbol{b}\cdot\boldsymbol{c} \\ \boldsymbol{c}\cdot\boldsymbol{a} & \boldsymbol{c}\cdot\boldsymbol{b} & \boldsymbol{c}\cdot\boldsymbol{c} \end{pmatrix} = \begin{pmatrix} a^2 & ab\cos(\gamma) & ac\cos(\beta) \\ ba\cos(\gamma) & b^2 & bc\cos(\alpha) \\ ca\cos(\beta) & cb\cos(\alpha) & c^2 \end{pmatrix} \tag{7}$$

3.1.1. Isometries

Isometries are distance and angle preserving transformations on a collection of points and can be represented by an affine transformation. For an affine transformation to be an isometry, the following statement must hold for all $X, Y \in A$ [6]:

$$\begin{aligned} d(X,Y) &= d(\mathcal{Q}X, \mathcal{Q}Y) \\ \left\|\overrightarrow{XY}\right\| &= \left\|\overrightarrow{\mathcal{Q}X\mathcal{Q}Y}\right\| \\ &= \left\|\boldsymbol{Q}\overrightarrow{OY} + \boldsymbol{q} - \boldsymbol{Q}\overrightarrow{OX} - \boldsymbol{q}\right\| \\ &= \left\|\boldsymbol{Q}\left(\overrightarrow{OY} - \overrightarrow{OX}\right)\right\| \\ &= \left\|\boldsymbol{Q}\overrightarrow{XY}\right\| \end{aligned} \tag{8}$$

As expected, the translation vector $\boldsymbol{q}$ is free, as a translation doesn't affect the distances between points.

Since X and Y are arbitrary points, so is the vector $\vec{v}$ between them. Using the definition of the metric:

$$\begin{aligned} \|\boldsymbol{v}\| &= \|\boldsymbol{Q}\boldsymbol{v}\| \\ \|\boldsymbol{v}\|^2 &= \|\boldsymbol{Q}\boldsymbol{v}\|^2 \\ \boldsymbol{v}^T\boldsymbol{G}\boldsymbol{v} &= (\boldsymbol{Q}\boldsymbol{v})^T\boldsymbol{G}(\boldsymbol{Q}\boldsymbol{v}) \\ &= \boldsymbol{v}^T\boldsymbol{Q}^T\boldsymbol{G}\boldsymbol{Q}\boldsymbol{v} \end{aligned} \tag{9}$$

Since $\boldsymbol{v}$ is arbitrary, this equation leads to the following condition on $\boldsymbol{Q}$:

$$\boldsymbol{G} = \boldsymbol{Q}^T\boldsymbol{G}\boldsymbol{Q} \tag{10}$$

Which might be more familiar to the reader in Cartesian coordinates where $\boldsymbol{G} = \boldsymbol{I}$

$$\boldsymbol{I} = \boldsymbol{Q}^T\boldsymbol{Q} \tag{11}$$

Which is the condition for $\boldsymbol{Q}$ to be orthogonal. Orthogonal matrices have determinant ± 1 the same is the case for the transformation matrix by:

$$\begin{aligned} \det(\boldsymbol{G}) = \det(\boldsymbol{Q}^T\boldsymbol{G}\boldsymbol{Q}) &= \det(\boldsymbol{Q}^T)\det(\boldsymbol{G})\det(\boldsymbol{Q}) \\ 1 = \det(\boldsymbol{Q}^T)\det(\boldsymbol{Q}) &= \det(\boldsymbol{Q})^2 \\ \Rightarrow \det(\boldsymbol{Q}) &= \pm 1 \end{aligned} \tag{12}$$

In addition to the orthogonality condition, we also know that the matrix $\boldsymbol{Q}$ must map integer vectors to integer vectors. In fact, for all space groups a representation using only matrices from $\{-1, 0, 1\}^{3\times 3}$ can be found.

For applications in computation, it is necessary to find a finite representation of space groups. One such representation can be found in what will be referred to as the normalized space group of $\mathfrak{G}$ denoted $\tilde{\mathfrak{G}}$. It is defined as the quotient group of $\tilde{\mathfrak{G}} = \mathfrak{G}/\mathfrak{T}$ where $\mathfrak{T} = \{(\boldsymbol{I}, \boldsymbol{v}) |\ \boldsymbol{v} \in \mathbb{Z}^3\}$ is the group generated by translations along the basis vectors. In the same fashion, the quotient vector space $\tilde{\vec{A}} = \vec{A}/\mathbb{Z}^3$ and the quotient point space $\tilde{A} = A/\mathbb{Z}^3$ is defined. The order of a space group is defined to be the cardinality of $\tilde{\mathfrak{G}}$.

3.2. Point Groups

To each space group, there exists a corresponding point group, which is the matrix group which is created by ignoring the translational part of the isometries in the space group. They are named point groups because at least on point remains unchanged after a transformation. When represented as a matrix group, the origin always remains unchanged. Note that the action of the point group on vectors is the equivalent to the action of the space group.

There exist 32 crystallographic point groups. These groups can further be classified into 11 Laue classes. Each Laue class is represented by one specific Laue group, which is obtained by adding the inversion matrix $\mathrm{diag}(-1, -1, -1)$ to the generators of the group. [7]

The Laue group is relevant in crystallography because some analysis techniques, like diffraction methods, have an inherent inversion symmetry and thus, crystals in the same Laue class produce diffraction patterns of the same symmetry.

In this work, the notation $L(\mathfrak{G})$ is used for the Laue group corresponding to the space group $\mathfrak{G}$.

3.3. Wyckoff Positions

In simple terms, Wyckoff positions describe how many times a site occurs in a unit cell and what symmetries it must follow.

Mathematically, a Wyckoff position can be described by an orbit and a stabilizer. An orbit $O_{\mathfrak{G}}(P) = \{\mathcal{G}P \mid \mathcal{G} \in \mathfrak{G}\}$ is the set of points which the point P is mapped to by elements of the group $\mathfrak{G}$. The stabilizer $S_{\mathfrak{G}}(P) = \{\mathcal{G} \in \mathfrak{G} \mid \mathcal{G}P = P\}$ is the subgroup of $\mathfrak{G}$ for which the point P remains unchanged. Wyckoff sites are commonly given in terms of their position in the unit cell. Thus, the normalized space group must be used to determine their multiplicity.

The multiplicity n of a site is given by $\left|O_{\tilde{\mathfrak{G}}}(P)\right|$, the orbit of the point P in the unit cell, and the site symmetry at point P is given by $S_{\mathfrak{G}}(P)$.

A position P is called general if the $S_{\tilde{\mathfrak{G}}}(P)$ only contains the identity, i.e. is the trivial group. By the orbit stabilizer theorem its multiplicity is $\left|\tilde{\mathfrak{G}}\right|$ the order of the space group $\mathfrak{G}$.

A special position P is a position with a nontrivial stabilizer $S_{\tilde{\mathfrak{G}}}(P)$. Any object at the point P in crystal must at least have an internal symmetry of its site symmetry, otherwise, the symmetry of the crystal is broken. [7]

3.4. Pairs in Crystals

A pair is an ordered collection of two positions. It can be represented by a two positions (P_1, P_2). Another defining feature of a pair is the pair vector $\overrightarrow{P_1P_2}$.

Two pairs (P_1, P_2) and $(P_{1'}, P_{2'})$ are considered equivalent if there exist a transformation $\mathcal{G} \in \mathfrak{G}$ for which $(P_1, P_2) = (\mathcal{G}P_{1'}, \mathcal{G}P_{2'})$ or $(P_2, P_1) = (\mathcal{G}P_{1'}, \mathcal{G}P_{2'})$

The pair multiplicity is defined to be the product of the cardinality of the expansion and the cardinality of the orbit of the first point of the pair. An additional factor of 2 is required for mixed pairs, as they can be constructed from either of the starting positions.

Let the expansion $E_{\mathfrak{G}}(P_1, P_2)$ be the set of all symmetry equivalent pairs to the pair (P_1, P_2), containing the point P_1. The expansion of a pair contains two types of pairs. The first type is generated by the stabilizer of P_1. The second type only exist in single site pairs and is produced by the symmetries which take P_2 to P_1.

$$M(P_1, P_2) = \begin{cases} \left|E_{\mathfrak{G}}\left(P_1, \overrightarrow{P_1P_2}\right)\right| \left|O_{\widetilde{\mathfrak{G}}}(P_1)\right| \text{ if } P_2 \in O_{\mathfrak{G}}(P_1) \\ 2\left|E_{\mathfrak{G}}\left(P_1, \overrightarrow{P_1P_2}\right)\right| \left|O_{\widetilde{\mathfrak{G}}}(P_1)\right| \text{ else} \end{cases} \tag{13}$$

4. Examples of Pair Multiplicities

4.1. One-dimensional Examples

In the one-dimensional case, there are only two line groups.

4.1.1. Line Group p1

In p1 all symmetry operations are translations by multiples of the lattice vector $\vec{l}$, it is isomorphic to $(\mathbb{Z}, +)$. Thus, all pairs of symmetry related position can be uniquely described by a position within the unit cell and the translation $n\vec{l}$ between them.

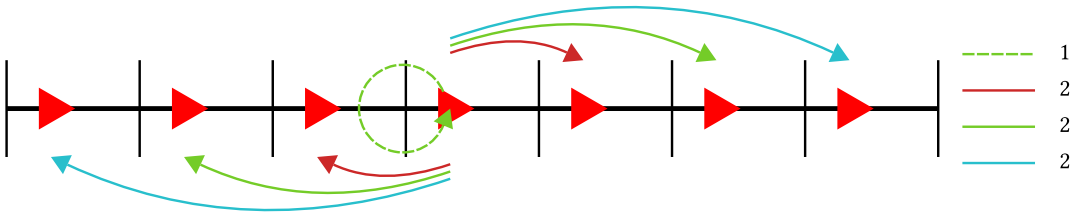


Figure 1: Pairs in line group p1

Figure Figure 1 shows how, starting from one position, there are two possibilities to construct each type of pair, except the zeroth pair from the starting position to itself, which only exist once. Thus, the multiplicity of such any pair in the line group p1 is 2 except that of the zeroth pair, which is two. Per unit cell, there is one starting position $\boldsymbol{p}$ with the paired position at $\boldsymbol{p}' = \boldsymbol{p} + n\vec{l}$.

4.1.2. Line Group p1m

In p1m additional to the translations by multiples of the lattice $\vec{l}$ there are mirror operations which mirror the line at positions $\boldsymbol{m} + n\vec{l}$. This group is isomorphic to the direct product $(\mathbb{Z}, +) \times (\{-1, 1\}, \cdot)$.

In p1m there are two Wyckoff positions. The general positions with Wyckoff multiplicity 2, in other words it exists twice per unit cell, and the special positions at $\boldsymbol{m}$ and $\boldsymbol{m} + \frac{1}{2}\vec{l}$, which have Wyckoff multiplicity 1 and the site symmetry m. Here the calculation of the pair multiplicity must be treated differently for origins at general and special positions.

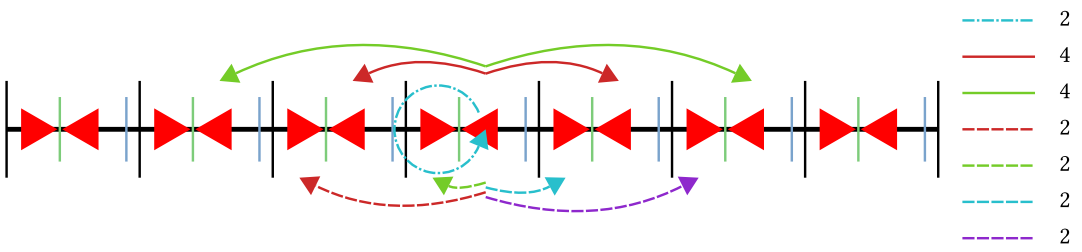


Figure 2: Pairs of general positions in line group p1m

The general position has site symmetry 1. For general positions, the pairs (shown in Figure 2 can be categorized further into two categories. For pairs of type $n\vec{l}$, there exist two possibilities for each pair, one in the positive direction on in the negative. Since there are two starting positions per unit cell, the pair multiplicity is 4.

For pairs which do not occur over a basis vector length, there is only one possibility for the construction of such a pair. Considering the two possible starting positions, the pair multiplicity is 2. The zeroth pair is again a special case since it is of type $n\vec{l}$ but still can only be built once from each starting position.

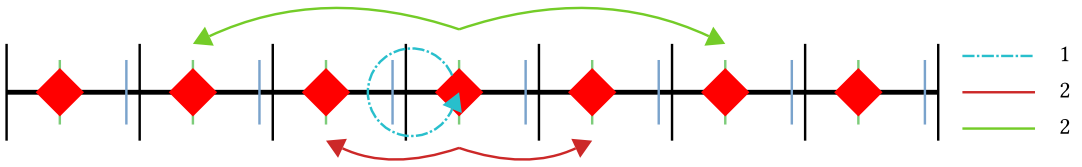


Figure 3: Pairs of special positions in line group p1m

Special positions in contrast have site symmetry m. Their pair expansions are shown in Figure 3. Thus, each pair can be built once in each direction. Except the zeroth pair, which can again only be built once from each position. Since the Wyckoff multiplicity is 1, pairs built from special positions have pair multiplicity 2, except the zeroth pair which has pair multiplicity 1.

4.2. Two-dimensional Example

As a next example, consider pairs of positions on the mirror axis of the wallpaper group p2mg. The coordinates of such a point are $(x, \frac{1}{4})$ with its symmetry equivalent point at $(1-x, \frac{3}{4})$.

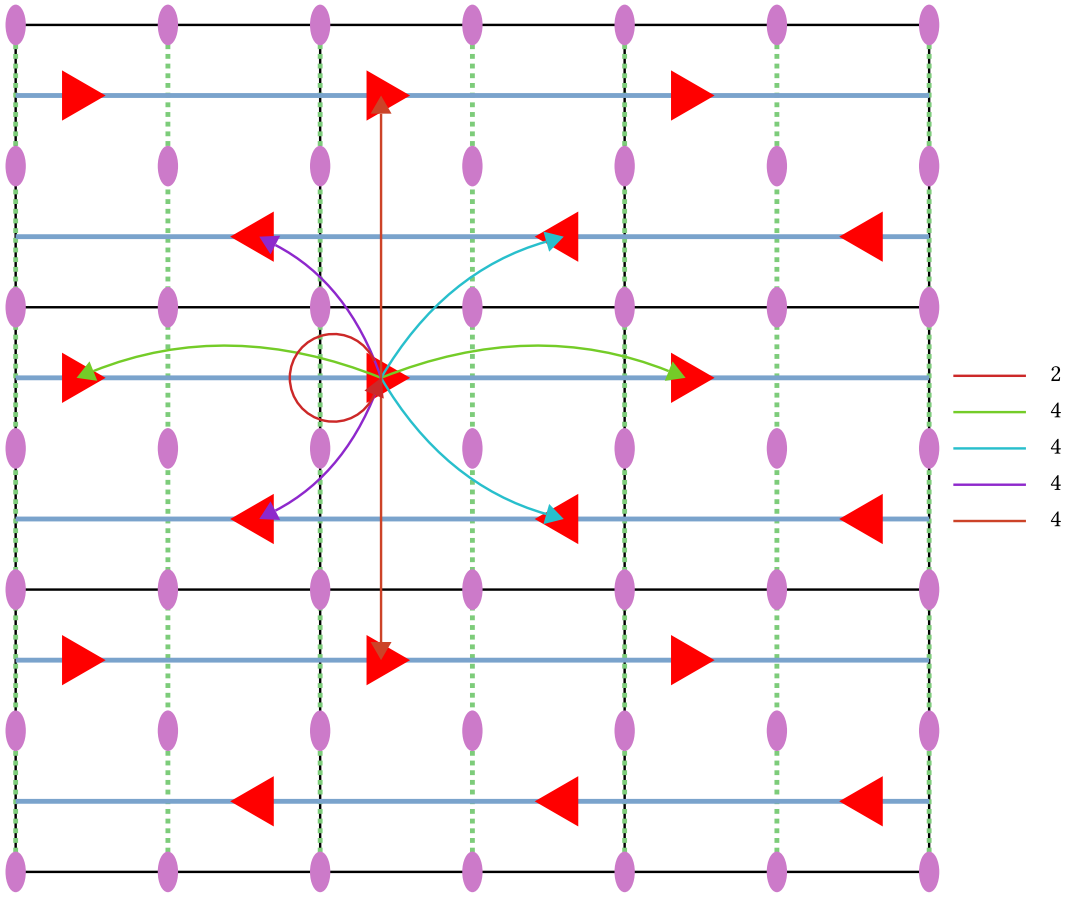


Figure 4: Pairs in the wallpaper group p2mg

Figure 4 shows the pair expansions of the shortest pairs of these sites. Similar to the pairs of general positions in the line group p1m the zeroth pair has multiplicity 2, because the Wyckoff multiplicity is 2.

In this two-dimensional example, it is apparent why we need to consider symmetries which move the end position of pair to the start position of the untransformed pair as the pair with the pair vector $\langle 10 \rangle$ and the pair with pair vector $\langle \bar{1}0 \rangle$ are equivalent even though there is no symmetry element in the stabilizer of P_1 which would transform one to the other.

4.3. Three-dimensional Example

The three-dimensional examples shown in Figure 5 are pair expansions from the site $[0, 0, 0]$ in a primitive cubic structure with the corresponding space group $Pm\bar{3}m$. As three-dimensional structures are harder to visualize, no more complicated examples are shown here.

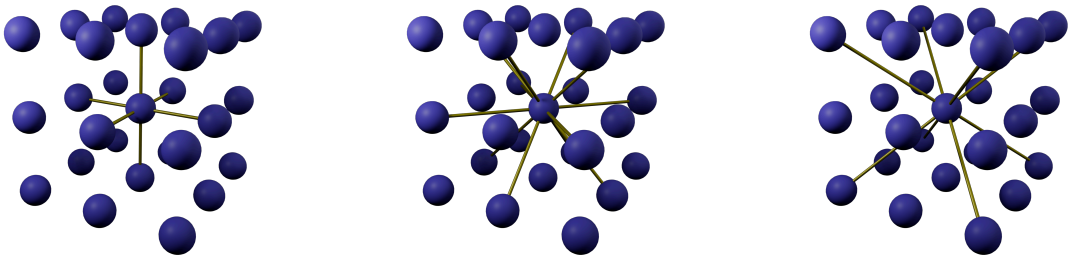


Figure 5: The First Three Nontrivial Pairs in a Cubic Lattice

Since there is only one $\langle 000 \rangle$ position in a primitive centered cell, the zeroth pair has multiplicity 1. The $\langle 100 \rangle$ pairs have a pair multiplicity of 6, the $\langle 110 \rangle$ pairs have a multiplicity of 12 and the $\langle 111 \rangle$ pairs have a multiplicity of 8. Note the correspondence between these pair of multiplicities and the number of faces, edges, and vertices of a cube.

5. The Program

The following section presents the implementation of the objects described in section Section 3. The code examples are written in Rust-like pseudo language, which ignores Rust borrow checker to simplify the code. Additionally, the code of the project includes some additional structs and steps in the algorithms to allow for more performant code. The code can be found on GitHub github.com/max-kay/bachelor_thesis.

5.1. Affine Space Implementation

All coordinates used in the program are rational numbers implemented as a pair of `i32`, which are always represented in reduced form and with a positive denominator. Rational numbers were chosen instead of floating-point numbers to allow for exact calculations and comparisons of coordinates. From triplets of rational numbers, vectors `Vec3` and positions `Pos3` and operations needed for them to form an affine space were defined.

Similarly, matrices `Mat3` and affine transformations `Affine3` were implemented, where a matrix is represented as a list of nine rational numbers and an affine transformation is represented as a pair of a matrix and a vector, as described above.

Additionally, a struct `Bounds3` was defined. It is used for the implementation of the remainder. Here 3 integers were chosen as it only makes sense to produce integer bounds in a crystal lattice.

The remainder operation was defined on positions `Pos3` such that each the result of the operation `Pos3(x', y', z') = Pos3(x, y, z) % Bounds3(a, b, c)` produces coordinates `x'`, `y'` and `z'` in the ranges $[0, a)$, $[0, b)$ and $[0, c)$ respectively. Note that if $a, b, c = 1$ the remainder produces positions in the unit cell.

For vectors `Vec3` the operation `Vec3(x', y', z') = Vec3(x, y, z) % Bounds3(a, b, c)` such that it produces coefficients `x'`, `y'` and `z'` in the ranges $\left(-\frac{a}{2}, \frac{a}{2}\right]$, $\left(-\frac{b}{2}, \frac{b}{2}\right]$ and $\left(-\frac{c}{2}, \frac{c}{2}\right]$ respectively.

The different implementation of the remainder for `Vec3` allows vectors to be represented by the shortest vector in the equivalence class.

The remainder operation was defined on affine transformations `Affine3` too. Here the operation leaves the matrix untouched but takes the remainder of the associated vector.

5.2. Space Group Implementation

The space group is represented as the normalized space group, where all operations take place modulo `Bounds3(1, 1, 1)`. Using this representation, there is a finite list of symmetry operations and all other symmetries can be easily generated by the multiplication of integer translation with the operations already present. The space group can be constructed from a list of generators by the following simple algorithm.

```
fn isometry_group_new(generators) -> IsometryGroup {
  bounds = Bounds3(1, 1, 1);
  symmetries = [];

  for sym in generators {
    normalized_sym = sym % bounds;
    if normalized_sym not in symmerties {
      symmetries.push(normalized_sym);
    }
  }

  added_new = true;

  while added_new {
    added_new = false;
    for sym_1 in symmetries {
      for sym_2 in symmetries {
        new_op = sym_1 * sym_2 % bounds;
        if new_op not in symmetries {
          symmetries.push(new_op);
          added_new = true;
        }
      }
    }
  }
  return IsometryGroup{ symmetries };
}
```

First, this algorithm brings the generator into the form required and checks for duplicates. Then it tries to close the group by going through all multiplications of elements. If the new operation is not present, it is appended to the list. This process is repeated until no new operation was added to the list in one pass through.

This algorithm is far from optimal, but because of the small numbers of elements in normalized space groups this is of no concern. If the wrong symmetries are supplied, the group might not be finitely closable. In this case, the algorithm described above results in an infinite loop. As a safety precaution, an upper limit of 10000 iterations was placed on the outer while loop.

The struct `IsometryGroup` contains the method `symmetries_in_bounds` which produces all symmetries within the bounds provided.

5.3. Point Group Implementation

The struct `PointGroup` is used to represent point groups, it can be constructed from its generators in the same fashion as the space group.

```
fn point_group_new(generators) -> PointGroup {
  symmetries = [];

  for sym in generators {
    if sym not in symmerties {
      symmetries.push(sym);
    }
  }

  added_new = true;

  while added_new {
    added_new = false;
    for sym_1 in symmetries {
      for sym_2 in symmetries {
        new_op = sym_1 * sym_2;
        if new_op not in symmetries {
          symmetries.push(new_op);
          added_new = true;
        }
      }
    }
  }
  return PointGroup{ symmetries };
}
```

Note that for point groups, no normalization is necessary.

The algorithm to produce the point group associated with a space group iterates through all space group symmetries, extracts the matrix part of the symmetry and deduplicates the resulting list.

```
fn to_point_group(space_group) -> PointGroup {
  new_symmetries = [];
  for sym in space_group.symmetries {
    matrix = sym.matrix;
    if matrix not in new_symmetries {
      new_symmetries.push(matrix);
    }
  }
  return PointGroup{ symmetries: new_symmetries };
}
```

Laue group can be created by a similar process but since the inversion element is pushed to the list of matrices, the list of symmetries needs to be sent through the `point_group_new` method, to ensure closure under multiplication.

```
fn to_laue_group(space_group) -> PointGroup {
  new_symmetries = [];
  for sym in space_group.symmetries {
    new_symmetries.push(sym.matrix);
  }
  symmetries.push(diagonal_matrix(-1, -1, -1));
  return point_group_new(symmetries);
}
```

5.4. Wyckoff Positions

For a position in the unit cell, the Wyckoff position can be calculated from the starting position `position` and the space group `group` by the following process.

```
fn site_new(position, group) -> Site {
  bounds = Bounds3(1, 1, 1);
  position = position % bounds;
  orbit = [position];
  stabilizer = [];

  for sym in group.get_symmetries() {
    new_pos = sym * position;
    if new_pos % bounds == position {
      stabilizer.push(symmetry_from_translation(position - new_pos) * sym);
    } else if new_pos % bounds not in orbit {
      orbit.push(new_pos % bounds);
    }
  }

  return Site { position, orbit };
}
```

The Wyckoff multiplicity is the length of the orbit, as they represent all symmetry equivalent positions in the unit cell. A struct called `Site` is used to collect the starting position, the orbit and the stabilizer.

The struct `Site` contains the method `orbit_in_bounds` which produces the orbit within the bounds provided. Moreover, the struct `Site` contains the method `contains_pos` which returns a `true` if the position is in the site and `false` otherwise.

5.5. Pair Expansion Implementation

The constructor for `Expansion` takes the starting position, the vector associated with the pair and the isometry group. After translating the starting position into the unit cell. The `Site` is constructed from the starting position and the space group. The pair vector is added to the starting position and the resulting position is checked against the orbit of the `Site`.

Then the following algorithm is applied to the pair to determine its expansion:

```
fn expansion_new(origin_site, end_position, group, bounds) -> Expansion {
  expansion = [];
  start_position = orgin_site.position;

  for sym in group.symmetries_in_bounds(bounds) {
    new_p1 = (op * start_position) % bounds;
```

```
    new_p2 = (op * end_position) % bounds;
    if new_p1 == start_position and new_pos2 not in expansion {
      expansion.push(new_p2);
    }
    if new_p2 == start_position and new_pos1 not in expansion {
      expansion.push(new_p1);
    }
  }

  return Expansion {
    is_ab_pair: not site_contains_pos(origin_site, end_position),
    origin_site,
    vec: (end_position - origin_position) % bounds,
    expansion,
  }
}
```

This algorithm applies each operation to the starting and end position of the pair and then checks either one of the positions was mapped to the original starting position, if so, it appends the other position to the expansion. Subsequently the end position is tested against the origin_site to see if the pair is a single site or mixed site pair.

The pair multiplicity now can now be calculated by multiplying the length of the expansion with the length of the orbit of the pairs and with an additional factor of 2 for mixed pairs, as each pair can be constructed from either of the two sites.

5.6. Pair Calculation

As arguments, the program takes a space group, the positions from which the pairs need be to generated, the bounds applied to the problem and optionally a boolean, which determines if pairs of mixed sites should be calculated.

The program starts by calculating and deduplicating the sites from the given positions.

Then the program produces all single site pairs and continues to do the same for all pairs of sites if mixed pairs should be calculated. The algorithm is implemented as follows:

```
fn calculate_multiplicities(
  group,
  positions,
  bounds,
  construct_mixed_pairs
) -> [Expansion]
{
  sites = [];
  for pos in positions {
    pos = pos % Bounds3(1, 1, 1);
    if not contains_pos(sites, pos) {
      sites.push(new_site(position, group, bounds));
    }
  }

  expansions = [];
  for site in sites {
```

```
    expansions.append(construct_expansions(site, site, group, bounds));
  }

  if construct_mixed_pairs {
    for i in 0..len(sites) {
      for j in (i+1)..len(sites) {
        expansions.append(
          construct_expansions(sites[i], site[j], group, bounds)
        );
      }
    }
  }

  return expansions;
}
```

The implementation of `contains_pos` iterates through all sites in the list and checks if the orbits of any of the sites contains the position.

The function `construct_expansions` iterates through the orbit of `site_2` expanded to the bounds, checks if any of the constructed expansions already contains the pair and if not, it pushes the expansion on the list.

```
fn construct_expansions(site_1, site_2, group, bounds) -> [PairExpansion] {
  expansions = [];
  origin_position = site_1.position;
  for pos in site_2.orbit_in_bounds(bounds) {
    if not contains_pair(expansions, origin_position, pos, bounds) {
      expansions.push(PairExpansion::new(origin_position, pos, group, bounds));
    }
  }
  return expansions;
}
```

`contains_pair` iterates through all expansions in the list and checks if any of the expansions contains the pair by calling `expansion_contains_pair`, which checks if the list of expansion positions of the pair contains the end position of the pair. Note that for single site pairs this requires the pair to be always started from the same position and for mixed site pairs that the pair is always started from the same position from the same site. This is guaranteed by the rest of the program.

5.6.1. Note on Bounds

The implementation of this program requires bounds to be specified to construct the pair expansions. This has consequences for the pair multiplicities.

As an example, consider the pair with vector `[2, 0, 0]`. In an infinite crystal, the pair with vector `[-2, 0, 0]` can always be constructed from this pair. But in the bounds `Bounds3(4, 4, 4)` these vectors are considered equivalent. Thus, the pair multiplicities constructed in bounds, are different from those in an infinite crystal. For the main application of this program for the correlation fitting program Yell, this is an advantage. As Yell works with similar bounds and needs the pair multiplicities calculated in bounds.

5.7. The File Format

The file format was implemented using the Rust crate `pest`. `pest` allows for the grammar to be defined in an external file, which then needs to be included in the code. The rest of the code needed for parsing is then automatically generated using Rusts macro system, allowing for easy changing of the grammar. Additionally, `pest` provides nice errors when a file cannot be parsed.

An example file is shown below:

```
// this is a comment
Space Group:
// this an example using the space group Cmcm

// pure translations are given like this
1/2,1/2,0;

// this is the affine transformation
// which maps (x, y, z) to (-x, -y, -z)
-x,-y,-z;

// this is the affine transformation
// which maps (x, y, z)
// to (-x, -y, z + 1/2)
-x,-y,z+1/2;

-x,y,-z+1/2;


Positions:
// Positions to form pairs from
0,0,0;
0,0,1/4; // must be given as rational numbers

Bounds:
// The bounds which are applied to the problem
5,5,5; // must be integers

Mixed Pairs:
// mixed fields are optional default = false
true;
```

The input file can be divided into four sections.

In the `Space Group` section, the symmetry elements are specified. The elements are given per index, separated by commas and terminated by semicolons. Pure translations can be written simply as vectors. Note that the identity `x, y, z;` and the translations `1, 0, 0;`, `0, 1, 0;` and `0, 0, 1;` are implicitly included in the space group.

In the `Positions` section, the positions are defined. Similarly to the symmetry operations, the positions are given as comma separated values terminated by semicolons. Note that all coefficients need to be given as rational numbers. The program does not support floating-point coefficients.

The `Bounds` section defines the bounds which are applied to the problem, as described in section Section 5.6.1. Here, the coefficients need to be positive integers.

The `Mixed Pairs` section is optional and can only contain `true` or `false`. This boolean specifies if mixed pairs should be calculated. If this section is not specified, it defaults to `false`.

6. Conclusion

A program to calculate the pair multiplicities was successfully developed. To this end, positions and vectors from a three-dimensional affine space, as well as transformations on them were implemented. Furthermore, algorithms to calculate Wyckoff positions and pair expansions were developed and implemented. These implementations were used to create a program to construct all possible pairs of positions in any bounded space group. Additionally, the pair multiplicities were explained using one-, two- and three-dimensional examples.

The code for this project (including the markup for this report and the slides for the talk) can be found in a GitHub repository github.com/max-kay/bachelor_thesis. The program can be used in two ways. Either it is used as a command line tool. Please follow the instructions in the `README.md` on GitHub to build the tool from source. Or in a simple website, hosted through GitHub Pages max-kay.github.io/bachelor_thesis.

Bibliography

[1] B. Bokstein, M. Mendelev, and D. Srolovitz, *Thermodynamics and Kinetics in Materials Science : A Short Course: A Short Course*. in Thermodynamics and Kinetics in Materials Science: A Short Course. OUP Oxford, 2005. [Online]. Available: https://books.google.ch/books?id=Uu0yVnKpgw0C

[2] A. Simonov, "Hidden diversity of vacancy networks in Prussian blue analogues," *Nature*, vol. 578.

[3] A. Simonov and A. L. Goodwin, "Designing disorder into crystalline materials," *Nature Reviews Chemisty*, vol. 4, pp. 657–673, Dec. 2020, doi: 10.1038/s41570-020-00228-3.

[4] L. Palatinus and G. Chapuis, "SUPERFILP – a computer program for the solution of Crystal Structures by charge flipping in arbitrary dimensions," *Journal of Applied Crystallography*, vol. 40, no. 4, pp. 786–790, Jul. 2007, doi: 10.1107/s0021889807029238.

[5] A. Simonov, T. Weber, and W. Steurer, "\it Yell: a computer program for diffuse scattering analysis \it via three-dimensional delta pair distribution function refinement," *Journal of Applied Crystallography*, vol. 47, no. 3, pp. 1146–1152, Jun. 2014, doi: 10.1107/S1600576714008668.

[6] M. Berger, *Geometry I*. Springer-Verlag, 2004.

[7] IUCr, *International Tables for Crystallography, Volume A: Space Group Symmetry*. Kluwer Academic Publishing, 2002.

Appendix

Proofs

Claim $\mathcal{QP} = (\boldsymbol{QP}, \boldsymbol{Qp} + \boldsymbol{q})$ is the formula for the composition of affine transformations.

Proof Let X be any arbitrary point.

$$\begin{aligned}\mathcal{QP}\overrightarrow{OX} &= \mathcal{Q}\Big(\boldsymbol{P}\overrightarrow{OX} + \boldsymbol{p}\Big) \\ &= \boldsymbol{Q}\Big(\boldsymbol{P}\overrightarrow{OX} + \boldsymbol{p}\Big) + \boldsymbol{q} \\ &= (\boldsymbol{QP})\overrightarrow{OX} + (\boldsymbol{Qp} + \boldsymbol{q}) \\ &= (\boldsymbol{QP}, \boldsymbol{Qp} + \boldsymbol{q})\overrightarrow{OX}\end{aligned} \tag{14}$$

Claim The inverse of $\mathcal{Q} = (\boldsymbol{Q}, \boldsymbol{q})$ is given by $\mathcal{Q}^{-1} = (\boldsymbol{Q}^{-1}, -\boldsymbol{Q}^{-1}\boldsymbol{q})$.

Proof Consider the following composition using the formula proofed above.

$$\begin{aligned}\mathcal{Q}\mathcal{Q}^{-1} &= (\boldsymbol{Q}, \boldsymbol{q})(\boldsymbol{Q}^{-1}, -\boldsymbol{Q}^{-1}\boldsymbol{q}) \\ &= (\boldsymbol{Q}\boldsymbol{Q}^{-1}, \boldsymbol{q} - \boldsymbol{Q}\boldsymbol{Q}^{-1}\boldsymbol{q}) \\ &= (\boldsymbol{I}, \boldsymbol{q} - \boldsymbol{q}) \\ &= \mathcal{I}\end{aligned} \tag{15}$$

Definition H is a normal subgroup of G if $ghg^{-1} \in H$ for all $g \in G$ and $h \in H$.

Claim Let $\mathfrak{H}$ be a space group and $\mathfrak{T} = \{(\boldsymbol{I}, \boldsymbol{v}); \boldsymbol{v} \in \mathbb{Z}^3\}$ the group of integer translation. Then $\mathfrak{T}$ is a normal subgroup of $\mathfrak{H}$.

Proof Let $\mathcal{H} \in \mathfrak{H}$ be an arbitrary symmetry element and $\mathcal{T} \in \mathfrak{T}$ be an arbitrary integer translation.

$$\begin{aligned}\mathcal{H}\mathcal{T}\mathcal{H}^{-1} &= (\boldsymbol{H}, \boldsymbol{h})(\boldsymbol{I}, \boldsymbol{t})(\boldsymbol{H}^{-1}, -\boldsymbol{H}^{-1}\boldsymbol{q}) \\ &= (\boldsymbol{H}, \boldsymbol{h})(\boldsymbol{H}^{-1}, -\boldsymbol{H}^{-1}\boldsymbol{q} + \boldsymbol{t}) \\ &= (\boldsymbol{H}\boldsymbol{H}^{-1}, \boldsymbol{H}(-\boldsymbol{H}^{-1}\boldsymbol{q} + \boldsymbol{t}) + \boldsymbol{q}) \\ &= (\boldsymbol{I}, \boldsymbol{H}\boldsymbol{t})\end{aligned} \tag{16}$$

To be a valid space group, $\boldsymbol{H}$ must map integer vector to integer vector. Thus, $\mathcal{H}\mathcal{T}\mathcal{H}^{-1} = (\boldsymbol{I}, \boldsymbol{H}\boldsymbol{t})$ is an element of $\mathfrak{T}$.

Example Inputs and Outputs

Input

```
Space Group:
0, 1/2, 1/2;
1/2, 0, 1/2;
-x, -y, z;
-x, y, -z;
z, x, y;
y, x, -z;
-x, -y, -z;

Positions:
0, 0, 0;

Bounds:
4, 4, 4;
```

Output

```
              Origin,               Vector, Multiplicity
          [0, 0, 0],            [0, 0, 0],            4
          [0, 0, 0],            [0, 0, 1],           24
          [0, 0, 0],            [0, 0, 2],           12
          [0, 0, 0],            [0, 1, 1],           48
          [0, 0, 0],            [0, 1, 2],           48
          [0, 0, 0],            [0, 2, 2],           12
          [0, 0, 0],            [1, 1, 1],           32
          [0, 0, 0],            [1, 1, 2],           48
          [0, 0, 0],            [1, 2, 2],           24
          [0, 0, 0],            [2, 2, 2],            4
          [0, 0, 0],       [0, 1/2, 1/2],           48
          [0, 0, 0],       [0, 1/2, 3/2],           96
          [0, 0, 0],       [0, 3/2, 3/2],           48
          [0, 0, 0],       [1, 1/2, 1/2],           96
          [0, 0, 0],       [1, 1/2, 3/2],          192
          [0, 0, 0],       [1, 3/2, 3/2],           96
          [0, 0, 0],       [2, 1/2, 1/2],           48
          [0, 0, 0],       [2, 1/2, 3/2],           96
          [0, 0, 0],       [2, 3/2, 3/2],           48
```

Input

```
Space Group:
1/2, 1/2, 0;
-x, y, -z;
-x, -y, -z;

Positions:
0, 0, 0;
1/4, 1/3, 0;

Bounds:
3, 3, 3;

Mixed Pairs:
true;
```

Output

```
              Origin,               Vector, Multiplicity
           [0, 0, 0],            [0, 0, 0],            2
           [0, 0, 0],            [0, 0, 1],            4
           [0, 0, 0],            [0, 1, 0],            4
           [0, 0, 0],            [0, 1, 1],            8
           [0, 0, 0],            [1, 0, 0],            4
           [0, 0, 0],            [1, 0, 1],            4
           [0, 0, 0],           [1, 0, -1],            4
           [0, 0, 0],            [1, 1, 0],            8
           [0, 0, 0],            [1, 1, 1],            8
           [0, 0, 0],           [1, 1, -1],            8
           [0, 0, 0],        [1/2, 1/2, 0],            8
           [0, 0, 0],        [1/2, 1/2, 1],            8
           [0, 0, 0],       [1/2, 1/2, -1],            8
           [0, 0, 0],        [1/2, 3/2, 0],            4
           [0, 0, 0],        [1/2, 3/2, 1],            4
           [0, 0, 0],       [1/2, 3/2, -1],            4
           [0, 0, 0],        [3/2, 1/2, 0],            4
           [0, 0, 0],        [3/2, 1/2, 1],            8
           [0, 0, 0],        [3/2, 3/2, 0],            2
           [0, 0, 0],        [3/2, 3/2, 1],            4
       [1/4, 1/3, 0],            [0, 0, 0],            8
       [1/4, 1/3, 0],            [0, 0, 1],           16
       [1/4, 1/3, 0],            [0, 1, 0],           16
       [1/4, 1/3, 0],            [0, 1, 1],           32
       [1/4, 1/3, 0],            [1, 0, 0],           16
<output continues>
```