Tutorial - Sample reconstruction from 2D EBSD data for PRISMS-Crystal plasticity finite element code using Dream.3D

Aaditya Lakshmanan[1] and Mohammadreza Yaghoobi[2]

[1]Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109, USA

[2]Department of Materials Science and Engineering, University of Michigan, MI 48109, USA

Introduction

This tutorial is a guide towards synthetic microstructure reconstruction using DREAM.3D© to generate the microstructure data in a format compatible with PRISMS-CPFE. PRISMS-CPFE uses 2 separate files - (i) **grainID.txt** and (ii) **orientations.txt**. **grainID.txt** contains the grainIDs for every voxel arranged in a specific order based on the convention chosen by DREAM.3D, the convention being specified in the file. **orientations.txt** contains the orientation information in terms of the Rodrigues vector for each grainID.

Procedure

The package on which the present procedure has been tested, is DREAM.3D Version 6.4.197. One should note that Dream.3D incorporates the passive[1] convention of rotations and the necessary transformations on the EBSD data must be performed to ensure this convention is obeyed. On running the DREAM3D executable two windows should pop up - (i) The DREAM3D toolbox containing all the filters required to construct a pipeline and (ii) The pipeline window where filters can be assembled and the pipeline can be created and tested . Filters will be copied from the toolbox to the pipeline window sequentially, after which the pipeline will be started to produce the intended result. For details pertaining to specific filters refer to the DREAM3D manual.

1. Copy the **StatsGenerator** filter to the pipeline window. This filter is useful to generate the statistics based on which the microstructure is reconstructed. The statistics referred to here is the grain-size distribution, ODF(Orientation Distribution Function), shape distribution and MDF(Misorientation Distribution Function). These can be specified either through the parameters window or an external file. Before we start specifying this microstructural information, we need to specify something about the material itself. This can be done by clicking on the yellow gear button on the top-left corner after which a window pops up to input phase properties. Select crystal structure as **Hexagonal**, **Fraction** as **1**(we are dealing with a single phase polycrystal) and **Phase Name** as '**Magnesium**', as shown in Fig. 1

2. Ensure that the tab displaying **Size distribution** is currently active or it can be made so by just clicking on it. The grain size distribution should be selected in a way that it can mimic that of the real specimen. In the present case, the selected values are - Mu = 4, Sigma = 0.5, Bin Step Size = 6.0000, Min Cut Off = 2 , Max Cut Off = 1.5(Fig. 2). This will result in an estimated 16 feature bins. For the **Preset Statistic models** choose **Primary Equiaxed** because we will be working with equiaxed grains. Then click on the **Create Data** button to update the statistics for the new set of parameters.

[1]Passive convention here refers to the passive transformation, in which orientation descriptors are specified in a way to transform the global coordinate system to a local coordinate system. In the present case it is the transformation from the sample reference frame to the crystal reference frame.

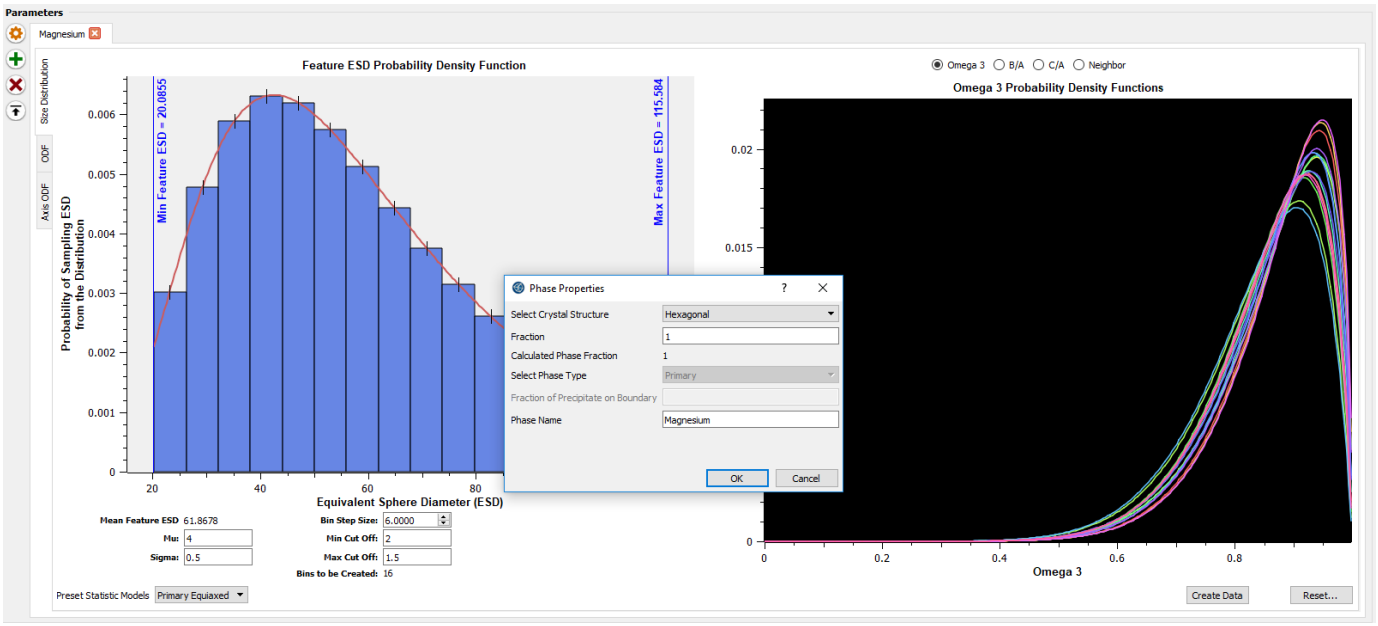

Figure 1: Size distribution

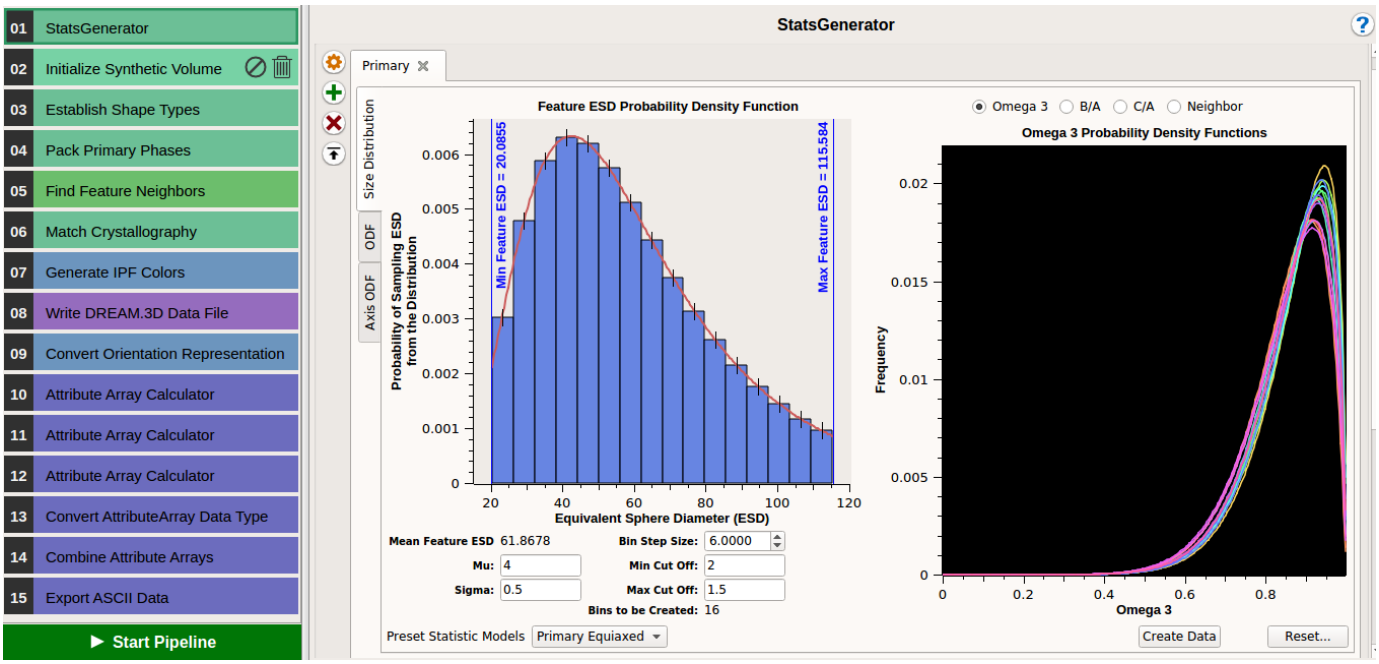

Figure 2: Size distribution

3. Notice that the 2 tabs displaying **ODF** and **Axis ODF** are accessible. Click on the **ODF** button and click on the radiobox under **OFD parameters** displaying **Bulk Load From File**. The first textbox asks for the input file which can be added by clicking on the **Select** button and choosing the required file. In the present case, the Euler angles are specified in radians hence, leave the box next to **Values in Degrees** unchecked. Choose the angle representation as **Euler** and the delimiter as **Space**, as in Fig. 3. Then click on the button displaying **Load Data** to generate the pole figures.

   Alternately, one can explicitly specify the texture data by specifying the Euler angles, the intensity

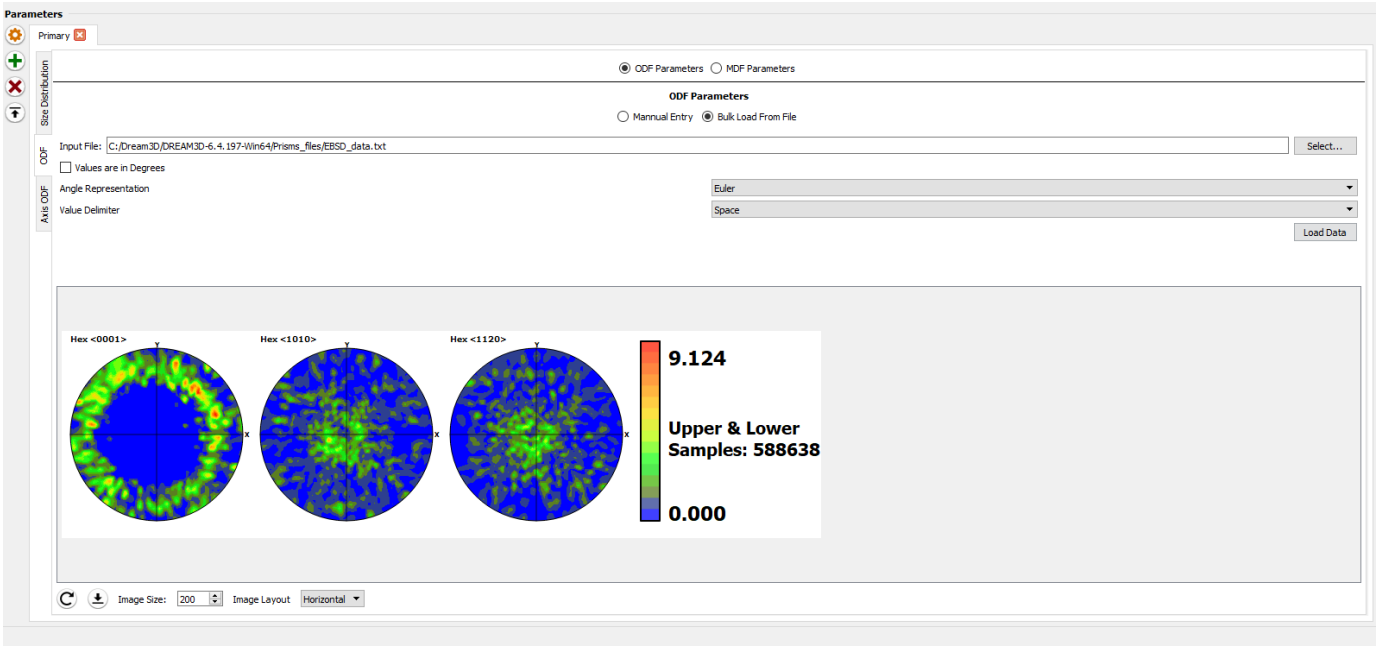

Figure 3: Reading orientation data from an external file

and spread via the set the parameters as follows - Euler 1 = 0 , Euler 2 = 0 , Euler 3 = 0 , Weight = 50000, Sigma = 14(Fig. 4. Pressing the keyboard **Enter** key will update the ODF and generate the corresponding pole figures. Crystallographic texture is controlled using these parameters. The 2D EBSD data should be used to generate the sample texture.

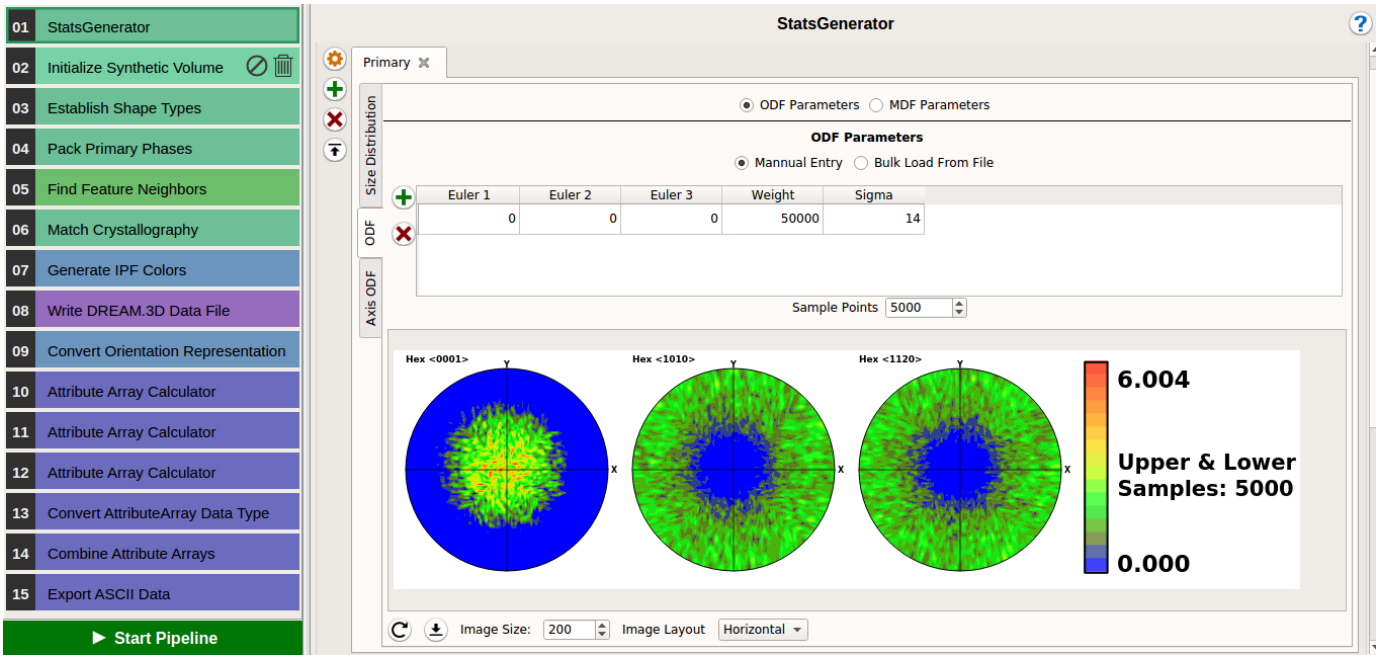

Figure 4: ODF

4. Click on the tab displaying **Axis ODF** and set the parameters as follows - Euler 1 = 0, Euler 2 = 0 , Euler 3 = 0 , Weight = 150 , Sigma = 3 (Fig. 5) - and press the keyboard **Enter** key to update the

axis ODF. Morphological texture is controlled using these parameters.

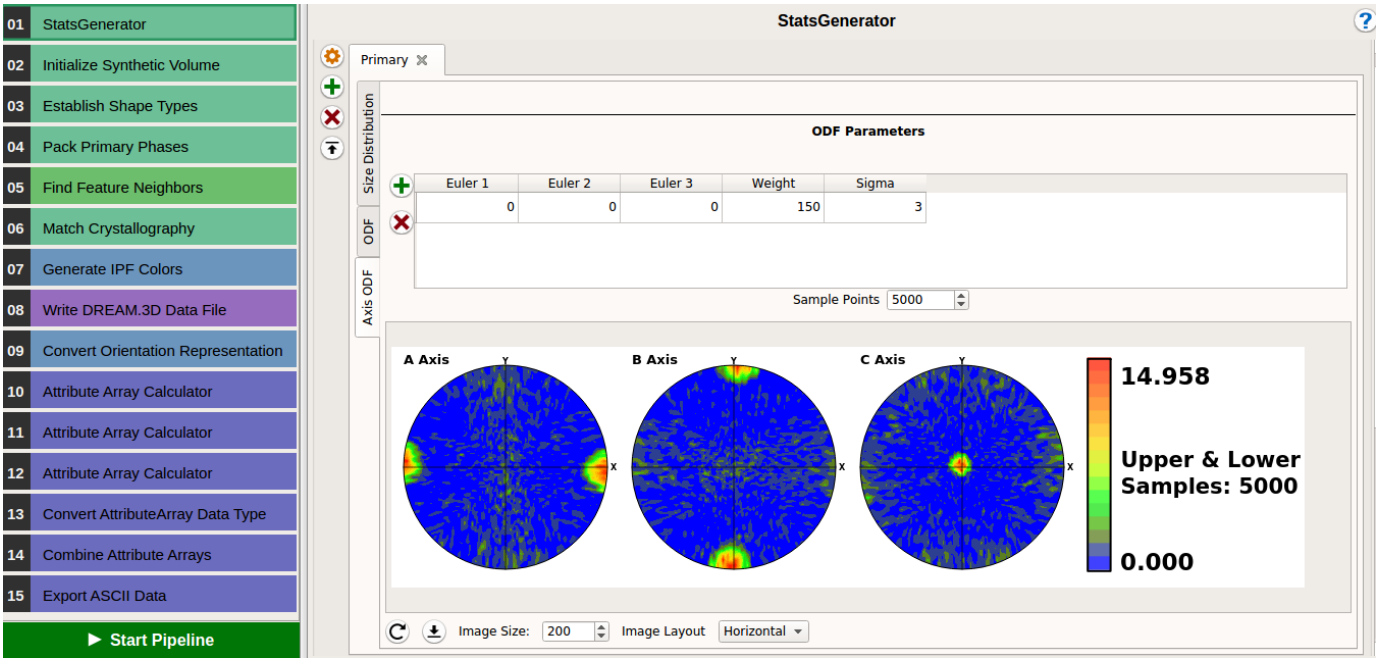

Figure 5: Axis ODF

5. Copy the **Initialize Synthetic Volume** filter to the pipeline window where the box dimensions can be specified. The box is split into voxels according to the dimensions as follows - if the dimension value in the first textbox is 50, then the box will be sectioned into 50 partitions along direction '1' to produce 50 voxels in that direction. Similarly the second and third textbox refer to the dimensions along directions '2' and '3' respectively. The resolution refers to the length of the box along each direction, so that the larger the resolution, the more grains generated for a given size. By ticking the checkbox next to **'Estimate number of features'** one can check the estimated number of grains that would be generated for the chosen dimensions and resolution. In the present case, we choose a dimension of 10 in each direction with a resolution value of 10(Fig. 6) . Note that the number of features estimated can change on multiple tests because the entire processing is essentially that of sampling based on a probability distribution.

6. Copy the **Establish Shape Types** filter to the pipeline window. Here one can choose the approximate shape of the grain. The available options are ellipsoid, superellipsoid, cube octahedron and cylinder(A,B,C). We choose the grain shape as an ellipsoid (Fig. 7).

7. Copy the **Pack Primary Phases** filter to the pipeline window. This filter will place primary Features with the sizes, shapes, physical orientations and locations corresponding to the goal statistics. Additionally, periodic boundary conditions are enforced by checking the box next to **Periodic Boundaries**(Fig. 8), because in most cases an RVE is to be modelled.

8. Copy the **Find Feature Neighbors** filter to the pipeline window(Fig. 9). This filter looks for features neighboring any particular feature, and this action is performed over all generated features.

9. Copy the **Match Crystallography** filter to the pipeline window, which attempts to match a predefined orientation distribution function (ODF) to a set of Features(Fig. **??**). Once the pipeline is started, this step is the most time-consuming process because a number of iterations are performed until a satisfactory match is established between the supplied statistics and the those synthetically generated.

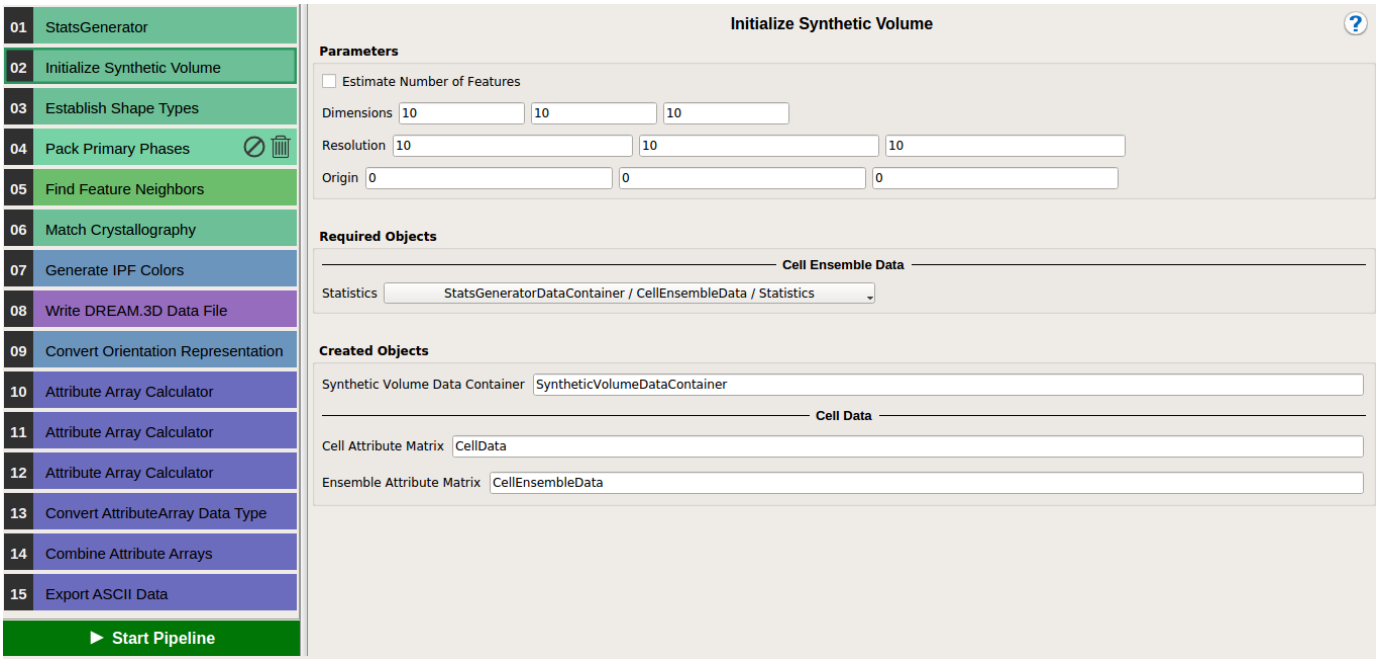

Figure 6: Initialize synthetic volume

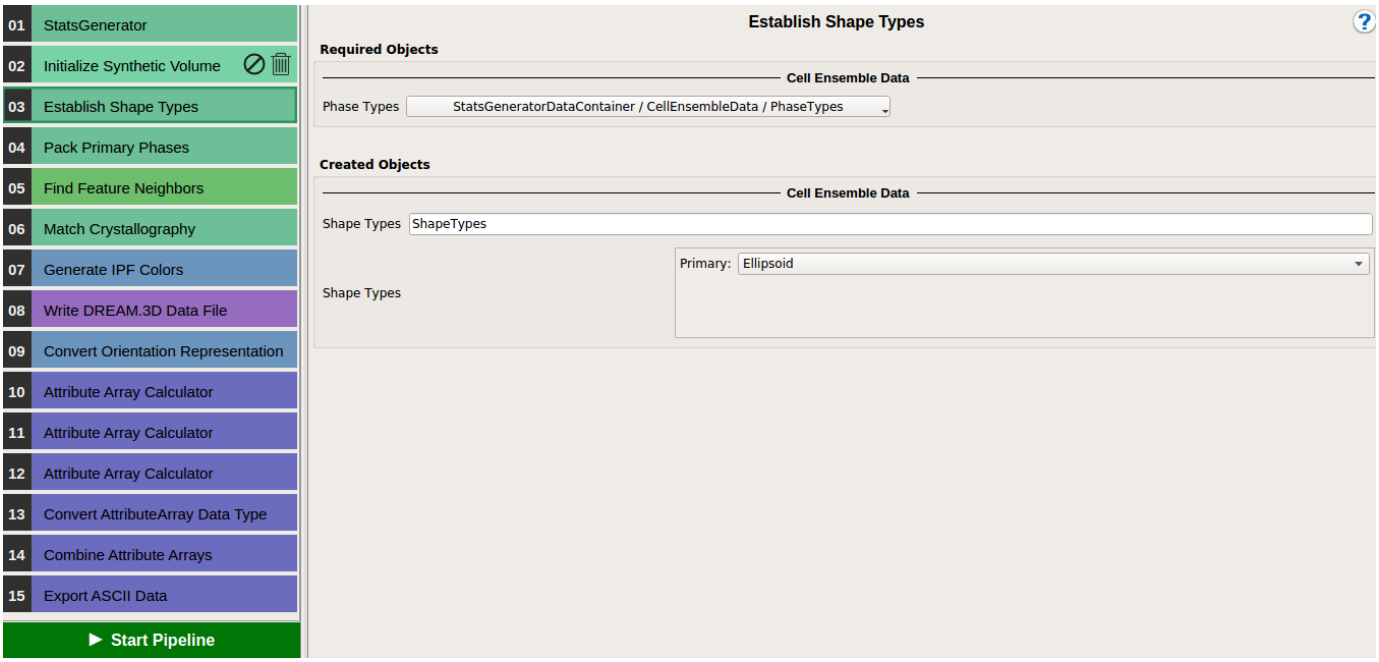

Figure 7: Establish shape types

10. Copy the **Generate IPF Colors** filter to the pipeline window to render colors to every grain depending on the orientation representation in the stereographic triangle. Set the reference direction to [**0 0 1**](Fig. 11).

11. Copy the **Export Dx File (Feature Ids)** filter to the pipeline window. This filter generates an output file containing the grain-ids for all the voxels in the format of **grainID.txt**. In the present case, the output file is specified with the name **grainID.txt** as in Fig. 12. This file includes the information

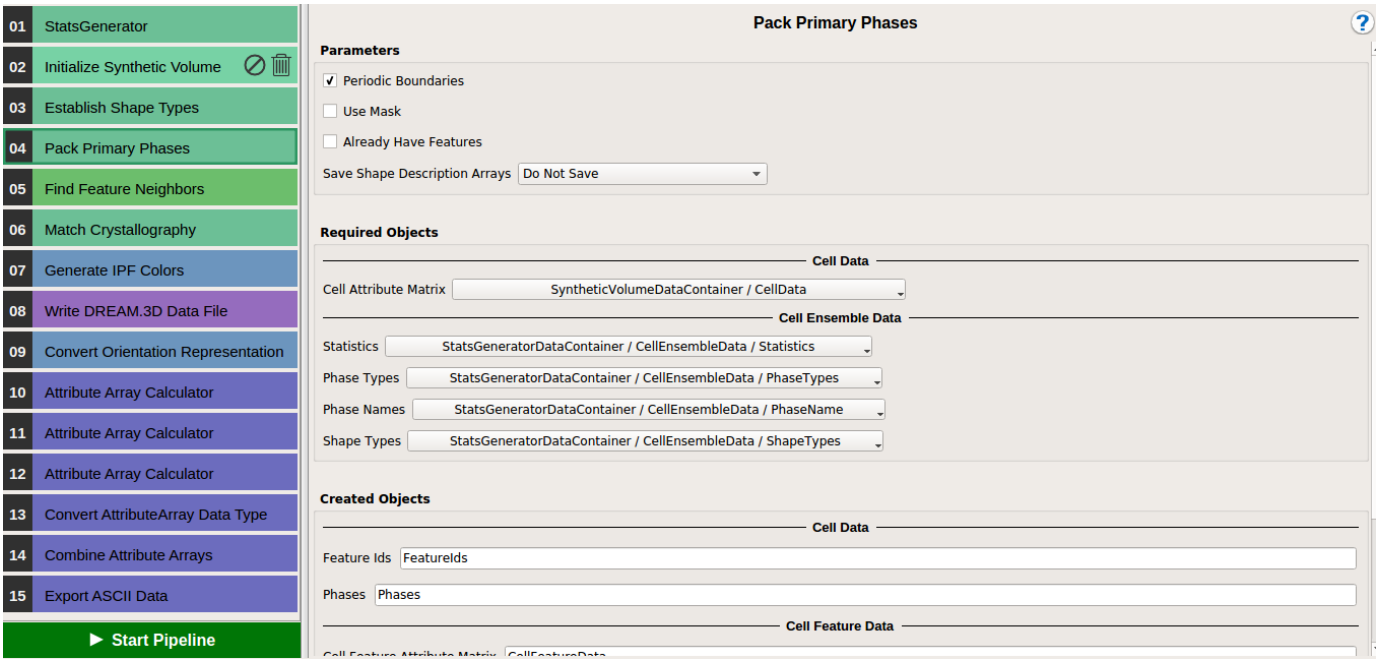

Figure 8: Pack primary phases

Figure 9: Find feature neighbors

for N1*N2*N3 voxels where N1, N2, and N3 are the values selected for **Dimensions** in **Initialize Synthetic Volume** (Fig. 12). The numbering convention is included in Appendix B.

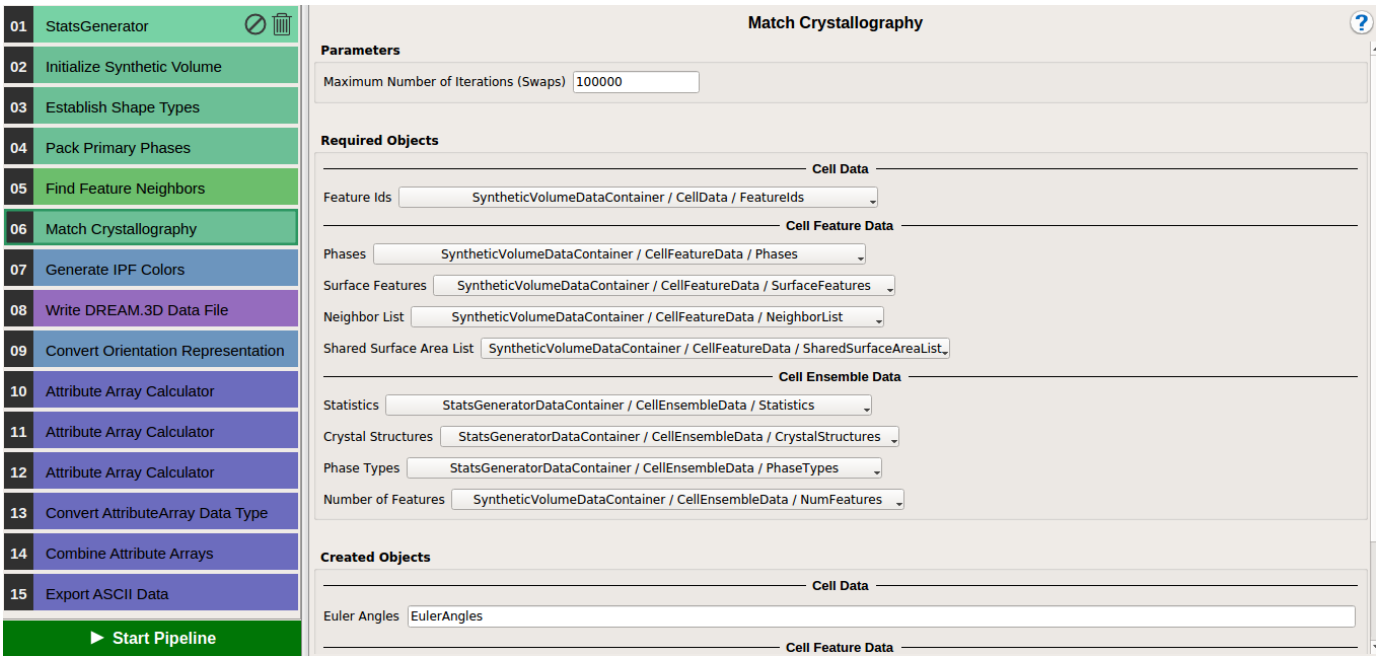

Figure 10: Match crystallography

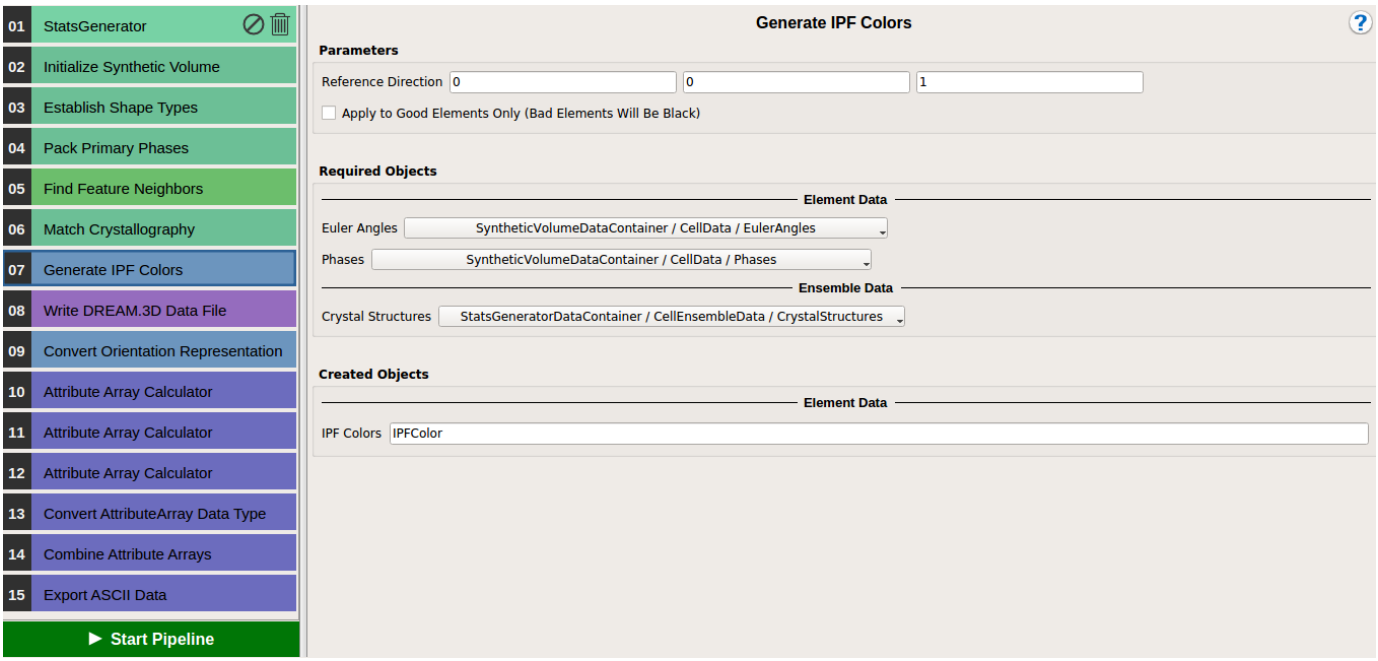

Figure 11: Generate IPF colors

12. Copy the **Write DREAM3D Data File** filter to the pipeline window(Fig. 13). This is performed so that the microstructure can be visualized in a visualization software like ParaView(Fig. 14).

13. We now focus on generating the input file **orientations.txt**. PRISMS-CPFE reads the Rodrigues vector for the **orientations.txt** input file. Accordingly, some post-processing steps should be performed on the existing data containers to convert orientation information from Euler angles(Bunge) to the corresponding Rodrigues vector. To do this, copy the **Convert Orientation Representation** filter to

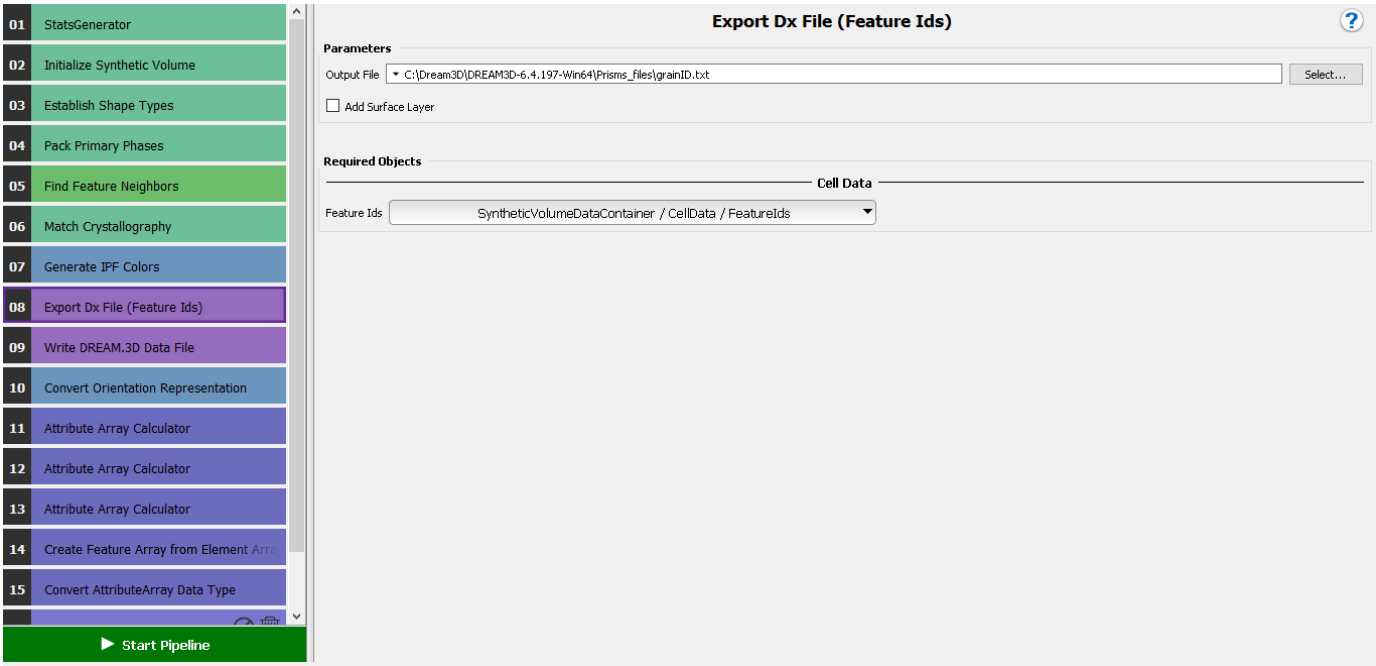

Figure 12: Export grainID.txt

Figure 13: Write DREAM.3D data file

the pipeline window. Then choose the input orientation type as **Euler** and the output orientation type as **Rodrigues**. For the **Input Orientations** options choose **SyntheticVolumeDataContainer / CellFeatureData / EulerAngles**, and prescribe a name for the output orientation. Here we choose the name **rvec**. We then obtain a 4-component vector in which the first 3 components are identified with the axis of rotation(a unit vector) and the 4^{th} component is $\tan(\frac{\theta}{2})$ where θ is the angle of rotation taken to be positive when anti-clockwise while staring into the vector. The output data container is

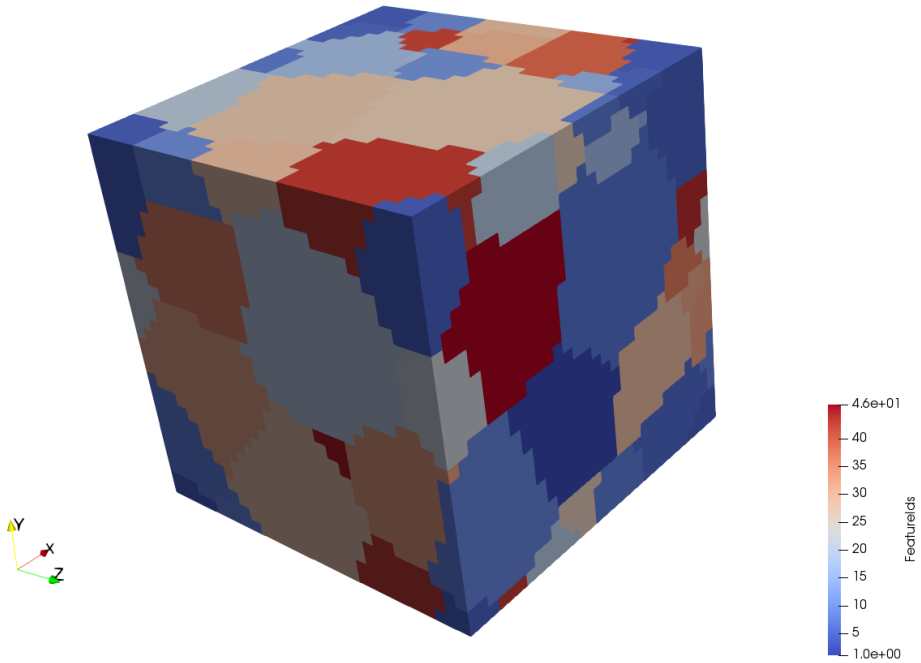

Figure 14: Sample microstructure visualized in ParaView

named **rvec**(Fig. 15).

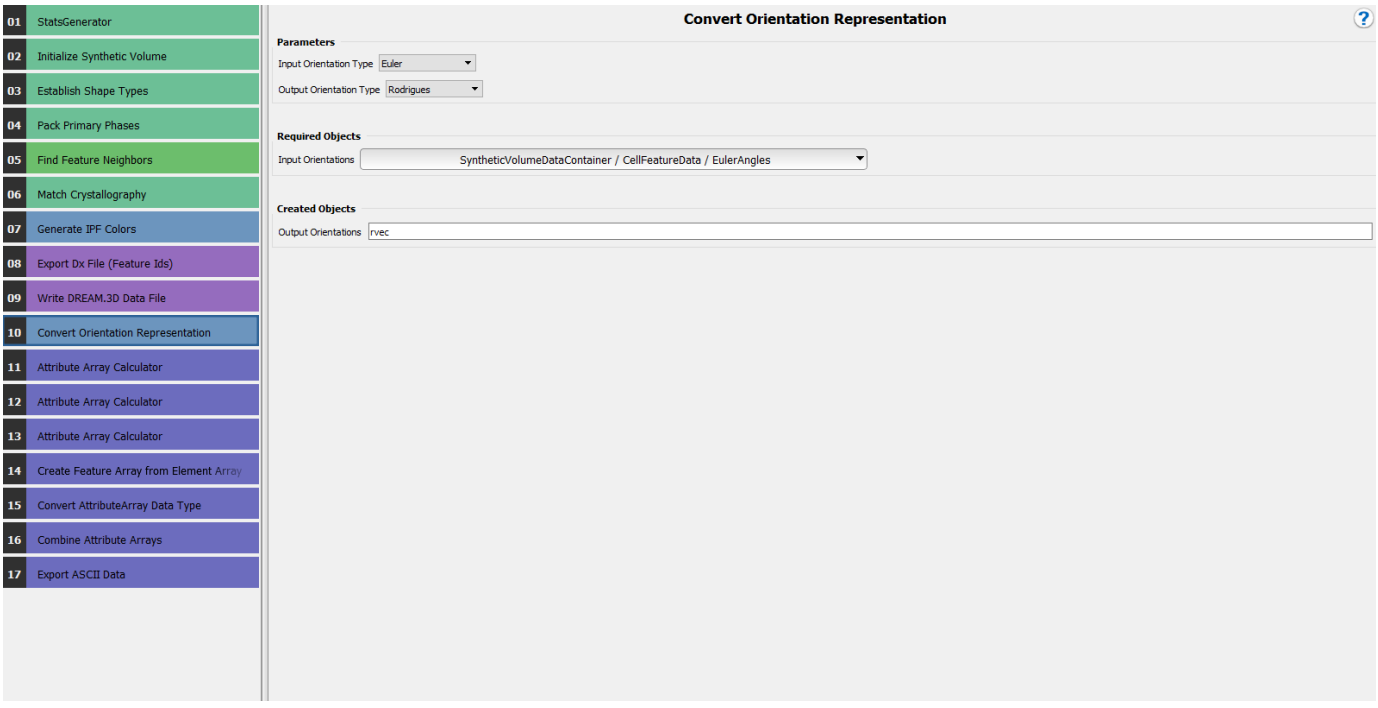

Figure 15: Convert orientation representation

14. In the previous step the Rodrigues representation of the Euler angles was obtained, which is a 4-component vector and not in its most concise form. PRISMS-CPFE reads the 'true' Rodrigues vector which can be obtained from a 4 component representation by multiplying the first three values (unit vector components) by the magnitude (the 4^{th} component). DREAM3D currently does not permit creating a vector of a smaller size from the given vector, hence we use a workaround. We separate out each individual component(which DREAM3D allows), perform some mathematical operations and recombine them into the desired 3-component vector. The following steps can be pursued in that direction.

    - Copy the **Attribute Array Calculator** filter to the pipeline window. Set **Cell Attribute Matrix** to **SyntheticVolumeContainer / CellFeatureData**. In the textbox type **rvec[0]*rvec[3]**, which means that we are multiplying the 1^{st} component of **rvec** with the 4^{th} component to generate the first component of the 'actual' vector. Under **Created Objects** set the option of **Calculated Array** to **SyntheticVolumeContainer / CellFeatureData** and the value as rvec_comp1(Fig. 16).
    - Copy the **Attribute Array Calculator** filter to the pipeline window. Set **Cell Attribute Matrix** to **SyntheticVolumeContainer / CellFeatureData**. In the textbox type **rvec[1]*rvec[3]**, which means that we are multiplying the 2^{nd} component of **rvec** with the 4^{th} component to generate the first component of the 'actual' vector. Under **Created Objects** set the option of **Calculated Array** to **SyntheticVolumeContainer / CellFeatureData** and the value as rvec_comp2.
    - Copy the **Attribute Array Calculator** filter to the pipeline window. Set **Cell Attribute Matrix** to **SyntheticVolumeContainer / CellFeatureData**. In the textbox type **rvec[2]*rvec[3]**, which means that we are multiplying the 3^{rd} component of **rvec** with the 4^{th} component to generate the first component of the 'actual' vector. Under **Created Objects** set the option of **Calculated Array** to **SyntheticVolumeContainer / CellFeatureData** and the value as rvec_comp3.

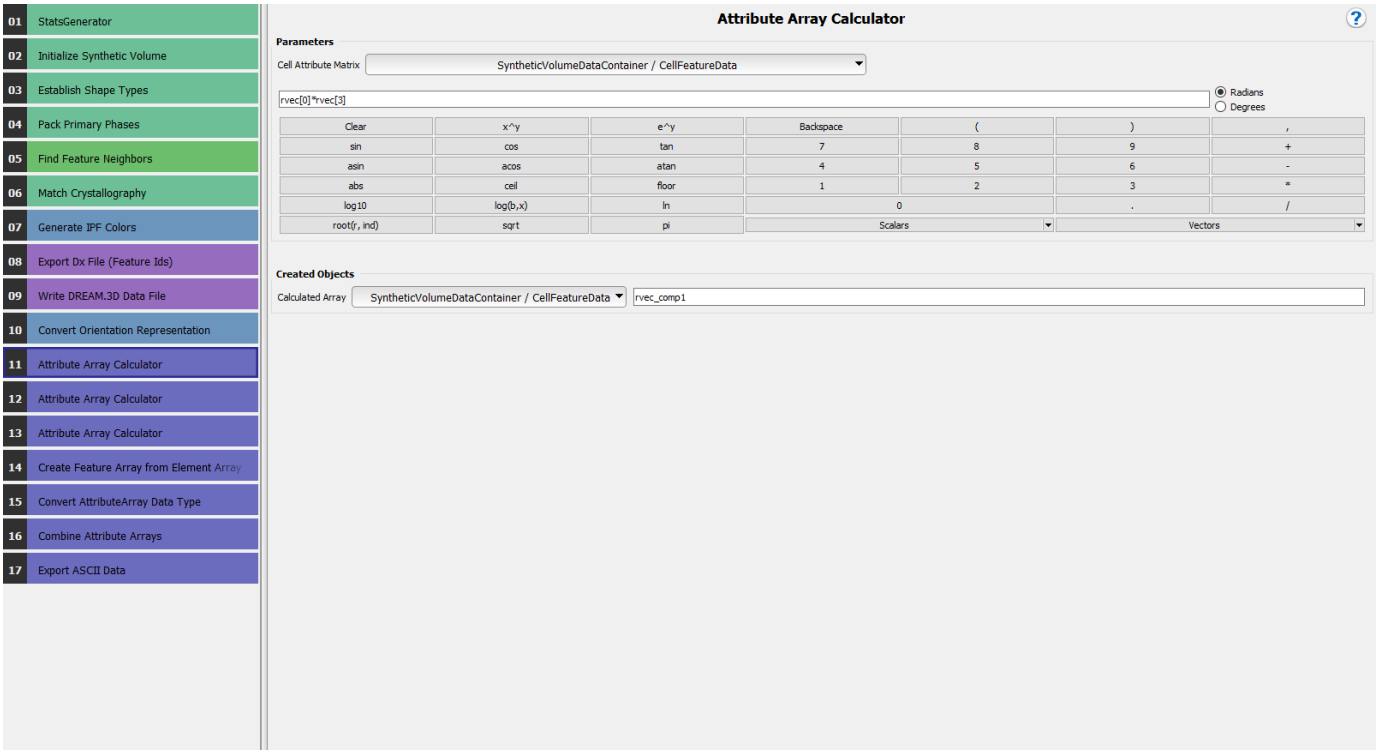

Figure 16: Attribute array calculator

15. From the previous procedure we have the Rodrigues vector. What we are left to obtain is the feature ids. For this copy the **Create Feature Array from Element Array** filter to the pipeline window. What we are doing here is to convert the feature ids available for every voxel into an array of distinct feature ids for the entire microstructure. In the **Required Objects** tab choose **SyntheticVolume-Container / CellData / FeatureIds** for both the options. In the **Created Objects** tab choose **SyntheticVolumeContainer / CellFeatureData** for the **Feature Attribute Matrix** option and name the **Copied Attribute Array** as **featids**(Fig. 17) .

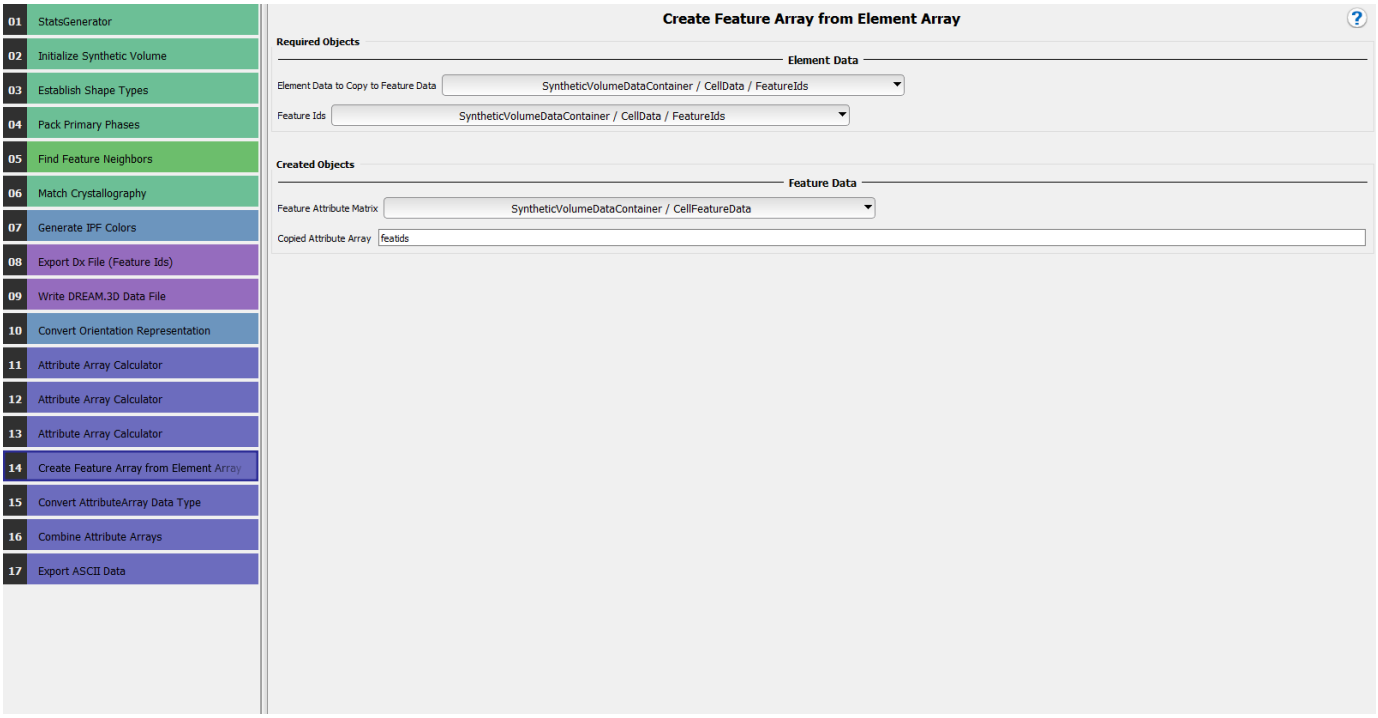

Figure 17: Attribute array calculator

16. We now wish to combine the feature ids and the Rodrigues vector into a single data container so that we can write it to an output file. However, the two data types don't currently match because the feature ids are 32 bit integers while the Rodrigues vectors are of type double. A conversion to data type double needs to be performed. To accomplish this, copy the **Convert AttributeArray Data Type** filter to the pipeline window. For **Scalar Type** choose **Double 64 bit**, for **Attribute Array to Convert** choose **SyntheticVolumeContainer / CellFeatureData / FeatureIds**, and name the **Converted Array Attribute** as **featids_conv**(Fig. 18). Any name can be specified though.

17. We are now ready to combine the required data into a single data container. Copy the **Combine Attribute Arrays** filter to the pipeline window. For **Attribute Arrays to Combine** choose **SyntheticVolumeDataContainer / CellFeatureData**. Then from the **Available Data Arrays** double-click on the names **featids_conv**, **rvec_comp1**, **rvec_comp2** and **rvec_comp3**. Name the new data container as **orientations**, beside **Combined Data** (Fig. 19). We are now ready to export this data to an ASCII file so that the PRISMS-CPFE code can read it in.

18. Copy the **Export ASCII Data** filter to the pipeline window. Specify the output path as required, file extension as **.txt**, maximum tuples per line as 1 and **(space)** as the delimiter. For **Attribute Arrays to Export** choose **SyntheticVolumeDataContainer / CellData**, and then double-click on the data container **orientations**(Fig. 20). This will eventually generate the file **orientations.txt**.

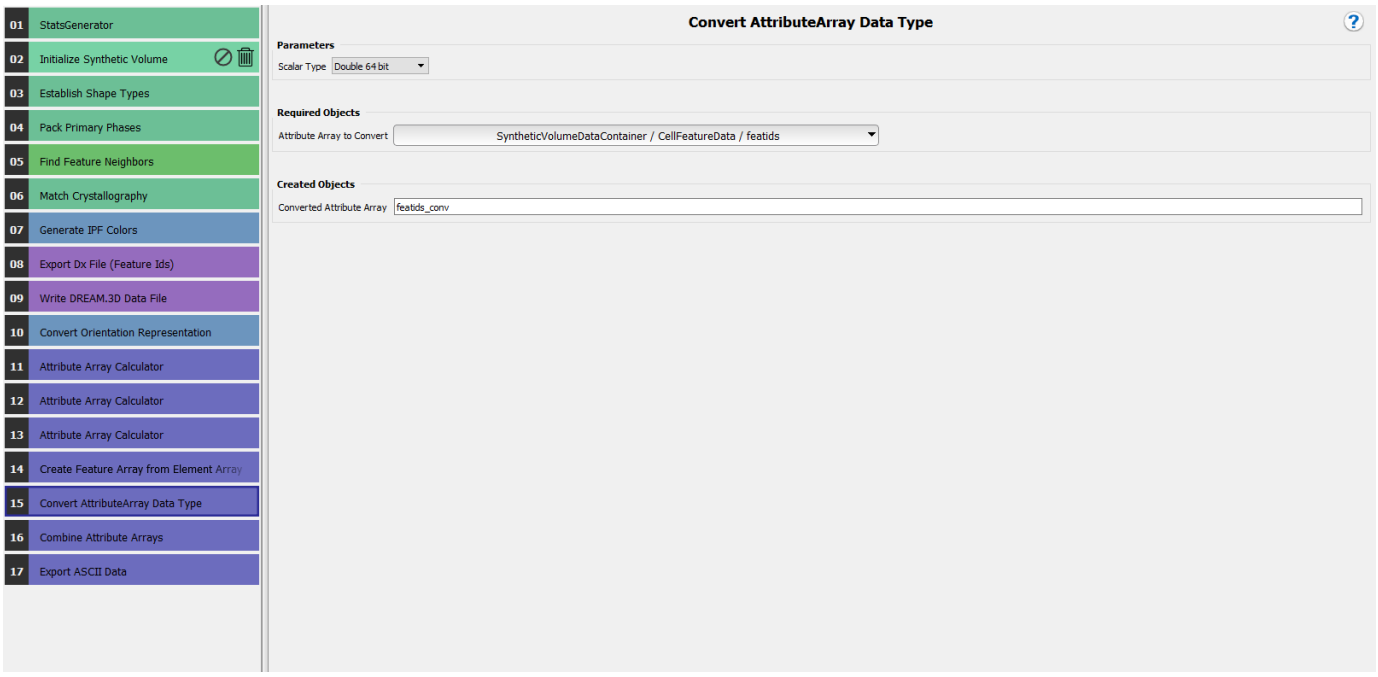

Figure 18: Convert attribute array data type

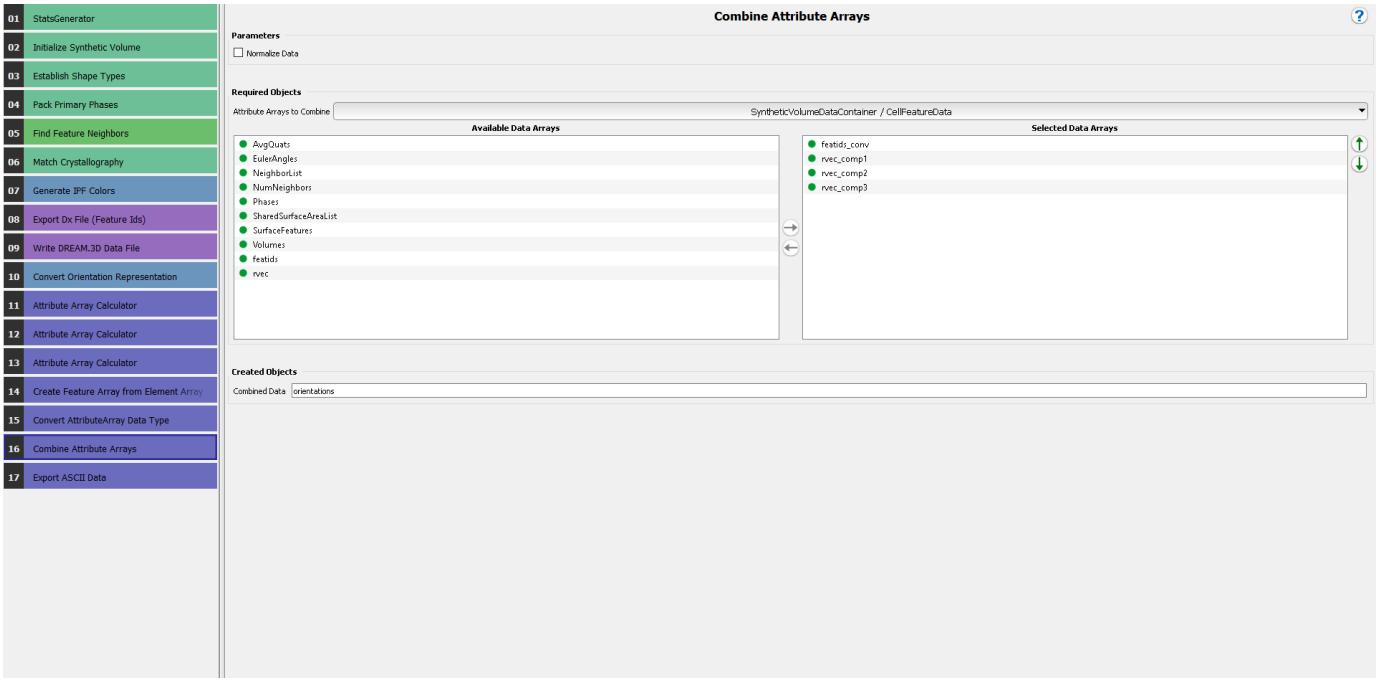

Figure 19: Combine attribute arrays

We have concluded the creation of the pipeline and are ready to generate the orientations file. Click on **Start Pipeline** and after some wait, the file **grainID.txt** and **orientations.txt** will be generated.

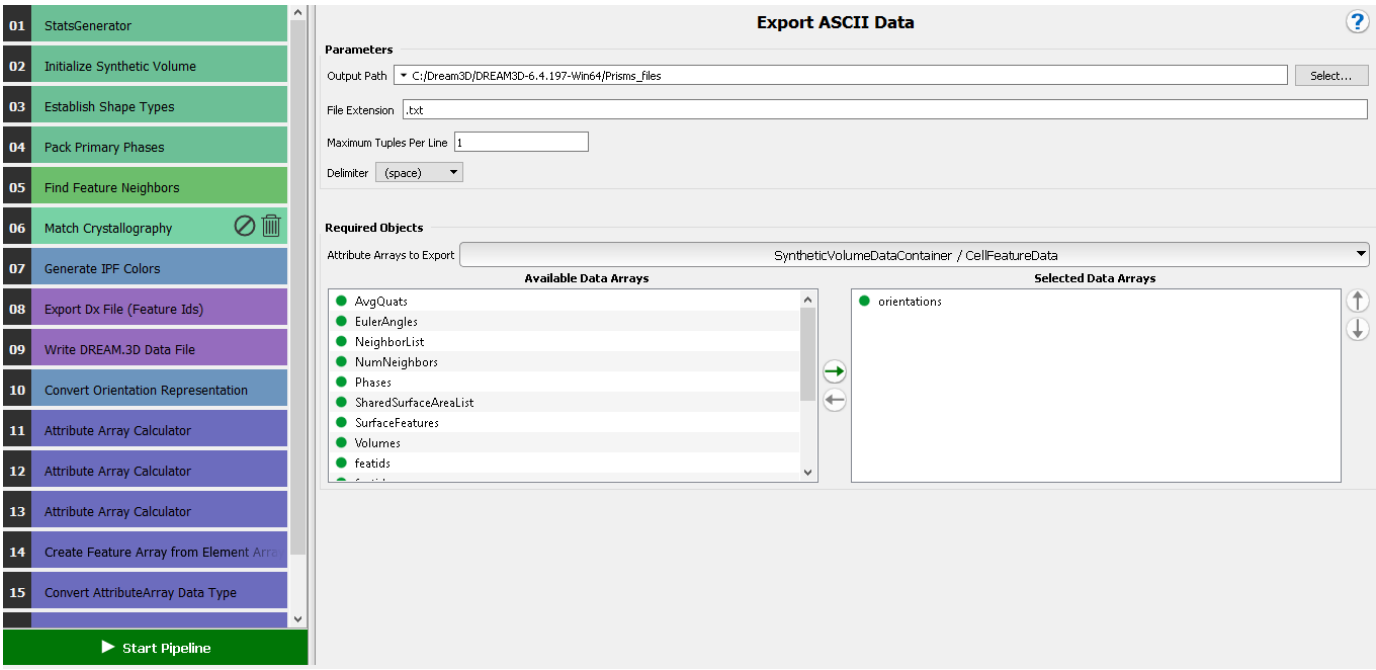

Figure 20: Export orientations.txt

Appendix A - Numbering convention in grainID.txt

Referring to the filter **Initialize Synthetic Volume**, let N_1, N_2, and N_3 be the number of voxels in directions 1,2 and 3 respectively. Then **grainID.txt** contains grain IDs for all the voxels specified in an arrangement consisting of N_3 columns and $N_1 * N_2$ rows. Referring to Fig. 14, there is a coordinate system associated with the microstructure aligned with the box edges. These are labelled as x, y and z. Every voxel can be uniquely identified by its position with respect to this coordinate system and for the structured grid, integer descriptors will suffice. For example, the corner voxel next to the origin may be indexed as $(1, 1, 1)$ because it is in the first voxel in x, y and z directions respectively. Using this notation, any voxel can be indexed as (i, j, k) with $i \in \{1, 2, \ldots, N_1\}$, $j \in \{1, 2, \ldots, N_2\}$, $k \in \{1, 2, \ldots, N_3\}$. Then the grain ID of such a voxel will be present in the $[j + (i - 1) * N_2]^{th}$ row and in the k^{th} column of **grainID.txt**. A snapshot of the file is included in Fig. 21.

```
# object 1 are the regular positions. The grid is 10 10 10. The origin is
# at [0 0 0], and the deltas are 1 in the first and third dimensions, and
# 2 in the second dimension
#
object 1 class gridpositions counts 10 10 10
origin 0 0 0
delta  1 0 0
delta  0 1 0
delta  0 0 1
#
# object 2 are the regular connections
#
object 2 class gridconnections counts 10 10 10
#
# object 3 are the data, which are in a one-to-one correspondence with
# the positions ("dep" on positions). The positions increment in the order
# "last index varies fastest", i.e. (x0, y0, z0), (x0, y0, z1), (x0, y0, z2),
# (x0, y1, z0), etc.
#
object 3 class array type int rank 0 items 1000 data follows
42 42 42 49 49 49 49 49 49 35
42 42 39 39 9 9 9 9 35 35
39 39 39 39 39 9 9 45 45 35
52 39 39 39 39 39 45 45 45 45
52 8 39 53 53 53 53 36 36 36
8 46 53 53 53 41 41 36 36 36
33 46 53 53 53 53 53 24 36 36
33 33 33 53 43 43 43 24 24 10
42 33 42 53 49 49 49 49 49 10
42 42 42 49 49 49 49 49 49 42
42 42 42 42 9 9 9 9 25 35
35 42 39 39 9 9 9 9 35 35
35 39 39 39 39 9 9 9 35 35
52 39 39 18 18 38 45 45 45 12
52 46 39 18 18 41 41 45 52 12
```

Figure 21: Snapshot - **grainIDs.txt**