Synopsis

Swapnil Ranadive
Department of Electronics and Computer Engineering, KJSCE

Implementing Polylines for the OpenFOAM GUI project

The research migration's aim is to develop a dictionary generator for the OpenFOAM's blockMesh dictionary's keyword "polyLine" leveraging the pyVNT library for automation and syntax accuracy and creating an intuitive GUI within Blender as the frontend for it. The GUI focuses on several core aspects:

- Data Storage Mechanism: Implemented using object-oriented programming principles, including classes of doubly linked lists to store the nodes and manage the sequences of nodes.
- Real-Time Visualization: Utilized OpenGL rendering for real-time updates and visualization of polylines within Blender.
- Dynamic Updates and Data Deletion: Enabled the dynamic updating of node locations and efficient deletion of data.

References

[1] Dmitri I Chitalov. "Development of an application with a graphical user interface (GUI) to compute in parallel in the OpenFOAM environment". In: Journal of Physics: Conference Series (2019) 1399 033001 doi:10.1088/1742-6596/1399/3/033/001

Semester-Long Internship Report

On

Implementing Polylines for the OpenFOAM GUI Project

Submitted by

Swapnil Ranadive
K J Somaiya College of Engineering

Under the guidance of

Mr. Rajdeep Adak
Developer, FOSSEE

and

Prof. Janani Muralidharan
Department of Mechanical Engineering
IIT Bombay

June 21, 2024

ACKNOWLEDGEMENT

I would like to express my appreciation and thanks to **Mr. Rajdeep Adak** for being a tremendous mentor for me. And for understanding my shortcomings, encouraging and allowing me to learn and grow as a fellow intern. Your mentorship has let an indelible mark on me. I would also like to thank OpenFOAM GUI team member, **Mr. Diptangshu Dey** for helping me understand complex codes and for providing invaluable advice throughout this journey. I thank **Ms. Payel Mukhurjee** and **Prof. Janani Muralidharan** for giving me the opportunity to work and for the perceptive insight on this project.

Lastly, this internship has been a boon for me. It has not only developed my interest in the field but also showed me where I need to improve upon and sharpened my intellect, instilling a passion for continuous learning and improvement. The experience has been invaluable and I'm eager to apply what I have learned to future endeavors.

CONTENTS:

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The following report documents a semester-long internship focused on enhancing the functionality of OpenFOAM, an open-source software suite for computational fluid dynamics, through the development of a graphical user interface using Python in Blender. The aim is to develop a dictionary generator for the OpenFOAM's blockMesh dictionary's keyword "polyLine" leveraging the pyVNT library for automation and syntax accuracy and creating an intuitive GUI within Blender as the frontend for it. The GUI focuses on several core aspects:

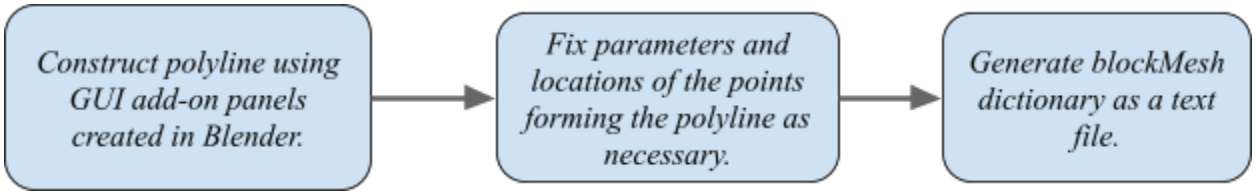
1. *Data Storage Mechanism: Implemented using object-oriented programming principles, including classes of doubly linked lists to store the nodes and manage the sequences of nodes.*
2. *Real-Time Visualization: Utilized OpenGL rendering for real-time updates and visualization of polylines within Blender.*
3. *Dynamic Updates and Data Deletion: Enabled the dynamic updating of node locations and efficient deletion of data.*

INTRODUCTION

2.1 OBJECTIVE

The focus is on developing a dictionary generator specifically for the blockMesh dictionary in OpenFOAM, utilizing the pyVNT library created by the FOSSEE team. By automating the generation of required OpenFOAM dictionaries, we intend to help new learners quickly create blockMesh dictionary and parameter files without needing extensive knowledge of meshing and solving techniques. This automation will adhere to OpenFOAM's meshing and solving rules. Additionally, a graphical user interface (GUI) will be developed in Blender using Python, allowing users to interact with the system seamlessly and trigger the generation of OpenFOAM dictionaries via the PyVNT library. The developed addon will support Blender v2.8+.

2.2 APPROACH



[Approach for generation of OpenFOAM dictionary]

BLENDER IMPLEMENTATION

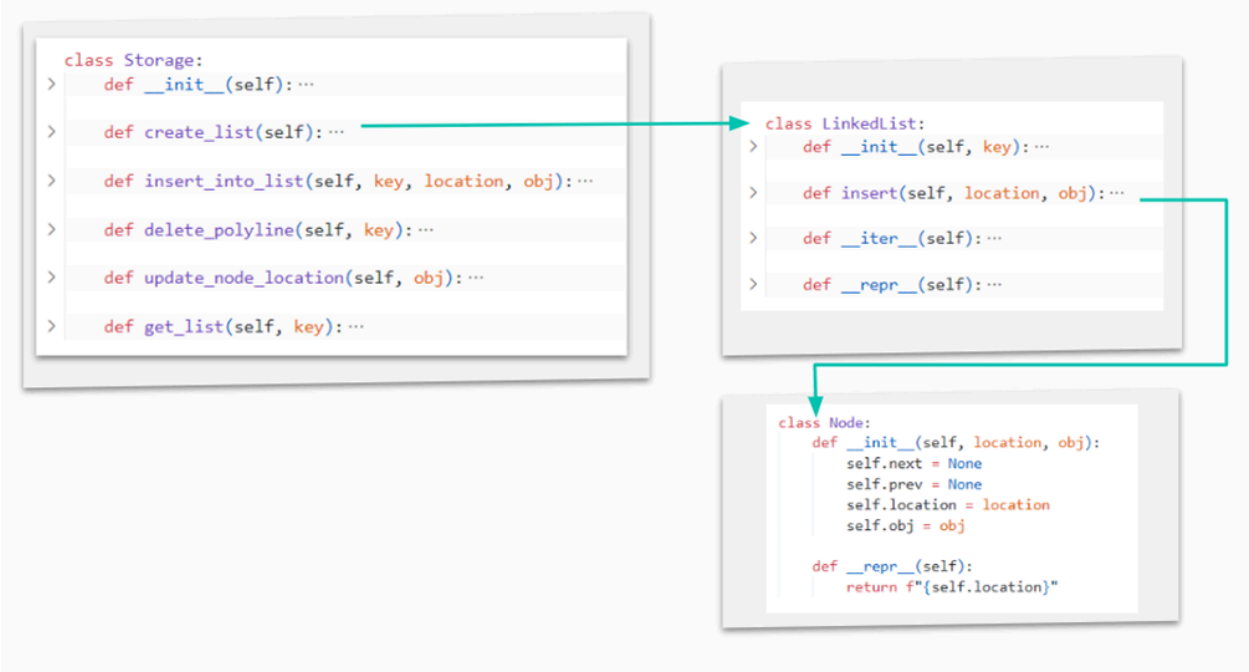
3.1 STORAGE MECHANISM

The system for managing polylines and their respective data is a sophisticated framework specifically tailored to work with Blender, the open-source 3D creation suite. This system leverages object-oriented programming principles and data structures to provide efficient storage, manipulation, and visualization of polylines.

At the core of the system is the ‘Node’ class, which encapsulates individual points in a polyline. Each ‘Node’ object contains a ‘location’ attribute that stores the 3D coordinates of the point, and an ‘obj’ attribute that references the corresponding Blender object created in the scene and 3D viewport. The ‘Node’ class also maintains pointers to the next and previous nodes in the sequence, facilitating its integration into a doubly linked list. This linked list structure is implemented in the ‘LinkedList’ class, which maintains a collection of ‘Node’ objects. The class itself features methods for inserting new nodes and iterating through the list. A doubly linked list is an optimal choice for this data structure due to its directional traversal capability and dynamic, efficient insertions and deletions without requiring reallocation or reorganization of memory. The flexibility and scalability support variable sizes without

performance degradation, while maintaining node integrity through pointers to both previous and next nodes ensure the continuity of the polyline during modifications.

The ‘Storage’ class acts as the primary interface for managing multiple polylines. It maintains a dictionary (‘polyline_storage’) where keys are unique identifiers generated sequentially, and values are instances of the previously mentioned ‘LinkedList’ class, each representing a distinct polyline. The ‘create_list’ method initializes a new ‘LinkedList’, assigns it a unique key, and stores it in the dictionary, returning the key for future operations. The ‘insert_into_list’ method facilitates the addition of new nodes to a specified polyline, ensuring that the polyline exists and then delegating the insertion operation to the corresponding ‘LinkedList’. The ‘delete_polyline’ method allows for the removal of an entire polyline, meticulously ensuring that associated Blender objects are removed to free up resources and that the UI is updated by removing the polyline's checkbox from Blender's scene properties.



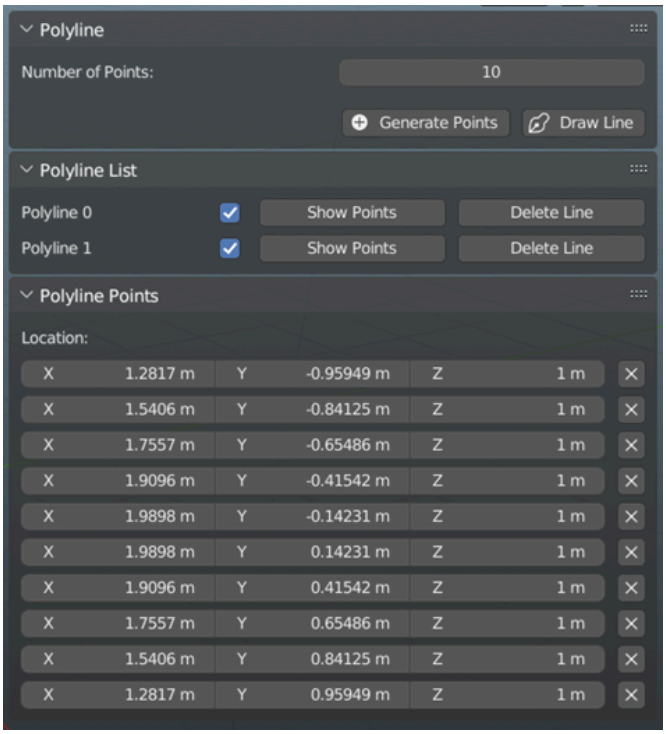
[Class relationship diagram for ‘Storage’, ‘LinkedList’ and ‘Node’]

3.2 USER INTERFACE ELEMENTS

The blockMeshDict Blender add-on specifically structured for polylines features three primary collapsible panels: ‘Polyline’, ‘Polyline List’ and ‘Points’. Each panel serves a distinct role in managing and visualizing polylines within the 3D viewport of Blender. The main ‘Polyline’ panel serves as the primary interface for users to manage the creation of the polylines. It contains a property field linked to allow users to adjust the number of points forming the polyline before generating them in the viewport.

BUTTONS	FUNCTION
Generate Points	Generate the number of points specified around the selected edge Adds a new polyline to the polyline list
Draw Line	Enables the draw handler to visualize the polyline
Show Points	Displays a third collapsible panel with points' location which forms the polyline. The points' location can be manipulated.
Delete Line	Deletes the lines from the dictionary storage, the viewport and memory of Blender.

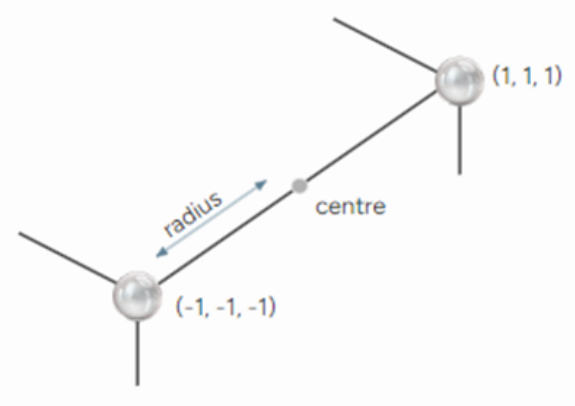
[Operator Functions]



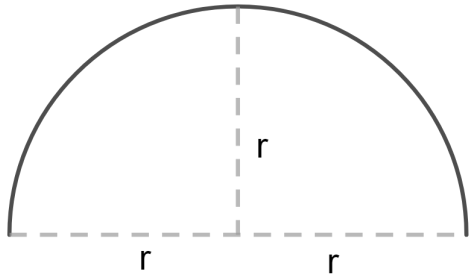
[GUI Interface]

4.3 GENERATION OF INTERPOLATION POINTS

The function in question is designed to generate a series of points along a curved line between two selected vertices (forming an edge in a 3D space. The function starts by identifying the edge that has been selected by the user. An edge defined by two vertices. The function checks the selection to ensure that exactly two vertices forming an edge are selected. If the selection is invalid, meaning that if points are selected or a face, the function halts and returns an error message instructing the user to select an edge. Once the coordinates of the two vertices have been selected, the center and radius of the curve is calculated considering the edge as a diameter. The radius is determined as half of the Euclidean distance between the two vertices.

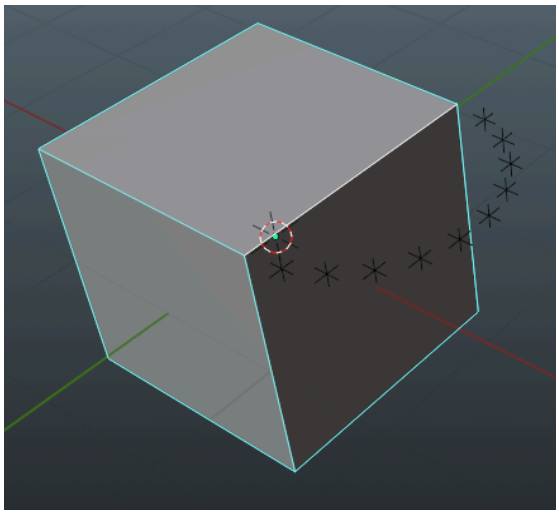


[Edge of a structure]



[The curve in which points are to be generated.]

An object of the Storage class is created and when the points are generated, a linked list is initialized to hold these points. The first point added to this linked list is one of the vertices of the edge, and the subsequent points generated along the curve are appended to this list. These points are generated along a semicircular path around the edge.



[Generation of points around an edge in Blender]

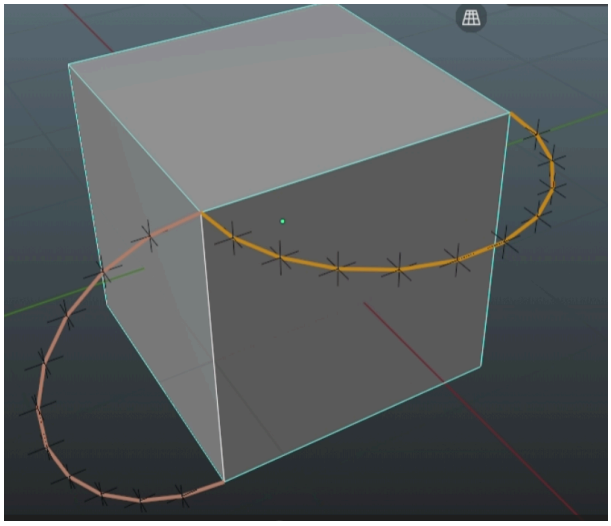
To achieve this, the angle for each point is calculated based on the total number of points to be generated. The semicircle is divided into as many segments as the number of points, ensuring even spacing along the curve. For each point generated, an empty object is added as a placeholder object that presents that point in the space in the viewport.

4.4 THE DRAW HANDLER

The drawing of polylines in the 3D viewport happens using a combination of python scripting and OpenGL (through Blender's bgl and gpu modules). It allows users to visually represent polylines between specified points directly. When the button for this operator is clicked by the user, a function is called. This specific function acts as a handler that keeps running in the background continuously once it is registered. It remains active until it is explicitly removed. This operation ensures that any changes or updates to the polylines are immediately reflected in the 3D viewport.

The storage contains all the polylines that have been generated and need to be potentially displayed. Each polyline is represented as a linked list of nodes, where each node contains the coordinates of a point on the polyline. For each polyline, the function retrieves the defined color, width, and other properties that dictate how the line should be rendered. These properties can include line thickness, color, and any additional visual effects. Using these properties, the function creates batches of vertices representing line segments between consecutive points of the polyline.

Using the configured shader and vertex batches, the function proceeds to draw the polylines on the 3D viewport. A shader is a type of program used in computer graphics to determine how vertices and pixels are processed and rendered. In this case, the shader is configured to handle the drawing of lines with the specified properties. The vertex batches are processed by the shader to render the line segments, creating a visual representation of the polyline in the 3D environment. The function also takes into account user preferences regarding which polylines should be visible. This is achieved through a linked list iteration where each stored polyline is checked against user-defined visibility settings. The line appears in the 3D view only when the corresponding polyline's checkbox is checked.



[Class relationship diagram for 'Storage', 'LinkedList' and 'Node']

4.5 DYNAMIC UPDATES

When a user changes a point's location in the panel within Blender's UI or in the 3D viewport, the corresponding Blender object's location attribute is updated through Python scripting. This change triggers the handler function to immediately redraw or update the object's position in the 3D viewport. Blender's viewport rendering engine continuously monitors changes to object properties like location, ensuring that any modifications made in the panel are reflected in real-time within the 3D environment.

4.6 DATA DELETION

Deleting points from a polyline involves targeted manipulation of the polyline's linked list data structure. When a point deletion is initiated using the operator, the script first identifies the specific point to delete based on its name. It then traverses the linked list associated with the polyline stored in storage.
The deletion process involves adjusting pointers within the linked list:

1. The operator locates the node (point) to delete and adjusts its predecessor's next pointer to skip over the node.
2. If the deleted node was the head or tail of the linked list, appropriate adjustments are made to ensure continuity (head or tail pointers).

Simultaneously, the Blender object representing the point is removed from the scene, freeing up associated resources.
Additionally, the operator updates any relevant UI elements, ensuring that the panel accurately reflects the current state of the polyline after the deletion.
When deleting a line, the operator identifies the polyline to delete using its unique key. It accesses the polyline's linked list data structure stored in storage, iterating through each node. For each node (representing a point along the polyline), the associated Blender object (point) is removed from the scene.

DICTIONARY GENERATOR

The pyVNT library, created by the OpenFOAM GUI Team, constructs node trees that replicate the format of OpenFOAM Dictionaries. It offers tools to easily manage these trees using straightforward Python scripts and produces serialized data to dynamically create graphical representations of the trees. pyVNT streamlines the process of interacting with OpenFOAM configuration files, making it simpler to visualize and edit complex data structures. By providing a flexible and user-friendly interface, pyVNT enhances the efficiency of managing and manipulating OpenFOAM Dictionaries within Python-based workflows.

pyVNT output is structured with a root node and edges defined using the polyLine keyword followed by indices and coordinates in a nested format. On the right, the OpenFoam dictionary syntax displays a similar structure where the edges keyword encompasses polyLine definitions with identical indices and coordinate data. Thus, we can see the congruence in both; showing how pyVNT helps in directly generating the OpenFOAM dictionary syntax without manual efforts.

```
root
└── edges
        (
            polyLine 0 1
            (
                (0.25 0.1 0)
                (0.5 0 0)
                (0.75 -0.1 0)
            )
            polyLine 4 5
            (
                (0.25 0.1 0.1)
                (0.5 0 0.1)
                (0.75 -0.1 0.1)
            )
        );
```

[pyVNT Output]

```
edges
(
    polyLine 0 1
    (
        (0.25 0.1 0)
        (0.5 0 0)
        (0.75 -0.1 0)
    )

    polyLine 4 5
    (
        (0.25 0.1 0.1)
        (0.5 0 0.1)
        (0.75 -0.1 0.1)
    )

);
```

[OpenFoam Dictionary Syntax]

CONCLUSION

6.1 CONCLUSION

Blender's geometry design functionalities with a graphical user interface to ease the entire process of geometry creation and further, text file generation within the software. This provides an user-friendly interface for parameter editing than manual modification and generation of dictionary files.

6.2 IN DEVELOPMENT

FEATURE	FUNCTION
Dictionary Generator	*Generates the text file for openFOAM dictionary blockMeshDict.*
Generation Capabilities	*The proper generation of points should extend beyond basic rectangular structures and even for other types of structures.*
Collision Detection	*Detecting points merging into adjacent objects to avoid their generation into them.*
Collapsible Popup Windows	*Managing visibility and interactivity with the points data.*
Labeling with indices	*For better identification of points through the panel.*
Labeling with respective 3D coordinates	*For better identification of points through the panel.*
Rotation of the polylines	*Rotating the generated points of the polyline around the edge to increase flexibility for editing.*
Merging of polylines	*To simplify complex mesh structures.*
Different curves of point generation	*Develop different functions beyond generation of points in the shape of a curve.*
Better GUI	*A more user-friendly and simplistic design for the GUI.*

[Features in development.]

6.3 FUTURE SCOPE

The future development of the add-ons would focus on enhancing current features, refining the already present code and introducing new tools. Enhancements to the polyline feature, the development of SimpleSpline and PolySpline dictionary generators and respective GUI panels as necessary for both features.

REFERENCES

CFD FOSSEE	https://cfd.fossee.in/
Blender	https://www.blender.org/
OpenFOAM	https://www.openfoam.com/
PyVNT	https://github.com/FOSSEE/pyvnt