RESEARCH ARTICLE

Dataflow programming for the analysis of molecular dynamics with AViS, an analysis and visualization software application

Kai Pua¤, **Daisuke Yuhara, Sho Ayuba, Kenji Yasuoka***

Department of Mechanical Engineering, Keio University, Yokohama, Kanagawa, Japan

¤ Current address: Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
* yasuoka@mech.keio.ac.jp

Citation: Pua K, Yuhara D, Ayuba S, Yasuoka K (2020) Dataflow programming for the analysis of molecular dynamics with AViS, an analysis and visualization software application. PLoS ONE 15(4): e0231714. https://doi.org/10.1371/journal.pone.0231714

Editor: Dennis Salahub, University of Calgary, CANADA

Received: December 11, 2019

Accepted: March 30, 2020

Published: April 21, 2020

Data Availability Statement: All relevant data are within the manuscript and its Supporting Information files. The source code and binaries for the software are hosted at https://github.com/avis-md/avis.

Funding: The author(s) received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

Abstract

The study of molecular dynamics simulations is largely facilitated by analysis and visualization toolsets. However, these toolsets are often designed for specific use cases and those only, while scripting extensions to such toolsets is often exceedingly complicated. To overcome this problem, we designed a software application called AViS which focuses on the extensibility of analysis. By utilizing the dataflow programming (DFP) paradigm, algorithms can be defined by execution graphs, and arbitrary data can be transferred between nodes using visual connectors. Extension nodes can be implemented in either Python, C++, and Fortran, and combined in the same algorithm. AViS offers a comprehensive collection of nodes for sophisticated visualization state modifications, thus greatly simplifying the rules for writing extensions. Input files can also be read from the server automatically, and data is fetched automatically to improve memory usage. In addition, the visualization system of AViS uses physically-based rendering techniques, improving the 3D perception of molecular structures for interactive visualization. By performing two case studies on complex molecular systems, we show that the DFP workflow offers a much higher level of flexibility and extensibility when compared to legacy workflows. The software source code and binaries for Windows, MacOS, and Linux are freely available at https://avis-md.github.io/.

Introduction

The study of molecular behaviour through analysis and visualization of Molecular Dynamics (MD) simulations is becoming more widely adopted with the increasing processing power of modern computers. A large number of software was developed for this purpose, each with a give-and-take balance between analysis and visualization flexibility. For example, analysis-focused libraries such as MDAnalysis [1] [2], MDTraj [3], and cpptraj [4] provide the necessary toolsets to analyze complex molecular systems to extract relevant information, while some analysis software such as VMD [5], AVOGADRO [6], PyMOL [7], and VESTA [8], apart from their analysis capabilities, also provide tools to visualize the results. While most analysis software offer very extensive analysis capabilities, and they allow complex modifications and extensions to the framework, a deep knowledge of their respective scripting interfaces are required, and hard constraints applied to valid scripts are a hurdle for novice users, even those well versed in bare-bone analysis programming. Therefore, common analysis methods such as hydrogen bonds or Common Neighbor Analysis are often provided in most applications and new algorithms are seldom implemented for integration with the software, except for advanced users. Other software such as OVITO [9] uses a combined visual workflow of analysis and visualization, and enabling complicated techniques to be separated and applied in steps. However, a large dependency on the API is still required when writing custom modules, and the flexibility of data transfer between functions is somewhat limited. In this paper, we seek to improve these limitations by proposing an Analysis and Visualization Software (AViS) application from scratch which utilizes the dataflow programming (DFP) paradigm and allows for graphical designing and debugging of algorithms. This study also demonstrates the flexibility and API simplicity of AViS and DFP in the analysis and visualization of MD data. By using AViS, we are able to simplify and accelerate various analysis and visualization tasks, as described in the sections that follow.

Methods

Software overview

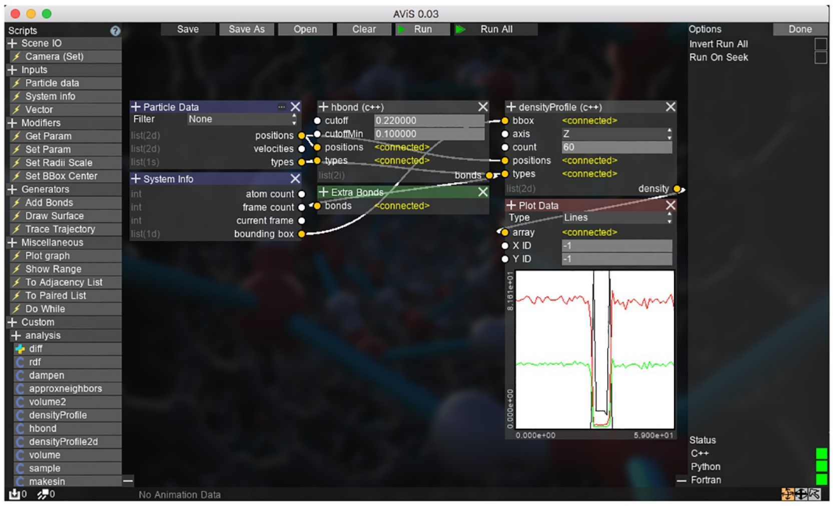
AViS is an OpenGL-based application written in C++. The software is portable and runs natively on various Microsoft Windows / macOS / Linux versions. As an integrated software application, AViS provides a unified environment for analysis and visualization. The user interface of AViS is shown in Fig 1.

Analysis

Application of dataflow programming. As existing analysis paradigms lack flexibility, we designed AViS with the DFP paradigm in its analysis framework, implemented as execution graphs. An execution graph contains multiple nodes, which are evaluated in order and their outputs are passed on to other nodes. Due to the black-box nature of DFP, users do not need to control data conversion and copying, and thus a combination of nodes implemented in different languages in the same graph is allowed. Therefore, new nodes can be created by the user using either Fortran, Python, or C++. Each node exposes its input/output variables in the user interface (UI), which can then be connected to any counterpart with a compatible type. In addition to integer and floating-point values, AViS also supports the data transfer of multi-dimensional variable-length arrays. A list of internal nodes provided by AViS is shown in S1 Table. Various examples of analyses using this software can be found in the documentation pages.

Scripting. Due to the multi-language compatibility, a generic and easy-to-understand format must be enforced for analysis scripts. In this case, we specifically designed the software with no API whatsoever; all relevant code is appended by the internal pre-processor. This makes writing scripts for AViS extremely easy as the user does not have to learn about any application-specific function calls or data structures. As a replacement, we require that all variables that the user wishes to expose to other nodes be preceded with a short comment. An example of valid analysis scripts written in Fortran, Python, and C++ is shown in S1, S2 and S3 Appendices, respectively, and they can be used together, as shown in S1 Fig. Compilers available in the system are used for compilations, with support for gcc, clang, MSBuild and Python 3.7. AViS can still work with other languages even if certain compilers are not

(a)



Particle Data
Filter None
list(2d) positions
list(2d) velocities
list(1s) types

System Info
int atom count
int frame count
int current frame
list(1d) bounding box

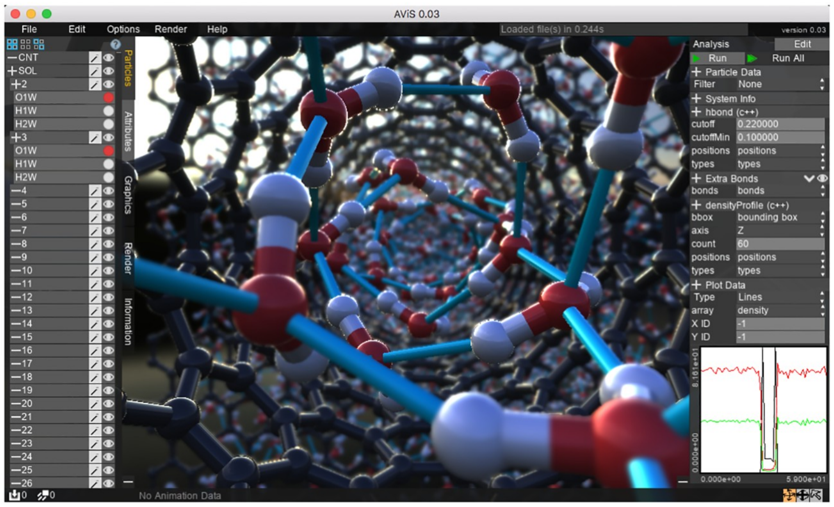
hbond (c++)
cutoff 0.220000
cutoffMin 0.100000
positions <connected>
types <connected>
list(2i) bonds

Extra Bonds
bonds <connected>

densityProfile (c++)
bbox <connected>
axis Z
count 60
positions <connected>
types <connected>
list(2d) density

Plot Data
Type Lines
array <connected>
X ID -1
Y ID -1

(b)



Particle Data
Filter None
System Info
hbond (c++)
cutoff 0.220000
cutoffMin 0.100000
positions positions
types types
Extra Bonds
bonds bonds
densityProfile (c++)
bbox bounding box
axis Z
count 60
positions positions
types types
Plot Data
Type Lines
array density
X ID -1
Y ID -1

Fig 1. Application screenshot. A screenshot of AViS, with its analysis and visualization modes. (a) Analysis mode of AViS. The graphical algorithm to compute the hydrogen bonds and density profile of carbon nanotubes are set up. (b) Visualization mode of AViS. The system as well as the results of the analysis is visualized. Various realtime effects such as Bloom, Ambient Occlusion, and Depth of Field are also applied to the 3D view.

https://doi.org/10.1371/journal.pone.0231714.g001

available. A summary of comment syntax for each language is shown in S4 Appendix through S6 Appendix.

To allow for interoperability between languages, C++ and Fortran scripts are edited internally before being compiled and loaded as dynamic libraries. Additional code is inserted in place of the comments, so the line numbers in the error messages will still match the original file. For Fortran scripts, subroutines that expose the shape and contents of arrays to AViS are automatically generated and appended. This allows AViS to interact with multi-dimensional variable-length arrays in Fortran scripts, with no effort on the user's part. For Python, all variables are accessed directly using the Python C API, and arrays use the Numpy [10] C API.

Attributes. Compared to classic data processing pipelines and single-file analysis workflows, the graph workflow provides a higher level of freedom for data transfer. As data flow is defined solely in the UI, very generic scripts and algorithms may be written, without special consideration of the data source. To ease the analysis of values assigned to each atom, such as order parameter values, AViS also allows for the definition of an arbitrary number of "attributes" in the UI. The values can be read from, and written to, at any point in the execution. Operations with attributes are defined in the graph UI, so complicated algorithms involving the reuse of multiple variables can be achieved easily. Attributes are also useful for algorithm debugging purposes as they allow for the visualization of intermediate outputs.

Data input

As many different types of simulation software give rise to a variety of file formats, it is vital that AViS supports as many formats as possible. AViS natively supports the importing of configuration and trajectory files from popular formats such as GROMACS [11], LAMMPS [12], and Protein Data-Bank [13] files. Other formats can also be expanded for AViS by writing a C plugin and putting the compiled library in the designated folder. Data from the trajectories can then be accessed from scripts using the input nodes, as shown in S2 Fig. A generic Space-Separated Values (SSV) format is also available for loading trajectories containing arbitrary attributes without writing a plugin. An example file is shown in S7 Appendix. By importing attributes, users can perform analysis on complex molecules without being constrained by the existing attribute list of the software or the file formats. An example of this usage for a system of liquid crystals as shown in S3 Fig, where the orientation vectors are imported as the rotx, roty, and rotz, attributes and may be analyzed and visualized without the need for software extensions. The data import and analysis frameworks are entirely seperated, so different files written in different formats can be easily analysed using the same analysis graph.

Visualization

As graphics cards become more powerful, visualization programs should exploit the capabilities of the hardware with the newest methods and APIs. AViS uses the OpenGL 3.3 standard, which is the most recent version that is also supported on most computers. All atoms and bonds are drawn as sprites using ray-tracing methods, and all polygon meshes, such as protein strips, are generated on-the-fly using shaders. The lighting model is physically-based and uses pre-processed radiance maps and a Fresnel term [14], improving the 3D perception of molecules. The deferred rendering method is adopted for the rendering path to allow for sophisticated lighting techniques and improved performance of the UI. Details of the rendering procedure are shown in S4 Fig. By combining these methods, we are able to achieve interactive to real-time frame-rates on typical computers, even for systems with millions of atoms. As the deferred rendering path saves intermediate results, we further improve performance of the interface by only redrawing the parts of the rendering path that are required. For example, when the user changes the lighting intensity, the geometry buffer is retained and only the lighting buffer is recalculated. The intermediate buffers also allow for sophisticated lighting effects such as Depth-of-Field and Screen-Space Ambient Occlusion.

In the place of a scripting interface, AViS provides a comprehensive collection of operation nodes. These nodes allow arbitrary script inputs/outputs to interact with the loaded data and visualization state. For example, the following non-exhaustive operations can be performed using nodes:

- Querying all positions of current atoms that are visible or in a certain region
- Modifying the color/radius/visibility of atoms
- Creating bonds between atoms

Results and discussion

Two case studies are used to demonstrate the flexibility and extensibility of AViS. Here, we look at the formation of Lennard-Jones (LJ) droplets and clathrate hydrates. We choose these systems as their analysis process is complex enough to show the capabilities of AViS. As the total file sizes of the two simulations considered are about 7 and 14 Gigabytes, respectively, the data for both simulations are prepared on a remote file storage system. However, we show that analysis and visualization can still be performed on them by utilizing the remote file handling capability of AViS. All scripts used in these studies are provided as supplementary information, implemented in C++. Some scripts are also translated to Python and Fortran for reference. The analysis results remain consistent when replacing nodes with one generated from a different language. We encourage the reader to refer to the comments in each script detailing the implementation for each analysis step.

Analysis flexibility

For the first case study, we performed the tracking of nucleated clusters in a LJ droplet system [15]. As the tracking of clusters requires the implementation of complex recurrent algorithms and extensive use of temporary variables, this case study is intended to test the flexibility of the analysis framework of AViS. For this study, we performed a simulation of 10976 LJ atoms using FDPS [16], and recorded a total of 10,000 snapshots over a simulation time of 21.5 ns.

First, we identified the largest nucleated cluster in the final snapshot. The flexibility of DFP allows us to split up the algorithm into different generic functions, as shown in Fig 2(a). We used three different functions written in C++. Their contents are listed as follows:

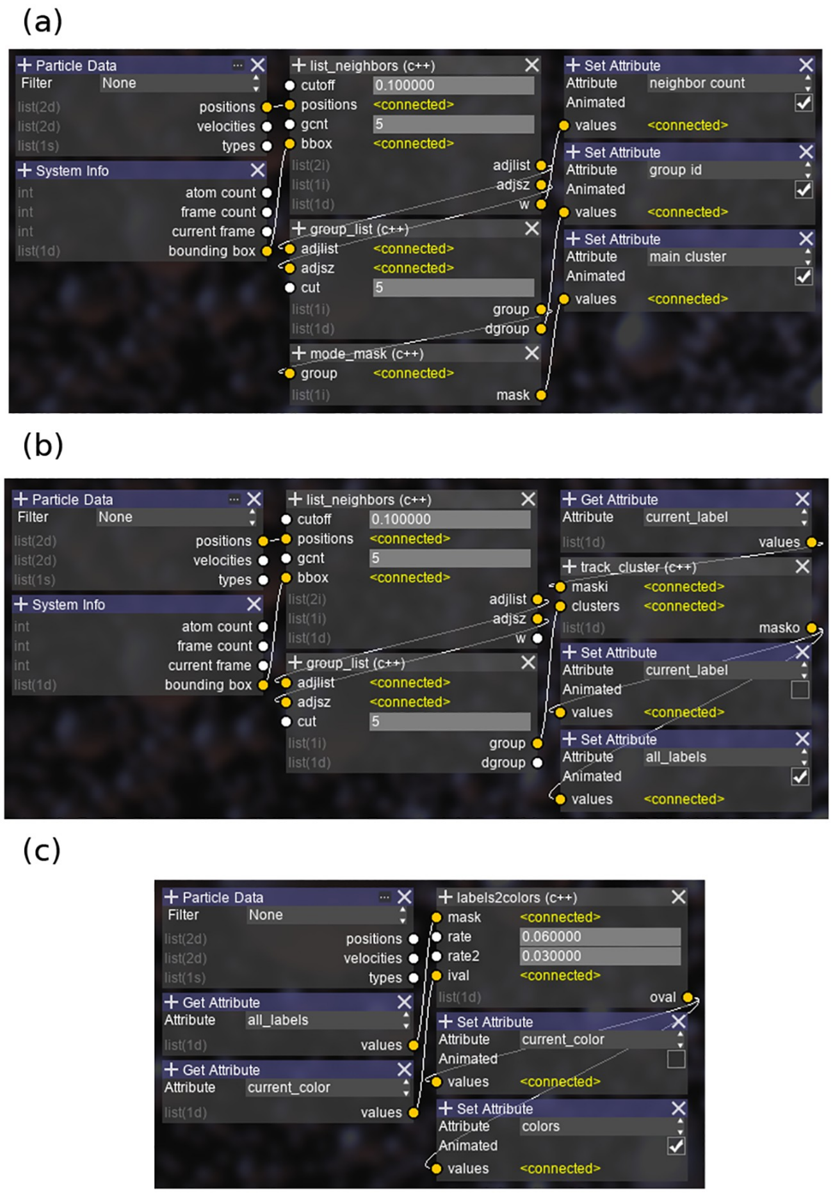
- *list_neighbors*: Create a neighbor-list for each molecule with a user-defined cutoff.
- *group_list*: Apply a unique ID to each group of connected molecules.
- *mode_mask*: Label the molecules with the ID that appears most often.

The results of each step are saved as a different attribute for debugging, with their visualization results shown in Fig 3. By splitting the algorithm into separate nodes, we are able to visually debug the results of each step. This is considerably easier than implementing the whole algorithm in one function.

Next, we tracked the formation of the identified cluster by devising a recurrent algorithm that updates the cluster label based on the next snapshot. The visual representation of the algorithm is shown in S5 Fig. First, the cluster IDs for the current frame are computed. Next, the number of molecules in each cluster that is labeled as a cluster in the previous frame are counted. Finally, the cluster with the largest number of counted molecules is identified as the main cluster, and the labels are updated accordingly. If the count is 0, the cluster has dissipated, and so the labels are cleared. This algorithm is implemented in a new node:

- *track_cluster*: Update the labels based on current cluster groups.

As this algorithm still requires the grouping of clusters in each snapshot, we exploit the flexibility of the graph system by reusing the grouping functions in the previous step. The complete setup is shown in Fig 2(b). To make the algorithm recurrent, the same attribute (*current_label*) is set as both the input and output target of the *track_cluster* node, which is allowed in AViS. An additional attribute is also used to save the output for the whole trajectory, whose results are shown in Fig 4.


(a)

Particle Data
Filter None
list(2d) positions
list(2d) velocities
list(1s) types

System Info
int atom count
int frame count
int current frame
list(1d) bounding box

list_neighbors (c++)
cutoff 0.100000
positions <connected>
gcnt 5
bbox <connected>
list(2i) adjlist
list(1i) adjsz
list(1d) w

group_list (c++)
adjlist <connected>
adjsz <connected>
cut 5
list(1i) group
list(1d) dgroup

mode_mask (c++)
group <connected>
list(1i) mask

Set Attribute
Attribute neighbor count
Animated ✓
values <connected>

Set Attribute
Attribute group id
Animated ✓
values <connected>

Set Attribute
Attribute main cluster
Animated ✓
values <connected>

(b)

Particle Data
Filter None
list(2d) positions
list(2d) velocities
list(1s) types

System Info
int atom count
int frame count
int current frame
list(1d) bounding box

list_neighbors (c++)
cutoff 0.100000
positions <connected>
gcnt 5
bbox <connected>
list(2i) adjlist
list(1i) adjsz
list(1d) w

group_list (c++)
adjlist <connected>
adjsz <connected>
cut 5
list(1i) group
list(1d) dgroup

Get Attribute
Attribute current_label
list(1d) values

track_cluster (c++)
maski <connected>
clusters <connected>
list(1d) masko

Set Attribute
Attribute current_label
Animated ☐
values <connected>

Set Attribute
Attribute all_labels
Animated ✓
values <connected>

(c)

Particle Data
Filter None
list(2d) positions
list(2d) velocities
list(1s) types

Get Attribute
Attribute all_labels
list(1d) values

Get Attribute
Attribute current_color
list(1d) values

labels2colors (c++)
mask <connected>
rate 0.060000
rate2 0.030000
ival <connected>
list(1d) oval

Set Attribute
Attribute current_color
Animated ☐
values <connected>

Set Attribute
Attribute colors
Animated ✓
values <connected>

Fig 2. The analysis setup for LJ droplets. (a) The largest cluster identification algorithm setup. (b) The cluster tracking algorithm setup. (c) The color graduation algorithm setup.

https://doi.org/10.1371/journal.pone.0231714.g002

Finally, we applied a gradient to each molecule based on its formation time, where atoms would fade to red as they join the cluster, and fade to blue when they leave. This is achieved by using a second recurrent function centered on a new node:

- *labels2colors*: Update the color of atoms based on their label.

By running the setup forward through time, as shown in Fig 2(c), atoms fade into red or blue as they join or leave the cluster. However, we also require that atoms that are about to join a cluster fade from blue and vice versa. This is easily achieved in AViS by running the exact same algorithm backwards by setting the appropriate option. This shows the advantage of exposing the algorithm as nodes, in that it allows for changes in the execution flow without

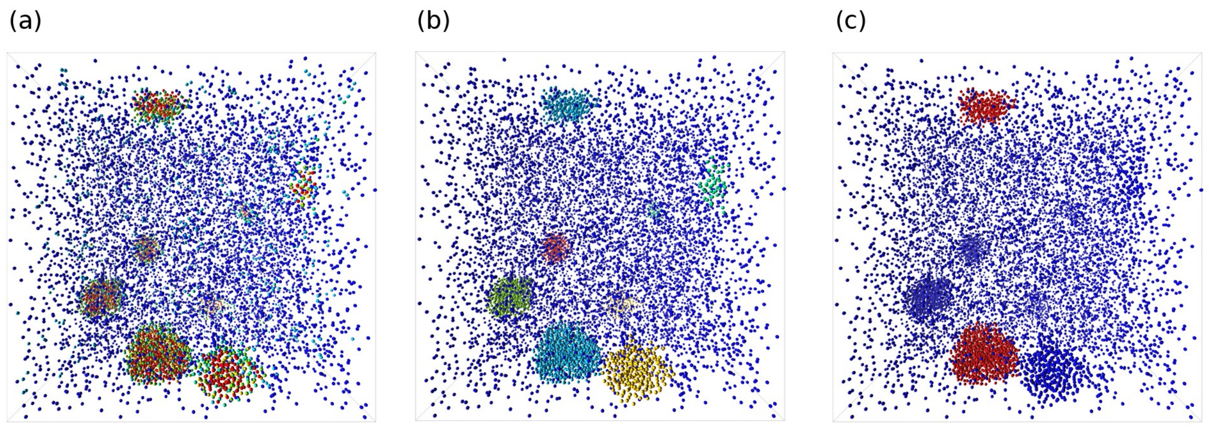


Fig 3. Visualization of the attribute results of the largest cluster identification algorithm. (a) The number of neighbors of each molecule. The color shifts from blue to red with an increasing number of neighbors. (b) The cluster index of each molecule. Each color corresponds to an unique cluster index. (c) The label mask for the largest cluster. The largest cluster is colored red.

https://doi.org/10.1371/journal.pone.0231714.g003

modifying the code. The attributes created from both loops are then combined, resulting in the output as shown in Fig 5.

Other common processes in cluster analysis include finding the largest cluster at every point in time. As we already have all the components, we simply plug the *group_ids* into the *mode_mask* node. We can also offset the camera to point to the labeled cluster by calculating its center with a self-evident script and passing the result to the *Set Camera Center* node.

To conclude, this case study showed that by using AViS' graph system, algorithms that involve loops over multiple modules that are otherwise difficult to achieve can be easily implemented. As the module implementations are completely isolated, each function can also be written with little to no reference to others, thus speeding up the analysis procedure.

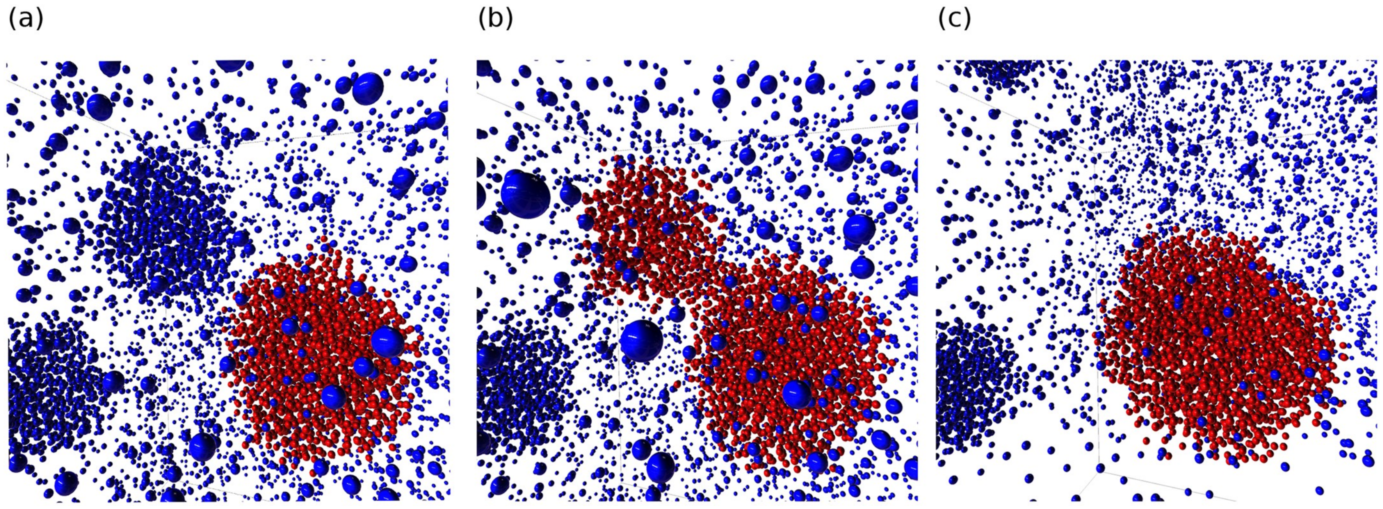


Fig 4. Visualization of the results of the tracking algorithm. In this algorithm, the main cluster is tracked backwards through time and colored red. (a) The final snapshot. The largest cluster is labeled. (b) The snapshot during the instant two clusters have combined. The clusters still share a common label. (c) The snapshot before the two clusters combined. The largest of the two is tracked and labeled.

https://doi.org/10.1371/journal.pone.0231714.g004

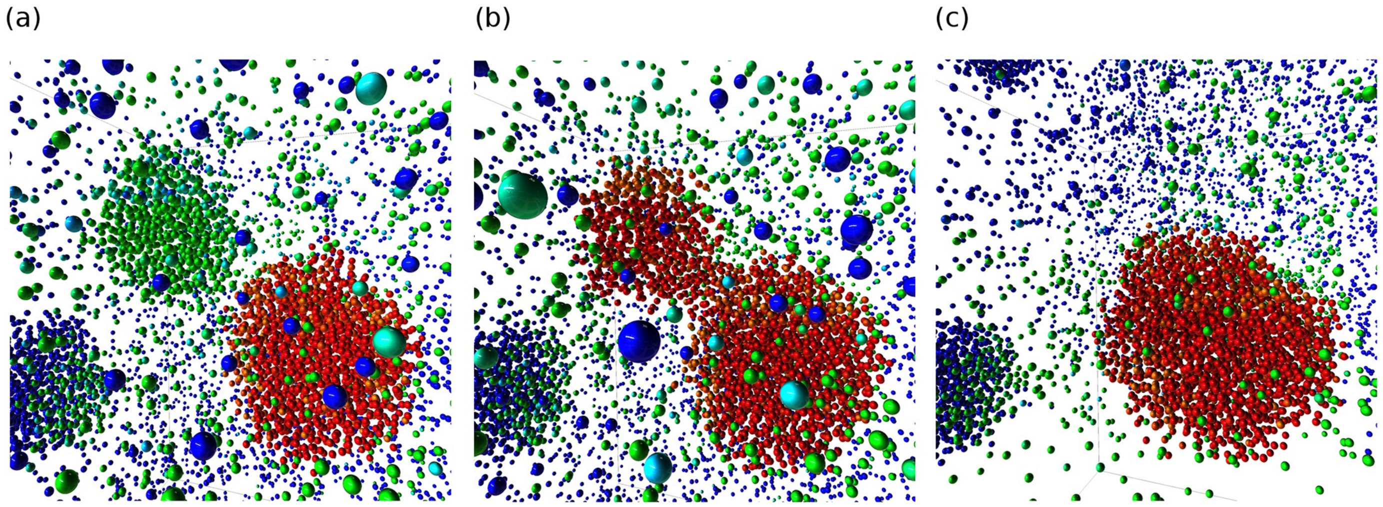


Fig 5. The results of the color graduation algorithm. A color gradient is applied so that atoms are colored blue to red based on how near they are to being a part of the main cluster. The same snapshots from Fig 4 is used. (a) A subcluster turns green right before merging with the tracked cluster (b). (c) A small green cloud surrounds the tracked cluster, indicating the constant flux of particles.

https://doi.org/10.1371/journal.pone.0231714.g005

Visualization flexibility

The Mutually Coordinated Guest (MCG-1) algorithm for the study of the growth of methane clathrate hydrates was presented by Barnes *et al.* [17]. For effective analysis of the results, we would like to change the appearance of atoms based on the results, as well as create bonds forming the hydrate-like structures. Therefore, high interoperability between analysis and visualization modules is required. In this study, we used a portion of the same data created by the authors of the MCG-1 algorithm [18], which contains a mixture of 2944 water atoms and 512 methane atoms, over 8000 frames.

The MCG algorithm looks for the structure shown in S6 Fig. The algorithm is split into separate modules for ease of debugging, with each module explained below. The completed analysis graph is shown in Fig 6.

Several nodes are created for this case study, one for each step of the algorithm:

- *filter_guests*: Find pairs of guests within 9Åof each other.
- *filter_waters*: Find water molecules that are 45° between each guest pair.
- *hbonds_filtered*: Find hydrogen bonds between selected water molecules.

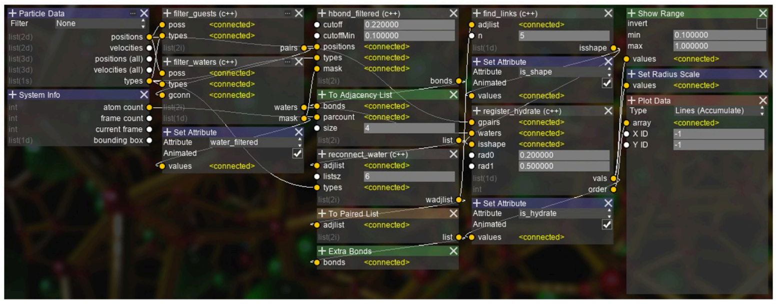

```
Particle Data: Filter None; list(2d) positions; list(2d) velocities; list(3d) positions (all); list(3d) velocities (all); list(1s) types
System Info: int atom count; int frame count; int current frame; list(1d) bounding box
filter_guests (c++): poss <connected>; types <connected>; list(2i) pairs
filter_waters (c++): poss <connected>; types <connected>; gconn <connected>; list(2i) waters; list(1d) mask
Set Attribute: Attribute water_filtered; Animated ✓; values <connected>
hbond_filtered (c++): cutoff 0.220000; cutoffMin 0.100000; positions <connected>; types <connected>; mask <connected>; list(2i) bonds
To Adjacency List: bonds <connected>; parcount <connected>; size 4; list(2i) list
reconnect_water (c++): adjlist <connected>; listsz 6; types <connected>; list(2i) wadjlist
To Paired List: adjlist <connected>; list(2i) list
Extra Bonds: bonds <connected>
find_links (c++): adjlist <connected>; n 5; list(1d) isshape
Set Attribute: Attribute is_shape; Animated ✓; values <connected>
register_hydrate (c++): gpairs <connected>; waters <connected>; isshape <connected>; rad0 0.200000; rad1 0.500000; list(1d) vals; int order
Set Attribute: Attribute is_hydrate; Animated ✓; values <connected>
Show Range: invert; min 0.100000; max 1.000000; values <connected>
Set Radius Scale: values <connected>
Plot Data: Type Lines (Accumulate); array <connected>; X ID -1; Y ID -1
```

Fig 6. The complete setup for the second case study. The analysis setup for the hydrate formation case study, namely the MCG-1 algorithm.

https://doi.org/10.1371/journal.pone.0231714.g006

- *reconnect_water*: Change O-H bonds to O-O bonds.
- *find_links*: Look for bonds making up a *n*-chain polygon. In this study, *n* is set to 5.
- *register_hydrate*: Assign a unique label for caged guests and their oxygen atom structures.

Additionally, the following operational nodes provided in AViS are used for modifications on the visualization state:

- *Extra Bonds*: Create additional bonds between atoms.
- *Show Range*: Hide atoms with an attribute value outside of a specified range.
- *Set Radius Scale*: Modify the radius of atoms.

As with the first case study, splitting the algorithm into parts allowed us to visually debug the validity of each step. As with the previous case study, all atom properties such as masks are saved as attributes, one of which can be seen in Fig 7(a). Also, the bonds created can be easily switched between O-H (Fig 7(b)) and O-O (Fig 7(c)) by simply changing the input source of the *Extra Bonds* node. This further shows the advantage of exposing critical functions in the UI instead of as an API.

By running the algorithm across the whole trajectory, the progression of the formation of hydrates can be visualized as shown in Fig 8. AViS automatically caches the result for each step, so when the calculation is finished, all results can be visualized in the window directly, including dynamic bonds, visibility, and size changes.

Additionally, a crucial step in hydrate nucleation analysis is identifying the formation time of the initial cluster by plotting the value of the order parameter through time. This can be achieved in AViS by using the *Plot Data* node set to the *Lines (Accumulate)* mode, taking the order parameter value as input. The output of the node is shown in S7 Fig, which shows the point of hydrate formation at approximately frame 2000.

To conclude, this case study showed that by exposing essential functions in the UI, changes to the visualization state can be made instantly, with no need for recompilations or API

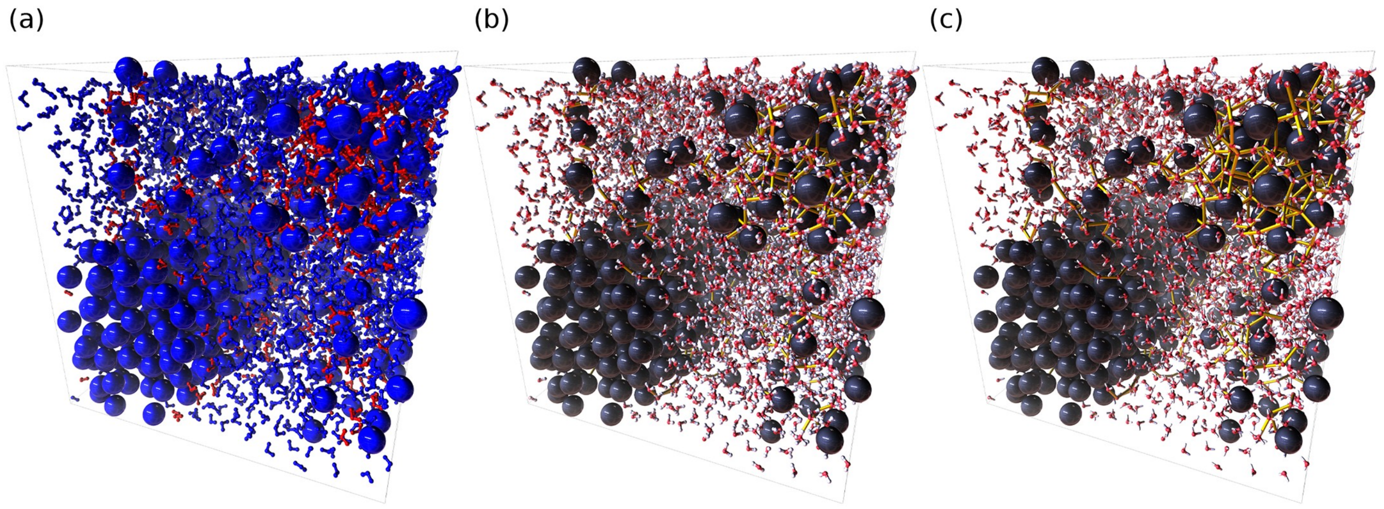


Fig 7. The intermediate results of the hydrate analysis. Large spheres are methane molecules, while small connected spheres are O and H atoms of water molecules. (a) Water molecules identified by the MCG-1 algorithm are colored red. (b) Cage bonds are drawn in yellow in the form of O-H bonds. (c) Cage bonds are drawn in yellow in the form of O-O bonds. AViS is able to change between (b) and (c) by simply switching the connection of nodes.

https://doi.org/10.1371/journal.pone.0231714.g007

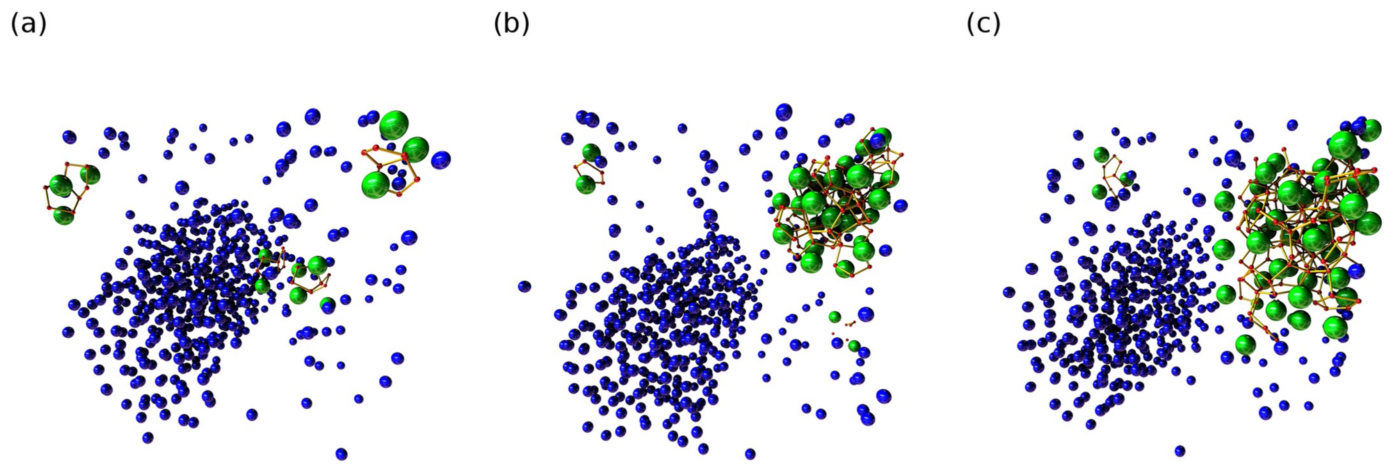


Fig 8. The progression of hydrate formation in the trajectory. Unidentified and identified methane molecules are colored blue and green respectively, and unindentified water molecules are not visualized. Snapshots (a), (b), and (c), are taken at arbitrary frames throughout the trajectory. AViS caches the results of each analysis frame, so modifications including radius, color, and additional bonds can be animated.

https://doi.org/10.1371/journal.pone.0231714.g008

lookups. As a result, visual debugging of algorithms can be performed easily, without extensive and potentially complex changes to the code.

From our case studies, we can say that our software is useful for students or researchers who would like to visually inspect their algorithm's results easily, or when designing and tweaking the algorithm itself, as changing and debugging parameters are very easily acheived, and the independance between functions make it very easy to rearrange, reuse, and understand snippets of code. However, such frameworks do suffer from a lower execution speed compared to raw analysis code, and the optimization of DFP code is a research topic we are currently tackling. We do think though that by using AViS to design analysis code, a lower performance in exchange for a much faster debugging time is acceptable for non-expert researchers.

Conclusions

We have introduced Dataflow Programming for the analysis of MD data by creating AViS and demonstrated its flexibility in two complex case studies. We also showed that by abstracting the analysis scripts, we are able to substantially reduce the complexity of user code, while retaining the ability to interact with the visualization system. We believe that the DFP method has great potential, and while manual parallelization of single nodes using OpenMP is possible, we expect future work on parallelization of graph executions to further improve analysis performance. The source code, binaries for Windows, Linux, and MacOS, as well as documentation of AViS, are hosted on Github at https://avis-md.github.io.

Supporting information

S1 Table. Examples of in-built nodes provided by AViS. Custom nodes can be implemented by the user to interact with internal nodes, regardless of language.
(PDF)

S1 Appendix. A valid fortran analysis script that generates a sine wave.
(PDF)

S2 Appendix. A valid Python analysis script that differentiates an array.
(PDF)

S3 Appendix. A valid C++ analysis script that applies an exponential decay to a signal.
(PDF)

S4 Appendix. Comment syntax for Fortran scripts.
(PDF)

S5 Appendix. Comment syntax for Python scripts.
(PDF)

S6 Appendix. Comment syntax for C++ scripts.
(PDF)

S7 Appendix. An example of a SSV file with rotational and potential attributes. Each entry in the first line represents the type of data in each column.
(PDF)

S1 Fig. Multi-language support for analysis. AViS handles data transfer between Fortran, Python, and C++ scripts automatically, including arrays as seen here.
(PDF)

S2 Fig. Accessing molecular data from scripts. The molecular data of the loaded configuration and trajectory can be accessed using the input nodes. The positions and velocities of the current visible frame, as well as of the whole trajectory can be accessed, as well as additional attributes that are imported, if available. The input values update to the data for each frame over the trajectory when the graph is executed for all frames.
(PDF)

S3 Fig. Visualization and analysis of molecules with attribute data. (a) By importing the orientation data as attributes with the Generic SSV format, liquid crystal molecules can be visualized without the need to write a custom plugin or importer. In this figure, a color gradient is also applied based on the *roty* attribute. (b) Individual attributes can be used in analysis by utilizing the *Get Attribute* node.
(PDF)

S4 Fig. The rendering pipeline. (a) Bounding quads are generated using a vertex shader. (b) (c) Surface information channels are generated using ray-tracing in a fragment shader. (d) A Physically-based Rendering (PBR) shader combines the channels into the final image. (e) By utilizing deferred shading, UI overlays and image effects can be added without re-drawing the whole scene, thus improving performance.
(PDF)

S5 Fig. The visual representation of the tracking algorithm. (a) The current frame. (b) The next frame. (c) The number of particles belonging to the previous cluster is counted. (d) The labels are updated for the cluster with the largest count.
(PDF)

S6 Fig. MCG-1 algorithm representation. A visual representation of the mutually coordinated guest order parameter used in the case study on clathrate hydrates. The algorithm is presented by Barnes *et al.* [17]).
(PDF)

S7 Fig. The value of the order parameter across the whole trajectory. By recording the number of classified molecules every step, we can identify the point of hydrate formation at about frame 2000.
(PDF)

S1 File.
(ZIP)

Acknowledgments

The authors would like to thank Y. Ono for generously providing us with the carbon nanotubes data for the creation of Fig 1, as well as Dr. T. Nozawa for the Liquid Crystal data for the creation of S2 Fig. Thanks are also given to Dr. M. Nakayama for his invaluable input on visualization techniques in OpenGL. We would also like to thank Prof. A. K. Sum for allowing us to use the data in the second case study on methane hydrates. Finally, we would like to thank all members of the Yasuoka Laboratory for trying out the software and giving invaluable suggestions and bug reports.

Author Contributions

Conceptualization: Kai Pua, Kenji Yasuoka.

Data curation: Daisuke Yuhara, Sho Ayuba.

Formal analysis: Kai Pua.

Investigation: Kai Pua.

Methodology: Kai Pua.

Project administration: Kenji Yasuoka.

Resources: Daisuke Yuhara, Sho Ayuba, Kenji Yasuoka.

Software: Kai Pua.

Supervision: Kenji Yasuoka.

Validation: Kai Pua, Daisuke Yuhara, Sho Ayuba, Kenji Yasuoka.

Visualization: Kai Pua.

Writing – original draft: Kai Pua.

Writing – review & editing: Kai Pua, Daisuke Yuhara, Sho Ayuba, Kenji Yasuoka.

References

1. Michaud-Agrawal N, Denning EJ, Woolf TB, Beckstein O. MDAnalysis: a toolkit for the analysis of molecular dynamics simulations. J Comput Chem. 2011; 32(10):2319–2327. https://doi.org/10.1002/jcc.21787 PMID: 21500218
2. Gowers RJ, Linke M, Barnoud J, Reddy TJ, Melo MN, Seyler SL, et al. MDAnalysis: a Python package for the rapid analysis of molecular dynamics simulations. In: Proceedings of the 15th Python in Science Conference. SciPy; 2016. p. 98–105.
3. McGibbon RT, Beauchamp KA, Harrigan MP, Klein C, Swails JM, Hernández CX, et al. MDTraj: a modern open library for the analysis of molecular dynamics trajectories. Biophys J. 2015; 109(8):1528–1532. https://doi.org/10.1016/j.bpj.2015.08.015 PMID: 26488642
4. Roe DR, Cheatham TE. PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data. Journal of Chemical Theory and Computation. 2013; 9(7):3084–3095. https://doi.org/10.1021/ct400341p PMID: 26583988
5. Humphrey W, Dalke A, Schulten K. VMD: visual molecular dynamics. J Mol Graph. 1996; 14(1):33–38. https://doi.org/10.1016/0263-7855(96)00018-5 PMID: 8744570
6. Hanwell MD, Curtis DE, Lonie DC, Vandermeersch T, Zurek E, Hutchison GR. Avogadro: an advanced semantic chemical editor, visualization, and analysis platform. J Cheminf. 2012; 4(1):17.
7. DeLano WL. The PyMOL molecular graphics system. http://wwwpymolorg. 2002.

8. Momma K, Izumi F. VESTA 3 for three-dimensional visualization of crystal, volumetric and morphology data. J Appl Crystallogr. 2011; 44(6):1272–1276.
9. Stukowski A. Visualization and analysis of atomistic simulation data with OVITO–the Open Visualization Tool. Modell Simul Mater Sci Eng. 2009; 18(1):015012.
10. Travis E O. A guide to NumPy. USA: Trelgol Publishing. 2006.
11. Hess B, Kutzner C, Van Der Spoel D, Lindahl E. GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. J Chem Theory Comput. 2008; 4(3):435–447. https://doi.org/10.1021/ct700301q PMID: 26620784
12. Plimpton S, Crozier P, Thompson A. LAMMPS-large-scale atomic/molecular massively parallel simulator. Sandia Natl Lab. 2007; 18:43.
13. Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, et al. The protein data bank, 1999–. In: International Tables for Crystallography Volume F: Crystallography of biological macromolecules. Springer; 2006. p. 675–684.
14. Schlick C. An Inexpensive BRDF Model for Physically-based Rendering. Comput Graph Forum. 1994; 13(3):233–246.
15. Ayuba S, Suh D, Nomura K, Ebisuzaki T, Yasuoka K. Kinetic analysis of homogeneous droplet nucleation using large-scale molecular dynamics simulations. J Chem Phys. 2018; 149(4):044504. https://doi.org/10.1063/1.5037647 PMID: 30068205
16. Iwasawa M, Tanikawa A, Hosono N, Nitadori K, Muranushi T, Makino J. Implementation and performance of FDPS: a framework for developing parallel particle simulation codes. Publications of the Astronomical Society of Japan. 2016; 68(4).
17. Barnes BC, Beckham GT, Wu DT, Sum AK. Two-component order parameter for quantifying clathrate hydrate nucleation and growth. J Chem Phys. 2014; 140(16):164506. https://doi.org/10.1063/1.4871898 PMID: 24784286
18. Barnes BC, Knott B, Beckham GT, Wu DT, Sum AK. Reaction Coordinate of Incipient Methane Clathrate Hydrate Nucleation. J Phys Chem B. 2014; 118. https://doi.org/10.1021/jp507959q PMID: 25347748