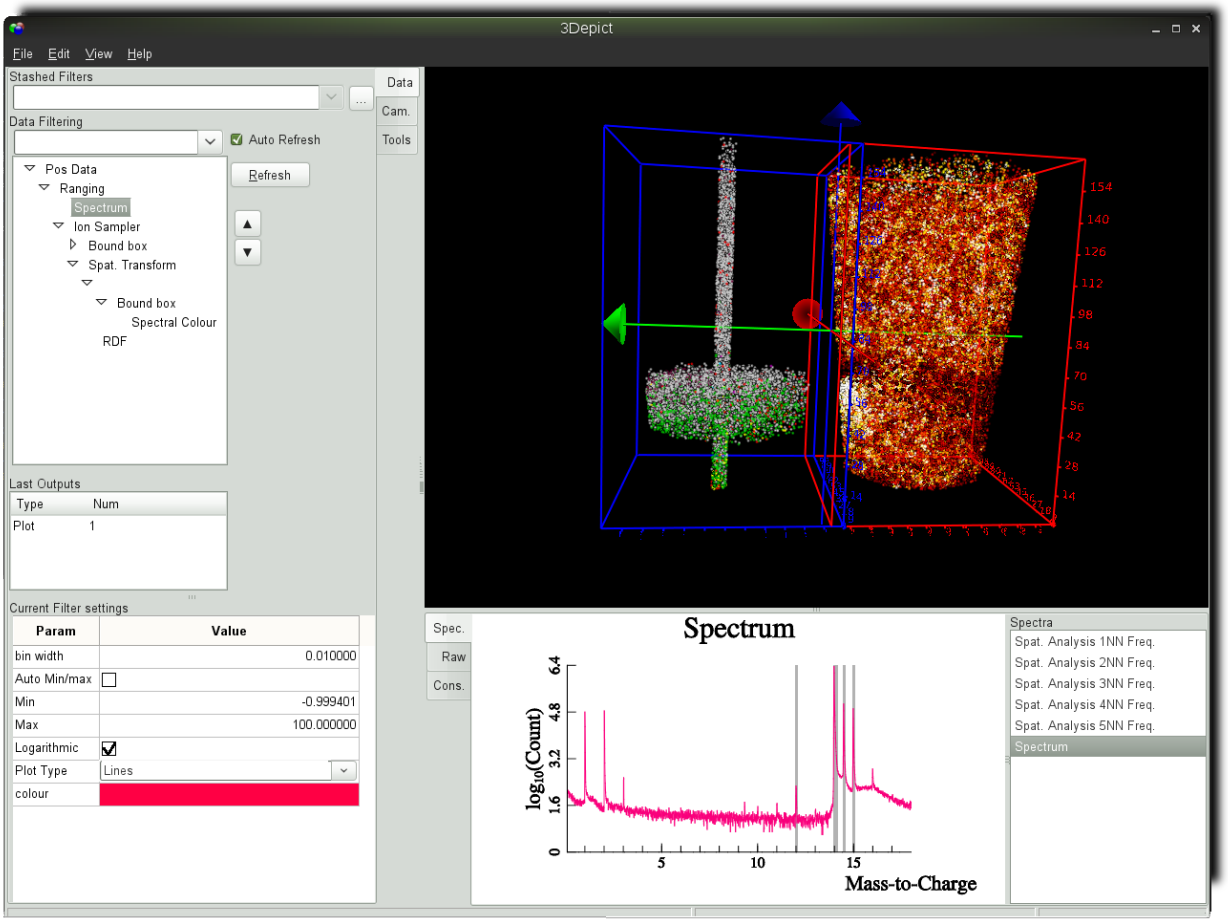
3Depict

Valued point cloud visualisation and analysis



User manual

Website:
`http://threedepict.sourceforge.net/`

Version:
0.0.6, May 2011

Contents

1 Foreword

1.1 Introduction

3Depict is an open source computer program designed for the analysis of point clouds. The program is designed around interactive data analysis, with a view to combine rapid feedback, ease of use and flexibility in a single system.

3Depict is designed purely for post-processing of 3D point data, and was originally primarily targeted to users of Atom Probe Tomography. Other users may find the program useful, and are encouraged to seek assistance.

1.1.1 Background

3Depict attempts to fill a perceived need for freely available flexible point data visualisation. This program is designed to manipulate and modify point data in a way which the author has otherwise not found a suitable program to do.

With this program, point data can be visualised using a fully implemented camera system, edited with directly interactive objects, and subjected to various analysis algorithms. A real-time plotting system is also provided to generate analyses of your data on the fly. External programs can be engaged as part of the system to create new analyses that "clip into" the analysis.

1.1.2 What is Open Source?

Open source programs are programs which distribute not only the executable code, which is understood by the computer (so called machine code), but also provides the version of the program as it was written by humans as well. This provides external users with the possibility of modification of the program behaviour or appearance, either by themselves, or by engaging a third party. With the source code one can verify the correctness of the system, alter behaviour or otherwise modify the program, or even reuse sub-sections of the program elsewhere.

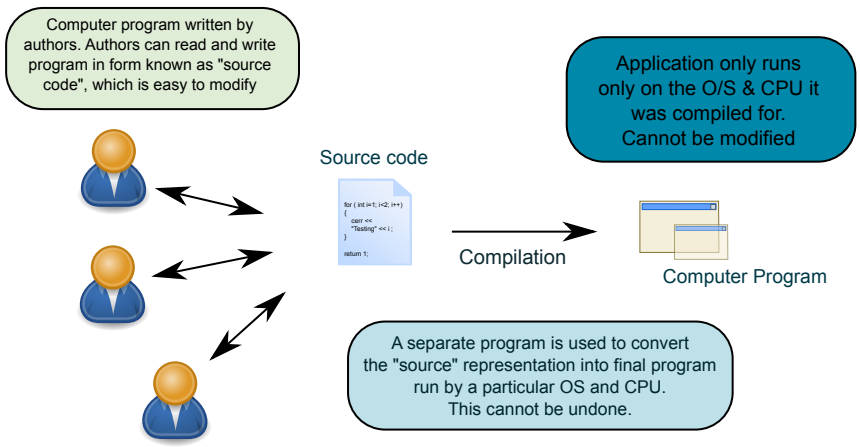


Figure 1: Closed-source programs only provide the final application, are neither human readable nor modifiable, and will only work on a specific platform. By contrast open source programs distribute the source-code as well as the application. The source code is the core logic which can be made to work on many platforms due to the invariance of the program logic.

Modifications to the program itself may include migrating the program to newer or older systems, adding new functionality, or correcting errors in the program implementation.

To provide the user with these capabilities, the program is distributed with a so-called *libre* copyright licence. The program is distributed at no cost to the end user, and the copyright attached to the program explicitly

allows modification and re-distribution (copying) of the program to other parties, without requiring the author's consent.

Note that there are restrictions on what may be done with the program, for example it is in violation of the licence to claim ownership of the program, or to use technical measures to prevent access to the program, or modification thereof. The licence used in the program is a generic one shared by many free (as in freedom) software programs

If you have been charged for this program, it is suggested that you request a refund and obtain a free copy from the main website. If you wish to have the full licence details (GNU General Public Licence Version 3 (or any later version)), please see the `COPYING` file distributed with this program. If this is not available, please see the project website, or perform an Internet search for the licence name.

1.2 Requirements

Due to the design of the program, the program should run under Linux, Mac, BSD and Windows machines. The program does not rely on CPU specific features, and thus should be able to be run under x86, x86-64, arm, or whatever. Basically, it should run just about anywhere. Every effort is expended by the author to ensure that the program can be run on as many devices as possible; if your platform is not supported, it may be possible for either you, or the author to generate executables for your system. See the section "Getting Help" for contact details.

The minimum requirements for running *3Depict* are not known. The author wrote a substantial portion of the program on a machine with a 4 and 12 GB drives, and a 1.6 GHz processor, which normally runs at 800 MHz and has 1 GB of RAM. I see no reason that it would not run on even lower-spec machines. Whilst a higher spec machine may run the program faster, intelligent use of the programs "filter" system may allow for complex analyses even on low-end machines.

If you are experiencing 3D graphics problems, first ensure that other 3D programs do not experience the same problems. Otherwise, please contact the author for assistance – there should be no requirement for vendor-specific hardware. Note however that the exact appearance of the 3D view is dependant upon your hardware, and may have small changes between different platforms.

1.3 Platform specific notes

Note that whilst every effort is made to ensure that the program will run on a variety of systems, small system-specific quirks may be evident, particularly on platforms to which the authors do not use regularly (*e.g.* windows). Secondly, due to slight differences between platforms some functions may be remapped to other mouse/key combinations.

Mac:

- `Ctrl` keys may sometimes be mapped to the `Command` (clover) key.

Windows:

- `Ctrl+Tab` cannot be used as a key combination, as this is reserved for switching between user interface elements. `Ctrl+Alt` is used instead.

1.4 Getting help

Assistance with this program may be freely obtained over the Internet at `http://threedepict.sourceforge.net`. Questions regarding use of the program, feature or bug reports will be attended to as soon as possible. Contact options include email, or an online forum.

If the program crashes in a predictable manner (*i.e.* you know how to trigger it), this is a bug and needs to be fixed. Please report the bug in this case, so we can fix it as quickly as possible. If the program crashes in an unpredictable fashion, please still report it as best you can, and we will try to fix it, if it is possible to isolate the problem from the description. For advanced users, we would appreciate backtraces, state files and any other relevant information in both of these cases.

1.5 Who wrote this program?

This program was written by D. Haley, in his spare time. A. Ceguerra provided development from Version 0.0.2 and provided assistance with debugging and fixing the Macintosh version, and providing executable versions of the program for OSX in 0.0.1.

1.6 Alternate documentation

For the more visually inclined, screencasts of the program have been created, and are available on the project website. These videos exhibit basic use of the program for various simple analyses. At time of writing, the only literature available for the program is this document, and the online screencasts. If you have questions, please contact us through the website, where we will reply as soon as possible.

1.7 Helping out

3Depict takes time to develop, and no doubt could be better than it is now. However, this doesn't all just magically happen – people have to put the work in. Development time by the authors is split between testing the program, coming up with new ideas for program changes, editing documentation, making pretty pictures, maintaining websites, and even developing the program.

We would always appreciate assistance with this work. You don't have to be able to write computer programs. For example, we would like to translate the program into other languages. If you can translate a spreadsheet table into another language, this is helpful. If you can work out what triggers particular bugs, this is helpful. If you can improve this document, this is also really helpful. Of course, if you can program (C/C++) and are willing to help grab a copy of source from our website, contact the authors, because a little code goes a long way.

2 Basics

2.1 Getting started

2.1.1 Licence

This program is distributed under the GNU General Public Licence Version 3 (GPLV3+), an *open-source* licence. Information on the copyright of this program is available under the `COPYING` file in the program directory, or online (*e.g.* `http://www.gnu.org/licenses/gpl-3.0.txt`).

The basic premise is that you may copy the program, modify and distribute such modified versions or derivitive works only under the same licence. The licence forbids technical restrictions on users further redistributing the program.

2.1.2 Installing the program

The installation method for the program depends upon your chosen operating system. The most up-to-date notes are available on the project website. It is highly recommended that, in general, you do not simply download random programs from the Internet and execute them if a version is available in a trusted software repository.

2.2 Understanding the interface

The program interface consists of three different views. On the left, there is the data, cameras and tools panes, with are used to generate data for visualisation, and to provide an interface into changing properties in a structured manner. On the right, the view is split into two sections; at the top, there is the 3D view. At the bottom are the plotting, raw data and console output panels.

Each pane may be hidden, either by double clicking the "sash" between the two panes, by selecting the respective item from the view menu or by its keyboard shortcut key as listed in the menu.

At the very bottom of the program, a status bar is shown – here messages are shown to provide hints on how to use the program, or to communicate information relating to the program's internal state.

2.2.1 The 3D View

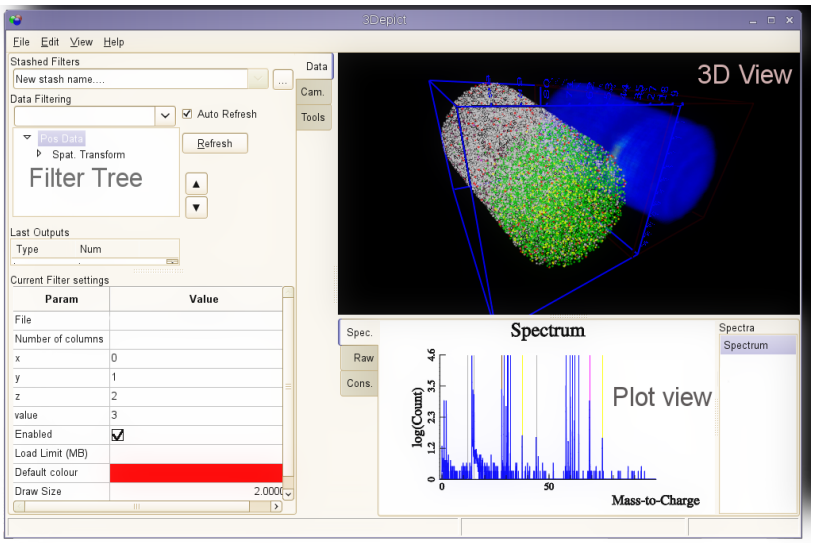


Figure 2: Interface layout. The 3D view, plot panel and filter tree are labelled.

The 3D view is used to show the three-dimensional objects generated during a data analysis, and provides a direct method of interaction with the 3D Scene. Through the use of the mouse (or other pointing device), the 3D view can be manipulated to change the view position and orientations. Some objects in the 3D view are interactive, and will be indicated by an overlay in the top right of the window when the pointer is on top of such an object.

2.2.2 Plot area

The available plots are listed on the right hand side of the plot view panel. You can select the active plot from the list. The items in the list take their name from the filter from which they originates name (there are exceptions to this rule, i.e. composition profiles). Several plots may be drawn at once by holding down the `Ctrl` key when selecting the plot to draw from the plot list box.

2.2.3 Basic movement

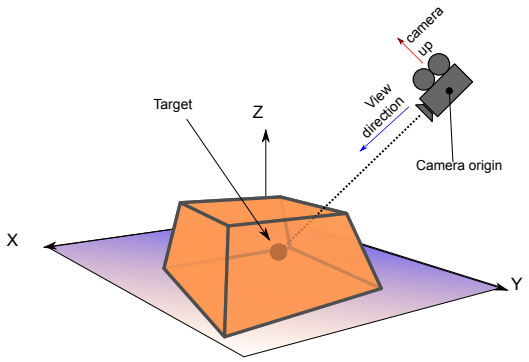


Figure 3: Basic camera layout. Each camera has a position, an up direction and a target. The 3D view is as seen by the camera. Cameras may be saved and recalled to return to specific views. Try to realise it is not the object that moves, but rather yourself.

The 3D view represents your camera into a 3D scene of your construction; it is by manipulation of cameras that the view is interacted with; so you may zoom, orbit, pan, roll or swivel the camera view. If you are lost at any time, you may reset the view by tapping the space bar. To change the axis along which the view is reset, hold the `Ctrl` or `Shift` buttons whilst resetting.

The basic 3D view consists of a "target" based camera, so when you move the camera, the camera will orbit around this target. To interact with a scene, hold down the left mouse button and move the mouse to control the camera.

The basic keys for controlling the camera move mode (left click) are :

- **No key**: Orbit camera
- **Ctrl**: Pan camera
- **Tab** : Swivel camera (Look about)
- **Ctrl +Tab** (Windows **Ctrl+Alt**): Roll camera around viewport centre. Note that the rolling motion is controlled by the position of the mouse click.
- **Space/Shift+Space/Ctrl+Space** : Reset camera bounds and position to look along X,Y or Z axes respectively.
- **+/-** : Zoom in/out.

For any motion, the `Shift` key may be used to increase the camera move speed. Scrolling on the window zooms in or out. For a perspective camera, zooming is performed by moving the camera closer to the object. For an orthographic camera, zooming simply scales the view, whilst holding the camera position constant.

3 Creating a scene

Initially the program window will appear with only the default world axes visible. To provide a more interesting view, it is necessary to inject data into the program. To do so, select the File menu, and then select using "Open". At time of writing, only two formats are currently supported. Firstly are "POS" files, and secondly are text files, each which consist of X,Y,Z and a values (usually mass-to-charge) [1]. To load a file, navigate to an existing POS file on your disk. If you do not have a POS or text file, small example files are available on the project website, on the documentation page.

Upon selecting the file and then `Open/OK`, the file will be loaded into the viewport. Note that the entire file is not loaded, but rather a random selection of elements in the file.

3.1 Basic manipulation

Loading this file populates a small treeview on the data pane (at the left). This tree is referred to as the "analysis" tree, and each item in the tree is called a "filter". The tree is responsible for producing the output data in the scene, and a good understanding of the behaviour of this tree is required to extract the maximum benefit from the program. Each item in the tree has a list of properties that can be modified. For example, the amount of data loaded by the "pos load" filter can be altered by selecting the "pos load" item from the tree, then in the grid below, entering in the new amount of data to load (you can set this to 0 to load the entire file).

Thus, each filter can be individually altered to change its behaviour. However, each filter acts upon the output of the filter that is a "parent" to it (in the case of not having a previous filter, each filter will act as if it had no incoming data). Thus the arrangement of each filter in the tree is critical to the output of the program. In order to modify the layout of tree, you may add new and move, copy or remove existing components of the tree. Changes to the tree, or any filter contained therein, may be undone using the "Undo" menu item, or with the keyboard shortcut `Ctrl-Z`. Each filter's behaviour is outlined in Section 4.8. More information on the tree behaviour is given in Section 4.2.

Note that with every modification of the tree, the 3D scene and any plots will be recomputed. The time of computation is dependent upon the amount of data that is to be analysed, and can be reduced through sampling or volume restriction methods. By default, each filter may cache its own output, in order to speed repeated computations.

To delete an item, simply select the item to delete with the mouse, and then use either the `Delete` or `Backspace` keys on your keyboard. Note that clicking on an already selected item will activate the name edit mode. To exit this mode, press `Escape`.

New items can be added to the tree by selecting the filter to add from the dropdown box immediately above the tree. When selecting a new filter to add; an element in the tree must be selected, where the new filter will be placed. If there is no item selected, an error will be shown in the status bar.

Once an item is added, the filter tree is thus modified and a recomputation of the scene will occur. Approximate progress on the filter update is visible in the status bar. During an update, only limited interaction with the program is permitted. An update may be cancelled at any time with the `escape` key.

3.1.1 Tools panel

The tools panel offers several options on changes to the way the program operates internally.

- **Smooth and Translucent objects**: This enables so-called "alpha blending" in the 3D scene, where appropriate which allows for non-opaque objects, and anti-aliased objects. This mode alters the way

[1] For a technical description of the POS file format see Section 7.1.3. For a description of suitable text formats see 7.1.4

in which objects are rendered in the 3D scene and is in effect a quality-appearance tradeoff. Most of the time you will probably want it set to ON. The program may render the 3D scene slightly faster if this is disabled.

- **Enable lighting** : 3D objects do not look very 3D if you are only seeing them on a 2D screen. Computer graphics works around this by simulating the effect of having a 3D lighting source. This might provide minor performance improvements if disabled, at the cost of clarity of rendering.
- **Enable filter caching**: This alters the way in which the program processes the filter tree. Normally, the program performs what is known as a depth-first search, and propagates data generated by the program from one filter to the other. Intermediate copies are kept by the filters themselves to speed up recomputation. However, this strategy has a large downside, which is memory consumption. Disabling this will reduce memory consumption by filters, but will mean that any change to the filter tree, no matter how small, will cause the entire tree to be recomputed, including data loading.
- **Weak and fast random**: This setting is a program wide setting that switches the strength of the random number generator. However, for more robust statistical results, it is recommended that this be disabled when computing final values. When enabled, the program will use a Linear Shift Feedback Register using a maximal length Galois polynomial to generate numbers required for random sampling. This has the advantageous property of being a somewhat random entirely non-repeating sequence, but having sufficient decorrelative strength against most inputs to provide the appearance of random sampling.

4 Understanding the program

4.1 Filters

Filters form the key component of the program. These are the tools by which data is analysed and modified, in order to generate the visual representation that is needed by the end users. The basic idea behind a filter is that each filter may perform arbitrary operations on data streams. These data streams are sent to and from each filter, flowing through the tree.

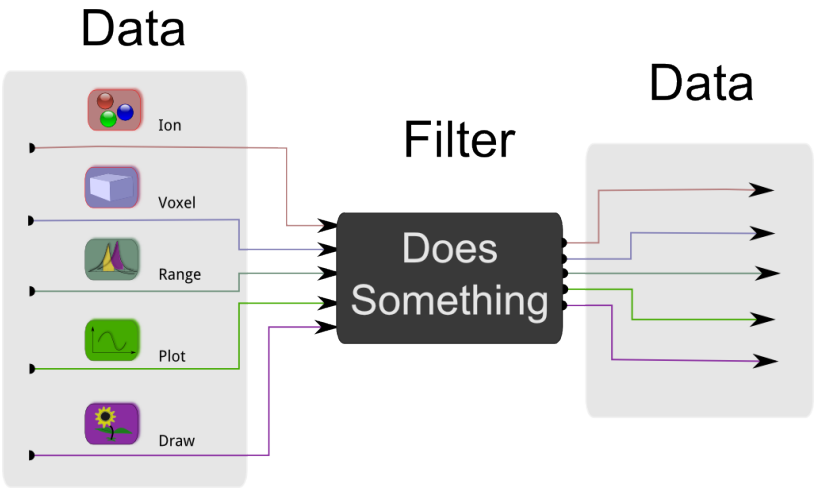


Figure 4: Basic concept of a filter. Data goes in, data comes out. The filter may perform any operation on the data coming in or out as it chooses. The data streams coming in are restricted to certain types of data, as shown.

The basic idea of a filter is illustrated in Figure 4. *3Depict*'s flexibility is that these filters can be arranged in any way that makes sense to the end user. There is no restriction on placement of filters – some placements may be totally useless, others may be exceedingly useful. It is up to the creativity of the end user to determine whether any single arrangements meets their needs.

4.2 Trees

The tree is a flexible and powerful system for constructing your own analyses, after some use this will become a familiar and readily modifiable system for performing your analyses, however the initial structure of the program may take some getting used to. If you are familiar with programs such as *Paraview*, you may already be familiar with this concept.

The filter tree essentially is a system for injection, manipulation and display of the data in the program. The tree becomes an "assembly line" for the view of data in the 3D and plot views. The nodes of the tree are the filters that act on or insert data into the analysis. Each node in the tree may be considered in what is called a "parent-child" relationship. Each element in the tree (except the first) has a "parent", and thus may have their own "child" elements. Each "parent" may, in fact, have many children. Data may be considered to propagate from the "root" of the tree downwards, with each filter in a direct line somehow modifying the data from above in some way. When data reaches the end of the filter tree it is "picked up" by any of the 3D view, plot or console panels, depending upon the nature of the data.

The basic method for data flow is that a parent gives a copy of the data it has processed to its "children" to modify in some way. Each "child" has its own copy[2]of the data from the parent, which it modifies. In turn this child then gives a copy of the data to each of its own children. If a filter has no children it then passes the data to either the 3D view, the plot view or the console view, depending upon the data type.

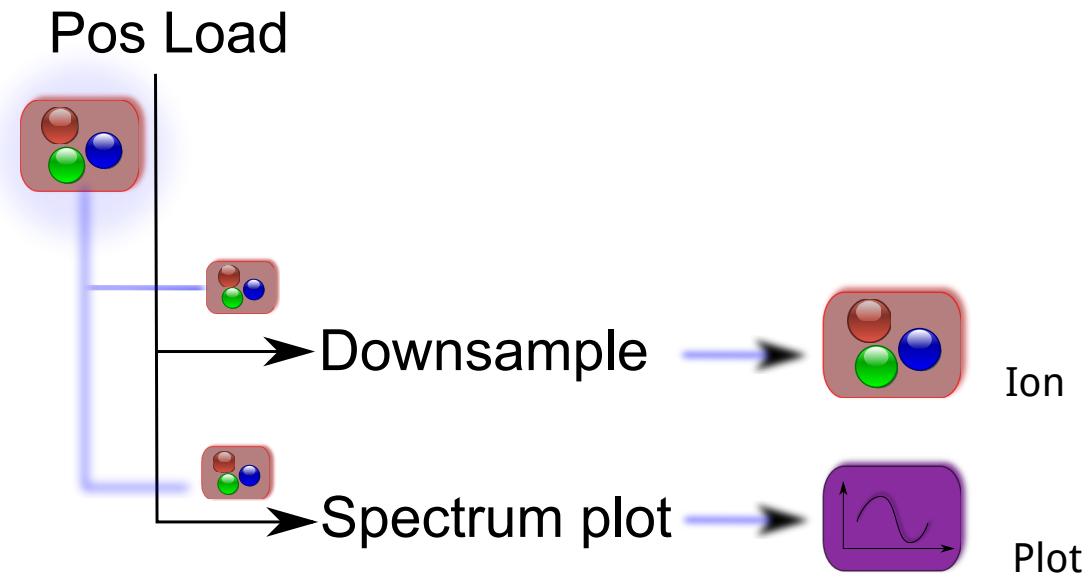


Figure 5: Data propagation in a tree for a particular arrangement of filters. Data is propagated from a parent filter to its children.

Using this method, one may create a variety of different analyses; for example, one may wish to subsample

[2]Technical note: the "copy" system is at the discretion of each filter. Child filters are given a reference to the parent data which restricts modification of the parent's data by the children; children may or may not duplicate this data, propagate or terminate the reference.

data before performing a time-consuming spatial analysis, or one may wish to clip the data to remove unwanted sections before generation of a mass spectrum. The flexibility of the filter system supports this concept.

Note that items in the filter tree can be moved. You may move any filter to a new parent by dragging with the mouse. In order to copy instead of move, hold down the `Ctrl` whilst moving to duplicate the filter, rather than moving it.

You may also rename filters in the tree; The filter name may be used by the filter to generate its output, *e.g.* spectrum plots will take the plot title from the filter name.

4.3 Stashes

Instead of enabling or disabling sections of the tree, the program supports "stashes" as a place to put sections of the analysis tree for later use without using them in the analysis section. To create a "stash", select a section of the filter tree to "stash", then in the "stashed filters" dropdown on the data tab, type the name of the stash you wish to create (this is up to you), and press `Enter`. Once done, a duplicate of the subtree specified (*i.e.* all the filters below the selected one, and the selected one too), is made. This process is shown in Figure 6. You can view the contents of the stash by selecting the button next to the stash dropdown, and you may delete stashes however you cannot edit them.

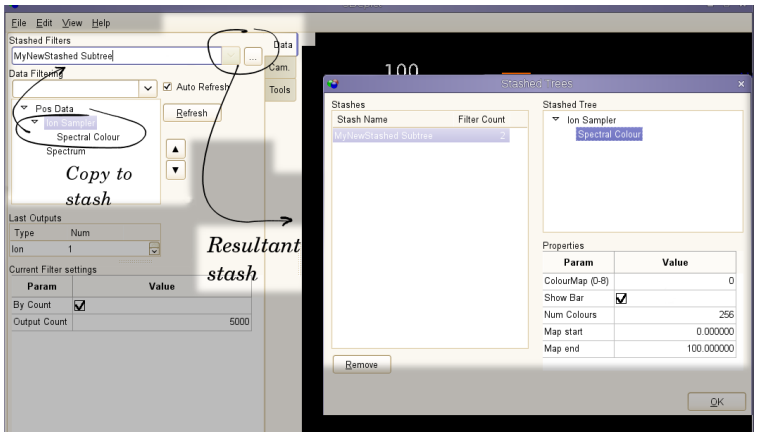


Figure 6: Creating a stash from the filter tree. New stashes will appear in the dropdown and can be selected to recall subtrees to insert into the filter tree.

To use a stash, select a filter in the tree and then click the dropdown button on the stash combo box, and then select the stash you wish to use. This will place the stash as a child of the selected filter. Note that the stash can be used multiple times.

4.4 Plots

Any plots generated by the filter system are displayed in the plot pane. It is possible to zoom or pan the view as required by dragging or `shift` dragging the plot respectively. Double-clicking the plot returns the plot back to its original scaling.

The associated numbers used to generate the selected plots are shown in the "Raw" tab. Note that plots can contain "regions", such as generated by a range file. In this case, each region may be manipulated in-situ, by dragging the regions sides, or its centre to alter or move the region respectively. These modifications will be propagated back to the original filter.

Each plot is either logarithmic, or linear in scaling. Mixing these two types of plot will result in the y-axis stating that there are mixed data types in the plot. The log/linear mode is determined by the filter that generates the plot. Note that due to internal limitations (fixed plot palette in the underlying library), the colours observed in the plot may be slightly different from those specified by the filter.

4.5 Cameras

To fully understand the camera model, it is necessary to understand the parameters in the camera property tab. Initially there is only the default camera, which is unnamed. By entering in a name for the camera, you can access the properties for that particular camera. By entering in more names, you can create multiple cameras, saving the position of existing cameras as you go. This can allow you to jump between different camera views as desired.

One can select the position of the camera, a position that the camera is always looking at (target), the camera "up" direction, and the field of view.

With the exception of the field of view, these parameters are dynamically modified when interacting with the 3D scene (see section X). The camera field of view, however requires special mention. The field of view of the camera is the angle that the camera look at. Human vision is around 120*, and is much narrower for suffers of tunnel vision (say, 30*). A bird has a full 360 degree field of view (it can see in all directions without needing to turn its head). By default the camera is set to 90*. To get the "fish-bowl" effect, where close objects appear very large, this number can be increased. To get an effective orthogonal camera, this number can be set very low. Note that changing this value will also have the apparent effect of zooming the camera in or out, so tapping `space` to reset the camera view is recommended for large changes.

4.6 Program actions

4.6.1 Save

The current programs state can be saved to an "XML" state file for later analysis[3]. Note that opening an existing program state file will erase your current state. If you wish to merge the two states together into a single analysis, use the "merge" option. Note that as this file references, but does not contain, the data files needed for the analysis, this file cannot be moved between computers and expected to "just work". However, to overcome this, the program provides the ability to export an analysis "package", which contains all the data necessary to move these files between computers with ease, regardless of platform. This feature is explained in the "Export" section.

4.6.2 Undo

The program has an undo feature which can be used to abort the last changes to the filter tree. Note that for memory reasons, the results of the computation are not stored, and will need to be recomputed. Note that there is also a "redo" function, which allows for undone changes to be restored.

4.6.3 Raw Data

The raw data pane may be used to obtain the raw XY data used to generate the plots. This can either by copied and pasted, or alternately saved to file.

[3]See Section 7.1.1 for more information.

4.6.4 Export Menu

Plots, images and ion data may be exported from the program. The output format for 3D images is the "Portable Network Graphic (PNG)" format; these are supported by almost all image viewers.

For plots, you may save in either (Scalable Vector Graphic (SVG)) or "PNG" forms. Note that due to the nature of the SVG files, no resolution is needed, and the image can be reproduced at any scale. Furthermore the SVG can be used later to generate PNG images at the required size for output (I recommend the program *Inkscape*). Alternately saving as PNG can be done, and you will be prompted for the desired image size.

Exporting Ion data can be done in several ways; you may export only the visible ions, or alternately, you may export only a subset (for example one or two ranges) of the data, depending upon the filter that the data emerged from (*i.e.* per leaf filter. The output format will be in Big-endian "POS" format., as detailed in the Appendix.

Modified range files may be exported in whole. Currently the only supported export format is the oakridge"RRNG" format

Finally one can export the entire analysis state, including all required data using the export analysis package option. This will create a folder which contains all the files needed to reproduce the current program state elsewhere. Note that this imports all referenced data files, so the package can become quite large.

4.6.5 Autosave

The program will generate an autosave file periodically. If the program crashes, it will look for an autosave file and prompt you to restore it. Note that only the program settings are saved, not the intermediate data, so recomputation will be necessary.

4.7 Data types

Different data can propagate through the filtering system before it is seen in the 3D view. The currently available types are ion, range and plot types. Although these are used internally by the program, understanding the type system may enable more advanced use of the program. If you are not interested in this, skip to the next section.

4.7.1 Ions

Each ion represents a point in space, which has a value type associated with the point. For example, one might consider a point in a dataset where positions represent atomic positions, and the value is the measured atomic mass. Ions are grouped together by different filters, and each group may be represented with a unique colour and size.

4.7.2 Plots

Plots can be passed between filters to allow for a 2D graphical representation of whatever it is that the filter computes. Plots are a X-Y paired set of scalar values, which are finally given a visual representation as a plot. Plots have a title, and a label, and may represented either on a linear scale, or a logarithmic one.

4.7.3 Range

This is a special datatype which propagates information through the filter tree. The data represents non-overlapping regions of the value space which are to be tagged as belonging to a certain group. This data type

has now actual output into the 3D scene, but can alter the manner in which "downstream" filters process incoming information. For example, if a profile filter is used after a range, it will split up its measurements into a per-tag "range" section. X data need not be sorted in an increasing fashion.

4.7.4 Voxels

Voxels is shorthand for "volume pixel" and is a rectilinear region of space, divided up into an equally spaced rectangular grid. Voxels can currently be represented by a point cloud, where each point has a given colour and transparency, or by a triangulated surface (an iso-surface) which represents the contouring surface for a given scalar value.

4.7.5 Drawables

3D primitives can be injected into the data stream to assist in the final representation of the scene. Items such as spheres, lines triangles or text can be placed in the final scene.

4.8 Available Filters

4.8.1 Pos load

The pos load filter injects ion data into the analysis tree. Ions are loaded from a file by one of several different methods. By default, random data is selected from the file. This filter can be created using the "load" function from the file menu.

- "Number of columns": Number of floating point values in a single record. Defaults to 4.
- "x": Position in record to use as X value. Defaults to 0.
- "y": Position in record to use as Y value. Defaults to 1.
- "z": Position in record to use as Z value. Defaults to 2.
- "value": Position in record to use as associated scalar value. Defaults to 3.
- "Enabled" : Disable/enable the filter.
- "Ion colour": Colour of the ions from the 3D view.
- "ion size" : Default size of points in 3D view.
- "filename" : name of the file to load the data form.
- "Load limit": The maximum quantity of data to load from the file. If set to 0, then the entire file is loaded. Otherwise a random sub-selection of the file is loaded. Note that random selection reduces memory cost, but if it is more than a few percent of the file size, may be slower to load.

4.8.2 Downsampling

Randomly samples ions from the input stream. Can operate either to generate a fixed number at the output, or to take a fixed percentage of the input

- "Fraction" : Approximate random fraction of the data to load. Must be between [0,1].
- "Max count": The approximate number of ions to load.
- "By count": Specifies whether to use a fixed count, or a fixed fraction

4.8.3 Ranging

This allows for the cropping and segregation of ions in 3D space by their scalar values.

Each range loaded from the file may be enabled, either at the ion level (groups of ranges) or at the range level. The range values may be altered; however these may not overlap at any time. Note that these can be edited graphically if used in a mass spectrum.

- "Filename": This is the name of the file to use as the range source. So-called ORNL "rng" files, Cameca "env" files and Imago/Cameca "RRNG" files are accepted. For information on the accepted file formats, see the Appendix.
- "Drop unranged": This causes any ions not ranged to be silently dropped from the filter output. This is best enabled for 3D viewing, and best disabled for spectrum plotting

4.8.4 Clipping

This filter allows for the rejection of data that does not lie within some given boundary. Possible boundaries are plane, sphere and cylinder. For example, if the sphere mode is set, ions within the sphere will be kept and propagated. Ions outside the sphere boundary will be dropped.

- Mode: Select the fundamental primitive used to divide the incoming ions into two groups (inside and outside). Sphere, Cylinder and Plane modes are available.
- Invert clip: Reverse the action of the filter, *i.e.* swap the definition of "inside" and "outside".
- Various positioning parameters; These can be typed in manually, or set by manipulating the clipping object in the 3D view with the mouse.

4.8.5 Spectrum

This will generate a histogram of the "value" of ions passing through the filter. Note that no output other than the histogram is generated. Plots can be assigned a colour, set to logarithmic or non-logarithmic mode, or restricted to only cover a specific region. The plot title is taken from the filter name, some limited LaTeXis supported (note that the "\" symbol is a special LaTeXcommand; you may need to use "\\" to represent a single "\" in the title), for example to type "My Spectrum A\B" you would actually name the filter "My Spectrum A\\B".

4.8.6 Profile

The profile filter conducts a density or "compositional" analysis of a given sub-region of 3D space. The action of the profile filter depends upon whether the incoming ions have been "ranged". If not, then the profile filter generates a density profile of the ions inside a cylindrical volume by count, which is visible in the 3D view. If the ions have been ranged, then the frequencies are on a per-species basis. *Properties*

- Normalise : The action of this option converts the density into a fractional one. For ranged ions, this is the local composition. For unranged ions this is the relative density.

4.8.7 Spatial analysis

This filter conducts spatially oriented data analysis of incoming ions, and reassigns the 'value' component of the ion data. The nature of the reassignment depends upon the selected algorithm and the incoming

data itself. Note that the exact values computed by the spatial algorithms may be affected by subsampling; however trends are usually unaffected, provided the number of incoming data elements is sufficiently large.

- Algorithms
  - Local Density: This computes the local density of the ions on either a nearest neighbour, or a fixed distance metric. The density is then assigned as the point value. Note that the number of points to be examined increases rapidly in the fixed distance metric, and may rapidly become untenable. Clipping the volume of data to reduce the time is an option, however surface effects can occur.

4.8.8 Clustering analysis

The cluster analysis filter is designed to aid in the detection and analysis of spatial clustering in segregated data. Cluster analyses are used to determine the extent of non-random spatial relationships between dataset members.

The method works by identifying two data types – "core" and "bulk" members of the dataset (in APT these are usually called "solute" and "matrix"). The program attempts to determine adjacencies between core elements, and to group them together, extracting them from the bulk of the dataset. To do this, the filter uses the scalar value associated with each point to classify it. Range data (which must be present as a parent filter), is used to identify regions of value to classify value regions. Each of these regions then can be selected to belong to either the "core" group or the bulk group – but not both.

The clustering algorithm implemented in *3Depict* is a modification of the clustering algorithm outlined in Stephenson *et al*[**?**], and to a lesser extent Hyde *et al* [**?**] and Vaumousse and Cerezo [**?**][4].

Parameter Description:

- **Core classification distance** : This distance is the maximum distance between which items initially marked "core" by their value can be separated from another core point (up to Core kNN Max) in order to not be discounted in the clustering. This aids in removing isolated points that are initially marked as core. This option is disabled if the value is set to 0.
- **Core kNN max** : The Core k-th nearest neighbour maximum for core classification. This modifies the core classification stage, only looking up to some max kNN (unclassified core only) for other core points

Algorithm Description; each of these is conducted in sequence to generate the final clustered output.

- **Core Classification (Optional,*Core Classify Dist* nonzero)**: Core classification; work only on core ions (bulk is ignored). Each "core" point has sphere of specified size placed around it, if point's kth-NN is within a given radius, then it is used as core, otherwise it is rejected to "bulk".
- **Cluster Construction**: A "backbone" is constructed using the core points (after classification). Each core point has a sphere placed around it of fixed size; if it contacts another point, then these are considered as part of the same cluster.
- **Bulk Inclusion (Optional, *Bulk Link Dist* nonzero)** : For each cluster, every point has a sphere placed around it. Bulk points that lie within this union of spheres are assigned to the cluster. This assignment is unambiguous *iff* this radius is smaller than that for the cluster construction step

[4]These sources are not freely available. Some of these concepts are discussed by in this work which is available online : "Design in Light Alloys by Understanding the Solute Clustering Processes During the Early Stages of Age Hardening in Al-Cu-Mg Alloys" ; `http://hdl.handle.net/2123/4008`.

- **Bulk Erosion (Optional, *Erode Dist* nonzero)** : Each unclustered bulk ion has a sphere placed around it. This sphere strips out clustered "bulk" points from the cluster and returns them to the unclustered data. This is only done once (*i.e.*, not iterative).

Note that there are more steps listed in the filter progress due to the need to generate data query structures.

Several post-processing options are available as part of the filter. The size distribution (number of items) can be computed, as can the composition. A frequency table is generated and printed to the program console. Note that the "count bulk" parameter specifies whether to include points classified as "bulk" in these frequency and chemistry tables or not.

Whilst much effort has been placed into optimisation of the clustering algorithm, the query itself is quite slow. The clustering algorithm is best operated on a small region of data to optimise the parameters prior to applying the algorithm to the full dataset.

4.8.9 External program

This allows the program to run external commands on the system in order to link into other programs. Note: Loading a state file with this filter will result in the user being prompted to the existence of "potentially hazardous elements" in the filter tree, and will give the user the option of removing them. If you are presented with this warning you are highly recommended to discard these elements unless you know better, as it is possible for arbitrary computer programs to be executed if you accept these elements.

Command syntax: The syntax for specifying the program command uses % as the escape character. If you wish to pass a single % to the command line, you can use %%. %i will be substituted with the first pos file's name, repeated uses will use the second, third and so on-th pos file name. If there are not enough incoming ion streams to be converted to pos files, then the filter will report an error. You can use %I to substitute all pos files (space separated) to the command line at once. Similarly %p and %P will substitute for plots. Unrecognised % sequences will be considered an error.

Prior to program execution: Ion data coming into this filter will be saved in the folder "inPos" inside the specified working directory, with the prefix "pos", in the pos format (Section 7.1.3). Plots will be saved as tab separated files with the prefix "xy". If there is no input to the filter, and thus no files, the program will not be run. By default, the program that is excecuted will have these files passed as arguments to the function, appended to the output if no % syntax is used.

At this point, the target command will be run. *3Depict* will halt at this point, and await the completion of the underlying program.

After program execution: Once the program is run, any .pos files (*i.e.* any files matching '*.pos') in the working directory will be loaded back as ion streams. Similarly any '*.xy' files will also be loaded. .xy files should be ASCII files, and should have a multiple of 2 columns (one for x, one for y) separated by a valid delimiter. The x and y column lengths must also match for each x-y pair. Valid delimiters are tab, comma and space. At time of writing, there is currently no way to specify the plot colour or style. The x-y values will, by default be connected with line, thus a single value will not be clearly visible

5 Attributions

- The source code image is a derivative work of `http://commons.wikimedia.org/wiki/File:User_icon_2.svg` and `http://commons.wikimedia.org/wiki/C_cplusplus_compilation_process.svg`.

- The camera image is a derivative work of `http://commons.wikimedia.org/wiki/File:Icon_Camera.svg`.

6 Licence

7 Appendices

7.1 File formats

7.1.1 State file

The state file is an XML file, which is generated by *3Depict*. XML is short for eXtensible Markup Language, and describes a basic format for data layout. In XML files, the file consists of elements, attributes and text. Each element is marked by the use of angle brackets, as in "¡element¿ ". Each element must have a start and an end marker, for example ¡element¿ is the start marker, and must be followed by its end marker ¡/element¿. These elements can be nested or in sequence, but cannot be mixed (it is wrong to say ¡element1¿¡element2¿¡/element1¿¡/element2¿).

A full description of the XML language is beyond the scope of this document, however many resources can be found online to explain the concept. The extensible bit means that *3Depict* can define its own elements. Hence the exact format is subject to change from version to version.[5]

This is due to the rapidly changing nature of the filter properties. Unfortunately the most up-to-date documentation for the file format is the source code itself. For those who may be inclined to try to emulate this, you may wish to look at the VisController::saveState routine, and the *Filter*::writeState routines.

However, in general the file is divided into several sections. Below is an example file.

```
<threeDepictstate>
    <writer version="0.0.1 Rev:232 (5e44e97bbba1)"/>
    <backcolour r="0" g="0" b="0"/>
    <filtertree>
        <posload>
            <userstring value=""/>
            <file name="/home/user/data/data.pos"/>
            <columns value="4"/>
            <xyzm values="0,1,2,3"/>
            <enabled value="1"/>
            <maxions value="327680"/>
            <colour r="1" g="0" b="0" a="1"/>
            <ionsize value="2"/>
        </posload>
        <children>
            <iondownsample>
                <userstring value=""/>
                <fixednumout value="1"/>
```

[5]Technical note: As of time of writing, I have not created a Document Type Descriptor (DTD) for the file which fully describes the file format. This may be done in future versions.

```
                <fraction value="0.1"/>
                <maxafterfilter value="5000"/>
            </iondownsample>
        </children>
    </filtertree>
</threeDepictstate>
```

The state consists of the program version, to check that the program can actually interpret the file, a background colour, and the filter tree.

Optional elements which are not shown in this example include the stash data, and the camera information (here there is only the default camera).

The filter tree is shown, with a pos load filter as the top level element, which has a child element of iondownsample. Note that the attributes of each element are dependant upon the filter. Again due to the rapidly changing nature of the program, this is subject to change.

7.1.2 Range files

My 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). Unfortunately, the specification given for the file is weakly stated, and is open to different interpretations. I have attempted to make*3Depict* as resilient as possible to variations that I have encountered, however it may be that there are alternate interpretations with which I am not familiar, and the code is thus unable to interpret.

A simple example file is given below, and is nominally in the ASCII 1 byte per character format. The original specification, to my knowledge, predates the UTF-8 and extended codepage support for non English languages. Thus non-English language support is not guaranteed,

```
1 2
Aluminium
Al 1 1 1 Al
------------- Al
. 100 150 1
. 150 200 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. The final string is, to my knowledge, vestigial.

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.

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.

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

In the *3Depict* implementation, 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
Alumnium
Mg 0.0 0.0 0.0
Copper
Cu 0.0 0.0 0.0
Nickel
Ni 0.0 0.0 0.0
------------ Al Cu Ni
. 25 27 1 0 0
. 25 33 0 1 0
. 55.6 59 0 0 1
```

Note that*3Depict* guarantees to be able to read its own range files, and will do its best to read files generated by any major external program (within reason). If you have a file that you believe should be accepted, please contact the author.

7.1.3 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. *3Depict* will accept files with different numbers of records (eg XYZ only, or XYZMI (where I is ignored)), but this must be manually specified.

7.1.4 Text files

The text files that are accepted by *3Depict* must be ASCII formatted, and consist of at least four columns of data, separated by an acceptable delimiter. The accepted delimiters are currently tab, space and comma. The numeric format must be in the English locale, i.e. with a period used as the decimal separator, consisting of the digits 0–9 and the + and - symbols.

Each file may have a contigous header that does not consist of this format, however if ANY portion of the header is interpretable as per the above, this will be considered to be the end of the header, and the remainder of the file is the file body. All lines in the file body must be intepretable as per the above.

Note that due to the need to do multiple passes over the text file to interpret it, and the need to do string to binary conversions, this will be considerably slower than using a POS formatted file for large inputs.

7.2 Modifying the program

As *3Depict* is an open source program, you are free to modify it, or to extract useful bits subject to the licence agreement (See Section 2.1.1). You will need a knowledge of C++ in order to reasonably understand the components of the program itself. A knowledge of OpenGL and wxWidgets is desirable, but you could pick this up as you went along, and don't really need it for many parts of the program.

This section of the manual is the hardest to write, and the most likely to not be applicable to your context, as it depends heavily on the computer system you are trying to use. Nevertheless, this section will attempt to explain how to get yourself set up to build. To modify the program, you must first be able to build the base version of the program from source code. This is by far easiest under a linux system, as packaging programs can allow you to auto-import all the needed components to build the program.

7.3 Development tools

The program was primarily developed using C++ (gcc), and utilises autotools for the build scripts. Some custom Bourne-again shell (BASH) scripts are used to do side tasks, such as dependency retrieval and compilation and .app package building (for OSX, really). Mercurial is used for version control. The program is developed using a private repository, which I sync up to the Sourceforge repository periodically. My personal tools for development are the VIM editor and the command line. This was primarily developed under a Debian squeeze (testing) system (EEE 901), with some development under OpenSuse. I actively maintain the programs' package for Debian, and this is periodically synchronised to Ubuntu's package database. The program is also available under the Fedora platform.

The main libraries used for the program are:

- wxWidgets
- mathgl
- ftgl
- libxml2
- qhull

7.4 Getting yourself set up

Compilation instructions vary from operating system to operating system. In increasing order of complexity to generate a compilation, Linux, mac, and windows versions can be built from source. Instructions for compilation change frequently, and the most up to date version is available on the project website.

If you are running a Debian or Debian derived program, all you need to do is to run these commands as an admin user `sudo aptitude install build-essential`, which will install a compiler and the needed build scripts. Then run `sudo apt-get build-dep 3depict`, this will install all the needed components to install the program.

Once this is done, you can download the latest source code from the website, unzip it, and then run `./configure && make`. This builds the program. You can now modify any of the files, then recompile it simply running `make`.

7.4.1 Changing stuff

All you have to do now is to modify the .cpp and .h files to do what you want, this is going to be specific to what you want to do, and thus it is impossible for a "walkthrough" to be reasonably written. This is only here as a guide. To get started, the easiest components to change are probably the filters. These have been written to be as independent of the user interface as possible, so you need to know very little about OpenGL or wxWidgets.

Each filter is an object derived from a base class. If you don't know what that means, you may need to revise your C. To implement a new filter, you have to derive a new class and implement the pure virtual functions to do what you want. To make it accessible from the user interface, you have to add a

new entry in `comboFilters_choices` in the function `MainWindowFrame::MainWindowFrame(...)`, and in `MainWindowFrame::OnComboFilter(...)` in `3Depict.cpp`. You can probably just copy the relevant bits from a neighbouring filter.

The main trick to realise is that each filter takes in something, and spits out something. That is, each filter does all of its work in the `::refresh` functions. The easiest examples of this will probably be the ion downsampling and transform filters. They might look a bit daunting, but much of the code is simply there to keep things running as fast as it can, and to provide many options. Each filter can only work with its own properties, and that of the FilterStreamData pointers passed into the refresh function. It can really do anything it wants here.

To have properties appear up in the left hand panel, you have to implement the `getProperties(...)` function – try copying one that seems closest to your situation. To have these properties take effect, you need to implement the `setProperties(...)` function. If you wish to "peek-ahead" at the filters coming into the filter (this is a little advanced, but can sometimes be necessary), you have to implement the `initFilter(...)` function.

References

[1] Leigh T. Stephenson, Michael P. Moody, Peter V. Liddicoat, and Simon P. Ringer. New Techniques for the Analysis of Fine-Scaled Clustering Phenomena within Atom Probe Tomography (APT) Data. *Microscopy and Microanalysis*, 13(06):448–463, 2007.

[2] J.M. Hyde, E.A. Marquis, K.B. Wilford, and T.J. Williams. A sensitivity analysis of the maximum separation method for the characterisation of solute clusters. *Ultramicroscopy*, In Press, Corrected Proof:– , 2010.

[3] D. Vaumousse, A. Cerezo, and P.J. Warren. A procedure for quantification of precipitates microstructures from three-dimensional atom probe data. *Ultramicroscopy*, 95:215–221, 2003.