Enhancing the OpenFOAM GUI: Migration and Improvements for Blender 4.2 Compatibility

Budhil Nigam
B.Tech - Computer Engineering
Jamia Millia Islamia

Acknowledgement

I would like to extend my heartfelt gratitude to **Prof. Chandan Bose**, **Mr. Diptangshu Dey** and **Mr. Rajdeep Adak** for their exceptional mentorship and constant support throughout the duration of my semester-long internship with the **OpenFOAM GUI Project at FOSSEE, IIT Bombay**.
Their depth of knowledge in computational fluid dynamics and open-source software development has profoundly influenced my understanding, and I am sincerely thankful for their patience, clarity, and encouragement during each stage of this journey.

I am equally thankful to the entire staff at FOSSEE, IIT Bombay for providing an inspiring and collaborative work environment. Their professionalism and enthusiasm made the experience not only productive but also deeply enriching. The opportunity to engage with a team so committed to innovation in free and open-source software has been both a privilege and a learning milestone.

This report is the culmination of numerous discussions, reviews, and refinements—each made possible by the generous guidance of the individuals acknowledged above. I remain genuinely appreciative of their unwavering support and contribution to my academic and professional growth.

Contents

List of Figures

1 Introduction

OpenFOAM[4] is a free open-source C++ toolbox for solving continuum mechanics problems. It requires the user to manually write the case files in text format, which may be inconvenient. Having a Graphical User Interface(GUI) for the software makes it easier to interact with it without having to learn the technicalities.

The OpenFOAM GUI project aims to develop a tool to alleviate the process of generating FOAM cases.

OpenFOAM does not have an integrated GUI, but there are many third-party GUI software products. However, they are inadequately comprehensive for the OpenFOAM solution workflow. To achieve that, an API was developed earlier to act as an intermediate piece of software between OpenFOAM and the GUI software named ‘pyvnt’.

Two GUI interfaces have been made previously under the OpenFOAM GUI project, a Blender addon and a PyQt application. Blender software provides a way to interact with its components through the ’bpy’ python package. Blender also has a rich node system to represent hierarchical data. Hence, a GUI is made as a Blender addon using the nodes to represent OpenFOAM case files.

However, the blender add-on ’Venturial’ was built upon Blender 3.6 LTS which will reach it’s end of support by June 2025. Therefore, there was a need to migrate the add-on to Blender 4.2 LTS which is the latest long-term version of the Blender software. Along this, several new bug fixes and new features have been implemented.

2 Migration to Blender 4.2 LTS

2.1 Fix: Curves Visualization Not Rendering in Blender 4.2 LTS

After upgrading the Venturial add-on to support Blender 4.2 LTS, a regression was observed in which curve visualizations, such as custom boundary or edge overlays drawn using GPU shaders, were no longer visible in the 3D viewport. These drawings are crucial for representing user-defined geometry and visual feedback in the GUI.

Cause:
Blender 4.2 LTS introduced an API change in the `gpu.shader` module [2]. The shader previously obtained using:

```
shader = gpu.shader.from_builtin('3D_UNIFORM_COLOR')
```

is now deprecated or behaves differently. Specifically, the API no longer accepts the string `'3D_UNIFORM_COLOR'` directly and requires updated naming conventions or alternatives.

Solution:
The shader initialization was corrected to use the appropriate shader key supported in Blender 4.2 LTS. The draw handler code was updated to explicitly fetch and bind the correct shader, ensuring that the 3D drawing overlays would render as intended.

The relevant draw function was modified as follows:

```
import gpu
from gpu_extras.batch import batch_for_shader

def draw_curve():
    # Corrected shader initialization for Blender 4.2 LTS
    shader = gpu.shader.from_builtin('UNIFORM_COLOR')

    batch = batch_for_shader(shader, 'LINE_STRIP', {"pos": verts})

    # Bind shader and draw the curve
    shader.bind()
    shader.uniform_float("color", col)
    batch.draw(shader)
```

Impact:
This fix restored the expected curve visualization behavior within the Blender viewport. The add-on now correctly renders overlay graphics such as boundaries and guides, maintaining its usability in Blender 4.2+.

2.2 Upgrade: Migration from Deprecated OpenGL Wrapper (bgl) to GPU Module

As part of the compatibility enhancement for Blender 4.2 LTS, deprecated usage of the `bgl` module in the 'Visualize' tab was identified and replaced. The `bgl` module, previously used for OpenGL state management and rendering, is deprecated in Blender's future API. It was necessary to transition the rendering logic to the modern `gpu` module to maintain add-on compatibility.

Cause:
The drawing logic for edges and bound properties used `bgl.glEnable`, `bgl.glDisable`, and `gpu.shader.from_builtin("3D_SMOOTH_COLOR")`, which are no longer fully supported. These calls caused instability and missing visualizations in Blender 4.2 LTS.

Solution:
All `bgl` state control calls were replaced with `gpu.state` equivalents. Additionally, the shader call was updated from the deprecated `"3D_SMOOTH_COLOR"` to `"SMOOTH_COLOR"`, as per updated GPU API requirements.

Code Changes:
The following function demonstrates how both OpenGL state control and shader usage were updated:

```
def draw_edge_properties(self, operator, context, geo):
    # Set GPU state
    # Before
    bgl.glEnable(bgl.GL_BLEND)
    bgl.glEnable(bgl.GL_LINE_SMOOTH)
    bgl.glEnable(bgl.GL_DEPTH_TEST)

    # After
    gpu.state.blend_set('ALPHA')
    gpu.state.depth_test_set('LESS_EQUAL')

    for point in self.get_edge_properties(geo):
        self.draw_line_3d((0.0, 1.0, 0.0, 0.7), point, geo.location)

    # Reset state
    # Before
    bgl.glDisable(bgl.GL_BLEND)
    bgl.glDisable(bgl.GL_LINE_SMOOTH)
    bgl.glDisable(bgl.GL_DEPTH_TEST)

    # After
    gpu.state.blend_set('NONE')
    gpu.state.depth_test_set('NONE')
```

Additionally, the shader used in the `draw_bound_properties` function was updated as follows:

```
# Old
self.shader = gpu.shader.from_builtin("3D_SMOOTH_COLOR")

# New
self.shader = gpu.shader.from_builtin("SMOOTH_COLOR")
```

Impact:
These changes ensure that the 'Visualize' tab's visual elements render correctly in Blender 4.2 LTS and remain functional in future releases. By eliminating all deprecated `bgl` usage, the add-on now aligns with Blender's recommended GPU drawing pipeline.

3 Bug Fixes & Development Updates

3.1 Fix: File Path Error in Add-on Initialization

During the enhancement of the Venturial add-on for Blender 4.2 LTS compatibility, an issue was identified where the add-on failed to locate the `user_custom_settings.json` preferences file during initialization. This resulted in errors when attempting to load or save user preferences, thereby disrupting the add-on's functionality.

Cause:
The problem originated from the method used to define the default path for the preferences file within the `VNT_OT_save_preferences` operator. Specifically, the code utilized:

```
pref_loc: StringProperty(
    default=bpy.utils.script_paths(subdir='addons')[1]
    + "/venturial/preferences/user_custom_settings.json"
)
```

This approach assumed that the desired add-on directory was always located at index `[1]` of the list returned by:

`bpy.utils.script\_paths(subdir='addons')`

However, the order of paths in this list can vary across different systems and Blender installations, leading to incorrect path resolutions and subsequent file not found errors.

Solution:
To rectify this, the index was changed from `[1]` to `[0]`, ensuring that the path points to the first directory in the list, which typically corresponds to the user-specific add-ons directory. The updated code is:

```
pref_loc: StringProperty(
    default=bpy.utils.script_paths(subdir='addons')[0]
    + "/venturial/preferences/user_custom_settings.json"
)
```

This update helps ensure the add-on functions correctly by matching Blender's typical folder structure, which reduces the chance of file path errors during startup.

Impact:
This fix ensures that the Venturial add-on reliably locates and interacts with the `user_custom_settings.json` file across various Blender environments. It eliminates initialization errors related to file paths, thereby improving the add-on's stability and user experience.

3.2 Fix: Double Dialog Appearance on Directory Selection

During the enhancement of the Venturial add-on for Blender 4.2 LTS compatibility, a bug was discovered where the file browser dialog for selecting a directory would appear twice consecutively. This caused confusion and required multiple user interactions to complete a single selection.

Cause:
The issue originated in the `VNT_OT_select_mesh_filepath` operator, defined in the `file_handling_operators.py` module. The lack of proper context checks and unconditional re-invocation logic led to the execution of the operator multiple times.

Solution:
To eliminate this, three key modifications were made:

1. **Added a `poll()` class method** to restrict operator execution to valid contexts.

```
@classmethod
def poll(cls, context):
    """Ensure the operator can be executed."""
    return context.scene is not None
```

2. **Updated the `execute()` method** to safely complete the action and avoid reopening the dialog.

```
def execute(self, context):
    if self.is_dir:
        # Handle directory selection logic
        return {'FINISHED'}
    else:
        return {'CANCELLED'}
```

3. **Modified the `invoke()` method** to ensure the file selector is invoked just once.

```
def invoke(self, context, event):
    self.is_dir = True
    context.window_manager.fileselect_add(self)
    return {'RUNNING_MODAL'}
```

Impact:
This update guarantees that the file selection dialog appears only once per user action, removing redundancy and significantly improving the user experience.

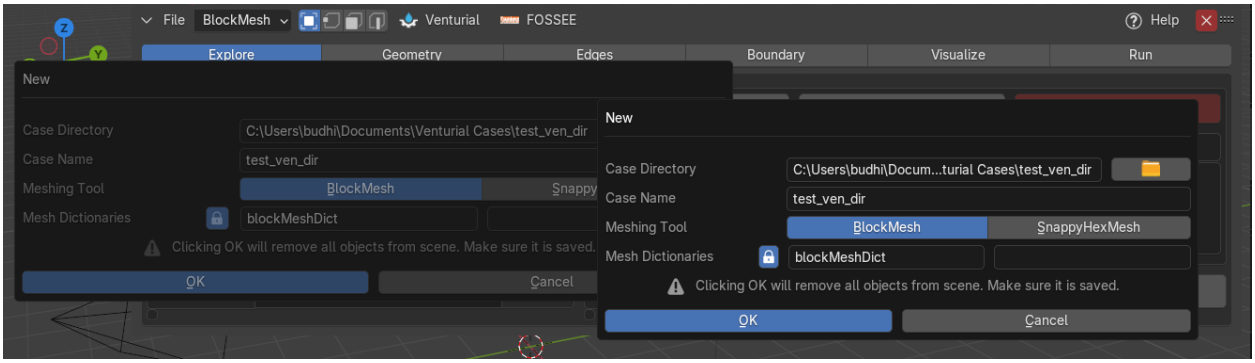


Figure 1: Before: Double Dialog appearing after confirming directory

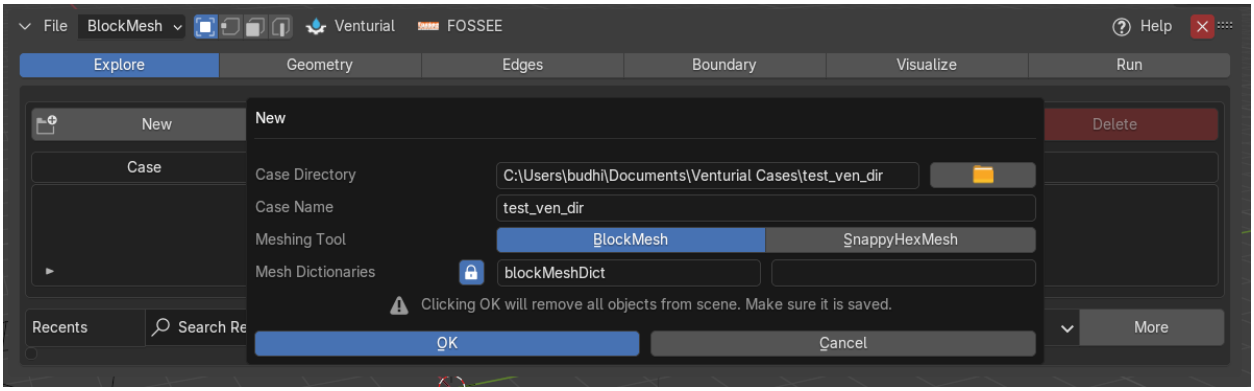


Figure 2: After: Single dialog appearing after fix

3.3 Persistent Data across saved .blend file

It was observed that certain custom data, specifically edge and vertex information used for visualization, was not persisting across sessions. Upon saving and reopening a `.blend` file, the custom draw handlers and associated data were lost, leading to a degraded user experience.

Cause:
The issue stemmed from the storage of temporary drawing data in global variables or in contexts that are not preserved when a `.blend` file is saved and later reopened. Blender's architecture requires that persistent data be stored within its data structures, such as the `Scene`, to ensure longevity across sessions.

Solution:
To address this, the following changes were implemented:

1. **Storing Draw Data in the Scene:** A function `get_edge_draw_data()` was defined to retrieve or initialize a dictionary within the `Scene` to store draw handler IDs and vertex data.

```
def get_edge_draw_data():
    """
    Returns a dict used to store temporary drawing data.
    This dictionary is attached to the scene and reinitialized on file
    load.
    """
    scn = bpy.context.scene
    if "_edge_draw_data" not in scn:
        scn["_edge_draw_data"] = {
            "draw_handlers": [],
            "verts": []
        }
    return scn["_edge_draw_data"]
```

2. **Resetting Draw Handlers on File Load:** A load handler function `load_post_handler()` was added to clear existing draw handlers and vertex data upon loading a `.blend` file, ensuring a clean state.

```
def load_post_handler():
    """
    Handler that resets the draw handlers for edges after a .blend file
    is loaded.
    """
    # Clear any existing draw handler references in our scene data.
    edge_data = get_edge_draw_data()
    edge_data["draw_handlers"].clear()
    edge_data["verts"].clear()

    # Re-run the drawing function for each edge.
    scn = bpy.context.scene
    for i in range(len(scn.ecustom)):
        try:
            # Assume draw_edge is a function that sets up drawing for
    an edge
            draw_edge(scn.ecustom[i])
        except Exception as e:
            print(f"Error drawing edge: {e}")
```

3. **Registering the Load Handler:** The `load_post_handler()` was registered using Blender's `bpy.app.handlers.load_post` to ensure it is called after a file is loaded.

```
import bpy
from bpy.app.handlers import persistent

@persistent
def load_post_handler(dummy):
    # Handler code as defined above

bpy.app.handlers.load_post.append(load_post_handler)
```

Impact:
These modifications ensure that custom edge and vertex data used for visualization are preserved across Blender sessions. By storing the data within the `Scene` and properly managing it upon file load, the add-on now provides a consistent and reliable user experience.

3.4 UI Changes

3.4.1 Header Overlap

Cause:
With Blender 4.2, Venturial's top-bar layout suffered from self-overlapping header elements. This issue originated from the use of multiple uncoordinated layout rows in the header section—specifically, each UI element was being added using separate `layout.row()` calls. Blender's updated UI behavior, particularly in the top-bar header region, enforces single-row alignment, making such an approach prone to overlap and layout breakage.

Solution:
To resolve this, the header drawing logic was refactored into two clearly structured parts:

1. **`header_layout`**: This function collects all left-aligned elements and places them within a single row using `layout.row(align=True)`. For instance, the "File" menu and the tool selection popover are now drawn sequentially in the same horizontal row:

```
class header_layout:
    def draw(self, layout, context):
        cs = context.scene
        row = layout.row(align=True)
        row.menu(VNT_MT_file_menu.bl_idname, text="File")
        row.popover(VNT_PT_uicategory.bl_idname,
    text=cs.current_tool_text)
        # Second row: Mode propertyAdd commentMore actions
        row = layout.row(align=True)
        row.prop(cs, "mode", icon_only=True, expand=True)

        # Third row: Venturial and FOSSEE menus
        row = layout.row(align=True)
        row.menu(
            VNT_MT_about_venturial.bl_idname,
            text="  Venturial  ",
            icon_value =
    custom_icons["venturial_logo"]["venturial_logo"].icon_id
        )
        row = layout.row(align=True)
        row.menu(
            VNT_MT_about_fossee.bl_idname,
            text="  FOSSEE  ",
    icon_value=custom_icons["fossee_logo"]["fossee_logo"].icon_id
        )
```

2. **`header_preset_layout`**: Introduced for placing right-aligned controls in the header area, such as help links and panel close buttons. This follows Blender's convention where 'draw_header_preset()' is used for right-end justified content. The added code looks like:

```
class header_preset_layout:Add commentMore actions
    """Preset Class that consists of methods to define venturial's
    header layout"""

    def draw(self, layout, context):
        # Fourth row: Help menu and close operator
        row = layout.row(align=True)
        row.menu(VNT_MT_help_menu.bl_idname, text="  Help  ",
    icon="QUESTION")
        row = layout.row(align=True)
        row.alert = TrueAdd commentMore actions
        row.operator(VNT_OT_close_venturial.bl_idname, text="",
    icon="PANEL_CLOSE")
        row.alert = False
```

And the panel now uses:

```
class VNT_PT_usermodeview(Panel):
    """Main Panel Layout of User Mode"""
    def draw_header(self, context):
        layout = self.layout
        getattr(header_layout(), "draw")(layout, context)

    def draw_header_preset(self, context):
        layout = self.layout
        getattr(header_preset_layout(), "draw")(layout, context)

    def draw(self, context):
        layout = self.layout
        getattr(mainPanel(), "draw")(layout, context)
```

This approach ensures that secondary actions (Help and Close) appear on the far right of the header bar, separated from the primary workflow controls.

Impact: The updated structure improves visual clarity and UI consistency with Blender's evolving design standards. By distinguishing between primary (left-aligned) and secondary (right-aligned) controls, the new layout prevents header self-overlap and enhances usability.

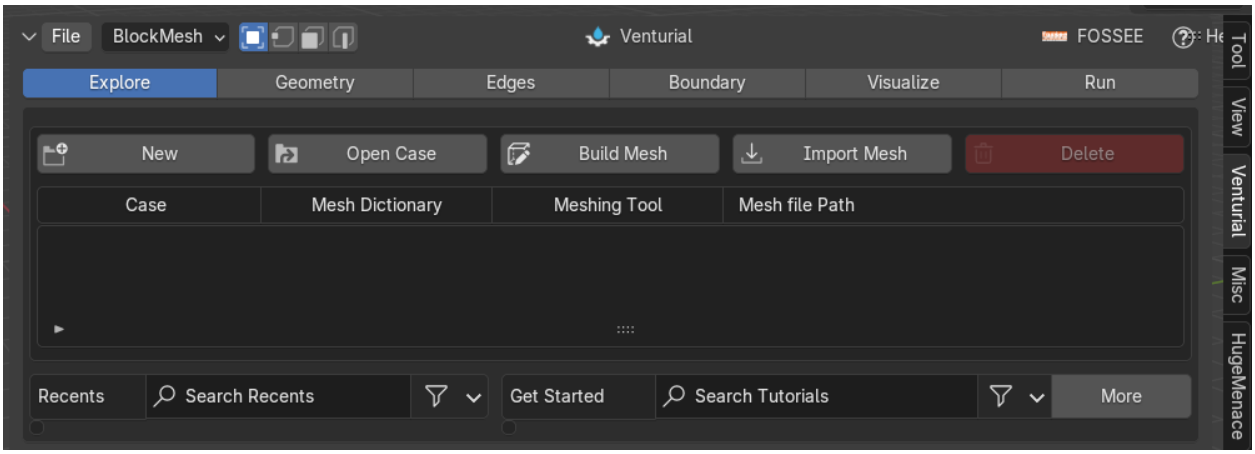


Figure 3: Old Header of "Venturial"

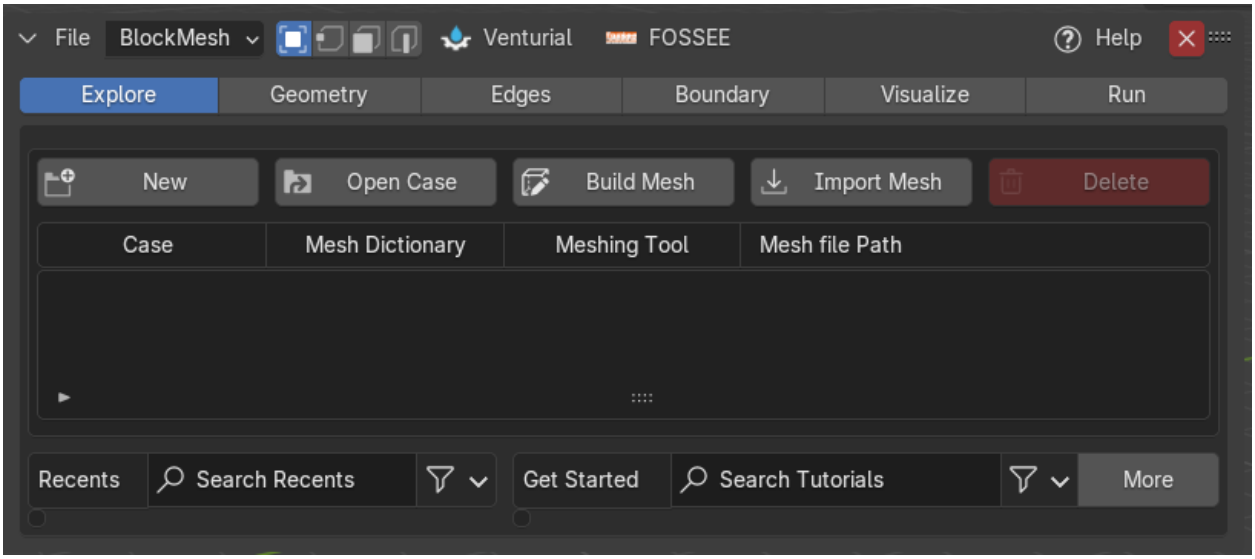


Figure 4: New Header of "Venturial" without self-overlap

3.4.2 Adequate Ratio of the Edges List and Vertices list under the 'Edges' Menu

To improve visual balance and readability in the Edges panel, the layout ratio between the label and corresponding UI elements was adjusted.

The split factor in the layout definition was updated:

```
split = layout.split(factor=0.4)
```

This change provides better horizontal space allocation for vertex and edge lists, preventing label truncation and enhancing usability.

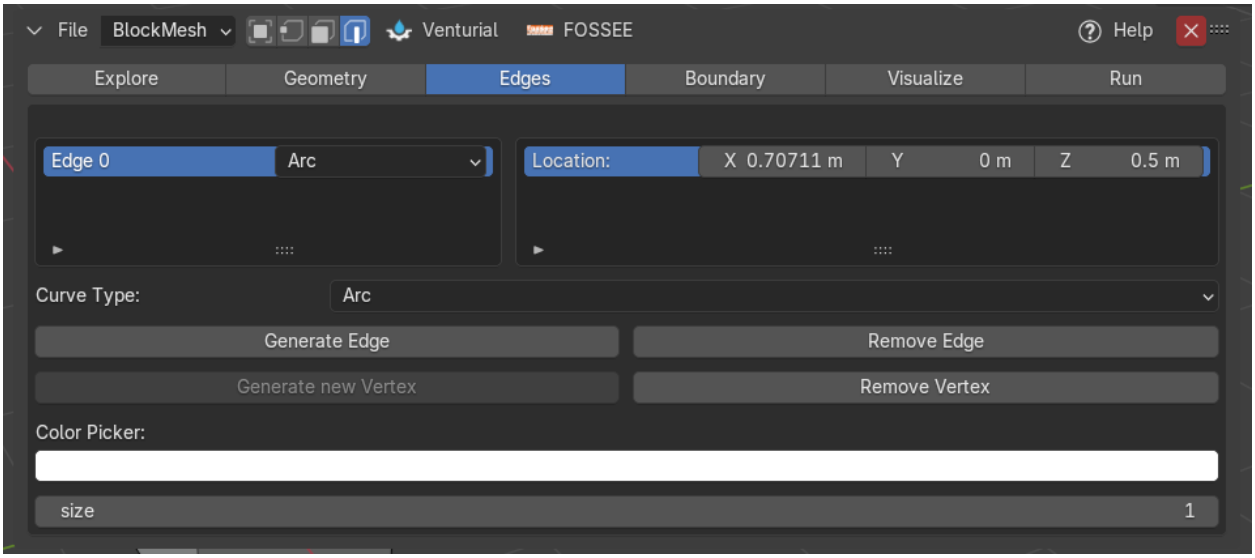


Figure 5: Adequate ratio to view both the edges and vertices list

3.4.3 Separate row for 'Curve Type' drop down

In the Edges panel, the `Curve Type` dropdown was previously placed on the same row as the edge operation buttons (`Generate Edge`, `Remove Edge`), leading to a cramped layout and reduced clarity.

To enhance readability, the layout was restructured to place the `Curve Type` selection on its own row. The relevant code changes are as follows:

```
row1 = layout.row()
row1.prop(cs, "curve_type", text="Curve Type")

row2 = layout.row()
row2.operator('vnt.new_edge')
row2.operator('vnt.remove_edge')

row3 = layout.row()
row3.operator('vnt.new_vert')
row3.operator('vnt.remove_vert')
```

Impact:
This adjustment improves the visual organization of the Edges panel, making the interface more intuitive and user-friendly.

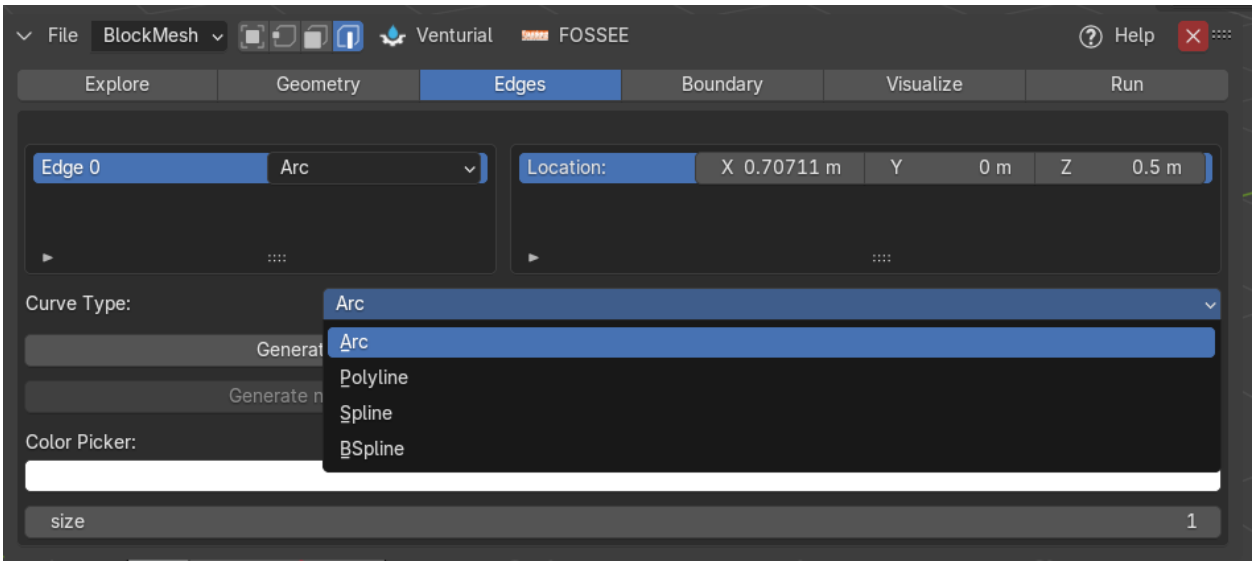


Figure 6: Separate row for 'Curve Type' drop-down making it a cleaner UI

3.5 Dynamic Face Selection for 'Merge Face' Operation

Cause:
Previously, the "Merge Face" operation in Venturial's Blender add-on required users to select faces through drop-downs of static UI lists maintained in scene properties. This approach was cumbersome and prone to synchronization issues between the 3D viewport and internal data.
Furthermore, the earlier implementation lacked geometric validation for face overlap, risking erroneous merges.

Solution:
The migration introduced in this feature modifies the "Merge Face" workflow to use dynamic, edit-mode-based selection and geometric validation.

- **Dynamic Selection:** The operator now directly queries the currently selected faces in edit mode via bmesh, enforcing exactly two face selections:

```
bm = bmesh.from_edit_mesh(obj.data)
selected_faces = [f for f in bm.faces if f.select]
if len(selected_faces) != 2:
    self.report({'ERROR'}, "Exactly two faces must be selected.")
    return {'CANCELLED'}
```

- **Geometric Utilities:** Mathematical functions made to verify the intersection of the two faces completely ensuring a face is smaller(slave) and the other is larger(master) and are completely overlapping into one another and not partially.

```
def project_point_onto_plane(point, plane_point, plane_normal):
    vec = point - plane_point
    distance = vec.dot(plane_normal)
    projection = point - distance * plane_normal
    return distance, projection

def point_in_polygon_2d(pt, poly):
    x, y = pt
    inside = False
    n = len(poly)
    for i in range(n):
        xi, yi = poly[i]
        xj, yj = poly[(i + 1) % n]
        if (yi > y) != (yj > y):
            x_intersect = (xj - xi) * (y - yi) / (yj - yi + 1e-10) + xi
            if x < x_intersect:
                inside = not inside
    return inside

def faces_intersect(faces, tol=1e-6):
```

```
    if len(faces) != 2:
        raise ValueError("Exactly two faces are required.")
    f1, f2 = faces
    def face_completely_inside(inner, outer):
        u = (outer.verts[1].co - outer.verts[0].co).normalized()
        v = outer.normal.cross(u).normalized()
        poly2d = [
            ((vtx.co - outer.verts[0].co).dot(u),
             (vtx.co - outer.verts[0].co).dot(v))
            for vtx in outer.verts
        ]
        for vtx in inner.verts:
            dist, proj = project_point_onto_plane(vtx.co,
   outer.verts[0].co, outer.normal)
            if abs(dist) > tol:
                return False
            pt2d = ((proj - outer.verts[0].co).dot(u),
                    (proj - outer.verts[0].co).dot(v))
            if not point_in_polygon_2d(pt2d, poly2d):
                return False
        return True
    return face_completely_inside(f1, f2) or face_completely_inside(f2,
   f1)
```

- **Face Validation and Mapping:** The selected faces are validated against the add-on's custom face list, ensuring only faces recognized by the system are merged:

```
selected_face_indices = [[v.index for v in f.verts] for f in
    selected_faces]
fcustom_faces = [face_strtolist(f.name) for f in cs.fcustom]
if not all(face in fcustom_faces for face in selected_face_indices):
    self.report({'ERROR'}, "Selected faces must exist in the face
    list.")
    return {'CANCELLED'}
```

- **Master/Slave Determination by Area:** The operator determines which face is master/slave by comparing their calculated areas:

```
face_areas = [f.calc_area() for f in selected_faces]
master_face, slave_face = (selected_face_des[0], selected_face_des[1])
    if face_areas[0] > face_areas[1] else ( selected_face_des[1],
    selected_face_des[0])
```

- **Mapping Storage and Confirmation Message:** The merge result is stored in a custom mapping list and an informative message is reported:

```
item = cs.fmcustom.add()
item.master_face = master_face
item.slave_face = slave_face
self.report({'INFO'}, f"Merged {item.master_face} (master) with
    {item.slave_face} (slave).")
```

Impact:
This migration delivers a seamless, Blender-native workflow by allowing users to select faces directly in the 3D viewport, eliminating the need for error-prone static UI lists. The introduction of geometric validation routines such as plane projection and polygon containment ensures that merges only occur between genuinely overlapping faces, reducing the risk of invalid operations. Master/slave face assignment is now automatic and robust.

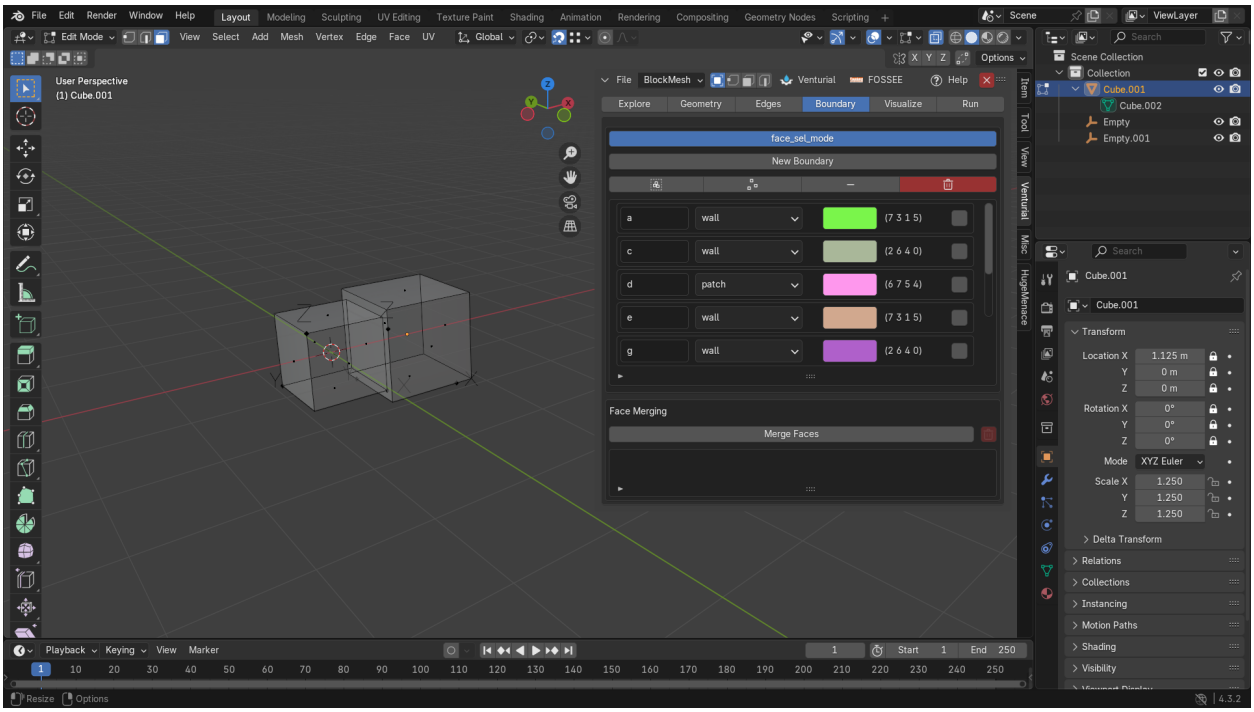


Figure 7: Selecting the two faces we want to merge

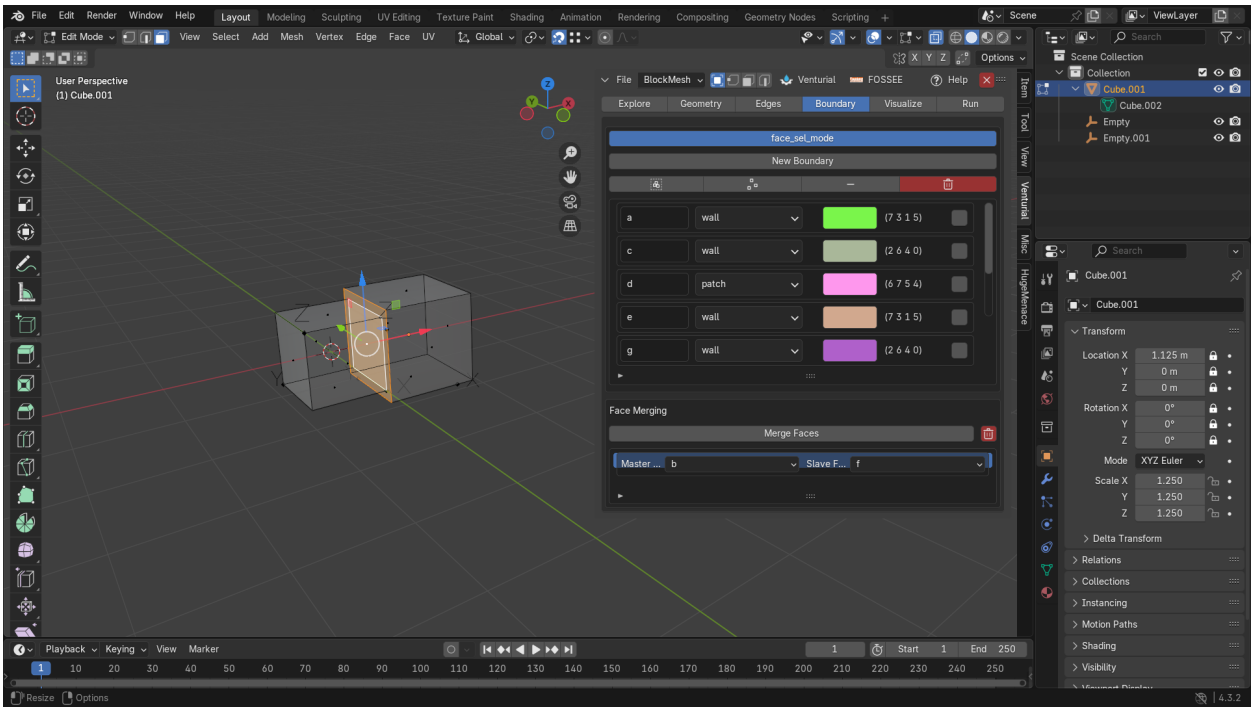


Figure 8: Merging the selected faces with automatic master and slave face recognition

3.6 Updated New Vertex Formula to form symmetric curves

Cause:
The previous method for calculating the position of a new vertex for a new edge/curve of type 'Arc' created in 'Edges' menu was insufficient for ensuring geometric symmetry and alignment with the faces of the 3D Shape/Mesh. The old approach averaged the coordinates of the two edge endpoints, adjusting the z-coordinate differently, as shown below. This method often produced vertices that were not properly aligned with the face's geometry, especially for non-square or rotated cubes, leading to inconsistent and non-symmetrical results.

```
coord = [None, None, None]
for i in range(3):
    coord[i] = (x[i] + y[i])/1.5
```

Solution:
To address these issues, a new formula was implemented. This formula constructs a vertex position that is always on the perimeter of an ellipse (approximated as a circle for square faces), symmetrically around the upper face of the cube. The key steps are:

1. Compute the center of the upper face of the cube:

```
cube_center = context.active_object.location
face_center = (cube_center[0], cube_center[1], cube_center[2] +
    context.active_object.dimensions.z / 2)
```

2. Calculate the radius as half the diagonal of the face, ensuring the new vertex lies on the ellipse boundary:

```
face_diagonal = (context.active_object.dimensions.x ** 2 +
    context.active_object.dimensions.y ** 2) ** 0.5
radius = face_diagonal / 2
```

3. Determine the midpoint of the selected edge, then compute the direction vector from the face center to this midpoint:

```
midpoint = [(x[i] + y[i]) / 2 for i in range(3)]
direction = [midpoint[i] - face_center[i] for i in range(3)]
direction[2] = 0  # Project onto XY plane
```

4. Normalize this direction and use it to place the new vertex symmetrically around the face:

```
length = (direction[0] ** 2 + direction[1] ** 2) ** 0.5
direction = [direction[i] / length for i in range(2)] + [0]
coord = [
    face_center[0] + direction[0] * radius,
    face_center[1] + direction[1] * radius,
    (x[2] + y[2]) / 2
]
```

Impact:
This updated approach guarantees that the generated vertex is always positioned on a symmetrical ellipse (or circle) around the upper face, regardless of the cube's orientation or dimensions. As a result, the add-on produces more accurate, visually consistent, and mathematically robust geometry, which is crucial for procedural modeling workflows. The symmetry and precision introduced by this new method enhance both the reliability of the add-on and the quality of the generated meshes, supporting advanced modeling task.

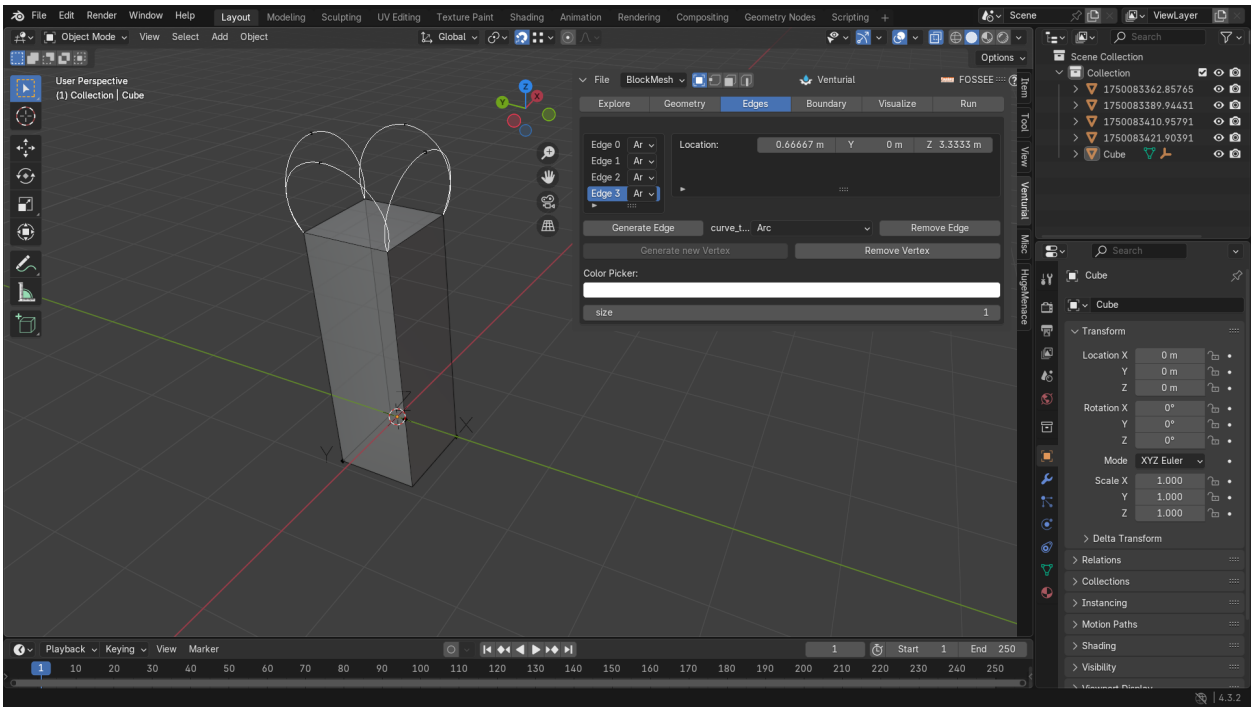


Figure 9: Before: Camera View of arcs created without any user intervention

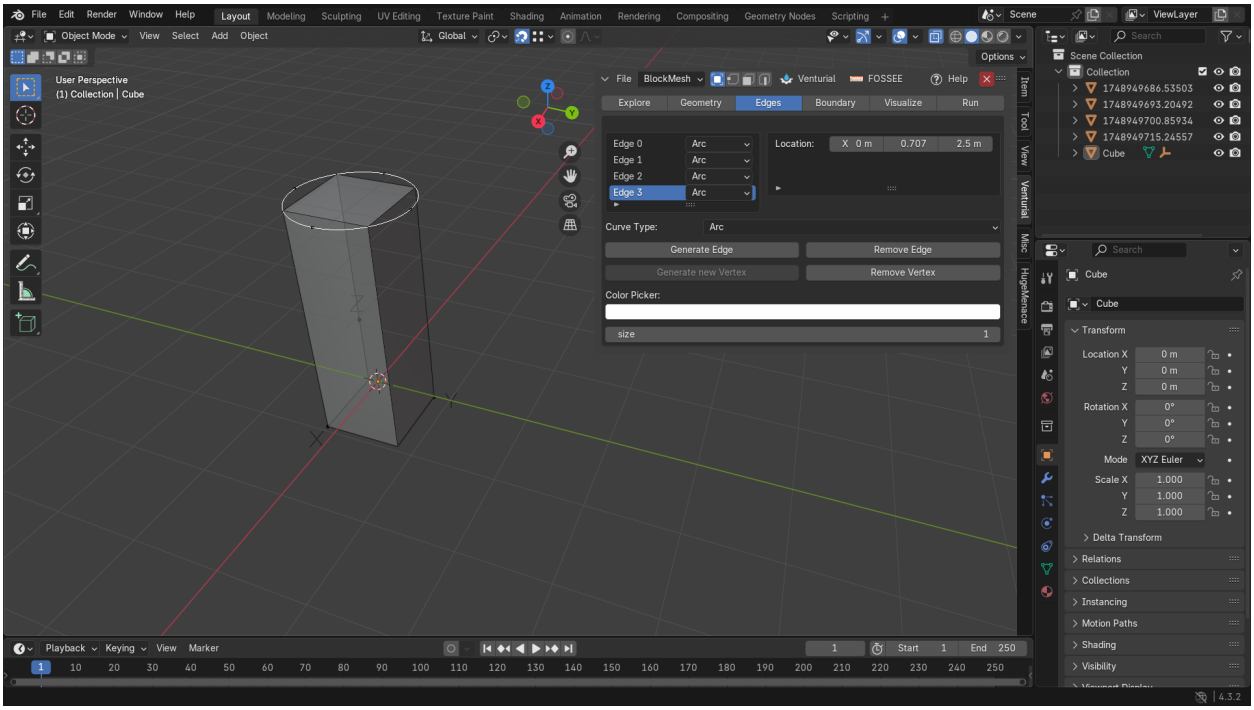


Figure 10: After: Camera View of the created new arcs without any user intervention

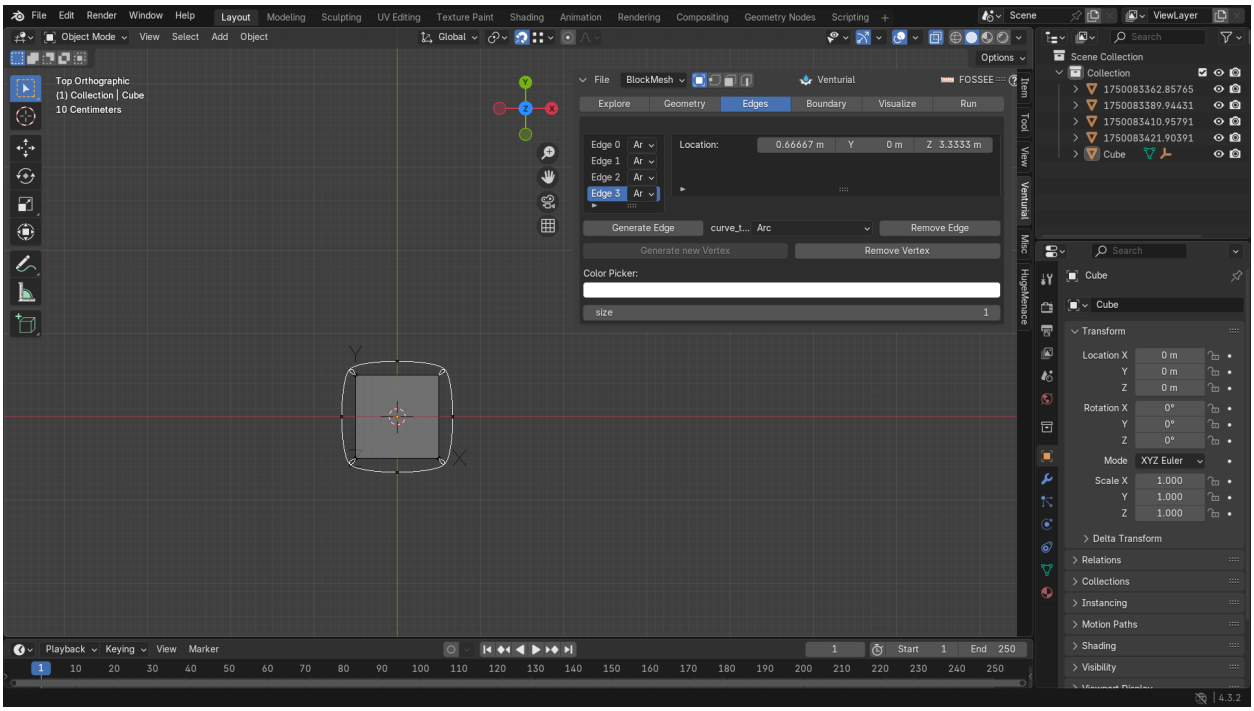


Figure 11: Before: Upper z-axis view of the created new arcs

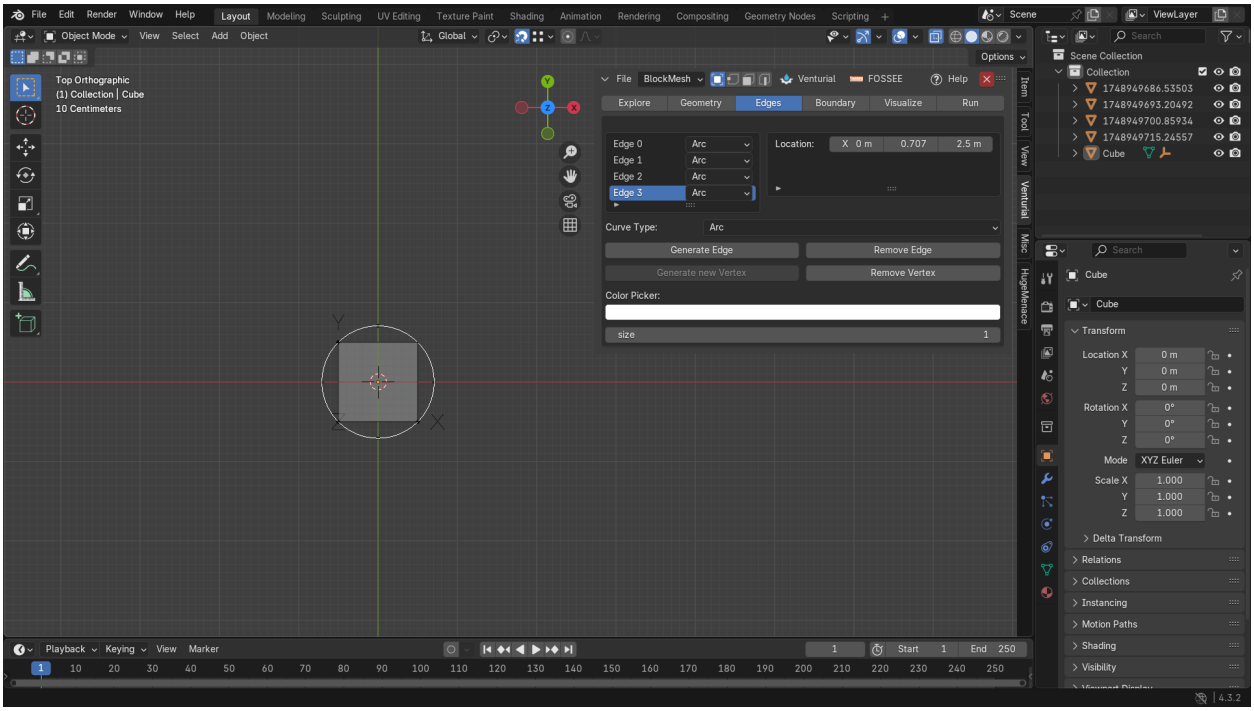
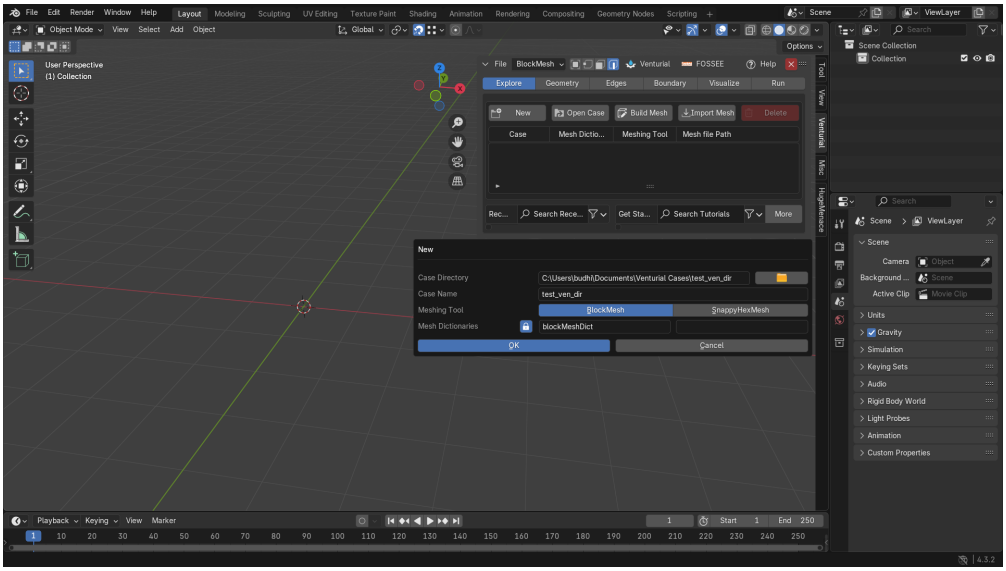


Figure 12: After: Upper z-axis view of the created new arcs

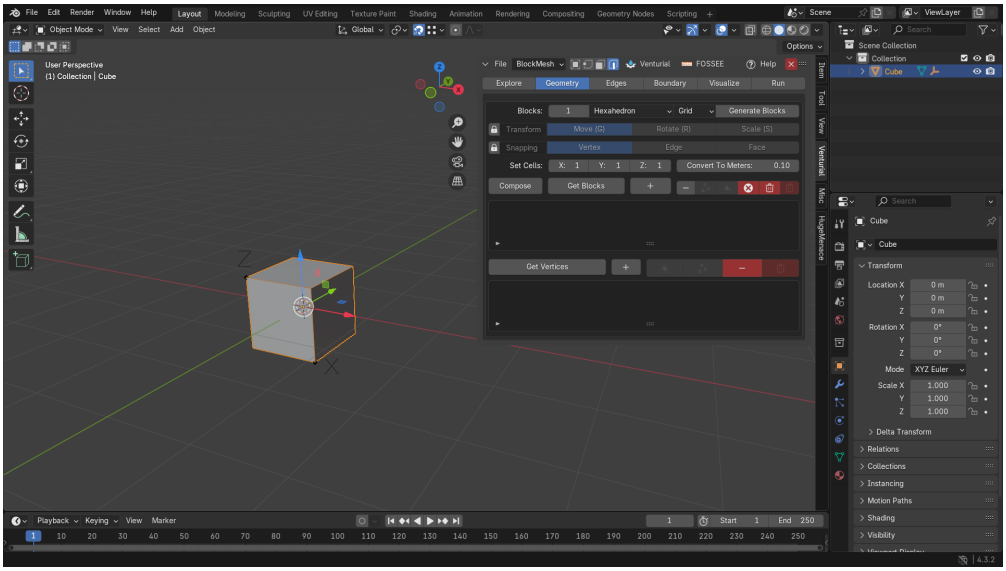
4 Results

A step by step process of creating a block with associated curves and assigned boundaries and then generating it's **blockMeshDict** file.

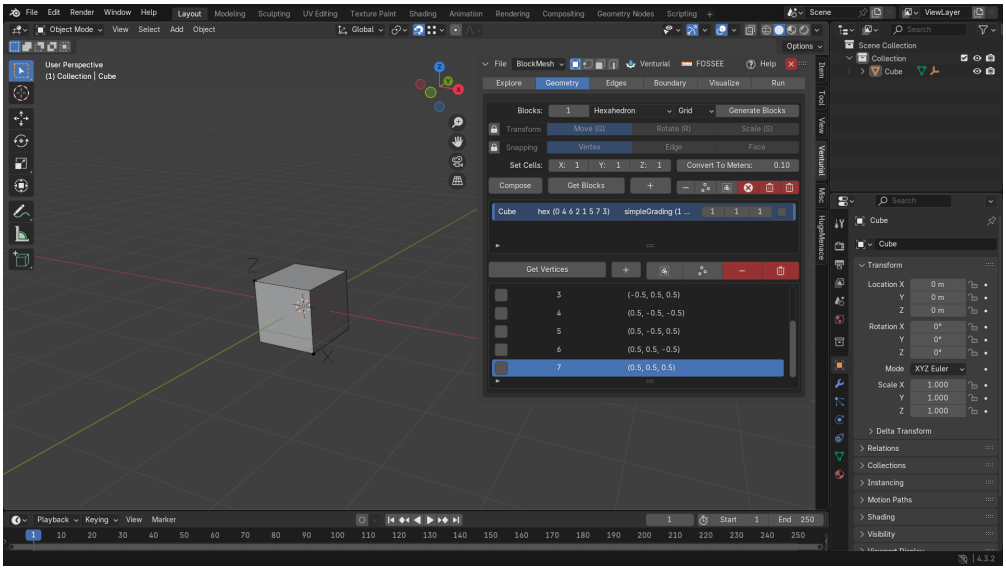
1. **Opening a new case directory to save blockMeshDict file**
   We first go to the 'Explore' tab of our 'Venturial' add-on in Blender. Click on 'New' Button which will open a dialog box asking for the case directory to store blockMeshDict file.



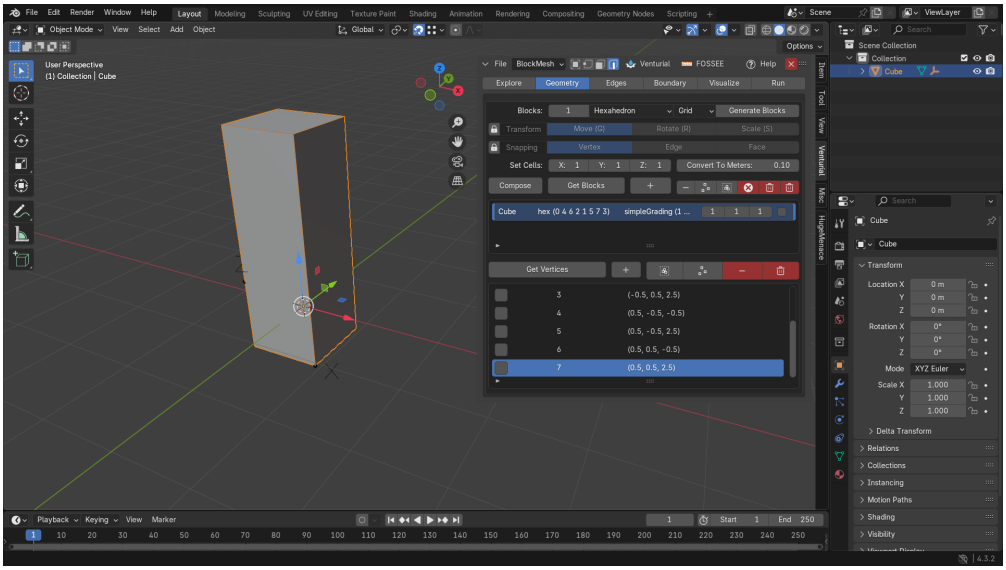
2. **Generating a new block object to work on**
   Go to the 'Geometry' tab and click on 'Generate Blocks' button which will generate the number of blocks we have selected in the integer input box, default being 1.



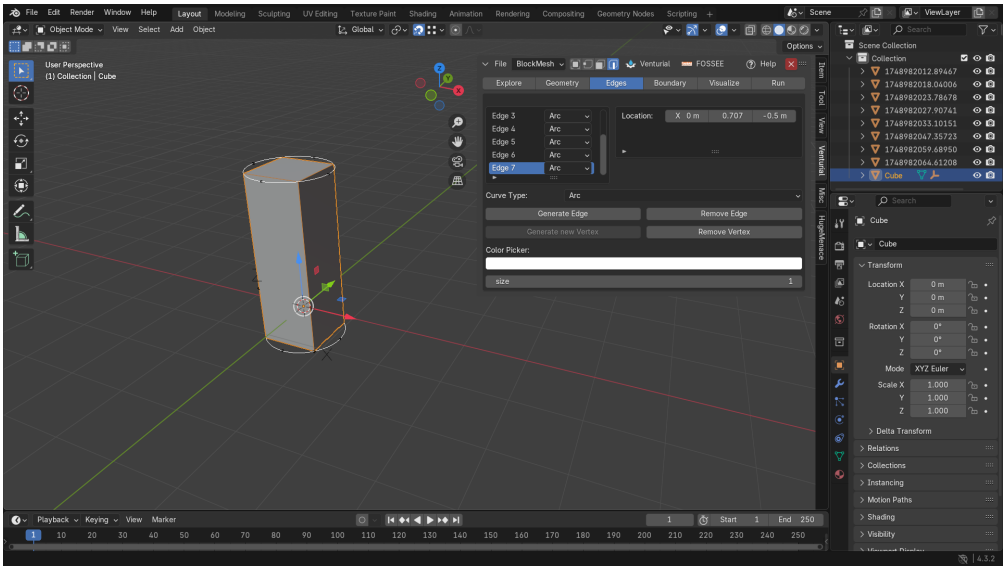
3. **Reading all block and vertices data in the scene**
   Click on 'Get Blocks' and then 'Get Vertices' to collect all the block and vertices data present in the scene.



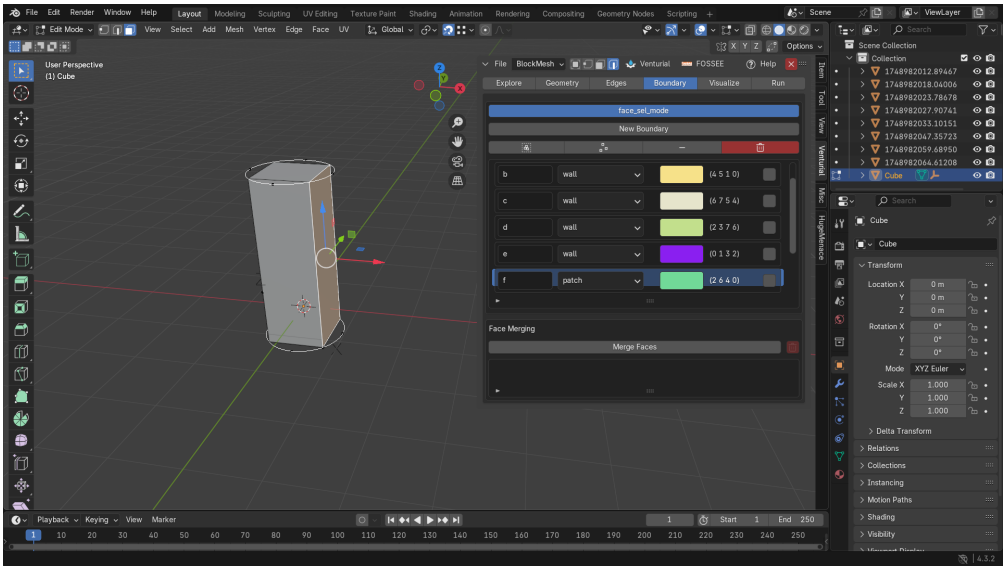
4. **Modifying the block to an elongated shape**
   We can also modify the block as we wish and repeat the process of reading the block and vertices data.



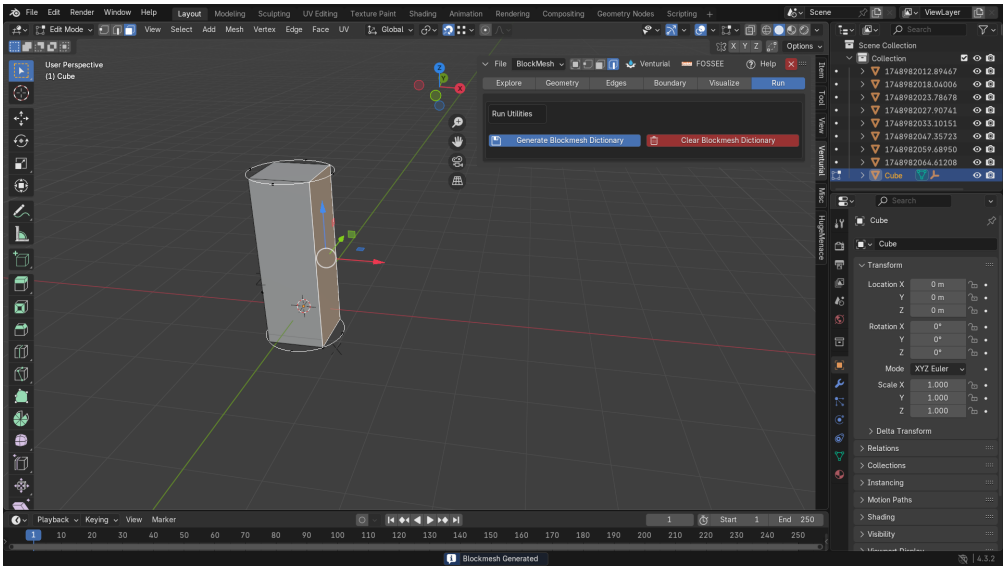
5. **Creating edges around the block**
   Go to 'Edges' tab and select the edge of the block on which you want to draw your desired curve via 'Edit Mode' of Blender. Simply click on 'Generate Edge' button to generate a curve on the selected edge and then 'Generate new Vertex' to create a vertex along the curve.



6. **Assigning boundaries to faces of the block**
   To assign boundaries for blockMeshDict we have to select the faces of the block and assign them boundary name and their type. Simply select a face of the block in 'Edit Mode' and click on 'New Boundary', a dialog box will appear to take the name and type of boundary.



7. **Run Utility to generate blockMeshDict file**
   Go to 'Run' tab and click on 'Generated BlockMesh Dictionary' to generate the blockMeshDict file of the block and associated curves you have created till now. The blockMeshDict file will be saved in the same directory you selected in the beginning.



8. Generated blockMeshDict file

The contents of the generated blockMeshDict file in the selected directory after running the ”Run Utilities” is as follows:

```
/*--------------------------------*- C++
    -*----------------------------------*\\
| ========= | |
| \\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
| \\ / O peration | Version: 9 |
| \\ / A nd | Web: www.OpenFOAM.org |
| \\/ M anipulation | |
\*---------------------------------------------------------------------------*/
//*******This file is generated by Venturial*********//
FoamFile
{
       version 2.0;
       format ascii;
       class dictionary;
       object blockMeshDict;
}
convertToMeters 0.10000000149011612 ;
vertices
(
       ( -0.5 -0.5 -0.5 )
       ( -0.5 -0.5 2.5 )
       ( -0.5 0.5 -0.5 )
       ( -0.5 0.5 2.5 )
       ( 0.5 -0.5 -0.5 )
       ( 0.5 -0.5 2.5 )
       ( 0.5 0.5 -0.5 )
       ( 0.5 0.5 2.5 )
);
blocks
(
       hex ( 0 4 6 2 1 5 7 3 ) ( 1 1 1 ) simpleGrading (1 1 1)
);
edges
(
       arc 7 5 ( 0.7071067690849304 0.0 2.5 )
       arc 5 1 ( 0.0 -0.7071067690849304 2.5 )
       arc 3 7 ( 0.0 0.7071067690849304 2.5 )
       arc 1 3 ( -0.7071067690849304 0.0 2.5 )
       arc 4 6 ( 0.7071067690849304 0.0 -0.5 )
       arc 0 4 ( 0.0 -0.7071067690849304 -0.5 )
       arc 2 0 ( -0.7071067690849304 0.0 -0.5 )
       arc 6 2 ( 0.0 0.7071067690849304 -0.5 )
);
boundary
```

```
(
        a
        {
                type patch;
                faces
                (
                        ( 7 3 1 5 )
                );
        }
        b
        {
                type wall;
                faces
                (
                        ( 4 5 1 0 )
                );
        }
        c
        {
                type wall;
                faces
                (
                        ( 6 7 5 4 )
                );
        }
        d
        {
                type wall;
                faces
                (
                        ( 2 3 7 6 )
                );
        }
        e
        {
                type wall;
                faces
                (
                        ( 0 1 3 2 )
                );
        }
        f
        {
                type patch;
                faces
                (
                        ( 2 6 4 0 )
```

```
            );
        }
);
mergePatchPairs
(
);
//
    *********************************************************************
    //
```

5 Conclusion

We were successful in enhancing the existing OpenFOAM GUI add-on *Venturial* to make it compatible with Blender 4.2 LTS. The work included fixing critical bugs, updating deprecated Blender API usages, and improving the overall user interface layout. These changes ensure that the add-on remains functional and stable with the latest Blender release.

The results demonstrate a complete pipeline—from generating a basic geometry like a cube to exporting its corresponding **blockMeshDict** file—using the improved add-on. This workflow showcases the practical utility of the GUI for OpenFOAM case setup. Continued development can help evolve this prototype into a robust tool for CFD case preparation.

References

[1] Blender OpenGL Wrapper Module: `https://docs.blender.org/api/current/bgl.html`

[2] Blender GPU Shader Utilities: `https://docs.blender.org/api/current/gpu.shader.html`

[3] Blender API Documentation: `https://docs.blender.org/api/current/`

[4] OpenFoam Documentation: `https://www.openfoam.com/documentation/overview`