Posgen User Guide

D. Haley

1 Introduction

Posgen is a command-line only tool for the manipulation and creation of "POS" (position) files from Atom Probe Tomography (APT). These files contain a list of x,y,z and mass-to-charge data, which are in the form of a 3D point cloud (a series of points in space). Using *posgen* a wide variety of "operations" can be performed, where an Operation is a small subroutine that work on theis point cloud. Operations in *posgen* include geometric operations (cutting, scaling, rotating), compositional (obtain compositions or profiles, drop unranged data), file format conversions (pos↔tapsim, pos↔text), data synthesis (crystal generation), statistical analyses (clustering), etc. The program is primarily aimed at users who undertake data analysis of atom probe datasets, whilst minimising the level of programming they need to undertake.

Each operation can be connected to another allowing for a practically infinite number of combinations. By joining operations together, new datasets can be created or queried, in a rapid fashion. *Posgen* is designed to be fast, and to save having to re-write yet another data analysis tool. You can replace many complex scripts with simple *posgen* calls. For an example of using *posgen*, see the Example session6.1.

2 Copyright

Posgen is free software - you can copy, modify and distribute the program as you see fit, with no charge, within the constraints of the GNU General Public Licence v3.0. Similarly, this document is copyright under the Creative Commons 4.0 Attribution-ShareAlike licence (CC by-sa-4.0 BY: ⓢ). Information on how to locate the full licence terms is available in Appendix 8.3. If you have paid for this software, please request a refund and notify the Authors.

3 Installing

Posgen is provided as source-code for most platforms. You need to follow your platforms guide to installing a compiler, and building the program. For the latest instructions on how to install *posgen*, it is recommended to refer to the website `http://apttools.sourceforge.net/documentation.html`.

4 Questions

- *Why would I use this program?* – The program is designed to speed-up data processing on your own. Without this program, when performing calculations on your own data files, you would need to write your own programs by hand. Here, you can just use *posgen* to do it - whether you want to rotate, scale, random relabel, or perform any other common APT operation. *Posgen* is fully automatable, meaning you can perform the same operation (e.g. composition measurements) across many datasets. In short **Posgen will save you time**.

- *What operating systems does* posgen *work with?* – You can use any "POSIX" compatible operating system, such as Linux and OSX. Additionally, windows is supported.

- *Is there a graphical interface for* posgen*?* – At this time there is no graphical interface for posgen. It is a command-line only tool. You must write a script, in "XML" format - and then give it to *posgen* to process.

- *How do I view the output?* — you can view the output using any program that can open `.pos` files. If you don't have your own favourite, you can use the open-source *3Depict* program to view and manipulate the output[1] .
- *I'm getting an error :* `I/O error : failed to load external entity "posscript.dtd"` – This "DTD" file contains the list of available operations for posgen, and the syntax for each. The file is normally shipped with *posgen* and is required for the program to function correctly. Either put this file in the same path (folder) as you are currently using, or have your administrator install it into your XML catalogue, or set the `XML_CATALOG_FILES` environment variable [?].

5 Help

Whilst this document attempts to be as complete a reference as possible, it cannot fully answer all questions that one might have when using the software. Examples for most operations can be found in the `examples/` folder, or online at `http://sourceforge.net/p/apttools/posgen/code/ci/default/tree/examples/`.

Additional help is available online at the website forums [2]. If posting a problem, please try to be as specific as you can regarding what you are trying to do, so we can assist as quickly and accurately as possible. If you think you have found a bug, please also use the forums to notify us - even if you are not entirely sure, and we will attempt to resolve the problem as soon as possible. If you have an operation you think could be implemented, please contact us and we can identify whether or not this can be included in *posgen*.

6 Quick start

This document forms a technical reference for the suage of *posgen*. In this section the program usage is demonstrated, with worked examples of analyses.

6.1 Example session

When running *posgen*, the command line interface is used. Without any additional "arguments", the program runs as in the following:

```
Version: 0.0.1 (revision 226:11ebe5dbeff6) Dec 2013
USAGE : posgen [-pos|-text|-tapsim] xmlfile
```

The program states that it can be run in several ways, such as `posgen someXMLFile.xml`, or `posgen -text someXMLFile.xml`. Using an example from the `examples/` folder, we obtain the following ($ indicates an interactive command line, i.e. where the command has been typed in):

[1]`https://threedepict.sourceforge.net` - Disclosure: this program is also developed by the *posgen* authors
[2]`http://sourceforge.net/p/apttools/discussion/general/`

```
$ posgen examples/cubicTest.xml
Parsing File: examples/cubicTest.xml
Running first-pass checks...OK
Processing...
Processing node: cubic  (Line 6) ...Done


Parse successful
Final number of generated atoms:1670625
output mode not specified -- won't create final output posfile.
Possible options are:
        -pos : Pos file generation
        -text : text file generation
        -tapsim : tapsim binary file generation
```

The program first examines the input file `examples/cubicTest.xml`, and after some initial checking, no errors are found in the file (indicated by `Running first-pass checks...OK`). The program then intially starts with no data, and rather generates the data as a cubic lattice, as specified in the example. Once generated, the end of the input file is reached, and no further oeprations are done. The number of atoms at the end of the dataset is reported, however as no option for output is given, the program exits.

Using the following command, we request that the result be printed as text. This can result in a very large output.

```
$ posgen -text examples/cubicTest.xml
0        0        0        1
0        0        0.405        1
0        0        0.81        1
0        0        1.215        1
...
29.97        29.97        118.665        1
29.97        29.97        119.07        1
29.97        29.97        119.475        1
29.97        29.97        119.88        1
```

This output can be saved to a file, in either pos, tapsim or text format. Here we save the data as pos[3].

```
$ posgen -pos  examples/cubicTest.xml  > outputFile.pos
```

The resultant file appears as in Figure 1 - the cubic lattice is visible in the inset, and the overall bounding box for the dataset is visible from the main figure :

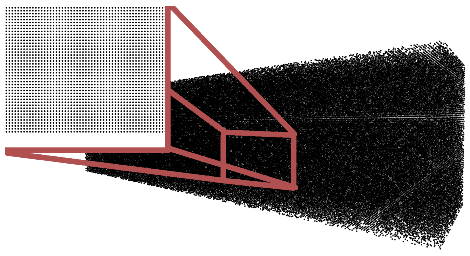

Figure 1: Cubic dataset generated from cubicTest example

[3]Saving to binary formats under windows may result in a corrupted file. Windows users should use the internal save routines, e.g. `possave`

7 Operations

This section outlines the various "operations" that are provided by *posgen*. Each operation allows for the manipulation of datasets in a specific ways, depending on the operation type. An *Operation* takes in the data from the previous *operation*, forming a chain 2. In this section, and for each operation, a short example of the tag usage itself is given with some brief details about what the operation does. Complete example files can be found in the *examples/* folder in *posgen* packages.

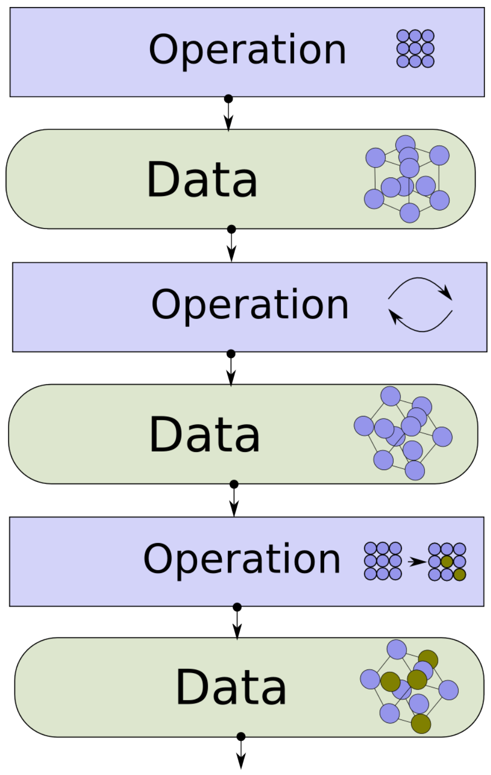

Figure 2: Chain of oeprations modifies data from preivous operation, linking these together allows for novel analyses or data volumes to be constructed

The operation files are in XML format [?], and are directly human readable – thus it is possible to hand-write these files using a text editor[4] As a short technical note, the XML syntax is largely specified in the DTD [?] file that is shipped with posgen. This DTD file enforces syntax and thus in the case of a conflict between this docuement and the DTD, the DTD takes precedence over the description of operations given here.

7.1 Information

7.1.1 posinfo

This operator details some information regarding the current pos file that is in-memory. It does not modify the current data in any way. Information such as the minimal axis-aligned bounding box, number of ions and basic per-species counts can be obtained using this operation.

Example : By using a `<posinfo/>` or `<posinfo rangefile="myrangefile.rng"/>` operation, the following output can be generated.

[4]We recommend to use an editor with syntax colouring, such as Notepad++`http://notepad-plus-plus.org/`.

```
Dumping pos information
Ion Count: 1213560
Pos File limits
Min: (0.000120085,1.32434e-05,1.14832e-05)        Max: (40,30,30)
```

When using `<posinfo/>` alone, the number of ions in the dataset (at that point in the script), and the bounding box (lower and upper corner) are shown. By providing an additional (and optional) rangefile, using this syntax : `<posinfo rangefile="myrangefile.rng"/>`, information about the number of atoms in each range is shown.

```
Dumping pos information
Per Ion :
A:        1213560
B:        0
Per Range:
A:
0.5 , 1.5        1213560

B:
1.5 , 2.5        0
```

7.2 Generators

The generators allow for the creation of lattices from several predefined tile sets. At time of writing, only a limited number of crystal systems are available.

7.2.1 cubic

The cubic operation creates a fixed simple-cubic lattice within a specified bounds, and of a specified mass. The lower bound for the generation of the crystal is (0,0,0), and the upper bound is given by the `<bounds x="`*someval*`" y="`*someval*`" z="`*someval*`"/>"`. The cube side-length is set via the "A" spacing value. Only a single mass can be assigned for all points, as there all points in a simple-cubic lattice are self-similar under translation (*i.e.* there is only one unique location in the crystal). The index must be (000).

As an example:

```
<cubic>
        <bound x="5" y="5" z="5"/>
        <spacing>
                <A value="0.405"/>
        </spacing>

        <atom index="(000)" mass="1"/>
</cubic>
```

7.2.2 fcc

This generator creates an FCC crystal lattice. Similarly to the cubic lattice, the tiling is performed in a rectilinear manner in 3D space. The masses for each unique position in the unit cell of the FCC crystal is specified via the `<atom ··· />`. The positions that can be specified are the (000), (202), (220), and (022) positions. The (000) is the corner position, and the remaining three are the three perpendicular faces. the lattice parameter and bounding colume can be set using the `<spacing>···</spacing>` and the `<bound x="`*value*`" y="`*value*`" z="`*value*`"/>` attributes.

As an example, the following is given:

```
<cubic>
        <bound x="5" y="5" z="5"/>
        <spacing>
                <A value="0.405"/>
        </spacing>

        <atom index="(000)" mass="1"/>
        <atom index="(022)" mass="5"/>
        <atom index="(202)" mass="3"/>
        <atom index="(220)" mass="17"/>
</cubic>
```

7.2.3 spatrand

This function generates spatially random data within a rectangular bounding box. The randomisation implies that there is no correlation between point's spatial positions. The only parameters that affects the spatial coordinates in the dataset are the density value, ρ, and the bounds for generation. The mass data assigned to each point can be set by adjusting the count and atom parameters.

A simple example snippet is as follows :

```
<spatrand>
        <bound x="40" y="30" z="30"/>
        <density value="33.71"/>

        <atom index="(000)" mass="21"/>
        <count value="1"/>

        <atom index="(000)" mass="22"/>
        <count value="2"/>
</spatrand>
```

This example generates a dataset from (0,0,0) to the (40,30,30) coordinate with a spatial density of 33.71[5]. Note that for this dataset he `index` attribute is ignored for this operation. The mass and count occur in pairs, and indicate the relative fraction of atomic masses to assign. The total sum of counts (in this case, $1 + 2 = 3$) is used to normalise each count, generating a composition that is approximately $\frac{1}{3}$ mass 21, and $\frac{2}{3}$ mass 22.

As the points are randomly generated, the number of *total* points is computed, and thus fixed between runs. The position and mass assignments (including composition fraction) will change randomly (as a binomial distribution) across repeated runs of the program. This can of course be locked by setting a fixed seed (Section 7.4.6) before this operation.

7.2.4 generic-rect

The `generic-rect` operation allows for a user-specifiable generic rectangular unit cell to be provided. Users can provide the atomic coordinates, and then the operation will proceed to tile the given unit cell across 3D space. Care must be taken that duplicate points are not generated (due to repeated tiling at walls), or that duplicate points are removed, such as via a `<cleanup/>` operation (Section 7.6.1).

To specify an individual atom's XYZ coordinate, in fractional values [0,1] are used, and the mass must also be supplied. The volume to which the fractional coordinates are in reference to is given by the `<spacing>....</spacing>` parameter. The `<bound x="`*dx*`" y="`*dy*` z="`*dz*`"/>`

An example which creates a simple rectangular lattice is given:

[5]There can be a slight difference between the requested and the generated density, due to the requirement that only whole points can be generated

```
<generic-rect>
   <bound x="2" y="2 z="2"/>

   <spacing>
     <A value="0.2"/>
     <B value="0.4"/>
     <C value="0.42"/>
   </spacing>

   <genericatom x="0.5" y="0.5" z="0.5"/>
</generic-rect>
```

7.3 Loading and Saving

7.3.1 posload

This operation allows for the loading of an arbitrary "position" file (POS-file). The POS file format is specified in Appendix 8.2.1, and contains XYZ and mass data. Previous data is appended to, but otherwise unaltered.

The only required parameter is the path to the file, in either relative or absolute form. URLs are not allowed. If the file cannot be found, processing will halt. An example is as follows:

```
<posload file="somefile.pos"/>
```

7.3.2 possave

This operation allows for the saving of the current in-memory data for *posgen* to a file. The only parameter required is the output filename. Upon completion, a complete pos file is generated at the specified location. **Warning:** *files will be overwritten without notice.*

An example is as follows:

```
<possave file="somefile.pos"/>
```

7.3.3 tapsimload

This operation allows for the loading of *TAPSIM*[**?**] formatted binary files. Only the input filename is required as a parameter. Existing data is appended to, but not cleared.

```
<tapsimload file="sometapsimfile.bin"/>
```

7.3.4 textload

This operation allows for the loading of delimited text files. Each file must have 4 columns of data, and be delimited using a chosen delimiter. The text data must have four columns, which are interpreted as x,y,z and mass respectively. Allowed column separators are given by `delimiter="`*someChars*`"/>`, and more than one delimiter may be given.

A sample dataset that can be loaded is given below, using the space character as the delimiter:

```
colA colB colC colD
1 2 3 4
1 2 3 1
2 5 2 1
```

This can be loaded with (assuming the data is stored in *somefile.txt*:

```
<textload file="somefile.txt" delimiter=" "/>
```

7.3.5 store

The store operation allows the current dataset to be saved using a specified name. Saved datasets can be later recalled using `<recall ···/>`. This can be useful if one, for example, wishes to perform repeated calculations, or alterations to a dataset, and wishes to save the original for subsequent processing. The dataset will be saved using the *name* parameter.

To save a dataset, use `<store name="`*somename*`"/>`. Optionally, the current data can be erased after the save by providing the *clear* parameter, e.g. `clear="true"`. An example is as follows:

```
<store name="uniquename" clear="true"/>
```

As a technical point, the data is saved to disk, and will no longer be stored in memory.

7.3.6 recall

This allows for previously named datasets to be restored back into the current state. The *name* parameter sets the stored data name. The current data can be erased prior to recalling by using the option `cleardata="true"`, and the name can be deleted (i.e. it cannot be recalled again) by using `clearname="true"`. The `clearname` and `cleardata` attributes are optional, and default to false. The recall operation can only be used if a previous `<store ··· />` operation has been given, using the specified *name* parameter. An example is as follows :

```
<recall name="uniquename"  clearname="false" cleardata="false"/>
```

7.4 Generic Data operations

7.4.1 clear

This operation erases the current data. Data stored using `<store/>` is not affected. There are no options. To use this operation, simply use the `<clear/>` element.

7.4.2 extractfield

This operation allows the output of a specific field (x,y,z or m) from the dataset to a text file. The output file is specified using the `file="`*somefile*`"`, This can be used, for example, to process data with other applications. To select the desired field to extract, the attribute `field=`*integer*`"/>`, where the digits $0, 1, 2, 3$ correspond to x,y,z and m respectively. Optionally a min and max value can be specified to limit the data that is extracted.

Note: *no warning will be given when writing to the file. Any existing file will be replaced.*

An example is as follows:

```
<extractfield file="somefile.txt" field="2" min="100" max="200"/>
```

7.4.3 massclip

The `<massclip />` operation allows for the cropping of a dataset by its mass-to-charge value. There are two parameters, `min` and `max`, which set the lower and upper limits for cropping. Only points whose masses lie within these limits (inclusive) will be retained.

7.4.4 mathtransform

Note: This is not available for windows based systems

This operation allows for the application of an arbitrary transformation equation to each field. The operation utilises the MuParser library. As such, the full syntax is given by the muParser documentation[6]. Each point is operated on independently, so "array" type operations (mean, std deviation, min, max) will not work. However, operations that utilise multiple fields will work (*e.g.* $x^2 + y^2$), as will

[6] `http://muparser.beltoforion.de/`

operations such as log and exp, trigonometric operations sin and cos are supported. As such, to assign a field, the variables x, y, z and m can be used. An equation such as $\mathrm{sqrt}(x)$ is valid. Other valid equations could include $x^2 + z^2 + \exp(m/10)$, or similar. The result of the equation is stored in the value specified in field, which can be $0, 1, 2$ or 3. These values correspond to xyz and mass respectively.

An example is as follows:

```
<mathtransform field="0" expression="sqrt(x^2+1)"/>
```

7.4.5 setfield

This can be used to set any desired field to a constant value. Valid fields are $0, 1, 2$ or 3. These values correspond to xyz and mass respectively. As an example, for setting the "y" field, the following is given:

```
<setfield field="1" value="12.4"/>
```

7.4.6 seed

This operation does not modify data directly - instead this sets the so-called *random seed* used when generating random sequences. The generated random sequences determine, for example, how shuffling is performed, how spatially random data is generated, and similarly controls any computationally "random" process.

As computers follow fixed algorithms, true randomisation is complicated, and outside the scope of this document. For here, it is sufficient to say that a single number - the "seed" sets the randomisation sequence for each time the program is run. Normally this is selected from the current clock time, and will change between runs. This operation allows overriding this seed value. Only operations *after* the seed are affected. To set the seed value, simply use `<seed value="123"/>`, where 123 is replaced with the desired seed. Often the actual value itself does not matter, as it may only be desired that the exact same operation is performed each time the program is run.

7.5 Geometric operations

7.5.1 arraysplit

This operation divides the dataset into rectangular regions, and will save each region as a new file. There are two modes possible. In the first mode, "dimension", a fixed width for the clip is used - the size of the rectangular region to be cropped is given, using the x dimension via `<x value="">` (and similarly for y and z). In the second "count" mode, the x,y,z values are interpreted as the number of cuts to make, and the dimensions of the cut are determined by the minimal box that contains all the current data.

To set the file names for output, the `<naming prefix="someprefix" suffix="blah"/>` values can be used. Whilst the prefix is mandatory, the suffix is optional, and defaults to ".pos". This will save the files in the form `someprefix-`*number*`.blah`. This should be used after setting the xyz values. After this field is set, optionally a `<count/>` value can be provided, to enable the "count" mode - without this, the "dimension" mode is used.

An example of using the "count" mode is given - this will cause the dataset to be split into 125 files:

```
<arraysplit>
        <x value="5"/>
        <y value="5"/>
        <z value="5"/>
        <naming prefix="split-"/>
        <count/>
</arraysplit>
```

7.5.2 clip

The clipping operation can be used to crop regions of the dataset based upon the x,y,z coordinate of points. Allowable clipping entities are:

- `sphere` - A simple sphere, given by origin and radius.

- `box` - an axis-aligned box, specified by two coordinates.
- `plane` - A 2D plane in 3D space, specified by normal (in spherical coordinates) and a point on the plane.
- `cylinder` - An arbitrarily oriented cylinder.
- `mesh` - Any closed triangle mesh, in GMSH format. Triangle orientation must be self-consistent, and mesh must be "manifold" (realisable as a "simple" 2D surface - holes are permitted).

Normally, the interior of the clipping object is retained, however optionally, each clip operation can be inverted, using the `invert` attribute. i.e. the intterior is clipped (and exterior retained) by using the `invert` attribute.

Each primitive is defined slightly differently. For the sphere primitive, the following example is given:

```
<sphere radius="0.5">
    <x value="1.0"/>
    <y value="1.2"/>
    <z value="-4.0"/>
</sphere>
```

A box can be described via the x/y/z for two corners:

```
<box>
    <x value="3.0"/>
    <y value="2.2"/>
    <z value="-8.0"/>

    <x value="4.0"/>
    <y value="3.2"/>
    <z value="-4.0"/>
</box>
```

A cylinder is given by specifying the origin, the axial vector (from centre to one end), and the radius of the cylinder:

```
<cylinder radius="1.4">
   <!-- centre -->
    <x value="3.0"/>
    <y value="2.2"/>
    <z value="-8.0"/>

     <-- axial vector -->
    <x value="4.0"/>
    <y value="3.2"/>
    <z value="-4.0"/>
</cylinder>
```

A mesh object must be a triangle mesh of GMSH form, and can be loaded as in the below. The mesh may be optionally recentred around the origin (by vertex-based weight).

```
<mesh file="somemesh.msh" recentre="false"/>
```

A full example is given here:

```
<clip invert="false">
  <cylinder radius="1.4">
    <!-- centre -->
      <x value="3.0"/>
      <y value="2.2"/>
      <z value="-8.0"/>

      <-- axial vector -->
      <x value="4.0"/>
      <y value="3.2"/>
      <z value="-4.0"/>
  </cylinder>
</clip>
```

7.5.3 geotransform

This operation allows for simple linear transformations of the x/y/z data, such as rotation, translation and scaling. The layout for each operation is slightly different, and is presented here separately - however these individual sub-operations must be present inside a `<geotransform>···</geotransform>`. Each sub-operation (rotation/translation/scaling) can be performed in sequence, without requiring a new `geotransform` operation. All transformations use a right-hand coordinate convention.

rotation : this operation performs a rotation around the origin, using vector-angle notation. Rotation angles are given in radiians, and the rotation is anti-clockwise as viewed from the positive end of the rotation axis. An example of rotation (90 degrees around the 111 axis) is given below:

```
<rotate>
        <x value="0.1"/>
        <y value="0.1"/>
        <z value="0.1"/>
        <angle value="1.50708"/>
</rotate>
```

translate : Translation can be performed in two ways, either by translating to the "centre-of-mass" (assuming each point has unit mass), or by specifying the translation vector. The translation vector will be added to each point in the dataset. The translation can be performed as follows:

```
<translate>
        <x value="0.1"/>
        <y value="20.1"/>
        <z value="15.1"/>
</translate>
```

scaling : Similarly to translation and rotation, scaling of the entire dataset can be performed using `scale`. Scaling is performed around the origin, and can be performed anisotropically, with differing X, Y and Z scaling.

```
<scale>
      <x value="1.1"/>
      <y value="-2"/>
      <z value="1"/>
</scale>
```

A complete example, with sequentialg translate, rotate and scale transformations, is given below:

```
<geotransform>
    <translate>
      <centreofmass/>
    </translate>

    <rotate>
      <x value="1.0"/>
      <y value="1.0"/>
      <z value="1.0"/>
      <angle value="1.50708"/>
    </rotate>

    <scale>
      <x value="2.0"/>
      <y value="2.0"/>
      <z value="2.0"/>
    </scale>
</geotransform>
```

7.5.4 noise

The noise operation allows for the addition of randomised positional noise to data. Noise operations are useful for determining how robust algorithms are in the presence of spatial uncertainty and can be used in many statistical analyses.

Either White or Gaussian noise can be added using this operation. For white noise, a random vector contained within the box $\pm dx, \pm dy, \pm dz$ is added to each point the data. For Gaussian noise, the x/y/z values refer to the standard deviation of a normalised Gaussian, from which the noise vectors will be computed. Unlike white noise, Gaussian noise will affect each point isotropically. Examples of white and Gaussian noise are given below:

```
<noise>
        <white>
                <x value="0.1"/>
                <y value="0.1"/>
                <z value="0.1"/>
        </white>
</noise>

<noise>
        <gaussian>
                <!-- values are one standard deviation -->
                <x value="0.1"/>
                <y value="0.1"/>
                <z value="0.1"/>
        </gaussian>
</noise>
```

7.5.5 posset

This operation performs a so-called "set" (as in set-theory[7]) operation on points $S(A, B)$. Currently supported operations are "subtract" and "intersect". For the purposes of the set operation, only the positions (the xyz, but not the "m" value) is considered. The current in-memory data is considered as the A set, and a specified file is considered to be the B set - the result of the set operation is stored in-memory.

[7] https://en.wikipedia.org/wiki/Set_(mathematics)

7.6 Data quality

7.6.1 cleanup

The cleanup operation performs a simple "tidying" of the dataset. At this time, the only modification made to the data is the removal of points whose x/y/z values overlap, within some tolerance. To use the cleanup operation, use `<cleanup removeduplicates="true" duplicatetol="`*someTolerance*`"/>`, where *someTolerance* is greater than zero.

7.6.2 surfremove

This operation removes the surface of a dataset, by a convex-hull reduction method. The convex hull is computed, and re-scaled around its centre of mass, such that the the specified removal distance is guaranteed to be removed. This can, for example, be combined with the `pointdensity` and `posset` commands to remove surface artefacts (at the cost of statistics).

The only parameter for the `surfremove` operation is the distance to be removed, and is specified as an attribute, i.e. `<surfremove value="`*somevalue*`"/>`.

7.7 Randomisation

7.7.1 randreplace

The `randreplace` operation performs random replacement of the "m" value (xyz-m) associated with each point. Each point can be randomly assigned a new mass, as given using the `mass` attribute. To set the fraction of points replaced, a `frac` option must be given. Note that the fraction is applied randomly (ie, each point has a probability of `frac` for reassignment) - thus the number of points that are reassigned is governed via a Binomial distribution, and can vary slightly from run to run. The `frac` value must lie in the range $[0, 1]$. An example is given below:

```
<randreplace frac="0.12" mass="3"/>
```

7.7.2 randrm

Random removal of ions from within the dataset - each ion is examined in the dataset and randomly erased. The only parameter is `fracrm="`*value*`"`, where *value* lies between zero and one, e.g. `<randrm fracrm="0.5"/>` will remove approximatey 50% of the points in the dataset.

Note that the random selection is performed per-ion, and the result will randomly change from run to run - both in which points will be removed, and in the number of points that are removed (which is governed by a binomial distribution. If it is desired that the same "random" removal is performed each time, the random "seed" must be set, as detailed in Section 7.4.6.

7.7.3 shuffle

Shuffle simply randomly-orders the in-memory points - this can be useful when writing data to a file for input into programs which otherwise are order sensitive. The spatial positions of the data, and the associated mass information is unchanged. The only allowable usage is `<shuffle/>`.

7.8 Compositional operations

7.8.1 range

The ranging operation applies a so-called "rangefile" to the current data, and uses this to drop points whose mass value does not lie within a set of specific "ranges". Each range refers to a region of the mass between some lower and upper bound. The rangefile must be a valid "RNG", "RRNG" or "ENV" file (See Appendix 8.2.2) - files which are not correctly formatted will be considered an error and processing will be halted. The file type (RNG/RRNG/ENV) will be automatically detected, so there is no requirement to specify the rangefile type.

The mass ranges can be inverted, by using the `invert="`*false/true*`"` attribute- if this is supplied (and set to `true`) then points whose values correspond to the ranged mass values are dropped, and the unranged data is kept.

The usage for this operation is as follows:

```
<range file="someRangeFile.rng" invert="false"/>
```

7.8.2 cluster

The cluster operation performs a clustering analyses. The only currently supported algorithm is the Maximum-Separation method, (`<algorithm value="maxsep"/>`) as described by Vaumousse [?], Hyde [?] and Stephenson et al [?]. This algorithm first specifies whether certain points are considered "core" or "bulk". In the simplest mode of operation, "Core" points can be used to generate cluster backbones, by linking together points that lie within a specified d_{core} (sometimes referred to as $d_{\max}$) of one another. These cluster backbones can have non-core (aka "bulk") material attached to them by using the d_{bulk} parameter - any points that are within d_{bulk} of the cluster backbone's points will be added to the cluster. Additionally, the "erosion" post-processing step of Vaumousse, and the "classification" pre-processing step of Stephenson is supported, via the d_{erosion} and d_{classify} values.

The attributes that must be specified are as follows:

- **dcore value=**`"`*number*`"` **knn=**`"`*integer*`"` - The value specifies the "coring" distance, as described by Stephenson. To disable this step, set the value to zero. This parameter can be used to refine which points may be utilised as clustering "core" points. The `knn` attribute specifies that in order to be allowed, this k-th nearest neighbouring point must be found within the core radius. If not, it cannot be used as a "core" clustering point.
- **dmax value=**`"`*number*`"` - This value specifies the distance within which points that are nominated to be "core" points may be found, in order to be considered part of the same cluster.
- **dbulk value=**`"`*number*`"` - This value specifies the distance from a cluster within which "bulk" points, can be considered to be part of that cluster.
- **derode value=**`"`*number*`"` - This value specifies how much distance from non-clustered material that bulk points otherwise associated to clusters should be rejected from clustering.

This allows for the extraction of clusters of points from datasets, and can be used in conjunction with the `<relabel/>` operation to allow for statistical examination of cluster distributions against randomised distributions, similar to the method of Stephenson [?].

A rangefile is mandatory, as specified via a `<range file="somefile.rng"/>` element. This must be followed by a `<core>` ··· `</core>` section, and similarly a `<bulk>` ··· `</bulk>` section, which contains a list of `<atomtype symbol="`*RangeSymbol*`"/>` elements.

Some additional, but optional parameters are available. These allow for finer control over the clustering process and its output. If given, these must be given in the order as shown here (omissions are permitted).

- `<sizeclip nmin="`*integer*`" nmax="`*integer*`"/>` - The minimum/maximum allowable size of a cluster. Clusters less than this size will be rejected from the analysis. Both `nmin` and `nmax` are optional, and one or the other can be omitted.
- `<clusterstats core="`*true/false*`" bulk="`*true/false*`" percluster="`*true/false*`" file="`*filename*`"/>` - This optional parameter causes the program to emit information regarding the statistics of the clustering process, such as the number of ions of each type in the clusters and the radius of gyration. If `percluster` is set to true, then additionally the centroid of each cluster will be printed. This data can be quite large, so optionally, this can be saved to a text file, by supplying the `file="`*filename*`"` attribute. Other than for `file`, each attribute is required.
- `<unclusterstats file="`*filename*`"/>` - This option will cause the program to print out the number of ions that were not counted as part of a cluster. The `file` attribute is optional, and without this, the result will be printed to the terminal.

- `<sizedist file="`*filename*`"/>` - This option will cause the program to generate a cluster size distribution - the `file` attribute is optional, and without it the results will be printed to the terminal.
- `<clustered-pos file="`*filename*`" retain="`*true/false*`"/>` - This option has two effects. If the `file` attribute is specified, then points that are considered part of the clustering are saved to the named file. If the `retain` parameter is set to false, then the output of the clustering is dropped from the in-memory dataset. This means the clustered data will *not* be passed on to other operations. If the `retain` value is omitted, it defaults to true.
- `<unclustered-pos file="`*filename*`" retain="`*true/false*`"/>` - Similarly to the `<clustered-pos ···>` element, this allows for the saving, or retainment of the ions that are not considered part of a cluster.

An example of using the clustering operation is given below:

```
<cluster>
        <algorithm value="maxsep">
                <!-- Coring (pre-cluster) dist; zero to disable this step-->
                <dcore value="0" knn="1"/>
                <dmax value="0.5"/> <!-- Max sep dist -->
                <dbulk value="0.2"/> <!-- AKA envelope-->
                <derode value="0.2"/> <!-- erosion distance-->
        </algorithm>
        <range file="../AB.rng"/>
        <!--select ions for cluster core (aka solute)-->
        <core>
                <typelist>
                        <atomtype symbol="B"/>
                </typelist>
        </core>
        <!--select ions for bulk (aka matrix)-->
        <bulk>
                <typelist>
                        <atomtype symbol="A"/>
                        <!-- Could have more ions here, like so:
                        <atomtype symbol="C"/>
                        <atomtype symbol="D"/>
                        ...        -->
                </typelist>
        </bulk>

        <sizeclip nmin="2"/>

        <clusterstats core="true" bulk="true" percluster="true" file="cluster-stats.txt"/>
        <unclusterstats file="unclustered-stats.txt"/>

        <sizedist file="sizedist.txt"/>
        <clustered-pos file="cluster.pos" retain="true"/>
        <unclustered-pos file="uncluster.pos" retain="true"/>

</cluster>
```

7.8.3 composition

This operation computes the total composition of the dataset, using the ranging assignments provided. A rangefile must be provided, via `rangefile="`*somefile*`"` attribute. Background correction can optionally be performed. Currently multiple ions cannot be automatically split into their components.

At this time, the only allowable background correction modes are mode “0” - no correction, and mode “1” - which is a TOF-noise correction.

The TOF-noise correction works on the assumption that the background is formed is flat in TOF space, and when converted to mass-to-charge space (using a sqrt type extraction, typical for APT data) the noise follows a simple analytical curve (equation below, where k_{fit} is the estimated noise intensity, and m is the mass). A region for noise selection must be given, to allow for background fitting, which the start and end are given as `backstart="`*value*`"` and `backend="`*value*`"` - the start value must be lower than the end. The attribute `backbins value="`*value*`"` determines how data is to be binned when performing the TOF fit and statistical test; a value of no less than 10, and usually 20 is recommended. If any peaks are within the noise region, the statistical test will fail, and the compositional estimate will be aborted.

$$N_{\text{background}} = \frac{k_{\text{fit}}}{2\sqrt{m}} \tag{1}$$

Integrating this equation yields a total noise within an $[A, B]$ range as:

$$N_{\text{background}} = k_{\text{fit}} \left[\sqrt{m}\right]_a^b \tag{2}$$

This total noise is subtracted from each range to determine the final count, and thus composition. Statistical checks (Anderson-Darlington) are performed to ensure that the region selected for background correction is Gaussian when converted from m/c to TOF-space - an error is reported if the data is not Gaussian. For APT most users, fitting between 1.5 and 2 should yield good results, as no peak is expected at this mass.

Composition data can be in un-normalised (raw count) or in normalised form (with counts) by setting the `count="`*0/1*`"` parameter. Normalised results are given in atomic percent. An example of usage is as follows:

```
<composition rangefile="yourRange.rrng"
                        backmode="1" normalised="1" count="1"
                        backstart="1.1" backend="1.5" backbins="20"/>
```

7.8.4 compprofile

This operation computes a composition profile from a dataset. At this time only spherical profiles (radially increasing) are supported - if requested, we may implement axial cylindrical profiles. The profile result is saved to the file specified in the mandatory `file` attribute.

The `<primitive .../>` tag, and the `type="`*sometype*`"` attribute are required. As previously stated, the only supported primitive is “sphere”. The input rangefile for assigning masses to species is given via the required `<range file="`*someRangeFile*`">`, and must be a validly formatted rangefile (See Section 8.2.2). The `<vectors>` section lists any vector properties for the primitive - similarly scalar values are set by the `scalar` tag. The vector section must precede the scalar section.

For spherical profiles, the properties that can be set are are:

- **vector:** Position vector to sphere origin.
- **scalar:** Radius of sphere.

Finally the number of bins to take is given by `<nbins value="`*integer*`"/>`. The step size is computed automatically.

An example of a spherical composition profile is given here:

```
<compprofile file="profile.txt">
        <primitive type="sphere"/>
        <range file="../AB.rng"/>
        <vectors>
                <!-- sphere origin -->
                <point3d x="1.5" y="1.5" z="1.5"/>
        </vectors>
        <scalars>
                <!-- Radius -->
                <scalar value="1.5"/>
        </scalars>
        <nbins value="50"/>
</compprofile>
```

7.8.5 relabel

The `relabel` operation performs a random shuffling of the "m" value across the spatial positions, commonly referred to as "random relabelling". Commonly this operation is used for random-comparator statistical trials, particularly when looking for spatial correlations within a dataset (shuffling provides a good null-comparator) [?].

This operation has no parameters, and the only usage is given below:

```
<relabel/>
```

7.8.6 proxigram

The `proxigram` operation computes either a density or concentration profile as the distance from some surface. Each point in the point-cloud is assigned a so-called *signed distance* from the surface, and a histogram of these distances are generated (which is the proxigram function [?]). To do this, a mesh is required. *posgen* will accept any GMSH[8] formatted *triangle mesh*. The mesh must be non-intersecting and 2-manifold, but need not be closed - that is it must be a 2D surface, which can be physically realised. The mesh should be additionally consistently oriented, otherwise the mesh will be forced into a consistent orientation, and may end up reversed from the orientation that is actually desired.

The attributes for the proxigram itself include the maximum distance to which to compute the proxigram, `maxdist`, the file to which to save the histogram, `outfile`, and either a bin width, `binwidth`, or a bin count `nbins` (but not both). Note that the `nbins` value is only for one side of the proxigram. The `mesh` element is used to load the mesh from the specified file. This can optionally be recentred around the origin, but in most cases this will not be desired.

The proxigram may optionally include a `<range file="`*someRangeFile*`"/>` element, if provided the resultant proxigram will be computed for each species in the rangefile. If not provided, then only a count as a function of the distance from the surface will be given. An example is as follows:

```
<proxigram outfile="proxigram-output.txt" maxdist="4.0" binwidth="0.1"/>
  <range file="someRangeFile.rng"/>
  <mesh file="myMesh.msh" recentre="false"/>
```

7.9 Value transformations

7.9.1 pointconc

The point concentration operation computes a localised concentration using a ratio approach. This can allow for point-based identification of regions of differing composition, or compositional ratio. to compute compositions, a valid rangefile must be supplied, as well as a listing of the numerator, and denominator species. With this list, the localised composition C_x will be computed, per point (x), as per the following equation:

[8] `http://geuz.org/gmsh`

$$C_x = \frac{\sum_{i=1}^{n} \mathrm{Num}_i}{\sum_{j=1}^{m} \mathrm{Denom}_j} \tag{3}$$

Numerator and denominator species are specified in a `<typelist>···</typelist>` section, and both must be provided. Within the `<typelist>···</typelist>`, each species may be specified (if it is desired to be part of the numerator/denominator respectively), using an `<atomtype symbol="`*symbol*`/>` element.

An example operation is given:

```
<pointconc radius="5.0">
        <range file="someRange.rng"/>
        <!--source points (numerator) -->
        <typelist>
                <atomtype symbol="A"/>
                <atomtype symbol="B"/>
        </typelist>
        <!-- dest points (denominator) -->
        <typelist>
                <atomtype symbol="A"/>
                <atomtype symbol="B"/>
        </typelist>
</pointconc>
```

7.9.2 pointdensity

This operation computes the local density for each point in the dataset. This is done by placing a sphere around each point, of a specified radius, r. The number of points, N, that lie within this sphere are then counted and the density computed as $\rho = \frac{3N}{4\pi r^3}$, which is then used as the mass value for the point. Alternately if the count, N, is desired – rather than the sphere-normalised density (ρ), the attribute `normalise="false"` can be given. An example of the use of the `pointdensity` operation is given:

```
<pointdensity radius="1.2" normalise="false"/>
```

Care must be taken when interpreting results from this operation. Points that lie on the exterior of the point cloud will have a lower density if there is a non-physical end to the point cloud (i.e. if there are locations in your data where more data should be, but your dataset stops). This effect is limited in distance from the free surface by r, and becomes more noticeable as r is increased. Discussion regarding analytical correction factors for these surface effects for some geometries can be found in Lei et al [**?**].

8 Appendices

8.1 Glossary

- **Attribute** - parameters that are given to an XML tag
- **DTD** - Document Type Descriptor file . This file contains the formal syntax for operations, and is required for *posgen* to operate. DTDs are a standardised format.
- **Element** - A node in an XML file, usually appears as `<element>` $\cdots$ `</element>` or `<element/>`.
- **Operation** - An element in a *posgen* XML file that is an instruction to *posgen* to perform some action.
- **POS file** - A file that stores X/Y/Z/V data, with a specific format.
- **TAPSIM** - A program for evaporation simulation in atom probe
- **Tag** - Synonym for Element
- **XML** - eXtensible Markup Language - a simple structure for text files, which allows for simultaneous human and machine-readable files.

8.2 File formats

8.2.1 Pos Files

This file is a four-field fixed width record file, with an integer number of entries. The file is uncompressed raw 32 bit IEEE754 floating point data, and can be loaded using most languages relatively easily. Note that the order of the floating point numbers "endian-ness" is fixed as big-endian. The floating point values are X,Y,Z and an arbitrary scalar value. The file may not contain invalid (Not-a-Number "NaN") values.

8.2.2 Range Files

The program's interpretation of the Oak-Ridge format for range files is given below. The original specification is available in the book *Miller, Atom probe: Analysis at the atomic scale*, (Kluwer Academic/Plenum Publishers, ISBN 0306464152). Additional information on the format is given by the PoSaP program, which to the author's knowledge is not online. Unfortunately, the specification given for the file is weakly stated, and is open to different interpretations. It may be that there are alternate interpretations with which the authors are not familiar, and the code is thus unable to interpret — please report these.

A simple example file is given below, and is nominally in the ASCII 1 byte per character format. The original specification, to the authors knowledge, predates the UTF-8 and extended codepage support for non English languages. Thus non-English languages are not part of the file format - each should use the "C" locale for reading and writing, to avoid localisation concerns.

```
1 2
Aluminium
Al 1 1 1 Al
------------ Al
. 10.0 150 1
. 150 200.2 1
```

The first line consists of two unsigned integers, separated by a space. The first integer is the number of unique ion types, and the second is the number of ranges. The next lines are taken as pairs. The first entry in the pair is the name of the "ion". The next entry consists of four parts. The first entry is a space terminated string, and is the shorthand name for the element. The next three elements are floating point values in the range of $[0, 1]$, and are the colour of the ions that are ranged, with each element being the red, green and blue component in turn (*i.e.* cubic RGB space). The final string is, to the authors' knowledge, unused, and is ignored.

This is repeated for each element pair, as specified by the first integer in the file. Each entry must be uniquely named, both in short and long names.

The next line can nominally be ignored, however it should contain the dash character from positions 1 to 13, followed by a space separated list (with leading space) of the short names, as specified above. Sequence positions are *not* obtained from the dash list, but rather from the order they appear in the file.

Following this is a 2D table (space separated). The first column appears vestigial. The second and third columns contain the start and end "range" values for each ion. Note that these do not have to be in the same sequence as the original specification. These range values must be non-overlapping, and can be any 32 bit floating point number (other than NaN).

The next columns are the range table, and specify which ions the range corresponds.

In the implementation here, the table should have only entries of 0 or 1, and the row (from column 3) should to exactly 1. Files where this is not the case may be accepted, however the exact interpretation for non 0/1 entries is unclear, and not specified in the file, so will be essentially treated as either a 0 or 1 value.

A more complex example is given below.

```
3 3
Magnesium
Mg 0.0 0.0 0.0
Copper
Cu 0.0 0.0 0.0
Nickel
Ni 0.0 0.0 0.0
------------- Mg Cu Ni
. 25 27 1 0 0
. 25 33 0 1 0
. 55.6 59 0 0 1
```

8.3 Copyright

8.3.1 GNU General Public Licence, 3.0

The *posgen* program itself is distributed under the GNU General Public Licence, Version 3.0. This licence can be found in the COPYING file, distributed with *posgen*, or at `http://www.gnu.org/licenses/gpl-3.0.html`

8.3.2 Creative Commons ShareAlike

The documentation for *posgen* - *i.e.* this PDF file, and the associated source-code for generating the PDF is distributed under the Creative commons attribution-share-alike 4.0 CC-BY-SA licence, given at `https://creativecommons.org/licenses/by-sa/4.0/`: