Semester Long Internship Report

On

Implementing Splines and Usage Tutorial System for OpenFOAM GUI

Submitted by

Patel Aarsh Miteshkumar
Vellore Institute of Technology, Chennai

Under the guidance of

Mr. Rajdeep Adak
OpenFOAM FOSSEE GUI Developer

&

Prof. Janani Srree Murallidharan
Mechanical Engineering Department
IIT Bombay

Contents

List of Figures

Abstract

This paper describes the software development process of an OpenFOAM spline interpolation tool in Blender [5] and a teaching system in PyQt5. OpenFOAM, a popular open-source computational fluid dynamics (CFD) program, uses splines as one of the curve definition methods. However, for users without prior expertise or specialized knowledge, building and manipulating splines in OpenFOAM might be difficult.

To overcome this issue, we created a Blender-addon that facilitates spline construction for OpenFOAM. Users can create, edit, and change splines with the simple controls and tools. It provides options for defining handles, control points, and spline interpolation techniques. The panel improves productivity by streamlining the process.

Additionally, we used PyQt5 to integrate a tutorial system into Blender to aid in the learning process for OpenFOAM's spline construction. Users get the ability to efficiently design complex shapes, which alleviates the learning curve and enhances productivity. It is possible to effortlessly install the addon and the tutorial system into Blender.

FOSSEE's OpenFOAM GUI Project

The OpenFOAM GUI project at FOSSEE aims to develop tools to conveniently generate OpenFOAM cases. Presently, Venturial [1], an addon in Blender provides interactive mesh handling tools that minimises the effort to generate blockmesh dictionaries. This document presents the work done to develop some of Venturial's features that extend its capabilities to address more complex geometries. Venturial is inspired from reynolds-blender [3] (Surti et al., 2017), and has been in development since 2021, when the groundwork for using Blender's UI for generating blockmesh dictionaries commenced (R. Adak & K. Kumar Thakur, *OpenFOAM GUI development using Python on Blender - Fossee* 2021) [2]. Since then Venturial has acquired several features that make the workflow of writing mesh dictionaries more productive. Venturial is being developed further to provide a comprehensive interface to address the task of meshing, solving and post-processing within the same interface.

Introduction

Splines are widely used in a variety of industries, including computational fluid dynamics (CFD), animation, and computer graphics. OpenFOAM has become a well-known open-source software program for simulating fluid flows in the field of CFD. Splines play a significant role in OpenFOAM's ability to precisely specify complex geometries and boundary conditions. Spline creation and manipulation in OpenFOAM can be complicated and time-consuming, which can be difficult for users with little background in or skill with spline manipulation.

To address these challenges and provide a user-friendly solution, we present a Blender panel for efficient spline creation in OpenFOAM, by a tutorial system developed using PyQt5. Blender, a popular 3D computer graphics software, serves as the platform for seamlessly integrating spline creation tools and facilitating an intuitive user experience [6]. Leveraging the power of PyQt5, a Python library for developing graphical user interfaces, a comprehensive tutorial system that guides users through the intricacies of spline manipulation.

Complementing the spline creation panel, we introduce a tutorial system built using PyQt5 that seamlessly integrates into Blender. This tutorial system serves as a valuable learning resource, catering to users with varying expertise in spline manipulation for OpenFOAM. It offers step-by-step instructions to guide users through the intricacies of spline creation techniques. The tutorial system follows a structured learning path, starting with fundamental concepts and gradually progressing towards advanced workflows, empowering users to gain proficiency at their own pace.

By simplifying the spline creation process, reducing the learning curve, and enhancing user productivity, this solution enables users to generate complex geometries accurately and efficiently. Moreover, the seamless integration of the panel and tutorial system within Blender ensures accessibility and ease of adoption for a broad range of users in the CFD community.

Implementing OpenFOAM Splines in Blender

The Spline manipulation tool allows the users to create and manipulate the spline using Blender's interface. With comprehensive geometry manipulation capabilities and visualization features, users can overcome the lack of a GUI for spline operations in OpenFOAM. This method facilitates a user-friendly and streamlined workflow, allowing users to interactively manipulate spline-based geometries, while seamlessly integrating their work with the OpenFOAM simulation process.

Design Considerations:

1) **Usage Convenience**: A user-friendly interface that allows users to easily interact with the spline creation tool. It organizes controls and functions logically, providing clear labels and tooltips that guide users through the spline creation process.
2) **Real-time Feedback**: It offers immediate visual feedback during spline creation, such as updating the spline curve in real-time as control points are adjusted. This allows users to observe the effects of their actions instantly and make necessary adjustments.
3) **Visual Clarity**: The UI elements and controls are visually distinct and easy to locate. Appropriate iconography and colour schemes are used to enhance the clarity and legibility of the spline creation tool.
4) **Customizability**: Options are provided for users to customize the appearance of the spline curves, control points, and other visual elements. This allows the users to tailor the visual representation to their specific preferences and requirements.
5) **Compatibility**: The spline-generation algorithm utilized is different than that of OpenFOAM. Consequently, the users may observe a slight deviation in spline during the simulation compared to the one they created using the GUI. This can be rectified in the future iterations by utilizing the same algorithm as that of the OpenFOAM.

Underlying dependencies

The GPU and BGL libraries found in the Blender API are being utilized to fix this problem.[4] The GPU module speeds up rendering and computations in Blender by utilizing the graphics processing unit's (GPU) capabilities. This module includes practical texture mapping tools like texture mixing and texture filtering, all of which improve the aesthetic appeal of Blender-rendered scenes. The BGL module of the Blender API acts as a low-level interface, providing access to OpenGL resources[7] and functions created especially for Blender graphics development. Custom rendering and visualization techniques in Blender can be built by using this module's functions for drawing geometry, lines, points, and textures.

Spline Manipulation Tool (architecture)

A MVC (Model View Controller) has been implemented using Blender API to decouple user-interface as “View”, temporary data as “Model” and application logic as “Controller”. A Spline model belonging to a hexahedral block is defined using a collection of splines each having their own set of interpolation points. The controller maintains a coherence with the Blender Panel and the 3D viewport. The interaction between Model and View is not completely separated but minimised as much as possible.

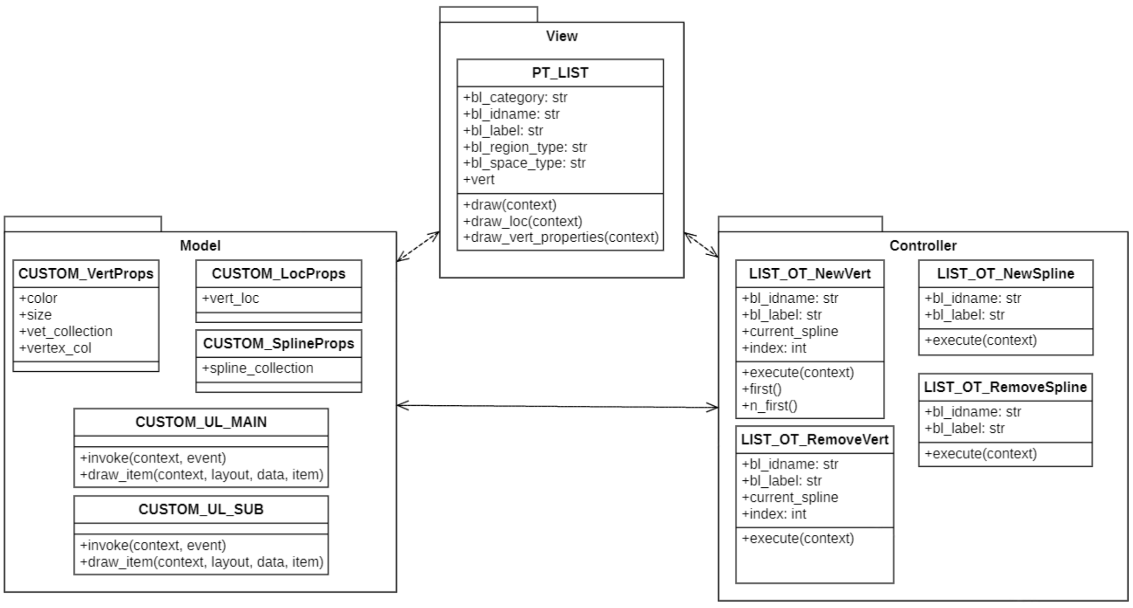


Figure 1 Spline Tool MVC Architecture

Spline Manipulation Tool (Panel layout)

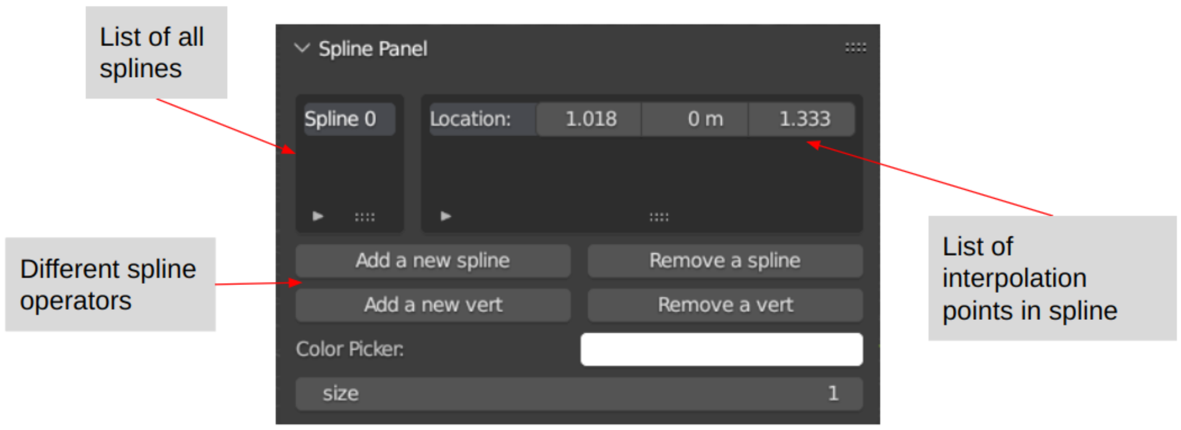


Figure 2 Spline Panel View

In the View part of the MVC, a class called "PT_List" that represents a panel within the Blender user interface is created. Its primary functionality involves displaying and manipulating a collection of spline objects along with their associated vertex collections. The "draw" method is responsible for defining the layout of the panel and rendering the user interface elements. Upon execution, it retrieves the necessary data from the "context" and "scene" variables of the Blender API. The "layout" variable is employed to arrange the UI components within the panel.

To create the layout, a split layout is employed using the "layout.split()" function[4]. On the left side of the split, a list template for the spline collection is presented. The type of list to be displayed is specified as "CUSTOM_UL_Main", a custom-defined list type for all the splines. The active item in the list is determined based on the scene's spline index, which can be set using the "active_dataptr" variable, which forms the "Model" part of the MVC.

If there are splines in the collection and a spline is selected, an additional list template on the right side of the split is displayed. This new list represents the collection of interpolation points associated with the selected spline. Similar to the spline list, the vertex list is defined using "CUSTOM_UL_Sub" as a custom-defined list type for all the interpolation points in the current spline.

```
class CUSTOM_LocProps(bpy.types.PropertyGroup):
    vert_loc: FloatVectorProperty(name='verts')

class CUSTOM_VertProps(bpy.types.PropertyGroup):
    vert_collection: CollectionProperty(
        name = "Vert_Collection",
        type = CUSTOM_LocProps)
    vertex_col: CollectionProperty(
        name = "Vert_Collection_for_changing_and_storing_intermediate_values"
        type = CUSTOM_LocProps)
    color : FloatVectorProperty()
    size : IntProperty()

class CUSTOM_SplineProps(bpy.types.PropertyGroup):
    spline_collection: CollectionProperty(
        name = "Spline_Collection",
        type = CUSTOM_VertProps)
```

Figure 3 Collection Property Pseudocode

Pseudo code for adding spline

```
add a new element in spline collection
```

Pseudo code for removing interpolation point:

```
get the current spline index
remove the index from spline collection
```

Figure 4 Adding / Removing Spline Pseudo

```
Pseudo code for adding interpolation point
-----------------------------------------------------------------
  try:
    get the index of the spline from the scene
    get the current spline from the spline collection using the index
  except Exception then
    report an error
    return {'FINISHED'}
  if there are no vertices in the current spline then
    call the first() function
  else
    call the n_first() function
  end if
  return {'FINISHED"}
-----------------------------------------------------------------
def first():
  switch to edit mode
  get all selected edges
  if len(selectedEdges) $>$ 1 then
    report an error and return
  end if
  if len(selectedEdges) $<$ 1 then
    report an error and return
  end if
  self.current_spline.add()
  create two more vertices and add them to the current spline
  self.current_spline[0] = Selected_Vert[0]
  self.current_spline[2] = Selected_Vert[1]
  self.current_spline[1] = (self.current_spline[0] + self.current_spline[2])/2
  create a new interpolation point at the location of the second vertex
-----------------------------------------------------------------
def n_first():
  set the gizmo object translate to true
  len1 = len(current_spline)
  current_spline.add()
  for each current_spline index do
    current_spline[i] = current_spline.vertex_col[(i+1)*len1//(len1+2)]
  end for
  create a new vertex object
  set its name to be the name of the current spline plus its length
  for every vertex in current_spline do
    vertex_location = current_spline[i]
  end for
```

Figure 5 Adding Interpolation Point Pseudocode

- Get index of current spline.
- If element in current spline is 0 then get all the selected edges location and put in spline collection
- If element present in current spline, then add a new interpolation point in collection property and re-calculate location of each interpolation point.

Psedo code for Removing interpolation point

```
get the index of the current spline
get the index of the last vertex in the current spline's vertex collection
remove last vertex from current spline
for each vertex in the current_spline:
  vertex.location = current_spline[i]
end for
if len(current_spline) is equal to 0 then
  spline_collection.remove(current_spline)
  remove draw handler
end if
```

Figure 6 Removing interpolation point pseudocode

- Get the index of current spline and the last vertex of current spline.
- Remove the last vertex from spline collection property.
- If spline has no interpolation points left delete the spline as well.

Pseudo code for draw handler

```
select GPU shader of "3D_UNIFORM_COLOR"
set the linewidth
set the color from input
input the vertices in batch for shader
bind the shader
draw the shader using "draw" function
```

Figure 7 Draw Handler Pseudocode

Buttons are incorporated for adding and removing splines and vertices. These buttons are associated with their specific operators. Furthermore, if there are splines in the collection, properties of the selected spline, such as color and size, are displayed within the panel using the “active_dataptr” variable. For the “Controller” part, whenever the spline is modified by the user, an update function is called via the Blender API. This update function uses the modified values of the interpolation points and CubicSpline interpolation of Scipy is used to generate the spline. The equation governing cubic spline is given below [9]. [8]

$$S(x) = \begin{cases} C_1(x), & x_0 \leq x \leq x_1 \\ C_i(x), & x_{i-1} \leq x \leq x_i \\ C_n(x), & x_{n-1} \leq x \leq x_n \end{cases}$$

where C_i is a cubic function that has generalized form

$$C_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$$

The Catmull-Rom spline algorithm used by OpenFOAM for interpolation could not be implemented. However, the current implementation of CubicSpline yields comparable results. The equation governing Catmull-rom spline is described below.

$$p(s) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau & -2\tau \end{bmatrix} \begin{bmatrix} p_{i-2} & p_{i-1} & p_i & p_{i+1} \end{bmatrix}$$

The "tension" parameter τ controls how aggressively the spline curves at the interpolation points in Catmull-Rom spline where P_i are the interpolation points. [10]

After the interpolation points are calculated, the BGL module in Blender can be used to display the spline in the 3D viewport using the “draw_handle_add” function call. The final output would be as described below.

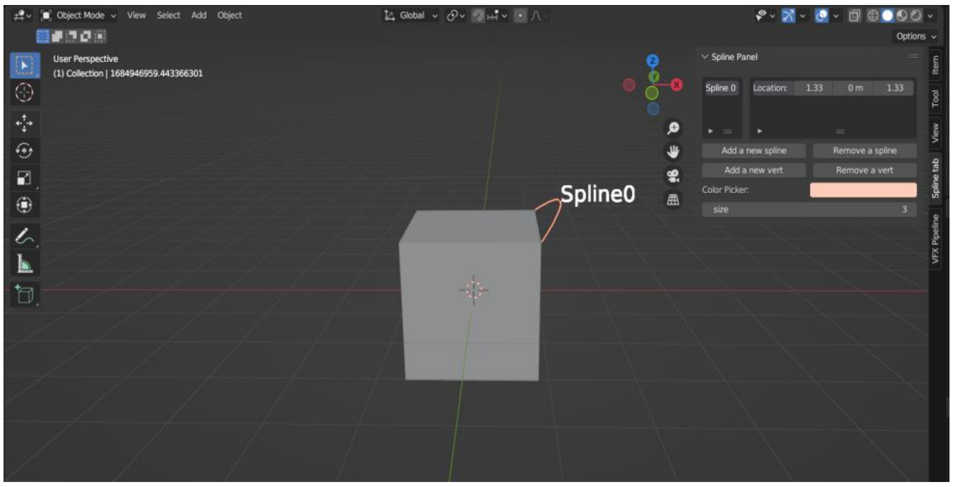


Figure 8 Spline Manipulation Tool Layout

Operation Flowchart

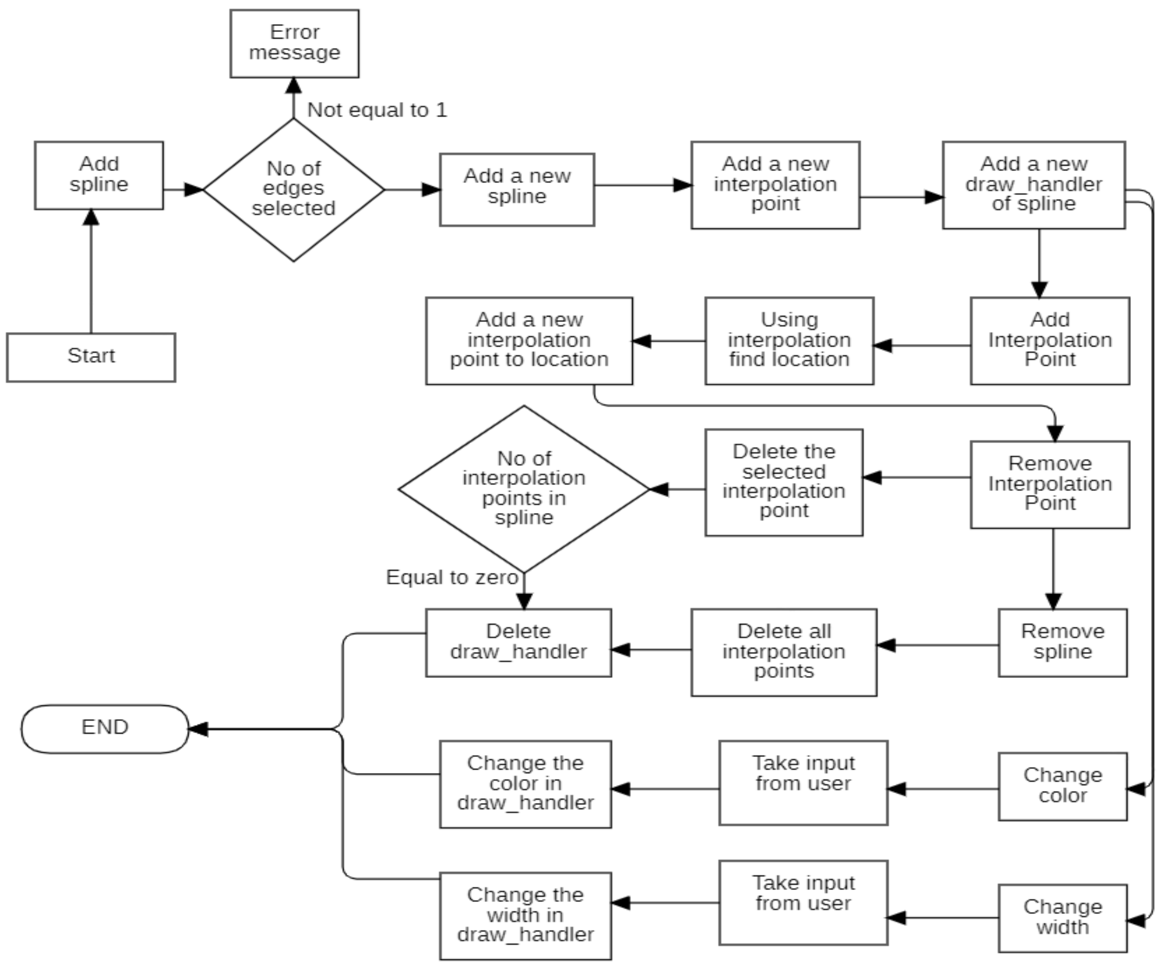
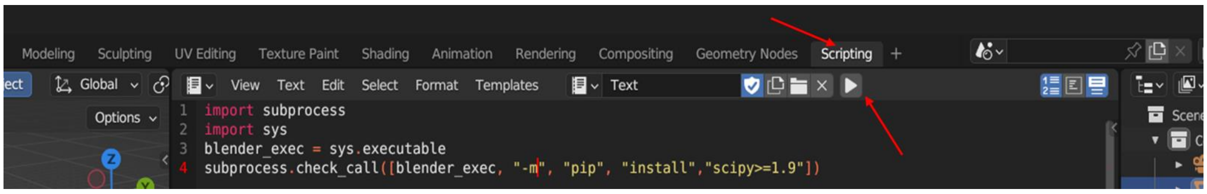

Error message
Not equal to 1
Add spline
No of edges selected
Add a new spline
Add a new interpolation point
Add a new draw_handler of spline
Start
Add a new interpolation point to location
Using interpolation find location
Add Interpolation Point
No of interpolation points in spline
Delete the selected interpolation point
Remove Interpolation Point
Equal to zero
Delete draw_handler
Delete all interpolation points
Remove spline
END
Change the color in draw_handler
Take input from user
Change color
Change the width in draw_handler
Take input from user
Change width

Figure 9 Flowchart of Operation (Spline-Manipulation tool)

Installation Instructions

1. Blender (Version 3.4 or above)
2. SciPy (Version 1.9 or above)
3. Open “Scripting Panel” in the Header section.
4. Press “+ New” Button.
5. Write the following commands in the text editor and press the play button

```
import subprocess
import sys
blender_exec = sys.executable
subprocess.check_call([blender_exec, "-m", "pip", "install","scipy>=1.9"])
```

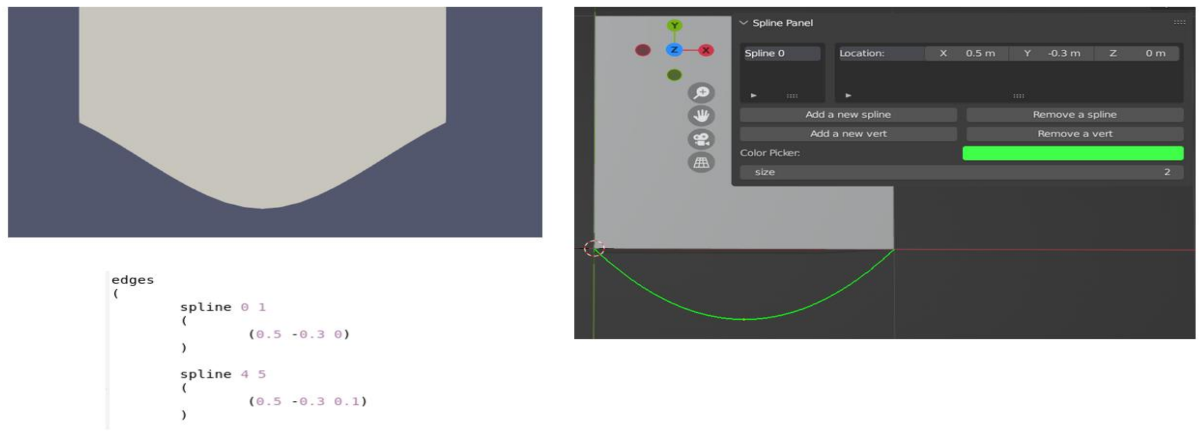


```python
import subprocess
import sys
blender_exec = sys.executable
subprocess.check_call([blender_exec, "-m", "pip", "install","scipy>=1.9"])
```

Figure 10 Scipy Installation in Blender

Validation Cases

The scipy spline interpolation method produces a set of estimated points that closely resemble to that of Catmull-Rom splines, however, there are visible areas where the splines fail to reproduce the geometry in Paraview. In test, with large number of interpolation points, such inaccuracies are not noticeable (as shown in Figure 12), however, for small number of interpolation points there are identifiable changes in curvature. A possible solution to this problem would be to replicate the same Catmull-Rom spline interpolation method in Python, however, there may be variations in spline implementation with other OpenFOAM versions. That said, the scipy interpolation method is replaceable in case a better, already available, interpolation method is found, or until Catmull-Rom splines can be accurately written for Python. Another possible option would be creating a wrapper around OpenFOAM's source Catmull-Rom splines. The output from this wrapper can be interpreted into points for BGL to generate a curve from infinitesimally small lines. The data within the spline manipulation panel UI isn't writable. The interpolation points usually require to be eyeballed to place them correctly. Estimators or convenience tools that enable snapping to pre-defined surfaces, edges or corners could also be implemented to aid in accurately modifying the spline. Nonetheless, the interpolation point coordinates can be controlled either via the panel or Blender's inherent transformation tools.



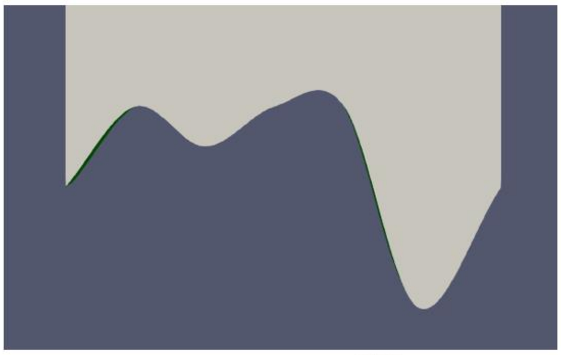
Spline 0 — Location: X 0.5 m Y -0.3 m Z 0 m

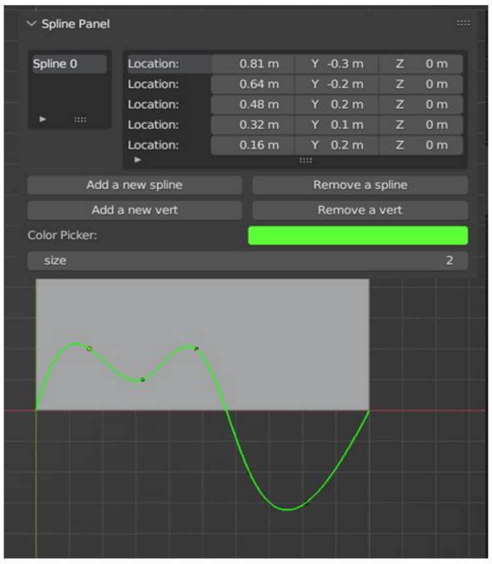
size 2

```
edges
(
        spline 0 1
        (
                (0.5 -0.3 0)
        )

        spline 4 5
        (
                (0.5 -0.3 0.1)
        )
```

Figure 11 Spline Validation Case-I: Interpolation Point



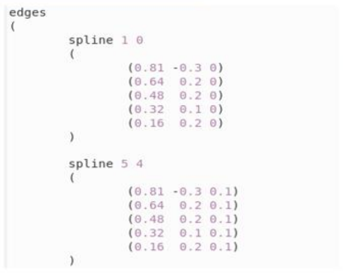


Spline Panel

Location:	0.81 m	Y	-0.3 m	Z	0 m
Location:	0.64 m	Y	-0.2 m	Z	0 m
Location:	0.48 m	Y	0.2 m	Z	0 m
Location:	0.32 m	Y	0.1 m	Z	0 m
Location:	0.16 m	Y	0.2 m	Z	0 m

Color Picker: 🟩

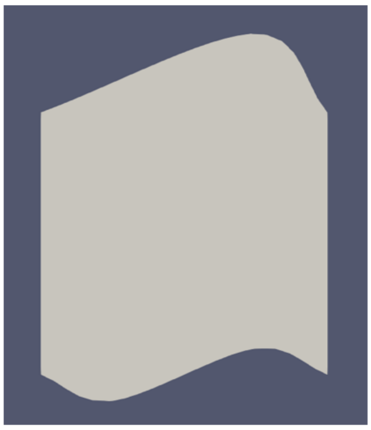
size 2

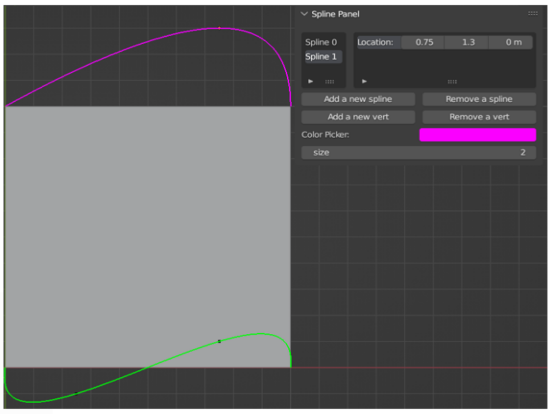


```
edges
(
        spline 1 0
        (
                (0.81 -0.3 0)
                (0.64  0.2 0)
                (0.48  0.2 0)
                (0.32  0.1 0)
                (0.16  0.2 0)
        )

        spline 5 4
        (
                (0.81 -0.3 0.1)
                (0.64  0.2 0.1)
                (0.48  0.2 0.1)
                (0.32  0.1 0.1)
                (0.16  0.2 0.1)
        )
```

Figure 12 Spline Validation Case-II: Multiple Interpolation points



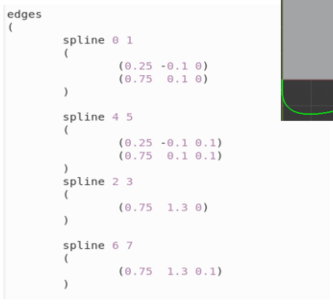


Spline Panel

Location: 0.75 1.3 0 m

Color Picker: 🟪

size 2



```
edges
(
        spline 0 1
        (
                (0.25 -0.1 0)
                (0.75  0.1 0)
        )

        spline 4 5
        (
                (0.25 -0.1 0.1)
                (0.75  0.1 0.1)
        )
        spline 2 3
        (
                (0.75  1.3 0)
        )

        spline 6 7
        (
                (0.75  1.3 0.1)
        )
```

Figure 13 Validation Case-III: Multiple Splines

Tutorial System (Blender-PyQt)

A Tutorial Viewer using the Model View Controller (MVC) design pattern is to be developed for Blender. By integrating this functionality, users' productivity will be enhanced, as they will no longer need to spend excessive time searching for GUI functionalities, resulting in a streamlined learning and workflow experience.

Design Considerations

1. **Better learning experience**: This instructional system offers structured assistance and explanations, which improves learning for beginners and aids in their better comprehension of difficult ideas. A tutorial system enables users to learn new skills fast and master numerous tools, techniques, and workflows by providing step-by-step explanations and interactive demos.
2. **Integration with new OpenFOAM versions**: As OpenFOAM continues to evolve and release new versions, the tutorial system can adapt and integrate with these updates, ensuring users have access to the latest features and capabilities.

Underlying dependencies:

PyQt5 is a Python package that provides Qt framework interfaces. It is used in the development of the novel Blender GUI plugin for OpenFOAM. This enables the construction of graphical user interfaces (GUIs). During the implementation of the PyQt5 component, the primary objective was to ensure that the GUI window was integrated without interrupting Blender's main loop.

"Modal operators" are employed to accomplish this. Modal operators are designed to operate independently of the primary Blender loop. When a modal operator is activated, Blender waits for user input and allows for scene updation, perform calculations, or modify objects based on that input. Thus, by utilizing modal operators, the PyQt5 window can appear and interact with users without disrupting the central operations of Blender.[11]

Tutorial System (Design)

The tutorial system also follows the Model-View-Controller design pattern, however, there is complete seperation between model and view. The controller allows both reading and writing of Models represented within the view. Models can be modified graphically but when written into memory, are routed via the controller.

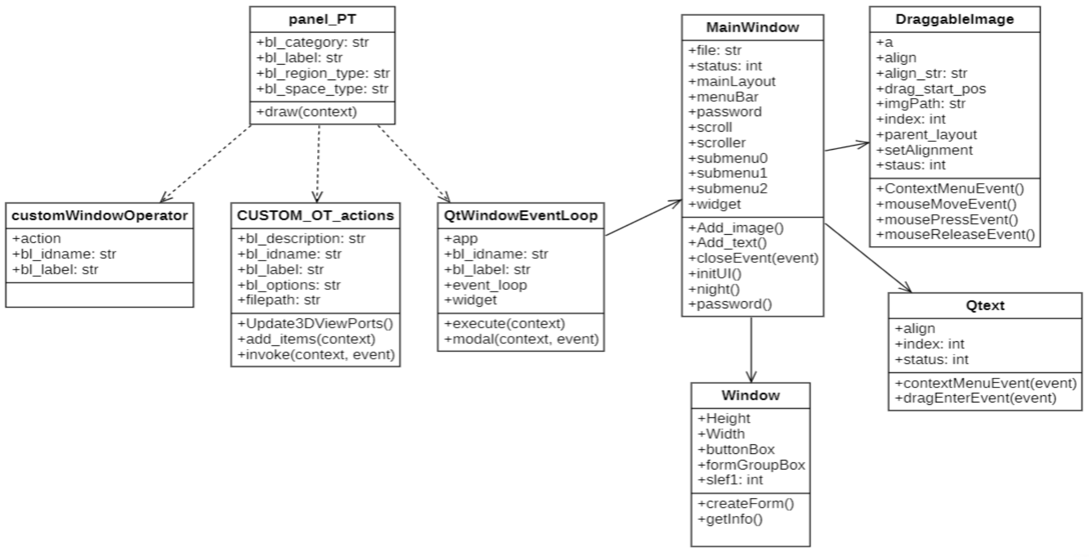
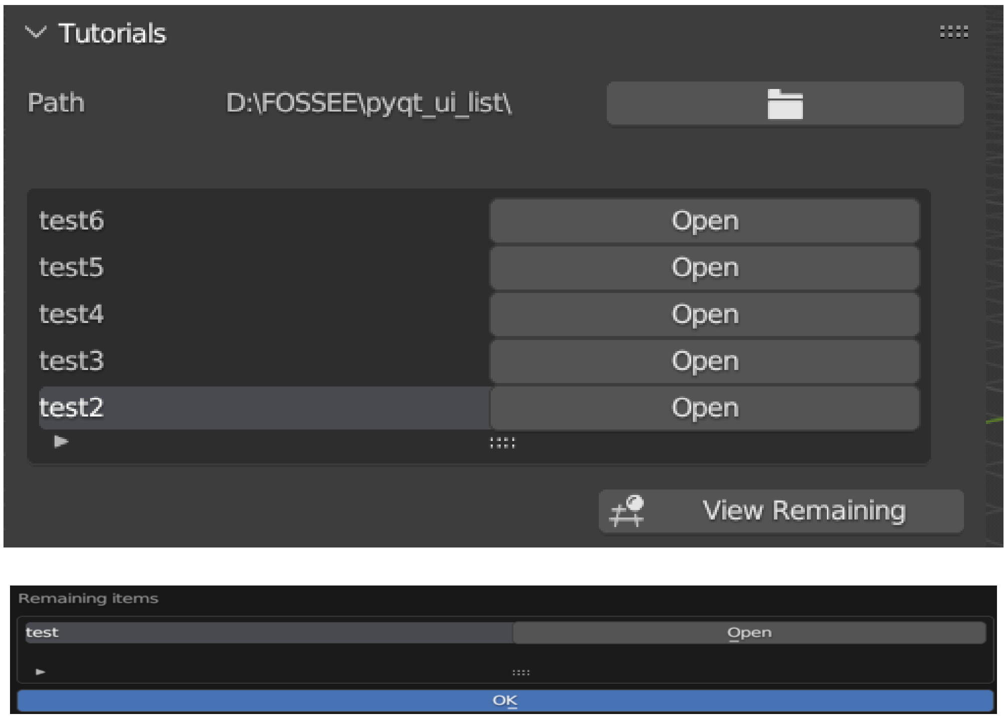


Figure 14 Class Diagram (Tutorial System)

Tutorial System (Blender - view)

A panel class in Blender API is implemented whose primary functionality involves displaying a list of all the available tutorials in the tutorials folder. To make UI more intuitive and easier to use, only 5 recent tutorials are shown and for rest a button with a pop-up operator is implemented that displays a list of remaining tutorials.



Tutorials

Path D:\FOSSEE\pyqt_ui_list\

test6	Open
test5	Open
test4	Open
test3	Open
test2	Open

View Remaining

Remaining items

test	Open

OK

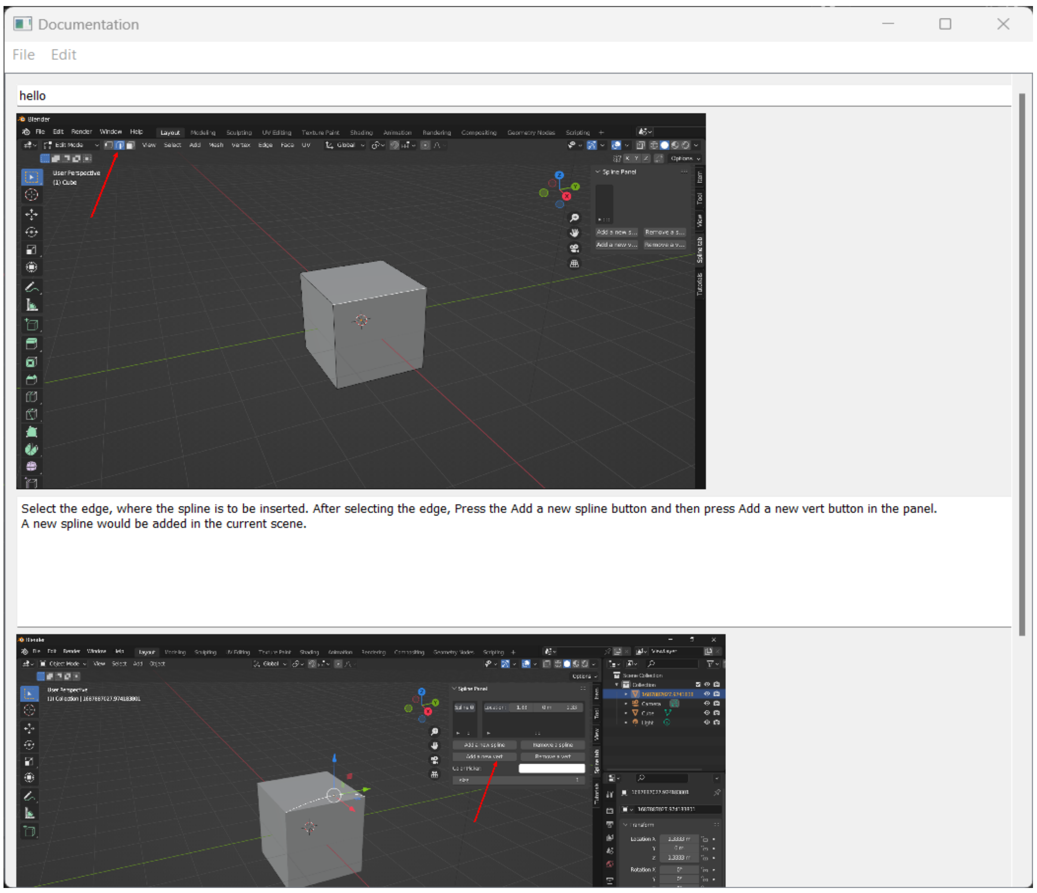
Figure 15 Tutorial System Blender Panel

When the "Open" operator is executed, it invokes the "screen.custom_window" function, which inherits the "Main window" class of PyQt5, which itself is a "modal Operator". As a result, the PyQt5 [6] window opens asynchronously. Within PyQt5, two classes, namely "DraggableImage" and "TextBox," are implemented to serve as the foundation for the tutorial system. The tutorial creator has the ability to drag images to their desired positions in the vertical axis and adjust their sizes based on user specifications.

The “DraggableImage” class extends the “QtWidgets.QLabel” to create draggable image widgets in Qt. “drag” method is created inside this class that is utilized to drag images in the vertical plane using a mouse. Context menus for actions like resizing, deleting, and alignment are also provided.

The “Qtext” class is a custom widget that extends “QtWidgets.QTextEdit” in Qt. It allows users to input and manipulate text. The widget provides a context menu with options like deleting and changing the font. It also allows users to select a font using a font dialog.

Tutorial View (PyQt window)



hello

Select the edge, where the spline is to be inserted. After selecting the edge, Press the Add a new spline button and then press Add a new vert button in the panel.
A new spline would be added in the current scene.

Figure 16 Tutorial PyQT Window

The tutorials are maintained using the YAML file format. The "ml-text" and "img" keys in the YAML dictionary are used to store text and images, respectively. The value for "ml-text" defines the tutorials' text content, while the value for "img" specifies the image's relative location in the tutorial folder.

A tutorial protection system has been developed utilizing the SHA256 algorithm that prevents users from modifying the tutorials. The SHA256 algorithm is utilized for cryptographic purposes. [12] In this algorithm, a message or data input is transformed into a fixed-size output, referred to as a hash value, which is 256 bits in length which is then stored instead of the original password.

Pseudo code for modal class

```
def init(self):
  create a window_manager variable
  return {'PASS_THROUGH'}
def execute(self,context):
  create a new instance of Qapplication
  set the stylesheet
  Create Qt event loop
  Add the window to the window manager
return {'RUNNING_MODAL'}
```

Figure 17 Modal class Pseudocode

- In init function create a new variable of type "window manager"
- In execute function Create a new instance of QAppplication and set the stylesheet.
- Create a new Qt event loop
- Add the window to the window manager.

Pseudo code for graphical operators

```
if left mouse button pressed do
   get the index of the object
   pop the object from the list
end if
if left mouse button released do
   get the index of object below mouse location
   insert the object between the index and index+1
end if
```

Figure 18 Graphical operator pseudocode

- If left mouse button is pressed get the index of the object.
- And pop the object from the list and apply hover effect.
- When the left mouse button is released get the index of the object below the mouse cursor.
- Insert the popped object after the index of the object below the mouse cursor.

Pseudo code for YAML file controller

```
for each object in view do
  if class of object is text_box do
    add a key in YAML file
    save the text as value for the key
  end if
  if class of object iis draggable_image do
    add a key in YAML file
    save the location of image as value of the key
  end if
```

Figure 19 YAML file controller pseudocode

- Loop through each object in the view and get the class of object.
- If class is text_box then in YAML file save key as ml-text and value as the text.
- If class is draggable_img then in YAML file save key as img and value as the image location.

Pseudo code for PYQT view

```
for each key in YAML file do
  if key is equal to "ml-text" then
    add a new text_box object with the text as the value
  end if
  if key is equal to "img" then
    add a new draggable_image object with value
  end if
```

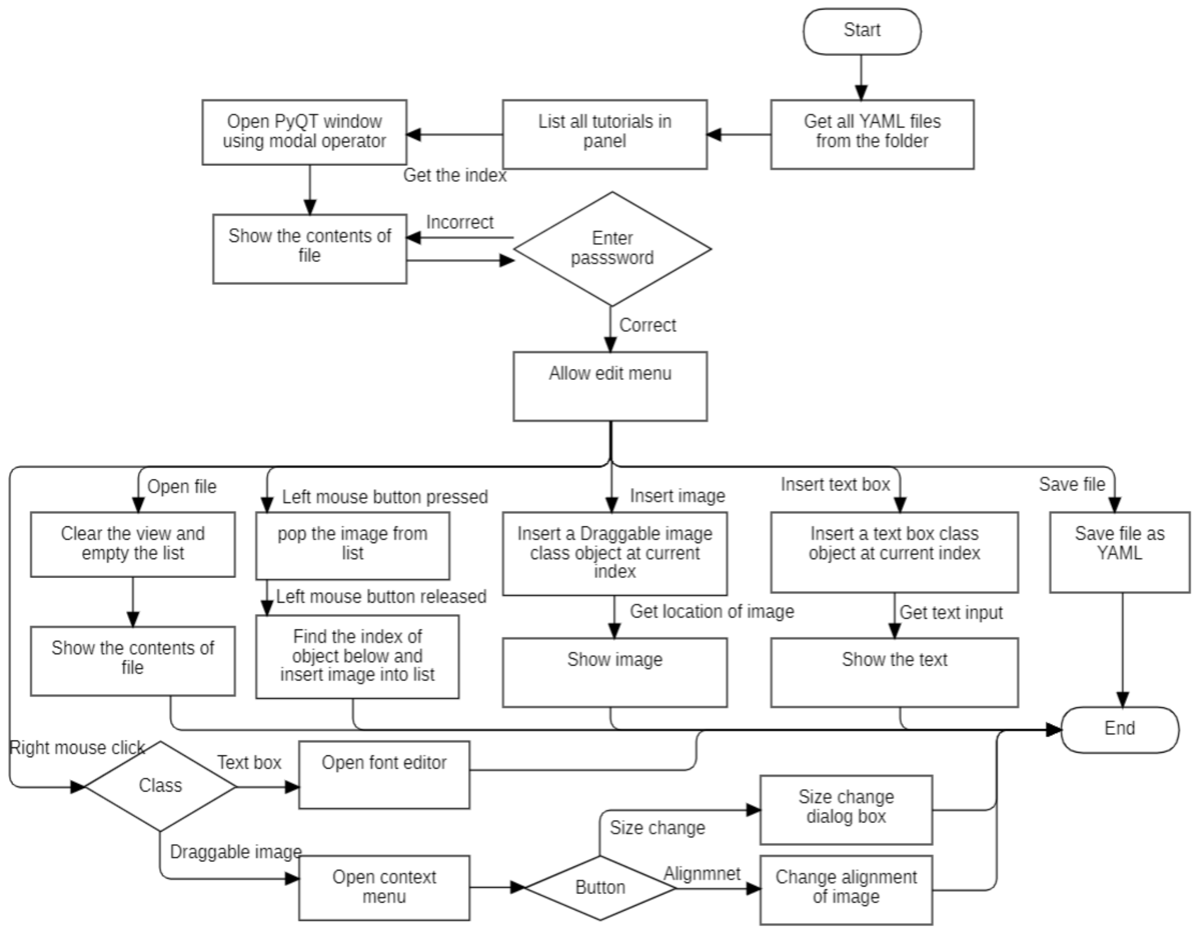
Figure 20 PYQT view pseudocode

Schema for YAML file

```
type: array
  items:
    type: object
    properties:
      ml-text:
        type: string
      img:
        type: string
```

Figure 21 YAML file schema

Operation Flowchart



Start
Get all YAML files from the folder
List all tutorials in panel
Open PyQT window using modal operator
Get the index
Show the contents of file
Incorrect
Enter passsword
Correct
Allow edit menu
Open file
Left mouse button pressed
Insert image
Insert text box
Save file
Clear the view and empty the list
pop the image from list
Insert a Draggable image class object at current index
Insert a text box class object at current index
Save file as YAML
Left mouse button released
Get location of image
Get text input
Show the contents of file
Find the index of object below and insert image into list
Show image
Show the text
End
Right mouse click
Class
Text box
Open font editor
Size change dialog box
Size change
Draggable image
Open context menu
Button
Alignmnet
Change alignment of image

Figure 22 Flowchart of operation (Tutorial System)

Installation Instructions

1. Blender (Version 3.4 or above)
2. PyQT5 (Version 5.15 or above)
3. Open “Scripting Panel” in the Header section.
4. Press “+ New” Button.
5. Write the following commands in the text editor and press the play button

```
import subprocess
import sys
blender_exec = sys.executable
subprocess.check_call([blender_exec, "-m", "pip", "install","PyQT5>=5.15"])
```

References

[1] R. Adak, J. S. Murallidharan, and P. Ramachandran, “Fossee/Venturial: Venturial is a GUI for openfoam.,” GitHub, https://github.com/FOSSEE/venturial (accessed Jul. 4, 2023).

[2] R. Adak and K. K. Thakur, “OpenFOAM GUI development using Python on Blender - Fossee,” https://fossee.in/semester-internship/2021, https://static.fossee.in/fossee/internship-reports/Python-Blender/Rajdeep_Adak.pdf (accessed Jul. 4, 2023).

[3] D. Surti, P. Ramachandran, and G. Shivasubramanian , “Dmsurti/reynolds-blender: A reference implementation using Blender to demonstrate integration of Reynolds with a 3D GUI interface.,” GitHub, https://github.com/dmsurti/reynolds-blender (accessed Jul. 4, 2023).

[4] Blender, “Blender 3.5 Python API Documentation — Blender Python API,” docs.blender.org, 2023. https://docs.blender.org/api/current/index.html

[5] Blender, “Blender 3.5 Reference Manual — Blender Manual,” Blender.org, 2023. https://docs.blender.org/manual/en/latest/

[6] Qt Group, “PySide2.QtWidgets — Qt for Python,” doc.qt.io, 2020. https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/index.html

[7] J. Rodríguez, “docs.gl,” docs.gl, 2012. https://docs.gl/

[8] SciPy. “SciPy.org — SciPy.org.” Scipy.org, 2020, scipy.org/

[9] Ohio University, “Polynomial and Spline Interpolation,” http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/lecture19.pdf

[10] Christopher Twigg, “Catmull-Rom Spline,” Mar. 11, 2003. https://www.cs.cmu.edu/~fp/courses/graphics/asst5/catmullRom.pdf

[11] Blender, “Blender wiki” Blender.org, 2023. https://wiki.blender.org/wiki/Source/Interface/Operators#Modal_Operators

[12] Dang, Q. H. (2015). Secure Hash Standard. CSRC. https://doi.org/10.6028/nist.fips.180-4