Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications 176 (2007) 486–506
Computer Physics Communications
www.elsevier.com/locate/cpc

Pattern information extraction from crystal structures ☆

Erhan Okuyan [a], Uğur Güdükbay [a,*], Oğuz Gülseren [b]

[a] *Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey*
[b] *Department of Physics, Bilkent University, 06800 Bilkent, Ankara, Turkey*

Received 2 May 2006; received in revised form 18 October 2006; accepted 4 January 2007

Available online 19 January 2007

Abstract

Determining the crystal structure parameters of a material is an important issue in crystallography and material science. Knowing the crystal structure parameters helps in understanding the physical behavior of material. It can be difficult to obtain crystal parameters for complex structures, particularly those materials that show local symmetry as well as global symmetry. This work provides a tool that extracts crystal parameters such as primitive vectors, basis vectors and space groups from the atomic coordinates of crystal structures. A visualization tool for examining crystals is also provided. Accordingly, this work could help crystallographers, chemists and material scientists to analyze crystal structures efficiently.

Program summary

Title of program: BilKristal
Catalogue identifier: ADYU_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/ADYU_v1_0
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland
Licensing provisions: None
Programming language used: C, C++, Microsoft .NET Framework 1.1 and OpenGL Libraries
Computer: Personal Computers with Windows operating system
Operating system: Windows XP Professional
RAM: 20–60 MB
No. of lines in distributed program, including test data, etc.: 899 779
No. of bytes in distributed program, including test date, etc.: 9 271 521
Distribution format: tar.gz
External routines/libraries: Microsoft .NET Framework 1.1. For visualization tool, graphics card driver should also support OpenGL
Nature of problem: Determining crystal structure parameters of a material is a quite important issue in crystallography. Knowing the crystal structure parameters helps to understand physical behavior of material. For complex structures, particularly, for materials which also contain local symmetry as well as global symmetry, obtaining crystal parameters can be quite hard.
Solution method: The tool extracts crystal parameters such as primitive vectors, basis vectors and identify the space group from atomic coordinates of crystal structures.
Restrictions: Assumptions are explained in the paper. However, none of them can be considered as a restriction onto the complexity of the problem.
Running time: All the examples presented in the paper take less than 30 seconds on a 2.4 GHz Pentium 4 computer.

PACS: 61.50.Ah; 61.68.+n; 07.05.Wr; 07.05.Rm; 61.66.-f

Keywords: Crystal; Crystallography; Chemistry; Material science; Pattern recognition; Primitive vectors; Basis vectors; Space group; Symmetry

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail addresses: okuyan@alumni.bilkent.edu.tr (E. Okuyan), gudukbay@cs.bilkent.edu.tr (U. Güdükbay), gulseren@fen.bilkent.edu.tr (O. Gülseren).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2007.01.002

1. Introduction

Obtaining the parameters for crystal structures is an important issue in crystallography. The physical properties of a material are closely related to its crystal structure. In material science, crystal parameters are used to classify materials. Such classification is quite useful in analyzing physical properties, particularly, in complex cases such as alloys whose atomic ratios can change.

Crystallography mainly uses X-ray diffraction techniques to determine the crystal structure of materials; X-ray absorbance data reveals crystal geometry [1,2]. These techniques generally give satisfactory results but sometimes the results are inadequate. In those cases, crystallographers must try several crystal geometries manually. At other times, scientists may work on theoretical materials where no sample is available and thus no X-ray diffraction data. Therefore, a tool that can determine crystal parameters from atomic coordinates could be very useful.

There is a considerable amount of research based on extracting pattern information from the atomic coordinates of a crystal structure; these studies can be grouped into three categories. The first category is crystallographic tools; *Computational Crystallography Toolbox* [3], an open source program, is in this category. These tools allow users to define their own unit cell by entering unit cell parameters. Users can examine atomic placements, perform several analyses, etc. Basically, these tools allow users to examine every known detail of unit cell structure to understand the crystal structure more clearly. However, since every essential unit cell parameter must be given to those tools as input, all they actually do is to provide a user interface where input parameters are converted to unit cell parameters.

There exists some notable work on identifying space group information from atomic coordinates [4,5]. For example, FINDSYM [5] identifies the space group symmetry and gives the lattice parameters and Wyckoff positions of atoms in a standard setting. ADDSYM [6–8] is a tool to search for additional symmetries in a given coordinate set. Hanneman et al. [9] describe two algorithms that allow the determination of the primitive cell of a structure, including its translational symmetry and the space group identification. However, due to the limitations of the simulation, the primitive cells identified are always triclinic and exhibits no symmetry elements. Hundt et al. [10] describe an algorithm to overcome the these limitations by employing some tolerance parameters, which are implemented in the program KPLOT.

The second category is crystallographic visualization tools, *RasMol* [11] is a well-known example. These tools provide a good understanding of crystal geometry by allowing users to examine crystal structures in 3D space. They provide several drawing models, the ball model, the ball-stick model, the wire frame model, etc. They allow users to disable the appearance of some atom types, changing their colors and sizes, etc. They allow the user to build multi-cells and to cut the crystal structure according to user-defined planes. They allow users to shape the crystal structure in any way they want. They also provide a sophisticated 3D visualization environment. Generally, crystallographic visualization tools are combined with crystallographic tools to maximize usefulness. *Crystal Maker* [12] and *Crystal Builder* [13] tools are two important examples of such combinations.

The tools in the third category are related to pattern recognition, computer vision and 3D shape matching. Since crystals follow some pattern, the proposed techniques can be used to find such patterns. Some notable examples of techniques in this category are proposed in [14] and [15].

The aim of this work is to extract pattern information from any crystal structure by using raw atomic coordinates to find the *primitive vectors* and *basis vectors*, and identify the *space group*. This task is relatively easy for a human for simple structures but it becomes quite difficult for complex structures. The molecular structure of crystals further complicate this process. These molecular structures can be simple molecules, such as H_2O, or they can be quite complex, like buckyballs, C_{60}, or biological materials, such as DNA or protein. A computerized approach is essential to handle such complex cases. To this end, we propose a framework that extracts crystal parameters for such complex structures. It finds primitive vectors and basis vectors, and identifies the space group; it also calculates additional unit cell parameters, such as the lengths of primitive vectors and the angles between them. The algorithms use atomic coordinates in crystal structure as input data.

Since crystal structures are repeated patterns of atomic positions in 3D space, a 3D visualization tool would be quite useful for observing crystal geometry. The second motivation of this work is to provide a good 3D visualization tool that allows users to explore crystal structure, such as cleaving surfaces or forming supercells. The proposed visualization tool works on unit cell data that is either extracted from atomic coordinates or provided by the user directly. The tool allows the user to look at unit cells from several angles, to combine several unit cells to obtain larger crystal segments, to show or hide several atom types, to cut crystal to obtain the desired surface, to dump the atomic coordinates that are shown, etc.

In this work, we assume that the atomic coordinates of atoms, which lie inside a sufficiently large volume of crystal structure, are available as input within a small error margin. For example, the atomic coordinates can be generated by using the packing of atomic spheres or X-ray diffraction data. It is important to identify the type of each atom in the input data. Since there can be more than one alternative combination of primitive vector triplets and basis vector sets that define a crystal structure, the tool is designed to be semi-automatic; throughout the analysis it asks the user the preferred primitive vector triplet alternative and preferred origin choice.

The organization of the paper is as follows. Section 2 presents the proposed framework. Section 3 gives implementation details. Section 4 presents experimental results and performance evaluation. Section 5 gives conclusions.

2. A framework for pattern information extraction

In any crystal structure, if a lattice point is translated by any integer combinations of primitive or lattice vectors, an identi-

cal point is obtained. In order for two atoms to be identical, they must belong to the same atom type and their position in the crystal structure must be the same. In other words, if A and B are identical, every atom C in the crystal structure has a corresponding atom C' of the same atom type, where C's position relative to A is the same as C''s position relative to B. This observation leads to the fact that for two identical atoms A and B in any crystal structure, the vector obtained by coordinate differences of A and B will be a combination of integer multiples of primitive vectors. Furthermore, by the definition of primitive vectors, for any atom A, there should be an identical atom $A_{i,j,k}$ for all integer values of i, j and k where $A_{i,j,k}$'s coordinates differ from A's coordinates by $i \times \vec{R}_1 + j \times \vec{R}_2 + k \times \vec{R}_3$, where $\vec{R}_1$, $\vec{R}_2$, and $\vec{R}_3$ are primitive lattice vectors. These observations establish the basis for the proposed framework. They imply that if identical atoms can be grouped together, difference vectors between every pair of identical atoms in each group can be extracted. The set of difference vectors will include all integer combinations of primitive vectors. The primitive vectors can be identified by using the difference vectors. The identification of basis vectors and space group will follow.

2.1. Data structures and indexing

Data structures and indexing methods are important in the efficient solution of this problem. Input data is queried several times throughout the analysis so that data structure significantly affects the runtime performance.

The analysis requires point search queries and three-dimensional range search queries for every point inside a cubic boundary. There are several data structure alternatives to store and index input data. We used an octree structure as the main data structure. The octree is a tree structure that starts with a cube enclosing the 3D region of interest (the root node). Then the cubes at each node are recursively subdivided into eight subcubes until the desired level of detail is reached.

The crystal segment given in the input data is assumed to have the shape of a cube. The boundary of the cubic volume to be indexed is known since the input data for the analysis is available. The cubic region enclosing the data is assigned to the root node. Each child of a node is assigned one eighth of its parent's volume. This is done by halving the volume of the parent node along each axis. All nodes at the same level of the octree correspond to the same amount of volume. The number of atoms per volume will not differ for different regions in the crystal since the crystal structures are usually homogeneous. These properties result in a balanced octree structure where the leaf nodes are all at the same level. This produces a better search performance than unbalanced octree structures. In our implementation, only the leaf nodes store data. Intermediate level nodes are used to direct the search.

In a general octree implementation, the number of records stored at each leaf node varies. This requires a linear search within the leaf nodes to find a record; this is an inefficient process. In our case, storing one record in each leaf node leads to a better performance since we perform a lot of point searches, for example, to identify space groups.

Point-search queries are performed by using an octree search algorithm. The search starts at the root node and is recursively redirected to one of the children of the current node until a leaf node is reached. The leaf node is checked to see whether it stores a record that satisfies the query constraint. If the answer is affirmative, a pointer to this record is returned. Otherwise, a null pointer is returned meaning that the search is unsuccessful. The complexity of this procedure is $O(\log(N))$, since the depth of the octree is proportional to $\log(N)$, where N is the number of nodes in the octree structure.

Range search queries can be efficiently answered by the octree structure. The search procedure finds all the points that lie inside the query range. Our range search algorithm forms a linked list of these points and returns a pointer to this linked list. The other algorithms that use the output of the range queries use this linked list structure. The search is performed recursively on every node whose volume intersects with the query range. On a leaf node, the algorithm checks whether the record satisfies the query constraint. If this is the case, the search routine appends this record to the output set. The worst case complexity of range searching is $O(N \log(N))$, where N is the number of nodes in the octree structure. Every node must be checked since the query volume can cover every node in the octree structure. However, since the range search queries performed during the analysis have small query volumes, the runtime performance is much better than $O(N \log(N))$. The query volumes in our case are all cubic volumes. Only $O(M \log(N))$ nodes will be visited in order to query a volume covering M leaf nodes. Thus, the expected complexity will be $O(\frac{V_Q}{V_T} N \log(N))$, where V_Q is the query volume and V_T is the total volume. The details of the octree structure and related algorithms can be found in [16].

2.2. The stages of the proposed framework

In the first stage, the atomic coordinates are read and indexed. Indexing should facilitate to retrieve points in a query region efficiently. In the second stage, the identical atoms are grouped together. Two atoms are identical if they are of the same type and have the same orientation in the crystal structure. After the grouping process is completed, difference vectors between every atom couple in every group are found. Then some of these vectors are eliminated since they are not qualified to be a primitive vector. Then, the user is asked to select one or more of the candidate primitive vector-set alternatives. Afterwards, the basis vectors can be calculated. Then the atoms that can be used to form a basis vector set are clustered. The atoms in a cluster are displayed so that the user selects the origin. Then the basis atoms are listed and the space group of the crystal structure is identified. This involves testing whether all symmetry operations of every space group are supported by the crystal structure. Fig. 1 summarizes the stages of the proposed framework.

Error handling is an important issue in this application. Due to the precision of the device or the imperfections in crystal structures, the input data may contain errors. In some cases, one may even prefer to introduce some error bars in order to give some flexibility to the input. For example, the atomic ra-

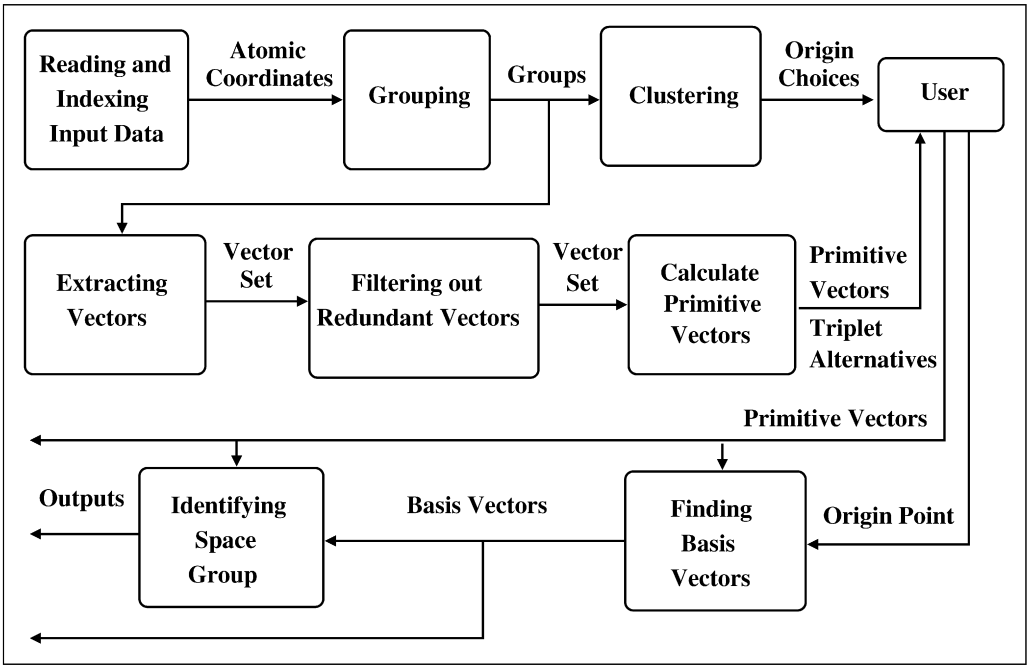

Fig. 1. The flow diagram of the proposed framework.

dius cannot be known for certain materials [17] and it changes in small amounts in different materials. For example, the Cl^- ion's radius is not the same in NaCl and KCl. Accordingly, several estimations are made in order to give approximate radius values of these atoms. There are several measurement standards and thus several atom-radius tables, such as CPK's, ionic, covalent and Van-Der-Walls radii [18], S&P [19] and VFI radii [20]. Generally, scientists can obtain approximate radius values from the appropriate table but these values will have a small margin of error. Accordingly, scientists sometimes prefer to introduce errors to input data to cover atomic radius errors. These errors are mostly small position differences of atoms.

Since small errors in atomic coordinates are very frequent, the parameter, *EPS*, is introduced for this purpose. This represents the amount of maximum coordinate error in each axis that can be seen in the input data. In other words, if an atom's coordinates are given as $[x, y, z]$ in the input data, the actual coordinates can be $[x \pm EPS, y \pm EPS, z \pm EPS]$. The value of the *EPS* parameter depends on the error range introduced in atomic coordinates. Ideally, *EPS* should be 0 but a value that is significantly smaller than the radius of the smallest atom in the input data will also work accurately. Setting the *EPS* parameter to high values may reduce accuracy. It is recommended that *EPS* be set at the maximum value of error margin in the atomic coordinates. It is also recommended that better data be used if the coordinate error is higher than 20% of the radius of the smallest atom.

Another type of error is missing atoms (i.e. vacancies) or impurities in the crystal. "Vacancy atoms" are the absence of an atom in a certain place in the crystal structure where it should exist in the perfect crystal. "Impurity atoms" are the substitution or insertion of another type of atom in the host crystal [21]. These imperfections are seen quite frequently. Scientists might wonder if a crystal structure is the material they expected with many imperfections, or is another material altogether. For such cases, the analysis proposed in this work helps scientists to reveal the actual pattern in the crystal data, thus revealing the crystal's form. Users can introduce such errors to the input data for this kind of analysis. All these errors cause the algorithms proposed in this framework to fail if they are not taken into account.

2.2.1. *Reading and indexing the data*

The input data contains one line for each atom's information. Each line contains an atom-type identifier string followed by three Cartesian coordinate values. Each line is read and an atom structure is generated based on this data. The atom structure contains the Cartesian coordinates of the atom, a unique identification number generated for each atom type, and several pointers to the related atoms. This procedure also determines the boundaries of the cubic crystal segment to be indexed.

After the input data is read, the atom records are indexed by using the octree structure. To do this, the octree is initialized by calculating the boundaries of the cubic crystal segment; this is calculated from the input data and the volume is assigned to the root node. Then, every atom record is iteratively inserted into the octree structure.

2.2.2. *The algorithm for grouping identical atoms*

Grouping identical atoms together is a crucial task in this analysis. For two atoms to be identical, they must belong to the same atom type and their relative orientation to their neighbors should be the same. If A and B are identical atoms, for every atom around A, there should be a corresponding atom of the same type around B with the exact relative positioning. Unless A and B are the same atoms, this definition requires

the crystal structure to be infinitely large in order for A and B to be identical. For practical purposes, it is sufficient to make sure that there is always a corresponding atom around B for all atoms around A within a pre-specified neighborhood.

If any point in a crystal structure is translated by integer multiples of primitive vectors, identical points are obtained. Three primitive vectors $\vec{R}_1$, $\vec{R}_2$, and $\vec{R}_3$ define a paralleloid, which can be considered as the unit cell of a crystal. If an atom A is taken as the origin, a volume V_A will be defined by the primitive vectors. The *matching volume* of an atom A is simply a list of all atoms whose coordinate differences with A at each axis are smaller than the matching range value. This list contains these atoms' relative coordinates with respect to A and their atom types. Calculating the matching volume of an atom is basically a range search query with the corresponding boundary parameters.

If we want to check whether A and B are identical, it is enough to check whether the matching volumes V_A and V_B are identical. This is because any point that does not lie inside V_A or V_B can be translated by integer combinations of primitive vectors to another point that does lie inside these volumes. Accordingly, any point outside the volume has an identical corresponding point inside these volumes. If any point outside causes a mismatch, it is guaranteed that some point inside the volume will also cause a mismatch.

Since the primitive vectors are unknown, it is impossible to determine the volumes of atoms. However, trying to match some larger volumes including these volumes will give correct results. We use cubic volumes around each atom because searching all atoms in a rectangular boundary is much more efficient than searching all atoms in any randomly-shaped volumes. We call half of the edge length of this cube the *matching range*. The boundary from the minus matching range to the matching range at each axis around the atom is used as this atom's matching volume. The user is asked to determine a value for the matching range parameter. The user should select a matching range parameter with a matching volume large enough to contain a unit cell of crystal. Values that are too low may produce inaccurate results, while values that are too high affect the performance. For most cases, selecting a matching range value that produces a matching volume covering about 10–20 atoms will give correct results, since atoms that are not identical tend to have significantly different positioning.

The algorithm for grouping identical atoms calculates the matching volume of each atom and groups atoms with identical matching volumes. The algorithm tries to match each atom with previously found groups. If an atom matches some group, it is included in this group. Otherwise, it forms another group. Only a subset of input atoms whose matching volumes are completely specifiable from input data are actually analyzed. In this way, complications caused by comparing partially specified matching volumes are avoided.

The grouping algorithm is severely affected by errors. Any error in the input data may cause two atoms that should belong to the same group to be put in different groups. Accordingly, the algorithm must be modified so that it calls two matching volumes identical even in the case of such errors in input data. Since the coordinates of atoms have an error margin $\pm EPS$, the relative coordinates of two atoms will have an error margin $\pm 2EPS$. The indexes on matching volumes are used to handle these errors. The number of groups will be much smaller than the number of atoms processed in the grouping algorithm. It is a good idea to index the matching volumes of the groups and search for a corresponding point in the group's matching volume for every point in an atom's matching volume. The creation and maintenance of an octree index on each group's matching volume is easy and inexpensive since the number of groups is low.

Missing atoms is the most important type of error for the grouping algorithm. Since the matching volume of an atom is a significantly big volume which contain several atoms, any atom is also placed in several other atoms matching volumes. Accordingly, if an atom is missing, none of the atoms that include this missing atom in its matching volume will be able to match its actual group. Accordingly, even a few missing atoms will cause the grouping algorithm to fail if no modifications are done. To handle this type of error, another parameter is introduced to specify how many atom mismatches are allowed to still consider two matching volumes identical. Most errors in the grouping algorithm can be corrected by allowing a small number of mismatches. The probability of a mismatch occurring in a matching volume is relatively small; and, the occurrence of more mismatches in the same matching volume is even smaller. By setting this parameter to a small value, most of the erroneous cases can be handled without scarifying accuracy.

In the grouping algorithm, if there are no errors, the first atom's matching volume for each group can be used as the matching volume for the group. Errors in atomic coordinates cause small errors in relative coordinates of the matching volumes but these errors are easily handled by using the input data with an error margin. However, if the matching volume contains a missing atom, this will cause a mismatch with any other atom's matching volume. Allowing a small number of mismatches handles most cases, but if the atom's matching volume also contains missing atoms, it may not match though it should. This problem is solved during the process of testing an atom against a group. If for a point P in the atom's matching volume, no corresponding point in the group's matching volume can be found, but the atom matches this group, P is inserted into the group's matching volume. Since the atom's matching volume cannot contain extra atoms that do not exist, the group's matching volume must have a missing atom corresponding to P.

Such modifications cover most of the cases that cause errors in the grouping algorithm. However, there can still be errors that may cause the formation of unwanted groups. These can be filtered out to obtain the desired groups. Since these error cases are rare, such unwanted groups contain only a small number of atoms. If the number of atoms belonging to a group is smaller than a specified threshold, the group is eliminated. In principle, the value of this threshold should be determined by input size and the expected number of groups. For example, NaCl should contain two groups. The actual groups usually contain about half the atoms given in the input data. A reasonable input contains thousands of atoms but unwanted groups mostly con-

```
GroupList=NULL;
foreach Atom A in ProcessVolume do
    MV=CalculateMatchingVolume(A);
    foreach Group G in GroupList do
        MismatchCount=0;
        NoMatchedAtoms=NULL;
        isMatched=true;
        foreach Atom B in MV do
            Boundary=B.Coordinates±4× EPS;
            R=RangeSearch(G.MatchingVolume,Boundary);
            if R==NULL then
                MismatchCount++;
                NoMatchedAtoms+=B;
            if MismatchCount > AllowedMatchCountDifference then
            isMatched=false;
            break;
        if isMatched then
            G.Insert(A);
            foreach Atom B in NoMatchedAtoms do
                G.MatchingVolume.Insert(B);
            break;
    if A not matches to any group then
        G=new Group();
        GMV=Duplicate(MV);
        G.MatchingVolume=Index(GMV);
        G.Insert(A);
```

Algorithm 1. The grouping algorithm.

tain a few atoms only. Therefore, using a value of around 20 works for most cases. However, if the input quality is poor, or the input crystal structure is not complete, there will be many groups with a relatively high number of atoms. For such cases, the value of the threshold should be increased. The grouping algorithm is given in Algorithm 1.

2.2.3. The algorithm for extracting primitive vectors

Translating any lattice point in crystal structure by an integer combination of lattice (primitive) vectors gives identical points. The vectoral distance between any two atoms A and B within the same group is equal to some integer combination of primitive vectors. Hence, a list containing vectoral distances between all atom pairs in a group can be constructed. It is possible to select any three vectors from the vector list and check if they can produce all other vectors in the list as their integer combinations. Accordingly, vector triplets that can be used as primitive vectors can be extracted from this list.

The proposed algorithm for extracting primitive vectors has three stages:

(1) A vector set containing vectoral distances between atom pairs of the same group is generated.
(2) The redundant vectors that cannot be a primitive vector are filtered out.
(3) Primitive vectors are found by testing vector triplets to see whether they can generate every other vector in the set as their integer combinations. This stage generates alternative primitive vector sets.

2.2.3.1. The vector set generation. This part takes each atom pair in a group and adds the vectoral distance between them to the vector list. The aim of extracting vectors is to construct a vector list that can be used to calculate primitive vectors. Therefore, the vector list should contain vectors to form a primitive vector triplet and several other vectors to test if a vector triplet can be used as a primitive vector set.

Every atom pair in a group can be used to extract a vector. Most of these are duplicate vectors. To improve the extraction process, a reference atom can be defined and the relative distances of all other atoms to this atom can be used as the extracted vectors. Since every atom's relative distance to the reference atom defines a vector, integer combinations of primitive vectors within some range are added to the vector list. It is quite unlikely that a desired primitive vector set contains a long vector. Accordingly, eliminating long vectors would reduce the number of extracted vectors to a reasonable number. In general, this approach works perfectly and extracting vectors from one group is sufficient. However, since the input coordinates may contain errors, a vector list obtained from only one group may cause problems. Additionally, missing atoms will cause some

```
VLIST=NULL;
foreach Group G do
    R=G.ReferenceAtom;
    foreach Atom A of Group G except R do
        V=A-R;
        if V.Length< C then VLIST+=V;
Sort(VLIST);
InterpolateIdenticalVectors(VLIST);
```

Algorithm 2. The vector set generation algorithm.

```
Sort(VLIST);
V1=VLIST.FirstVector;
V2=NULL;
while V1 != NULL do
    V2=V1.NextVector;
    while V2 != NULL do
        tmp=V2.NextVector;
        if V2 is an integer multiple of V1 then
            Remove(V2);
        V2=tmp;
    V1=V1.NextVector;
return VLIST;
```

Algorithm 3. The algorithm for filtering out redundant vectors.

vectors not to be extracted. The effect of such errors can be reduced by extracting vectors from each group and merging. The vector set generation algorithm is given in Algorithm 2.

2.2.3.2. Filtering out redundant vectors. Three primitive vectors should be able to produce all other vectors as their integer combinations. If a vector $\vec{R}$ is a scalar multiple of another vector, it is impossible for any vector triplet containing $\vec{R}$ to produce all other vectors as their integer combinations. The proof is given in [22]. Eliminating such vectors will reduce the size of the vector list, and significantly improve the performance. In this algorithm, every vector pair is checked to see if one of them can be generated from the other; if this is the case, the longer vector is eliminated. Checking whether a vector is a scalar multiple of another vector is done by comparing the proportions of the x-, y-, and z-components of each vector. If these proportions are equal (with some error tolerance), they are considered scalar multiples of each other and the longer one is filtered out. The algorithm for filtering out redundant vectors is given in Algorithm 3.

2.2.3.3. Finding primitive vector alternatives. After the redundant vectors are eliminated, a list of vectors is obtained that can form primitive vector-triplet alternatives. A simple way of calculating the primitive vectors is to take every vector triplet (derived from the vector list) and check whether all other vectors in the list can be produced by integer combination of these vectors. This procedure has a $O(n^4)$ time complexity where n is the number of vectors in the list; filtering redundant vectors reduces the list size significantly, but there might still be many vectors in the list. A more efficient solution is to sort the vector list according to the length of the vectors and limit the number of vectors in the set. A parameter value specified by the user eliminates long vectors. Parameter values of around 100 give quite good performance. Triple combinations of these shortest vectors are selected to form candidate primitive vector triplets.

In order to check if given three vectors $\vec{R}_1$, $\vec{R}_2$ and $\vec{R}_3$ can produce the vector $\vec{R}$, it is necessary to solve the following equation.

$$\vec{R} = i \times \vec{R}_1 + j \times \vec{R}_2 + k \times \vec{R}_3.$$

Since the vectors are 3-dimensional, the equation is linear with three unknowns: i, j and k. If integer solutions can be found for i, j and k, it is concluded that the vectors $\vec{R}_1$, $\vec{R}_2$ and $\vec{R}_3$ can produce the vector $\vec{R}$. Otherwise, the vectors $\vec{R}_1$, $\vec{R}_2$, and $\vec{R}_3$ cannot be a primitive vector set alternative.

Since each vector will have an error margin of $\pm 2EPS$, finding integer solutions might not be possible. However, since error margins will be much smaller than the sizes of the vectors, i, j and k values should be close to integer values if the primitive vector candidates can produce $\vec{R}$. Accordingly, it is possible to solve this equation and eliminate primitive vector candidates that do not produce i, j and k values that are sufficiently close to integer values. This eliminates most of the primitive vector set alternatives. Following this, another test is used to make the

```
SortByLength(VLIST);
PVLIST=NULL;
VLEN=min(VLIST.Count,MaxNumOfPVCandidates);
for i=0 to VLEN-1 do
    for j=i+1 to VLEN-2 do
        for k=j+1 to VLEN-3 do
            isPV=true;
            for t=0 to VLIST.Count-1 do
                if VLIST[i],VLIST[j] and VLIST[k] cannot produce VLIST[t]
                then
                    isPV=false;
                    break;
            if isPV then
                PVLIST+=
                    new PrimitiveVector(VLIST[i],VLIST[j],VLIST[k]);
return PVLIST;
```

Algorithm 4. The algorithm for calculating primitive vector alternatives.

final decision. The equation given above can be rewritten as

$$\vec{R} \pm 2EPS = i \times (\vec{R}_1 \pm 2EPS) + j \times (\vec{R}_2 \pm 2EPS) + k \times (\vec{R}_3 \pm 2EPS)$$

which is equivalent to

$$\vec{R} \pm \big(2 \times EPS \times (i + j + k + 1)\big) = i \times \vec{R}_1 + j \times \vec{R}_2 + k \times \vec{R}_3.$$

Since the calculated i, j and k values cannot be too far from their actual integer values, converting these values to the closest integer values is logical. Then, these integer i, j and k values and the vector parameters can be used to test if the given equation holds. If the answer is yes, then $\vec{R}_1$, $\vec{R}_2$, and $\vec{R}_3$ can produce $\vec{R}$ as their integer combination. The algorithm for calculating the primitive vector alternatives is given in Algorithm 4.

After finding all vector triplets that can be used as primitive vector sets, the user is asked to select one or more primitive vector set alternatives. The algorithm for extracting basis vectors and identifying space group continues according to the user's selections.

2.2.4. The clustering algorithm

Crystal structure is defined by lattice vectors and basis vectors. Basis vectors are atomic coordinate vectors of atoms that lie inside the paralleloid defined by lattice vectors. In order to define basis vectors, an origin must be defined. In principle, any point can be used as the origin from which the basis vectors that define crystal structure together with the primitive vectors can be calculated. However, scientists usually prefer to use the position of a certain atom or point as the origin; this results in a simpler geometric structure.

Since every atom in a group is identical, specifying one atom for each group is sufficient. However, specifying a random atom from each group is undesirable; this does not always allow the user to observe the relative positioning of specified atoms. Accordingly, atoms should be clustered as close to each other as possible so that the relative coordinates of atoms in a cluster can be easily observed.

The clustering is performed iteratively. Since each cluster has to have one atom from each group, the initial step is to assign each atom of the most crowded group to a different cluster. Then the remaining groups are iteratively processed. Processing group G is an example. For all clusters and atoms in G, the atom–cluster pair is found with the least distance between the atom and cluster center. Then, a direction vector is defined by using this pair as the atom's relative coordinate according to the cluster center. Then for all clusters C in the clusters list, an atom A whose relative distance to C is equal to the direction vector is assigned to C. It is necessary to find a direction vector since the clusters should have the same structure; every atom assigned to each cluster must have the same orientation.

After the cluster assignments are completed, the clusters that do not have atoms from each group are eliminated; a cluster without one atom from each group is incomplete. Such incomplete clusters can be seen on the surfaces of crystals but they are not suitable as origin alternatives. Once the clusters are obtained, they are sorted according to the distance between their center and the origin of input data. The cluster with the smallest distance is returned. The clustering algorithm is given in Algorithm 5.

The aim of this algorithm is simply to give the user a candidate basis set from which to select the origin. Taking the atoms closest to the origin of input data from each group could produce a good solution with much better runtime complexity. However, using this clustering algorithm will produce a geometrically more meaningful cluster. Since this cluster will be closer to the desired basis set, the relations between atoms will be easily visualized. After the coordinates of atoms in a cluster

```
ClusterList=NULL;
foreach Atom A in most crowded group do
    ClusterList+=new Cluster(A);
foreach Group G that is not processed do
    MinDist=∞ ;
    MinAtom=NULL ;
    MinCluster=NULL ;
    foreach Atom A in G do
        foreach Cluster C in ClusterList do
            D= distance between C and A ;
            if D < MinDist then
                MinDist=D;
                MinAtom=A;
                MinCluster=C;
    DV=MinAtom.Coordinates-MinCluster.Coordinates;
    foreach Cluster C in ClusterList do
        foreach Atom A in G do
            if A.Coordinates==C.Coordinates+DV then
                C.Assign(A);
                break;
    foreach Cluster C in ClusterList do
        if C.AtomCount < NumOfProcessedGroups then
            Remove(C);
Sort(ClusterList);
return First Cluster;
```

Algorithm 5. The clustering algorithm.

are shown to the user, (s)he can either select one atom from the proposed list as the origin, or manually enter the coordinates of the origin.

2.2.5. *The algorithm for finding basis vectors*

Basis vectors can be defined as the coordinates of atoms. The atoms lie inside the unit cell defined by the primitive vectors and the origin. With ideal data, it would be enough to find each point lying inside the unit cell. However, since the atomic coordinates may contain errors, a point that should be inside the cell might be placed outside or vice versa. This situation is quite common since the origin and primitive vectors are generally selected to place the atoms at the corners, edges or faces of the unit cells. Accordingly, a different approach must be followed.

Equal number of atoms belonging to each group should be placed in every unit cell. The clustering algorithm is used for this purpose. The desired basis vectors can be obtained with some modifications to the clustering algorithm. Basic modification is done on the direction vector calculation. In the original clustering algorithm, atom–cluster pairs with the smallest distances are found and used to calculate the direction vector. However, in the algorithm for finding basis vectors, the direction parameters come from the unit cell structure. Another modification is to use the origin of the unit cell as the cluster center, instead of using average coordinates of the atoms.

In the original clustering algorithm, the atoms belonging to the cluster closest to the origin are returned after the clusters are calculated. However, this approach may not work for finding basis vectors since a cluster contains one atom for each group. On the other hand, the user might use a vector set that is not primitive. For example, for the face-centered cubic crystal structure, the primitive vectors are $R_1 = \frac{1}{2}[0, a, a]$, $R_2 = \frac{1}{2}[a, 0, a]$, and $R_3 = \frac{1}{2}[a, a, 0]$. This structure contains one basis atom placed at the origin. However, due to its geometric simplicity, users often prefer the vector set $R_1 = [a, 0, 0]$, $R_2 = [0, a, 0]$, and $R_3 = [0, 0, a]$, with 4 basis atoms. Accordingly, the user is allowed to enter such vector triplets manually to be used as the lattice vector set. For this purpose, the user is expected to enter vectors that define a valid unit cell to guarantee that the unit cell defined by the vector set contains either the whole cluster or no part of that cluster. In other words, a unit cell will contain an equal number of atoms belonging to each group. The clustering process for finding basis vectors determines the shape of each cluster according to the volume defined by given vectors. Thus, a unit cell can be filled completely with clusters such that no part of these clusters left outside. Then the average coordinates of atoms are used as the cluster center and the atoms of every cluster whose center lies inside the unit cell are returned. The algorithm for finding basis vectors is given in Algorithm 6.

```
FindBasisVectors(Vects,Origin)
ClusterList=NULL;
foreach Atom A in least crowded group do
    C=new Cluster(A);
    C.Center=A.Coordinates;
    ClusterList+=C;
P=The paralleloid defined by Vects and starting from Origin;
C=Find the closest cluster to Origin whose center lies inside P;
DV=C.Coordinates-Origin.Coordinates;
foreach Cluster C in ClusterList do
    C.Center-=DV;
foreach Unprocessed Group G do
    MinDist=∞ ;
    MinAtom=NULL ;
    MinCluster=NULL ;
    foreach Atom A in G do
        foreach Cluster C in ClusterList do
            if A lies inside the paralleloid defined by Vects starting from
            C.Center then
                D= distance between C and A ;
                if D < MinDist then
                    MinDist=D;
                    MinAtom=A;
                    MinCluster=C;
    DV=MinAtom.Coordinates-MinCluster.Coordinates;
    foreach Cluster C in ClusterList do
        if There is an atom A in G at C.Coordinates+DV then
            C.Assign(A);
    foreach Cluster C in ClusterList do
        if C.AtomCount < NumOfProcessedGroups then
            Remove(C);
ReturnList=NULL;
foreach Cluster C in ClusterList do
    C.Center=Average atom coordinates of C;
    if C.Center lies inside P then
        ReturnList+=C.AtomList;
return ReturnList;
```

Algorithm 6. The algorithm for finding basis vectors.

2.2.6. The algorithm for identifying space group

The space group of a crystal structure is determined by checking whether it supports some symmetry operations. There are several symmetry operations: rotations, mirror operations, glide operations, etc. A crystal structure is tested to find out which symmetry operations it supports. Accordingly, it is classified into one of 230 predefined space groups. Any crystal structure should belong to one of these space groups [23]. The aim of this algorithm is to find the space group to which the crystal structure belongs.

A symmetry operation can be considered as a 3D coordinate operation, which translates a point into an identical point. In general, any symmetry operation can be defined in terms of rotation followed by translation [2]. Symmetry operations can be expressed by using a rotation matrix and a translation vector; applying a symmetry operation on a point can be expressed as a matrix vector multiplication and a vector addition.

In order to identify the space group of a crystal structure, it should be tested to find out whether it supports every symmetry operation of a given space group. If this is the case, it belongs to that space group. The space group with the highest group number that the crystal structure supports is returned as the space group of the crystal structure.

Checking if a symmetry operation is supported by a crystal structure can be done by applying this operation on several crystal points that cover the basis set. If the symmetry operation

```
ValidVects=NULL;
for i=-√K to √K do
    for j=i to √K do
        for k=j to √K do
            if i² + j² + k² < K then
                ValidVects+=i × V1 + j × V2 + k × V3;
Sort(ValidVects);
return ValidVects;
```

Algorithm 7. The algorithm for deriving valid vectors.

is supported, the translated point must be identical to the original point. There are two available coordinate systems for this procedure. These are the fractional coordinate system and the Cartesian coordinate system. For each alternative, appropriate space group symmetry matrices and vectors should be used. In this work, the fractional coordinate system is used since it requires fewer coordinate conversions. In addition, the symmetry matrices and vectors are generally given in fractional coordinates.

Testing whether a crystal structure supports a symmetry operation seems to be quite an easy task but it has some complications. Using arbitrary primitive vectors will not work for each space group. For example, in order to test if the crystal structure belongs to a space group from the cubic lattice class, a vector set defining a cubic unit cell must be used. But not just any vector set producing a cubic unit cell can be used. Primitive vectors should define the minimal cubic unit cell. Similarly, in order to test other lattice classes, vector sets defining the minimal unit cells of those classes should be used. Since there are seven lattice classes [1,17,21,23], seven different sets of vectors must be derived and used.

In order to derive such vector sets, primitive vectors must be used. Each integer combination of primitive vectors defines a valid vector. Accordingly, several choices of integer combinations of primitive vectors are used to define a set of valid vectors. Valid vectors are generated as integer combinations of primitive vectors. It is clear that infinitely many vectors can be generated by this approach. To keep numbers reasonable, valid vectors are limited to those vectors generated by using the integers i, j and k that satisfy inequality $i^2 + j^2 + k^2 < K$ where K is a predefined constant. After valid vectors are defined, they are sorted according to length in order to process shorter vectors earlier. Then every vector triplet is checked to see if it defines a unit cell belonging to one of seven lattice classes. If a vector triplet matches a class where no previous match has been found, it is recorded. The algorithm for deriving valid vectors is given in Algorithm 7.

Afterwards, each combination of three vectors from the set of derived valid vectors is checked to see if these vectors define a unit cell belonging to one of these seven classes. If this is the case, this unit cell is recorded and these vectors are used in space group tests belonging to this lattice class. To improve performance and ensure getting the minimal unit cell, vector triplets to be checked are sorted first. The algorithm is designed to check the vector triplets with smaller vectors before the vector triplets with larger vectors. Accordingly, the minimal unit cells belonging to each lattice class are obtained before other unit cells belonging to the same class. Once a unit cell is found for a class, other unit cells belonging to the same class are discarded. The algorithm for deriving unit cells of lattice classes is given in Algorithm 8.

After determining vector sets for each class, test points must be gathered. In principle, the set of test points should cover at least one point identical to each point in the basis set. However, it is also effective to use all atoms within some volume that can contain a unit cell. In this work, a cubic volume that is large enough to cover any possible unit cell is determined and all atoms lying in this volume are used as test points. Then, all symmetry operations of space groups are tested with all of the test points. Space groups are tested starting from the space group with the highest group number. When a crystal structure supports all the symmetry operations of a space group, this space group is returned as the space group of the crystal. The algorithm for identifying the space group is given in Algorithm 9.

Space groups are defined in order to classify the symmetry properties of crystal structures. A small difference in the primitive vectors or basis vectors can change the symmetry properties and hence, the space group. For example, the NaCl and TlF structures are quite close. NaCl has a cubic unit cell and its space group number is 225. TlF's structure can be considered as a slightly distorted NaCl structure. Its unit cell is not cubic. Axis lengths differ a small amount, making the unit cell orthorhombic. Thus, it cannot support the symmetry operations that cubic unit cells support; its space group number is 69. Distortions in TlF are as small as a certain level of noise can cause. In the presence of errors, it is quite easy to confuse the NaCl and TlF structures. If the error tolerance were set to a high level, distorted materials would be treated as higher symmetry materials. Otherwise, high symmetry materials would be treated as low symmetry materials. Unfortunately, there are no guidelines that can help to distinguish structural distortions and distortions caused by errors. It is therefore strongly recommended that ideal data be used and that the *EPS* parameter be set to a very small value in order to identify the space group correctly. With ideal data, it is enough to return the highest symmetry space group. However, we return all space groups that are supported for a material so that the user can further analyze the struc-

```
ValidVects=DeriveValidVects();
PVCNT=0;
for i=0 to ValidVects.Count-1 do
    for j=i to ValidVects.Count-1 do
        for k=j to ValidVects.Count-1 do
            PVCoefficients[PVCNT][0]=i;
            PVCoefficients[PVCNT][1]=j;
            PVCoefficients[PVCNT][2]=k;
            PVCNT++;
Sort(PVCoefficients);
for i=1 to 7 do Classes[i]=NULL;
for i=0 to PVCNT do
    V1=ValidVects[PVCoefficients[PVCNT][0]];
    V2=ValidVects[PVCoefficients[PVCNT][1]];
    V3=ValidVects[PVCoefficients[PVCNT][2]];
    if V1,V2 and V3 are not orthogonal then continue;
    ClsId=ClassOf(V1,V2 and V3);
    if Classes[ClsId]!=NULL then
        continue;
    else
        Classes[ClsId]=SetOf(V1,V2,V3);
return Classes;
```

Algorithm 8. The algorithm for deriving unit cells of lattice classes.

```
SpaceGroups=loadSpaceGroupData();
Classes=derivePVofClasses();
volume=determineBoundaries();
TestPoints=returnAllAtoms(volume);
for i=230 downto 1 do
    S=SpaceGroups[i];
    UC=Classes[ClassOf(S)];
    if UC==NULL then continue;
    isSupported=1;
    foreach Symmetry operation M,V of S do
        foreach Point P in TestPoints do
            C=getFractionalCoordinatesOf(P,UC);
            Q=M×C+V;
            C=getCartesianCoordinatesOf(Q,UC);
            if There is no atom of type P at C±2EPS then
                isSupported=0;
                break;
        if !isSupported then break;
    if IsSupported then Output+=S;
return Output;
```

Algorithm 9. The algorithm for identifying the space group.

ture manually. In spite of such modifications, the algorithm for identifying the space group is not very reliable in the presence of errors.

The algorithm for identifying the space group requires knowing the symmetry operations of each space group. In this implementation, we obtained the space group data of 273 space groups (some space groups have more than one form depending on the origin or the axis selections), from Bilbao Crystallography Server [24]. The source of symmetry operations stated in [25] are considered the most common reference tables used

in crystallography. Every time the tool is executed, this data is read and the space group data are loaded.

3. Implementation

We implemented the proposed framework as a software tool, called *BilKristal*. The tool consists of three programs, *Analyzer*, *VisualizationTool* and *UserInterface*. *Analyzer* performs the pattern extraction analysis and calculates the unit cell parameters. *VisualizationTool* uses unit cell parameters and provides facilities to visualize the crystal structure. *UserInterface* provides the necessary interface to interact with the user and other parts of the tool. We use three different programs, rather than one, to obtain a modular structure. This makes maintenance easier and produces high performance with a functional user interface. Since extracting pattern information from crystal structures is a computationally demanding process, we used C language in the implementation of *Analyzer* and *VisualizationTool*. Because *VisualizationTool* uses graphics to display crystal structures, the OpenGL and GLUT libraries are used in the implementation of this part. C++ with .NET Platform provides a suitable environment for developing programs with good user interfaces and so we used it for the implementation of *UserInterface*.

3.1. Analyzer

Analyzer is the program that executes the pattern extraction algorithms and finds the unit cell parameters. When it is executed, it obtains as arguments initial parameters, such as the input data path and analysis constants, such as *EPS*. Throughout the analysis, it requests some other parameters from the user, such as the origin choice or the primitive vector selections. It displays appropriate messages to inform the user, such as percentage values for the progress of the analysis. It outputs the results to files. *Analyzer* is a console application and it is not designed to be a stand-alone program.

3.2. VisualizationTool

VisualizationTool is responsible for visualizing the crystal structure. It uses the unit cell parameters as input and has several features to visualize the crystal structure more effectively. *VisualizationTool* is a console application and does not have a user interface. Both *Analyzer* and *VisualizationTool* are called from the *UserInterface* part.

VisualizationTool gets the required parameters from a file created by *UserInterface*. This file contain the unit cell parameters, such as the primitive vectors and the basis vectors, as well as atom parameters, such as the color and the radius of each atom type in the basis set. Initially, a single unit cell is shown. Afterwards, depending on the user's choices, the displayed structure is extended. The user can use the mouse or keyboard to rotate and zoom in/out on the displayed structure. The tool provides the following facilities:

(1) *Creating multi-cells*: *VisualizationTool* draws a single unit cell by translating the coordinate system by some integer combination of the primitive vectors and drawing each basis atom. Multi-cells are defined by determining the frequencies for each primitive vector. These frequencies represent the integer coefficients that are used to determine the translation vector of the coordinate system. There are two frequencies for each primitive vector. The first one is the minimum value and the second is the maximum value of those integer coefficients. In general, a unit cell is drawn for every integer-coefficient combination whose value for each primitive vector lies between these minimum and maximum values.
(2) *Changing drawing options*: the user can change the drawn size of atoms, enable or disable the display of sticks between atoms, the display of unit cells, the display of primitive vectors, and the display of controls.
(3) *Enabling or disabling the drawing of certain atom types*: A part of the interface can be considered the legend. In this part, the atom types in the crystal are listed. The colors of those atom types are also indicated so that user can identify them. In addition, the user can enable or disable the drawing of each atom.
(4) *Defining cut planes*: *VisualizationTool* can define *cut planes* that divide the crystal structure into two parts. It is defined by three numeric values and a cut operation. These three numeric values are given in the crystal indexing system [1]. The crystal indexing system takes primitive vectors as its three main axes. In order to define a plane in the crystal indexing system, the positions where this plane intersects with each of these main axes should be calculated. The inverses of these values by multiplication define this plane. For example, consider the primitive vectors $\vec{R}_1$, $\vec{R}_2$, and $\vec{R}_3$ as the coordinate axes. Let the plane P intersect each coordinate axis at points $(i\vec{R}_1, 0, 0)$, $(0, j\vec{R}_2, 0)$ and $(0, 0, k\vec{R}_3)$, respectively. Then, $(\frac{1}{i}, \frac{1}{j}, \frac{1}{k})$ are the values that define the plane P in the crystal indexing system; these are called Miller indices. Cut operations are simple comparison operators, such as $>$, $<$, $\geqslant$, $\leqslant$, and $=$. Any atom that does not satisfy a cut operation will not be drawn. In order to define cut planes, the crystal indexing system is used because it is the common indexing system for planes in crystallography. The user is allowed to define as many cut planes as (s)he desires. Accordingly, (s)he can give any convex polyhedral shape to the crystal structure.
(5) *Dumping atomic coordinates*: after performing several operations, users might want to obtain a list of atomic coordinates to be used as input to other utilities. Accordingly, the user is allowed to dump into a file the atomic coordinates (either fractional or Cartesian) of all atoms that are currently shown on the screen.
(6) *Animations*: The user can animate the crystal structure in 3D by simply rotating the structure around three principal axes in small amounts. The aim of the animation is to help the user to observe the crystal structure in 3D. The user can select animation styles and speed. The animation styles defined are: rotation as a combination of three principal

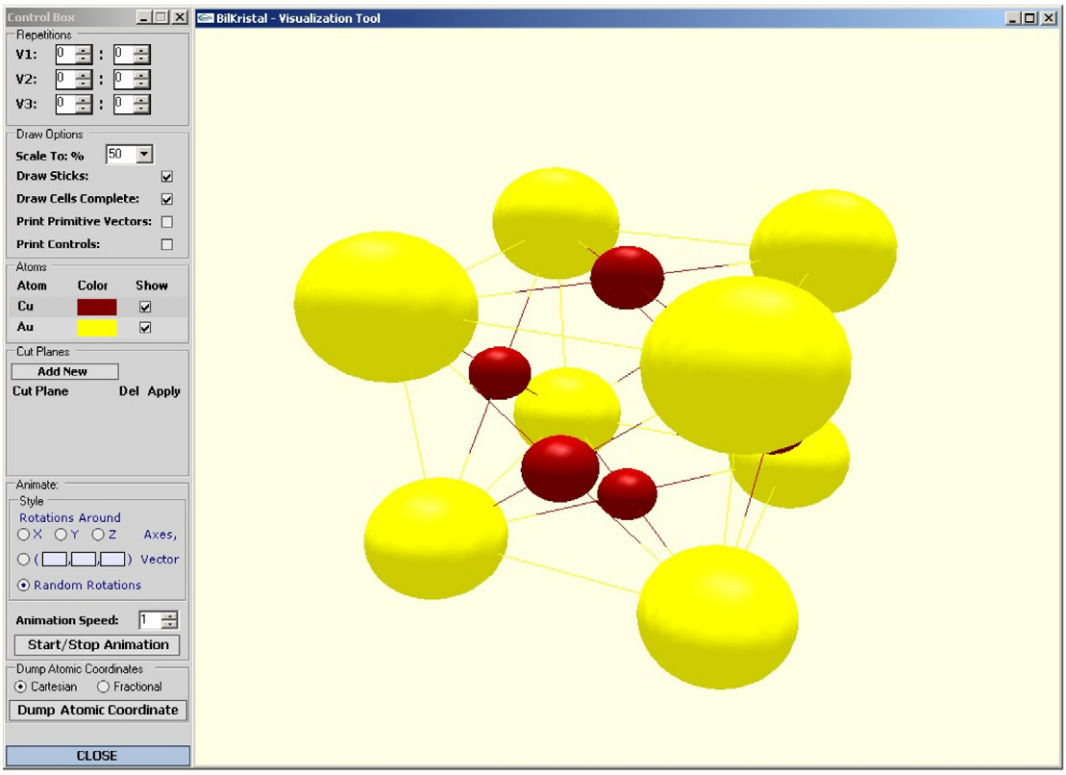

Fig. 2. The crystal visualization tool screenshot.

axis rotations (Euler angles); rotation around an arbitrary user-defined vector; and a random combination of rotations around the principal axes. Fig. 2 shows a snapshot of *VisualizationTool*.

4. Experimental results and performance analysis

We tested our framework with several test data that cover materials in most of the crystallographic classes (Fig. 3). Real crystal parameters of actual materials are used. Test data is generated automatically. To generate the data, each basis vector is translated by some integer combinations of primitive vectors. If the translated point lies inside a predefined cubic volume centered at the origin, the point is written to the file. This operation is repeated with different integer combinations of the primitive vectors, until the predefined volume is completely full. In this way, cubic crystal segments are obtained. The boundaries of this predefined volume are selected as -20.0 to 20.0 Å at each axis. In this way, we are guaranteed to obtain input data that can be used with the default threshold parameter, 20.0. Since the points that do not lie inside this predefined volume are not included in the output, there will be incomplete unit cells. In addition, the atomic ratios in the input data will not be equal to the atomic ratios in the unit cell. Two sets of data are generated for each kind of material. The first set is the ideal input data. The second set of data contains coordinate errors and missing atoms. The generation of the second set of data is a similar process to the generation of the first set, except that after calculating the actual position of a point, a small amount of noise (± 0.03) is added to the coordinate values. Another difference is that even though a point qualifies to be written into the output file, it is not written with a small probability (0.0001) to simulate the effect of missing atoms.

We tested our implementation with the materials (NaCl, La_2O_3, Cu_3Au, PtS, Al_3Ti, Mg, CoSn, αHg, and TlF) whose crystal parameters are obtained from [26], as well as MgF_2 and Ca_3SiBr_2 structures [5,9,10]. Both ideal and noisy data for these materials are analysed. During the analysis, default values of the analysis parameters are used. Detailed information about the analysis parameters can be found in [22]. Three sets of tests are performed. In the first part, primitive vectors and basis vectors are calculated and compared with the actual values. In the second part, the outputs of the intermediate stages for identifying the space group are produced. Finally, the tests for error handling performance is carried out.

4.1. Extracting primitive vectors and basis vectors

With the ideal data, the extracted primitive vectors are all accurate. The other primitive vector-set alternatives that the tool proposed are also accurate. However, the primitive vectors extracted from noisy data contain distortions. The vectors are close to the actual vectors, but not identical. Since the margin of error for each vector is $\pm 2EPS$ with the noisy data used in the tests, distortions up to ± 0.06 Å error margin is considered acceptable. No distortions higher than this limit are observed

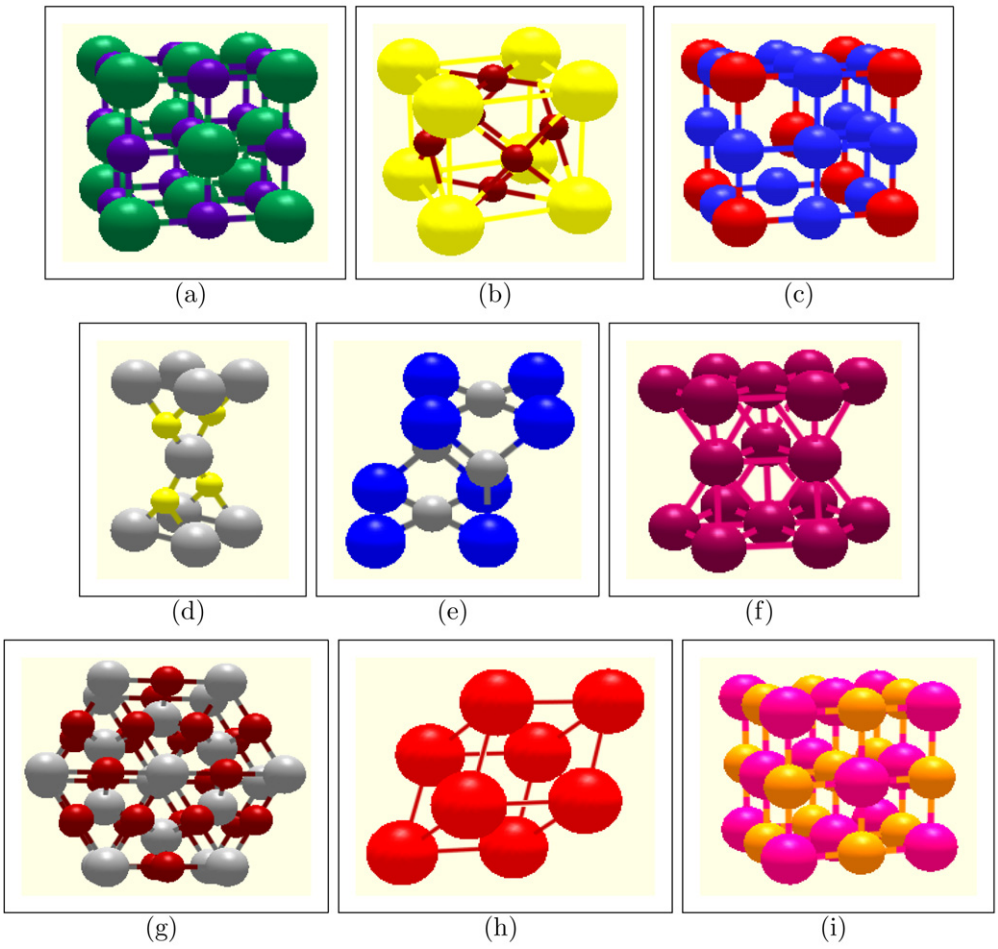

Fig. 3. Unit cells of crystals generated by *BilKristal*. (a) NaCl; (b) Cu_3Au; (c) La_2O_3; (d) PtS; (e) Al_3Ti; (f) Mg; (h) CoSn; (i) αHg; (j) TlF.

on the extracted primitive vectors. Therefore, the primitive vectors obtained with the noisy data are considered acceptable (see Table 1). The results of the basis vectors are quite similar to the primitive vector results. With the ideal data, exact basis vectors are obtained and with the noisy data acceptable distortions are observed (see Table 2).

4.2. *Space group identification*

The algorithm for identifying the space group is significantly affected by errors. Since the presence of errors is very important for this algorithm, three different tests are performed for each test material. In the first test, ideal data are used and *EPS* is set to a low value, 0.001. In the second test, ideal data and the default *EPS* values are used. Finally, in the third test, noisy data and default *EPS* values are used. Test results for some materials are given in Table 3.

The space groups are found correctly in all three tests for

- cubic structures: NaCl, La_2O_3, Cu_3Au,
- tetragonal structures: PtS, Al_3Ti, Ca_3SiBr_2,
- hexagonal structures: Mg, CoSn,
- trigonal structures: MgF_2.

None of these structures are a distorted form of a higher symmetry structure. Accordingly, these materials are in the highest symmetry form that their structures allow. There is no way that these materials can be confused with another higher symmetry structure even in the presence of a reasonable level on noise. Thus, their space groups are identified accurately. The space group of the trigonal structure, αHg, is also identified accurately in all three tests. However, errors may be observed with structures similar to αHg. The αHg structure can be considered a distorted simple cubic structure. Its unit cell can be considered as a simple cube with increased diagonal length. Accordingly, α, β and γ angles, which are the angles between each primitive vector pair, are all equal and smaller than 90 degrees. The αHg structure has different forms, depending on such angles. According to the environmental properties such as temperature and pressure, those angles change their values within some range. Accordingly, αHg's unit cell might become quite close to a simple cubic unit cell. In the test data, these angles are set to 70 degrees. However, if these angles are set to a value close to 90 degrees, it is quite possible to obtain a 221st space group in the second and third tests. This represents the simple cubic structure.

Some problems are observed with the TlF structure. TlF is a distorted version of NaCl. In other words, a high symmetry

Table 1
Primitive vectors of different material structures (actual, produced with ideal data input, and noisy data input)

Material	Vectors	Actual	Ideal data	Noisy data
NaCl	$\vec{R}_1$	[0.00 2.83 2.83]	[0.00 2.83 2.83]	[0.00 2.81 2.82]
	$\vec{R}_2$	[2.83 0.00 2.83]	[2.83 0.00 2.83]	[2.82 0.00 2.80]
	$\vec{R}_3$	[2.83 2.83 0.00]	[2.83 2.83 0.00]	[2.82 2.82 0.00]
Cu_3Au	$\vec{R}_1$	[3.14 0.00 0.00]	[3.14 0.00 0.00]	[3.13 0.00 0.00]
	$\vec{R}_2$	[0.00 3.14 0.00]	[0.00 3.14 0.00]	[0.00 3.13 0.00]
	$\vec{R}_3$	[0.00 0.00 3.14]	[0.00 0.00 3.14]	[0.00 0.00 3.13]
La_2O_3	$\vec{R}_1$	[−2.57 2.57 2.57]	[−2.57 2.57 2.57]	[−2.57 2.57 2.55]
	$\vec{R}_2$	[2.57 −2.57 2.57]	[2.57 −2.57 2.57]	[2.55 −2.56 2.57]
	$\vec{R}_3$	[2.57 2.57 −2.57]	[2.57 2.57 −2.57]	[2.60 2.55 −2.58]
PtS	$\vec{R}_1$	[1.48 0.00 0.00]	[1.48 0.00 0.00]	[1.46 0.00 0.00]
	$\vec{R}_2$	[0.00 1.48 0.00]	[0.00 1.48 0.00]	[0.00 1.47 0.00]
	$\vec{R}_3$	[0.00 0.00 3.29]	[0.00 0.00 3.29]	[0.00 0.00 3.28]
Al_3Ti	$\vec{R}_1$	[1.81 0.00 0.00]	[1.81 0.00 0.00]	[1.80 0.00 0.00]
	$\vec{R}_2$	[0.00 1.81 0.00]	[0.00 1.81 0.00]	[0.00 1.80 0.00]
	$\vec{R}_3$	[0.91 0.91 1.91]	[0.91 0.91 1.91]	[0.89 0.90 1.90]
Mg	$\vec{R}_1$	[0.86 −1.49 0.00]	[0.86 −1.49 0.00]	[0.83 1.48 0.00]
	$\vec{R}_2$	[0.86 1.49 0.00]	[0.86 1.49 0.00]	[−0.85 1.50 0.00]
	$\vec{R}_3$	[0.00 0.00 2.81]	[0.00 0.00 2.81]	[0.00 0.00 2.81]
CoSn	$\vec{R}_1$	[1.23 −2.13 0.00]	[−1.23 2.13 0.00]	[1.23 −2.12 0.00]
	$\vec{R}_2$	[1.23 2.13 0.00]	[1.23 2.13 0.00]	[1.23 2.12 0.00]
	$\vec{R}_3$	[0.00 0.00 4.02]	[0.00 0.00 4.02]	[0.00 0.00 4.02]
αHg	$\vec{R}_1$	[2.0 0.33 0.33]	[1.99 0.33 0.33]	[0.34 1.95 0.29]
	$\vec{R}_2$	[0.33 2.0 0.33]	[0.33 1.99 0.33]	[1.96 0.34 0.33]
	$\vec{R}_3$	[0.33 0.33 2.0]	[0.33 0.33 1.99]	[0.37 0.27 1.98]
TlF	$\vec{R}_1$	[0.00 1.17 1.08]	[0.00 1.17 1.08]	[0.00 1.17 1.06]
	$\vec{R}_2$	[1.11 0.00 1.08]	[1.11 0.00 1.08]	[1.11 0.00 1.07]
	$\vec{R}_3$	[1.11 1.17 0.00]	[1.11 1.17 0.00]	[1.11 1.14 0.00]
MgF_2	$\vec{R}_1$	[3.08 0.00 0.00]	[3.08 0.00 0.00]	[3.08 0.00 0.00]
	$\vec{R}_2$	[−1.53 2.69 0.00]	[−1.53 2.69 0.00]	[−1.51 2.69 0.00]
	$\vec{R}_3$	[0.01 −0.02 4.02]	[0.00 0.00 4.02]	[0.00 0.00 4.00]
Ca_3SiBr_2	$\vec{R}_1$	[4.49 0.00 0.00]	[0.00 4.49 0.00]	[0.00 4.48 0.00]
	$\vec{R}_2$	[0.00 4.49 0.00]	[4.49 0.00 0.00]	[4.49 0.00 0.00]
	$\vec{R}_3$	[−2.25 − 2.25 9.65]	[2.25 2.25 9.65]	[2.26 2.25 9.64]

All units are Å.

structure is converted into a low symmetry structure by small distortions. In the second and third tests, default *EPS* values are used to inform the *Analyzer* about the possibility of errors in the input data. The analysis is performed according to the specified error margin. This flexibility leads to matching the TlF structure with a higher symmetry group. Thus, the space group identified may not be reliable for materials that are distorted forms of a higher symmetry structure unless ideal data is used and the *EPS* parameter is set to a low value. If it is impossible to use ideal data and low *EPS* values, the user should manually check other lower symmetry space groups returned by the tool to ensure reliable results.

We also experimented with the two examples given in [6–8] for which ADDSYM finds the additional symmetry where FINDSYM failed in both cases. Although our system found the primitive vectors and basis vectors correctly, it could not identify the space group for these structures.

4.3. Error handling

The algorithms perform quite well in the presence of a reasonable level of coordinate errors. The noisy data used in the tests contain ±0.03 Å coordinate errors at each axis. This level of noise is sufficient to cover almost all of the cases that this framework would handle. But to assess the algorithms' error tolerance, the proposed framework is tested with further levels of coordinate errors. During the tests, calculated primitive vectors and basis vectors are examined. The NaCl structure is used as the test structure. The error margins ±0.2, ±0.4, ±0.6, ±0.8, and ±1.0 Å are used to generate the test data.

A vector may contain up to ±2*EPS* coordinate errors. Accordingly, any found primitive vectors and basis vectors should

Table 2
Basis vectors of different material structures (actual, produced with ideal data input, and noisy data input)

Material	Vectors	Actual	Ideal data	Noisy data
NaCl	$\vec{B}_1$	Na [0.00 0.00 0.00]	Na [0.00 0.00 0.00]	Na [0.00 0.00 0.00]
	$\vec{B}_2$	Cl [2.83 2.83 2.83]	Cl [2.83 2.83 2.83]	Cl [2.82 2.82 2.86]
Cu_3Au	$\vec{B}_1$	Au [0.00 0.00 0.00]	Au [0.00 0.00 0.00]	Au [0.00 0.00 0.00]
	$\vec{B}_2$	Cu [0.00 1.57 1.57]	Cu [0.00 1.57 1.57]	Cu [0.03 1.55 1.55]
	$\vec{B}_3$	Cu [1.57 0.00 1.57]	Cu [1.57 0.00 1.57]	Cu [1.55 −0.03 1.58]
	$\vec{B}_4$	Cu [1.57 1.57 0.00]	Cu [1.57 1.57 0.00]	Cu [1.57 1.54 −0.01]
La_2O_3	$\vec{B}_1$	La [0.00 0.00 0.00]	La [0.00 0.00 0.00]	La [0.00 0.00 0.00]
	$\vec{B}_2$	O [2.57 0.00 0.00]	O [2.57 0.00 0.00]	O [2.56 0.01 −0.01]
	$\vec{B}_3$	O [0.00 2.57 0.00]	O [0.00 2.57 0.00]	O [0.02 2.54 0.01]
	$\vec{B}_4$	O [0.00 0.00 2.57]	O [0.00 0.00 2.57]	O [0.01 0.03 2.53]
PtS	$\vec{B}_1$	Pt [0.00 0.74 0.00]	Pt [0.74 0.00 1.65]	Pt [0.00 0.00 0.00]
	$\vec{B}_2$	Pt [0.74 0.00 1.65]	Pt [0.00 0.74 0.00]	Pt [0.75 0.76 1.65]
	$\vec{B}_3$	S [0.00 0.00 0.82]	S [0.00 0.00 2.47]	S [−0.02 0.72 0.84]
	$\vec{B}_4$	S [0.00 0.00 2.47]	S [0.00 0.00 0.82]	S [0.01 0.71 2.46]
Al_3Ti	$\vec{B}_1$	Ti [0.00 0.00 0.00]	Ti [0.00 0.00 0.00]	Ti [0.00 0.00 0.00]
	$\vec{B}_2$	Al [0.91 0.91 0.00]	Al [0.91 0.91 0.00]	Al [0.92 0.93 −0.01]
	$\vec{B}_3$	Al [0.91 0.00 0.95]	Al [0.91 0.00 0.95]	Al [0.90 0.02 0.96]
	$\vec{B}_4$	Al [0.00 0.91 0.95]	Al [0.00 0.91 0.95]	Al [0.00 0.89 0.92]
Mg	$\vec{B}_1$	Mg [0.86 0.50 0.70]	Mg [0.00 0.00 0.00]	Mg [0.00 0.00 0.00]
	$\vec{B}_2$	Mg [0.86 −0.50 2.10]	Mg [0.86 0.50 1.40]	Mg [−0.03 1.01 1.41]
CoSn	$\vec{B}_1$	Sn [0.00 0.00 0.00]	Sn [0.00 0.00 0.00]	Sn [0.00 0.00 0.00]
	$\vec{B}_2$	Sn [1.23 0.71 2.01]	Sn [0.00 1.42 2.01]	Sn [1.27 0.71 2.01]
	$\vec{B}_3$	Sn [1.23 −0.71 2.01]	Sn [0.00 2.84 2.01]	Sn [1.24 −0.69 2.03]
	$\vec{B}_4$	Co [0.62 −0.36 0.00]	Co [−0.62 1.07 0.00]	Co [0.63 −1.07 − 0.02]
	$\vec{B}_5$	Co [0.62 0.36 0.00]	Co [0.62 1.07 0.00]	Co [0.62 1.05 −0.01]
	$\vec{B}_6$	Co [1.23 0.00 0.00]	Co [0.00 2.13 0.00]	Co [1.23 0.03 −0.03]
αHg	$\vec{B}_1$	Hg [0.00 0.00 0.00]	Hg [0.00 0.00 0.00]	Hg [0.00 0.00 0.00]
TlF	$\vec{B}_1$	Tl [0.00 0.00 0.00]	Tl [0.00 0.00 0.00]	Tl [0.00 0.00 0.00]
	$\vec{B}_2$	F [0.00 0.00 1.08]	F [0.00 0.00 1.08]	F [0.03 0.06 1.06]
MgF_2	$\vec{B}_1$	Mg, [0.00 0.00 0.00]	Mg, [0.00 0.00 0.00]	Mg, [0.00 0.00 0.00]
	$\vec{B}_2$	F, [0.66 0.33 0.23]	F, [0.66 0.33 0.23]	F, [0.67 0.32 0.26]
	$\vec{B}_3$	F, [0.33 0.67 0.77]	F, [0.33 0.66 0.77]	F, [0.32 0.65 0.78]
Ca_3SiBr_2	$\vec{B}_1$	Si, [0.00 0.00 0.00]	Si, [0.00 0.00 0.00]	Si, [0.00 0.00 0.00]
	$\vec{B}_2$	Ca, [0.51 0.51 0.00]	Ca, [0.50 0.50 0.00]	Ca, [0.50 0.51 0.00]
	$\vec{B}_3$	Br, [0.67 0.67 0.34]	Br, [0.68 0.67 0.66]	Br, [0.68 0.67 0.66]
	$\vec{B}_4$	Br, [0.33 0.33 0.66]	Br, [0.33 0.33 0.34]	Br, [0.33 0.34 0.34]
	$\vec{B}_5$	Ca, [0.16 0.16 0.32]	Ca, [0.16 0.15 0.68]	Ca, [0.16 0.16 0.69]
	$\vec{B}_6$	Ca, [0.84 0.85 0.68]	Ca, [0.84 0.84 0.32]	Ca, [0.83 0.84 0.32]

All units are Å.

Table 3
The results of the space group identification stage. The default *EPS* value is 0.05

	Actual space group number	Ideal data $EPS = 0.001$	Ideal data default *EPS*	Noisy data default *EPS*
NaCl	225	225	225	225
Mg	194	194	194	194
αHg	166	166	166	166
TlF	69	69	138	138

be considered acceptable if the coordinate distortions are all less than $\pm 2EPS$. For input data containing ± 0.6 Å or less errors, the highest coordinate distortions observed on primitive vectors are all smaller than $2EPS$ (see Table 4). For these input data, the basis vector results are also similar. The maximum distortions of fractional coordinates of basis vectors are all within acceptable limits (see Table 5). For input data containing ± 0.8 Å or higher errors, the tool does not produce very accurate results.

In general, the error tolerance of the framework is quite good. The tests show that even with the data containing ± 0.6 Å margin of error, acceptable results are obtained for the NaCl structure. Considering that Na's radius is 1.16 Å and Cl's radius is 1.67 Å in NaCl, ± 0.6 Å error in the coordinates can be considered a fairly large value.

Table 4
Primitive vector results of error handling tests (Cartesian coordinates are used and all units are Å)

Noise level	$\vec{R}_1$	$\vec{R}_2$	$\vec{R}_3$	Maximum distortion
±0.2	[2.63, −2.86, 0.00]	[2.76, 0.00, 2.77]	[0.00, −2.78, 2.80]	0.20
±0.4	[2.56, 2.79, 0.00]	[−0.17, 2.79, 2.66]	[2.69, 0.12, 2.78]	0.28
±0.6	[2.78, 2.77, −0.10]	[0.00, 2.85, 2.77]	[2.84, −0.06, 2.80]	0.10
±0.8	n.a.	n.a.	n.a.	n.a.
±1.0	n.a.	n.a.	n.a.	n.a.

Table 5
Basis vector results of error handling tests (fractional coordinates are used and noise level units are Å)

Noise Level	$\vec{B}_1$	$\vec{B}_2$	Maximum distortion
±0.2	Na [0.00, 0.00, 0.00]	Cl [0.55, 0.52, 0.48]	0.05
±0.4	Na [0.00, 0.00, 0.00]	Cl [0.55, 0.51, 0.55]	0.05
±0.6	Na [0.00, 0.00, 0.00]	Cl [0.50, 0.42, 0.55]	0.08
±0.8	n.a.	n.a.	n.a.
±1.0	n.a.	n.a.	n.a.

4.4. Performance evaluation

In this section, first the complexity analysis is provided and then the dissected runtime performance of each stage is evaluated.

4.4.1. Complexity analysis

The complexity of reading input data and inserting the records of atoms into the octree structure is $O(N \log(N))$ since each insertion has a complexity of $O(\log(N))$, where N is the number of atom records.

In the grouping algorithm, the matching volume of each atom to be analyzed is calculated. Since the crystal data is usually homogeneous, the number of atoms in each matching volume is constant. If we call this constant M, retrieving the matching volume of an atom has a time complexity of $O(\log(N) + M)$, since accesses to the boundaries of the matching volume can be done in logarithmic time and maintaining the linked list of output can be done in linear time. Since matching volumes of the groups are indexed with the octree structure, checking if two matching volumes are identical can be done in $O(M \log(M))$ time. The atoms can be inserted into the atom list of the corresponding group in constant time. However, if the atom does not match any group, a new group is created. The formation of a new group requires copying an atom's matching volume into a group's matching volume and indexing it; this can be done in $O(M + M \log(M))$ time. If G is the number of groups generated by the grouping algorithm and A is the number of atoms that are to be analyzed, the time complexity will be $O(A \log(N) + AM + GAM \log(M) + GM + GM \log(M))$. The first part, $A \log(N) + AM$, represents the complexity of obtaining matching volumes of every atom to be analyzed. The second part, $GAM \log(M)$, shows the matching volume comparison of every atom with every group. Finally, the last part, $GM + GM \log(M)$, represents the group formation times for each group. In the worst case, the number of groups is equal to N; this is the case when every atom defines a group and the complexity will be $O(NAM \log(M))$. Most of the time, however, a small number of groups will be generated. Thus, for most cases, G will be a small constant. Accordingly, the complexity reduces to $O(A \log(N) + AM \log(M))$.

If A is the number of processed atoms and G is the number of groups, a total of $A - G$ vectors are extracted by the vector set generation algorithm. Most of those vectors are eliminated since they are longer than a predefined threshold. However, the order, $O(A)$, is still correct considering G is a small constant. Whenever a vector is extracted, it is checked to see if it is a duplicate of a previously found vector. A range search performed on the vector set with an error margin of $4EPS$ at each axis gives the duplicate vectors if they exist. Since a range search in an octree has a logarithmic time complexity, the overall complexity is $O(A \log(A))$.

The complexity of the algorithm for filtering out redundant vectors is $O(V^2)$ where V is the number of vectors produced in the vector set generation algorithm. This leads to a worst case complexity of $O(A^2)$, since V can be as much as $A - G$. In general, the vector set generation produces much fewer vectors than A^2. Accordingly, the runtime performance can be expected to be reasonable, even though the worst-case complexity is quadratic.

The algorithm for calculating primitive vector alternatives first sorts the vector list. Sorting has a complexity of $O(V \log(V))$, where V is the number of vectors. The number of candidate vectors for a vector triplet, P, is relatively small. Accordingly, the total number of primitive vector triplets to be tested is limited to $O(P^3)$. The maximum number of primitive vector triplets is $P \times (P - 1) \times (P - 2))$. Thus, the sorting of primitive vector sets has a complexity of $O(P^3 \log(P))$. Therefore, the overall time complexity of the algorithm is $O(V \log(V) + P^3 V + P^3 \log(P))$. Since V can be at most $A - G$, the complexity can be rewritten as $O(A \log(A) + P^3 A)$.

In the clustering algorithm, the groups are sorted first according to the number of atoms in their lists. Then, a cluster is created for every atom in the least crowded group. This operation takes $O(G_0)$ time, where G_i represents the number of atoms belonging to the ith group. After that, every group is processed concurrently. To process a group, a direction vector is calculated. Finding the direction vector requires checking every cluster–atom pair and it can be done in $O(G_0 G_i)$ time, since the number of clusters can be as much as G_0. Then, an atom is found for each cluster whose distance to the cluster center is equal to the direction vector. This operation also has $O(G_0 G_i)$ time complexity. The overall time complexity for the clustering

Table 6
The execution times of the stages of the framework for different materials (execution times are milliseconds)

Material	Total # of points	# of points to process	Reading input	Indexing input	Grouping algorithm	Extracting vectors	Filtering out vectors	Finding primitive vectors	Clustering	Finding space group	Finding basis vectors
NaCl	3371	1331	130	0	40	10	0	20	10	141	0
La_2O_3	3375	2197	140	0	81	10	0	10	30	180	20
Cu_3Au	7813	4631	181	20	300	10	10	80	120	71	70
PtS	36396	17598	500	121	12958	20	10	2484	1612	30	741
Al_3Ti	41513	20213	571	110	16494	30	0	3204	2083	20	792
Mg	17752	8954	311	30	2333	20	0	2273	641	20	361
CoSn	17107	8979	300	50	2043	10	0	181	340	50	160
αHg	8865	4573	201	20	330	10	0	2053	0	20	0
TlF	46397	22707	621	140	14020	40	20	5729	5147	161	2033

algorithm is $O(G_i) + O(\sum_{i=1}^{G-1} G_0 G_i)$, which is equivalent to $O(G_0(A + 1 - G_0))$. This expression has its maximum value when $G_0 = (A + 1)/2$. Thus, the worst case complexity is $O(A^2)$. The complexity of the algorithm for finding basis vectors is the same as the complexity of the clustering algorithm.

The space group identification algorithm checks whether a symmetry operation is supported by a point. This process consists of two parts. The first part involves applying the symmetry operation and obtaining the translated point. In the second part, the coordinates of the translated point are checked for an identical point. The first part has constant time complexity while the second part has the logarithmic time complexity since it require a point search. There are 273 space groups to test and the number of symmetry operations of each space group is constant. Since the number of test points are also bounded by a constant value, the complexity of the algorithm for identifying the space group is $O(\log(N))$. The number of vector sets that are tested in order to find a vector set for each lattice class is also constant. The part that finds those vector sets has a constant time complexity. Thus, the complexity of this algorithm can be written as $O(\log(N))$.

Overall complexity can be found by adding the complexities of each stage. The overall complexity is

$$C = O(N) + O\big(N\log(N)\big) + O\big(A\log(N) + AM\log(M)\big) + O(A^2 + P^3A) + 2 \times O(A^2) + O\big(\log(N)\big),$$

where N represents the number of points in the octree structure, A represents the number of points to analyze, M represents the average number of points in the matching volume and P represents the maximum number of vectors that forms a primitive vector to test. The expression can be simplified as

$$C = O\big(N\log(N) + AM\log(M) + A^2 + P^3A\big).$$

4.4.2. Dissected execution times for different stages

Table 6 gives the execution times of each stage obtained with ideal test data. The results obtained with noisy data are similar to the ones obtained with ideal data. Figures are given in milliseconds.

The first two columns show the number of atom coordinates in the input data. The first row shows the number of all atoms in the input data and the second row shows the number of all atoms that are to be analyzed. The third column shows the time required to read the input data and the fourth column represent the time spent indexing the data by using the octree structure.

The fifth column shows the time spent grouping identical atoms. The grouping algorithm is based on comparing matching volumes in order to determine each atom's group. For the materials that are denser in terms of the number of atoms per volume, the grouping algorithm requires more time.

The most dominant term in the grouping algorithm's runtime complexity is $O(GAM\log(M))$, where G represents the number of groups, A represents the number of atoms to process and M represents the number of atoms in each atom's matching volume. Since the volumes of the crystal segments used in the analysis are equal for every test material, the values in the first column are roughly proportional to the number of atoms per volume of each material. Thus, they are proportional to the M value. As shown in Table 6, materials with high atom count values also have much higher execution times for the grouping algorithm. Since there are other factors in the algorithm's runtime complexity, the relationship between atom count and execution time is not quite clear. In general, the grouping process can be considered sufficiently fast. Nevertheless, this stage is the most time consuming stage for some cases.

The sixth, seventh and eighth columns show vector operations, namely extracting vectors, filtering out redundant vectors and calculating primitive vector alternatives, respectively. The stage of extracting vectors turns out to be quite fast. The stage of filtering out redundant vectors is even faster although its worst-case complexity is higher than that of extracting vectors. In most cases, its runtime is not measurable. The stage of calculating primitive vector alternatives is the dominant time-consuming vector operation. Basically, it checks every vector triplet obtained from the vector set to see if their integer combinations could produce all other vectors in that vector set. In order to reduce the runtime complexity, we limit the number of vectors that can be used to form a primitive vector alternative. This reduces execution times to reasonable levels.

The ninth column shows the clustering times. For some materials, such as αHg, the clustering is quite fast. During the clustering phase, each atom of the first group defines a cluster. Since the αHg structure has only one group, its clustering is trivial. The clustering process checks all atoms of the processed group against each cluster. This is done for every group except the first one. Since having a lot of groups reduces the number of clusters, it decreases the processing time. For CoSn, which has

6 groups, the clustering times are much smaller than those of other structures. In general, if the number of processed atoms is high, the clustering time tends to be higher. However, if the number of groups is high, the clustering time drops significantly.

The tenth column shows the processing times for the stage of identifying the space groups. This stage does not significantly depend on the input material. The runtime complexity of this part is logarithmic in the number of atoms. All materials should have similar processing times but there are some optimizations that change this situation. Finding one symmetry operation that is not supported by the crystal structure is enough to conclude that the crystal structure does not support the space group and it is unnecessary to check the rest of the symmetry operations. As the table implies, the materials whose processing times are higher are generally the higher symmetry structures.

The eleventh column shows the runtime of the stage that finds basis vectors. Since this part is quite similar to the clustering, execution times are similar to those of the clustering stage.

For most cases, the dominant time-consuming part is the grouping stage. The parts for calculating primitive vector alternatives and clustering also demand significant processing times for some input materials. However, it is not possible to generalize time requirements for the algorithms since they depend significantly on input characteristics. In general, the runtime performance of the whole analysis is satisfactory: under 30 seconds and accurate for any input data.

The memory requirement of the system is quite reasonable. The *Analyzer* program uses under 40 MB of memory with any input data. The *UserInterface* program requires about 20 MB of memory. The *VisualizationTool* program requires about 10 MB of memory. Thus, the overall system requires at most 60 MB; this is quite reasonable with today's computers.

4.5. The program files

The program archive file contains the source code, a readme file, executables, a user manual and input test data (organized in 11 folders and 186 files). The source code is located in the three directories: Analyzer, UserInterface, and VisualizationTool. They contain the following files.

BilKristal-Analyzer

```
Header Files
--------
Cluster.h, Groups.h, OglEngine.h,
SpaceGroup.h, Structures.h, UserInput.h,
Vectors.h, math_funcs.h

C Source Codes
---------
Cluster.c, Groups.c, OglEngine.c,
SpaceGroup.c, Structures.c, UserInput.c,
Vectors.c, chull.c, math_funcs.c
```

BilKristal-UserInterface

```
Header Files
--------
AVForm.h, AddNewBasisVector.h, AnimateForm.h,
Form1.h, NewCPForm.h, NewPriVectForm.h,
OriginSelecter.h, PriVectList.h, ResultForm.h,
ResultForm1.h, RunForm.h, Structures.h,
VTParameters.h, resource.h, stdafx.h

C Source Codes
---------
AVForm.cpp, AddNewBasisVector.cpp,
AnimateForm.cpp, AssemblyInfo.cpp,
Form1.cpp, NewCPForm.cpp, NewPriVectForm.cpp,
OriginSelecter.cpp, PriVectList.cpp,
ResultForm.cpp, ResultForm1.cpp, RunForm.cpp,
VTParameters.cpp, stdafx.cpp
```

BilKristal-Visualizer

```
Header Files
--------
IOHandler.h, Materials.h, OglEngine.h,
Structures.h, UserInput.h, math_funcs.h

C Source Codes
---------
IOHandler.c, Materials.c, OglEngine.c,
Structures.c, UserInput.c, chull.c,
math_funcs.c
```

4.6. Discussion

The results, the performance evaluation and the error analysis show that the proposed framework and the algorithms that are used in different stages are quite successful. Correct results are always obtained with ideal data and runtime performances are good. However, with noisy data some problems are observed. The primitive vectors and the basis vectors produced with noisy data contain small errors. These errors are in an acceptable range considering the margin of error in the input data.

The tool is semi-automatic in the sense that it requires input from the user during the analysis. Selection of the primitive vector sets by the user is based on the order produced by the algorithm for finding primitive vector alternatives. This order is much better for ideal data than noisy data. Another problem with noisy data is the possibility of obtaining invalid primitive vector sets. However these invalid primitive vectors do not cause any significant problem since the user can identify them quite easily. Finally, the most important problem with noisy data is the incorrect space group results. During the tests, we observed this with only the TlF structure. However, this problem can be avoided by setting the *EPS* parameter to a low value.

5. Conclusion

The aim of this work is to extract pattern information from crystal structures by using atomic coordinates. Determining

pattern information for crystal structures, such as primitive vectors, basis vectors and space group, has great importance in crystallography, chemistry and material sciences; the physical behavior of materials are directly related to these crystal parameters. This work provides a tool that can help scientists to identify and classify crystal structures. This work also provides a crystal visualization tool that allows scientists to observe crystal structures in a 3D environment.

Some approaches that are used in other areas such as 3D shape matching and pattern recognition are adapted to this problem and some new approaches are devised. There are two main challenges in proposing the framework. The first challenge is to obtain accurate results while achieving reasonable runtime performance. To this end, we make several critical decisions, such as limiting inputs in intermediate stages. We also propose several computational optimizations. The second challenge is the handling of erroneous input data. To overcome this problem, we re-write the algorithm for finding basis vectors and modify other algorithms.

This framework is tested with several data showing various characteristics. The test data are generated by using real crystal parameters, with different error levels. Experimental results and error analysis show that the framework can give accurate results even in the presence of reasonable levels of errors. Runtime performance is also quite satisfactory.

The implemented software can also be used to visualize crystals. The visualization part presents several features, such as defining multi-cells, defining cut planes, and animating crystals for 3D visualization.

Acknowledgements

The work described in this paper is supported by the Scientific and Technical Research Council of Turkey (TÜBİTAK) under Project Code 105E065. We are grateful to Miss Kirsten Ward for proofreading the paper.

References

[1] C. Hammond, Basics of Crystallography and Diffraction, Oxford University Press, NY, 1997.
[2] M. Ladd, Symmetry in Molecules and Crystals, Ellis Horword Ltd., West Sussex, 1989.
[3] R.W. Grosse-Kunstleve, N.K. Sauter, N.W. Moriarty, P.D. Adams, The Computational crystallography toolbox: crystallographic algorithms in a reusable software framework, Journal of Applied Crystallography 35 (2002) 126–136.
[4] D. Hatch, H. Stokes, Practical algorithm for identifying subgroups of space groups, Physical Review B 31 (1985) 2908–2912.
[5] H. Stokes, D. Hatch, FINDSYM-program for identifying the space-group symmetry of a crystal, Journal of Applied Crystallography 38 (2005) 237–238.
[6] ADDSYM, http://www.csb.yale.edu/userguides/graphics/pluton/html/pl000401.html, 1988.
[7] Y.L. Page, Computer derivation of the symmetry elements implied in a structure description, Journal of Applied Crystallography 20 (1987) 264–269.
[8] Y.L. Page, MISSSYM1.1—a flexible new release, Journal of Applied Crystallography 21 (1988) 983–984.
[9] A. Hanneman, R. Hundt, J. Schön, M. Jansen, A new algorithm for space-group determination, Journal of Applied Crystallography 31 (1998) 922–928.
[10] R. Hundt, J. Schön, A. Hanneman, M. Jansen, Determination of symmetries and idealized cell parameters for simulated structures, Journal of Applied Crystallography 32 (1999) 413–416.
[11] R. Sayle, Molecular visualization freeware (Protein Explorer, Chime and Rasmol), available at http://www.umass.edu/microbio/rasmol/, 2000.
[12] Crystalmaker Software Limited, Crystalmaker software: Crystal and molecular structures visualization and diffraction, available at http://www.crystalmaker.co.uk/, 2006.
[13] A. Lin, The MOE crystal builder, 2005.
[14] M. Pavel, Fundamentals of Pattern Recognition, Marcel Dekker Inc., New York, 1989.
[15] S. Rusinkiewicz, M. Kazhdan, T. Funkhouser, Symmetry descriptors and 3D shape matching, in: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP'04), 2004, pp. 115–123.
[16] H. Samet, The quadtree and related hierarchical data structures, ACM Computing Surveys 16 (2) (1984) 187–260.
[17] J.R. Bowser, Inorganic Chemistry, Brooks/Cole, Pacific Grove, CA, 1993.
[18] W. Reinhardt, E. Clementi, D. Raimondi, CPK atomic radii, Phase Transitions: A Multinational Journal 38 (1963) 2686.
[19] C. Prewitt, R. Shannon, Effective ionic radii in oxides and fluorides, Acta Crystallographica, Section B, Structural Crystallography and Crystal Chemistry 25 (5) (1969) 925–946.
[20] V. Indenbom, B. Vainshtein, V. Fridkin, Structure of Crystals, third ed., Springer-Verlag, Berlin, 1995.
[21] D. Schwarzenbach, Crystallography, John Wiley and Sons Ltd., West Sussex, England, 1996.
[22] E. Okuyan, Pattern information extraction from crystal structures, Master's thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, available at http://www.cs.bilkent.edu.tr/techreports/2005/BU-CE-0511.pdf, 2005.
[23] D. McKie, C. McKi, Essentials of Crystallography, Blackwell Scientific Publications, 1986.
[24] J.P. Mato, A. Kirov, C. Capillas, S. Ivantchev, E. Kroumova, M. Aroyo, H. Wondratschek, Bilbao crystallographic server: Useful databases and tools for phase-transition studies, Phase Transitions: A Multinational Journal 76 (1) (2003) 55–70.
[25] T. Hahn, International Tables for Crystallography, D. Reidel Pub. Co, Norwell, MA, 1987.
[26] Naval Research Laboratory, Crystal lattice structures, Center for Computational Materials Science, 1995.