



**Politecnico  
di Torino**

**Master of Science in Building Engineering**

A.Y. 2023/2024

Final examination session July 2024

# **Semi-Automatic Approach for the Scan-to-BIM of Built Heritage: The Case Study of Sacro Monte di Varallo**

**Academic tutor:**

Arch. Francesca Matrone

**Co-tutors:**

Prof. Andrea Maria Lingua

Ing. Matteo Del Giudice

**Candidate:**

Manuel Antonio Rico Carmona

# TABLE OF CONTENTS

<b>1</b>	<b>ABSTRACT</b> .....	<b>10</b>
<b>2</b>	<b>INTRODUCTION</b> .....	<b>12</b>
<b>3</b>	<b>CASE STUDY: SACRO MONTE DI VARALLO</b> .....	<b>14</b>
3.1	SACRO MONTE DI VARALLO HISTORY .....	14
3.2	CHAPEL XXIV .....	15
3.3	CHAPEL XXVIII .....	20
<b>4</b>	<b>STATE OF THE ART: SCAN-TO-BIM.</b> .....	<b>24</b>
4.1	CURRENT TECHNOLOGIES AND TOOLS. ....	24
4.2	METHODOLOGIES AND STANDARDS. ....	26
4.3	RECENT DEVELOPMENTS IN THE BUILT HERITAGE DOMAIN. ....	29
4.4	LATEST SEMI-AUTOMATED SCAN-TO-HBIM APPROACHES. ....	32
<b>5</b>	<b>METHODOLOGY</b> .....	<b>40</b>
5.1	FROM THE POINT CLOUD TO THE HBIM MODEL. ....	43
5.1.1	<i>Input reading.</i> .....	45
5.1.2	<i>ArCH dataset Categories</i> .....	46
<b>6</b>	<b>MODELLING PROCESS</b> .....	<b>52</b>
6.1	IMPORT DATA.....	52
6.2	DYNAMO ROUTINE FOR LOD B.....	53
6.2.1	<i>Floors</i> .....	53
6.2.2	<i>Columns</i> .....	55
6.2.3	<i>Walls</i> .....	56
6.2.4	<i>Roofs and Ceilings</i> .....	64
6.2.5	<i>Windows</i> .....	67
6.2.6	<i>Vaults</i> .....	71
6.2.7	<i>Arch</i> .....	77
6.2.8	<i>Stairs</i> .....	78
6.3	DYNAMO ROUTINE FOR LOD C .....	80
6.3.1	<i>Floors</i> .....	80
6.3.2	<i>Columns</i> .....	80
6.3.3	<i>Walls</i> .....	85
6.3.4	<i>Roofs</i> .....	85
6.3.5	<i>Windows</i> .....	85

6.3.6	<i>Vaults</i> .....	85
6.3.7	<i>Arch</i> .....	85
6.3.8	<i>Stairs</i> .....	85
<b>7</b>	<b>RESULTS</b> .....	<b>87</b>
7.1	CHAPEL XXIV: CATEGORY BASED LEVELS OF DETAILS COMPARISON.....	87
7.2	CHAPEL XXVIII: CATEGORY BASED LEVELS OF DETAILS COMPARISON.....	95
<b>8</b>	<b>DISCUSIONS</b> .....	<b>105</b>
8.1	PROS AND CONS OF THE PROPOSED WORKFLOW.....	105
8.2	FUTURE WORK.....	106
<b>9</b>	<b>CONCLUSIONS</b> .....	<b>108</b>
<b>10</b>	<b>BIBLIOGRAPHY</b> .....	<b>109</b>

## LIST OF FIGURES

Figure 1. Jesus before the court of Herod. ....	15
Figure 2. Aerial view of Sacro Monte di Varallo. ....	15
Figure 3. Chapel XVI and Chapel XXIV Complex. ....	16
Figure 4. Hypothetical reconstruction of Galeazzo Alessi’s project. ....	16
Figure 5. Chapel XVI Facade. ....	17
Figure 6. Location of Chapel XXIV. ....	17
Figure 7. Outside view of Chapel XXIV: Jesus at the court of Anna. ....	18
Figure 8. First design of Chapel XXIV. ....	19
Figure 9. Interior of Chapel XXIV. ....	19
Figure 10. Location of Chapel XXVIII. ....	20
Figure 11. Longitudinal section of Chapel XXVIII. ....	21
Figure 12. Plan View and Prospect of Chapel XXVIII. ....	21
Figure 13. Plan Views of Chapel XXVIII. ....	22
Figure 14. Outside views of Chapel XXVIII: Jesus before the court of Herod. ....	23
Figure 15: BIM-CAD chosen software in the years. ....	24
Figure 16. Autodesk Revit Architecture- Structure-MEP Disciplines. ....	25
Figure 17. Differences in Levels of Details. ....	26
Figure 18. LOD = LOG + LOI ....	27
Figure 19. LOD differences in Italy, U.S. and U.K. ....	29
Figure 20. ArCH dataset Class Definition. ....	30
Figure 21. ArCH dataset Categories. ....	30
Figure 22. Chapel XIV and Chapel XVIII semantic segmentation. ....	31
Figure 23. Comparison between analytical models and damages on wall structures. ....	32
Figure 24. 3D Model representing the Sacristy Mayor Painting. ....	33
Figure 25. Croce’s proposed methodology. ....	34
Figure 26. Automated Parametric Workflow. ....	35
Figure 27. Automated Parametric Workflow. ....	35
Figure 28. Generated model in Revit from Point Cloud. ....	37
Figure 29. Replication of template families using Grasshopper. ....	37
Figure 30. 3D Model of Casa di Pilatos in ArchiCAD by using meshes. ....	38
Figure 31. Manual segmentations of elements. ....	38

Figure 32. Semi-Automatic Scan-to-HBIM proposed methodology.....	40
Figure 33. Default Python node content using CPython3.....	41
Figure 34. Lists levels in Dynamo. ....	42
Figure 35. Python code generated by ChatGPT.....	43
Figure 36. Classification of Structural elements.....	44
Figure 37. Classification of Hosted Elements.....	44
Figure 38. Classification of Non-Hosted elements. ....	44
Figure 39. Reding Node for Point Cloud .txt file.....	52
Figure 40. Overall script for Floors - LOD B. ....	53
Figure 41. Floor Point filtering and clustering.....	53
Figure 42. Levels creation from Floors.....	54
Figure 43. Floor outline by Alpha Shapes. ....	54
Figure 44. Floor elements creation. ....	55
Figure 45. Add Base Offset to Floor.....	55
Figure 46. Column Point filtering and clustering. ....	56
Figure 47. Overall script for Walls - LOD B. ....	56
Figure 48. Filtering in Z axis for Wall points. ....	57
Figure 49. Level filtering and clustering for Wall points. ....	57
Figure 50. Planes calculation and filtering for Wall points. ....	58
Figure 51. Lists filtering for Wall points. ....	58
Figure 52. DBScan Clustering for Wall points.....	58
Figure 53. Join planes of Wall points. ....	59
Figure 54. Third clustering for Wall points. ....	59
Figure 55. Creation of cluster by a 6D HDBScan algorithm.....	60
Figure 56. Centroid calculation for Wall faces. ....	60
Figure 57. Centroid calculation for Wall points. ....	61
Figure 58. Average distance between Surface and Wall points.....	61
Figure 59. Middle line for Walls.....	62
Figure 60. Wall height calculation.....	62
Figure 61. Wall width calculation.....	62
Figure 62. Wall width String concatenation. ....	63
Figure 63. Wall family types creation.....	63
Figure 64. Wall creation by Curve and Height. ....	64

Figure 65. Base offset of Wall modification.....	64
Figure 66. Overall script for Roofs and Ceilings - LOD B.....	65
Figure 67. Roof Point filtering and clustering. ....	65
Figure 68. Roof points subsample node.....	65
Figure 69. Roof outline by Alpha Shapes. ....	66
Figure 70. Floor elements creation. ....	66
Figure 71 Set Base offset from level to Roofs. ....	67
Figure 72. Points addition to the Roof elements.....	67
Figure 73. Overall script for Windows - LOD B. ....	68
Figure 74.Windows Point filtering and clustering.....	68
Figure 75.Windows element location point calculation.....	69
Figure 76.Windows element dimension calculation.....	69
Figure 77.Windows element Window Type Name String creation.....	70
Figure 78.Windows element Window Type creation. ....	70
Figure 79.Windows element Hosting element calculation. ....	71
Figure 80. Window elements creation. ....	71
Figure 81. Overall script for Vaults. ....	71
Figure 82. Filter Vault Points and Get Clusters.....	72
Figure 83. Calculate profiles of main Barrel Vault.....	72
Figure 84. Get points of Cross Vault. ....	73
Figure 85. Get Diagonal Arcs of Cross Vault.....	73
Figure 86. Get Perimetral Arcs of Cross Vault.....	74
Figure 87. Filter Points.....	74
Figure 88. Create Cross Vaults Surfaces. ....	75
Figure 89. Cut Barrel Vault and Join to Cross Vault.....	75
Figure 90. Create Grid of Points in the Surface.....	76
Figure 91. Calculate Thickness of Vault.....	76
Figure 92. Create new Family Type.....	77
Figure 93. Create the Roof element and Add Points from Grid. ....	77
Figure 94. Overall script for Stairs - LOD B. ....	78
Figure 95. Stair Point filtering and clustering.....	78
Figure 96. Filter points in Z coordinates and get Convex Hull of whole element.....	79
Figure 97. Get the planes of each step. ....	79

Figure 98. Steps extrusion to the bottom and solid creation. ....	80
Figure 99. Overall script for Floors - LOD C. ....	80
Figure 100. Overall script for Columns - LOD C. ....	80
Figure 101. Column Point filtering and clustering. ....	81
Figure 102. Column division in two groups. ....	81
Figure 103. Selection of bottom half of column points. ....	82
Figure 104. Creation of Column Base. ....	82
Figure 105. Selection of top half of column points.....	83
Figure 106. Creation of Column Capital.....	83
Figure 107. Selection of shaft section of column points.....	83
Figure 108. Creation of Column Shaft.....	84
Figure 109. Joint of 3 different elements into a solid. ....	84
Figure 110. Overall script for Walls - LOD C. ....	85
Figure 111. Overall script for Roofs and Ceilings - LOD C.....	85
Figure 112 Overall script for Stairs - LOD C. ....	86
Figure 113. Average Error Value per Floor – Chapel XXIV.....	87
Figure 114. Identification of Floor Elements – Chapel XXIV. ....	87
Figure 115. Visual Error Representation for Floors – Chapel XXIV. ....	87
Figure 116. Average Error Value per Column – Chapel XXIV. ....	88
Figure 117. Identification of Column Elements – Chapel XXIV. ....	88
Figure 118. Visual Error Representation for Columns – Chapel XXIV.....	88
Figure 119. Average Error Value per Wall – Chapel XXIV. ....	89
Figure 120. Wall Instance Segmentation by using RANSAC + Clustering vs 6D Clustering – Chapel XXIV.....	89
Figure 121. Identification of Wall Elements – Chapel XXIV. ....	90
Figure 122. Visual Error Representation for Walls – Chapel XXIV.....	90
Figure 123. Average Error Value per Roof – Chapel XXIV. ....	91
Figure 124. Identification of Roof Elements – Chapel XXIV. ....	91
Figure 125. Visual Error Representation for Roofs – Chapel XXIV.....	91
Figure 126. Window Instance Segmentation by using 6D Clustering – Chapel XXIV.....	92
Figure 127. Window Instance Segmentation by using 3D Clustering – Chapel XXIV.....	92
Figure 128. Identification of Window Elements – Chapel XXIV. ....	92
Figure 129. Visual Error Representation for Windows – Chapel XXIV.....	93

Figure 130. Vault Instance Segmentation by using 6D Clustering – Chapel XXIV. ....	93
Figure 131. Average Error Value per Stair – Chapel XXIV.....	94
Figure 132. Identification of Stair Elements – Chapel XXIV.....	94
Figure 133. Visual Error Representation for Stairs – Chapel XXIV. ....	94
Figure 134. Comparison between Point Cloud and HBIM model – Chapel XXIV.....	95
Figure 135. Average Error Value per Floor – Chapel XXVIII.....	95
Figure 136. Identification of Floor Elements – Chapel XXVIII.....	96
Figure 137. Visual Error Representation for Floors – Chapel XXVIII. ....	96
Figure 138. Average Error Value per Column – Chapel XXVIII.....	97
Figure 139. Identification of Column Elements – Chapel XXVIII.....	97
Figure 140. Visual Error Representation for Columns – Chapel XXVIII. ....	97
Figure 141. Average Error Value per Wall – Chapel XXVIII.....	98
Figure 142. Wall Instance Segmentation by using RANSAC + Clustering vs 6D Clustering – Chapel XXVIII. ....	98
Figure 143. Identification of Wall Elements – Chapel XXVIII.....	99
Figure 144. Visual Error Representation for Walls – Chapel XXVIII. ....	99
Figure 145. Average Error Value per Roof – Chapel XXVIII.....	99
Figure 146. Identification of Roof Elements – Chapel XXVIII. ....	100
Figure 147. Visual Error Representation for Roofs – Chapel XXVIII. ....	100
Figure 148. Window Instance Segmentation by using 6D Clustering – Chapel XXVIII.....	100
Figure 149. Window Instance Segmentation by using 3D Clustering – Chapel XXVIII.....	101
Figure 150. Identification of Window Elements – Chapel XXVIII.....	101
Figure 151. Visual Error Representation for Windows – Chapel XXVIII. ....	101
Figure 152. Average Error Value per Vault – Chapel XXVIII.....	102
Figure 153. Identification of Vault Elements – Chapel XXVIII.....	102
Figure 154. Visual Error Representation for Vault – Chapel XXVIII.....	102
Figure 155. Average Error Value per Stair – Chapel XXVIII.....	103
Figure 156. Identification of Stair Elements – Chapel XXVIII.....	103
Figure 157. Visual Error Representation for Stairs – Chapel XXVIII.....	103
Figure 158. Comparison between Point Cloud and HBIM model – Chapel XXVIII.....	104



**LIST OF TABLES**

Table 1. Summary of Reference methodologies..... 36

Table 2. Pros and Cons of Reference methodologies. .... 39

Table 3. Pros and Cons of Proposed workflow..... 106

## 1 ABSTRACT

In the Architecture, Engineering, and Construction (AEC) sector, as well as in the cultural heritage field, the digitalization of the built heritage is an ever more increasing demand until reaching a high level of importance, which can be attributed to the necessity for more accurate and efficient methods for preserving and managing historical structures. Digitalization methodologies, such as Building Information Modelling (BIM) and Geographic Information Systems (GIS) provide a detailed and precise digital representation, enabling more effective documentation and preservation of the cultural heritage sites, ensuring that historical and architectural details are available for future generations.

The Scan-to-BIM process, between its diverse functions has developed the focus in the generation of parametric models focused on the existing historical buildings or so-called Historic Building Information Modelling (HBIM). This process starting from raw point cloud data and finishing in the virtual representation of historical buildings, has the capability of facilitating the planning for maintenance, metric computation specialists, and management bodies.

In recent years, the development of Artificial Intelligence (AI) algorithms has further improved this process. Particularly, through Machine Learning (ML) and Deep Learning (DL), the semantic segmentation process facilitates the division of the diverse classes of architectural elements such as Columns, Floors, Roofs, Walls. This automation of the element classification not only speeds up the process but also minimizes the potential for human error, leading to more reliable and consistent models.

This thesis focuses on the automation of the Scan-to-HBIM process, specifically targeting the modelling of various architectural classes of the Chapels XXIV and Chapel XXVIII of the Sacro Monte di Varallo case study (North-West Italy). For each category of the architectural elements, the optimal techniques for their digital and parametric reconstruction have been investigated.

Advanced techniques in Visual Programming Language (VPL) and ML frameworks have been thus analysed to transform point cloud data into detailed HBIM models efficiently. Starting from the state-of-the-art ArCH dataset, manually annotated, the so semantically labelled point clouds have been processed using Dynamo for Autodesk Revit with the help of AI, producing a semi-automated workflow for the creation of the HBIM categories at various Level of Detail (LOD). Additionally, the assessment of time efficiency and accuracy deviations when converting point cloud data into HBIM models has been considered.

The proposed workflow aims to enhance future HBIM projects and processes by enabling the creation of elements at a higher LOD than B and C, specifically LOD D and beyond. This could be ensured by applying further semantic segmentation techniques. The proposed methodology is also open to improvement by the reformulation of diverse methods for its geometry creation process. As a further development, ML techniques could also be used in the future for the insertion of non-geometric parameters regarding to the Level of Information (LOI) such as material, colours, etc.

## 2 INTRODUCTION

Built Heritage serve as a tangible connection to the cultural heritage and legacy. Given their age and detail, as well as the lack of detail and documentation for most, preservation of these buildings is difficult at best and in many cases impossible. Of these, the 59 UNESCO World Heritage Sites [1] are the best examples of historical representation.

Nowadays, BIM is one of the most powerful methodologies in the conservation and management of heritage buildings. Detailed digital representations provided by BIM also aid in the maintenance, refurbishing, and preservation of these structures. Still, the process to bring scan data into HBIM models, often referred to as the scan-to-HBIM workflow, is extremely time-consuming and must be done carefully, especially for more complex architectures.

This master thesis, performed at Politecnico di Torino, aims to automate the Scan-to-HBIM process in Revit using Dynamo software, as a powerful plug-in that speaks the same language as Revit. It includes the use of professional Python packages to perform complex calculations and high memory-consuming operations. This research proves how 3D models at a LOD B and a LOD C can be automated using Dynamo, enhancing efficiency and accuracy in the modelling process.

In the first chapter, is explored why the Sacro Monte di Varallo complex was built by diving into bibliographic research. Is also discussed the inspirations of the architects and how the design evolved from its initial concept to the final construction. A closer look at two Chapels in Piazza dei Tribunali, specifically Chapel XXIV and Chapel XXVIII is taken. For each Chapel, the architectural details and their locations are highlighted.

The second chapter provides a summary of the current technologies and tools. It then explains the current standards for the BIM process, focusing on Italy but also considering other countries. Towards the end, the chapter compares four of the latest research studies on the automation of the scan-to-HBIM process, identifying their limitations and deciding which limitation to address.

The third chapter aims to present current methods and technical solutions used in the scan-to-HBIM process for cultural heritage. It begins by identifying the primary challenges in automating the scan-to-HBIM process for complex buildings like the Sacro Monte Varallo Chapels. Following this, an automated workflow is suggested, combining advanced scanning techniques with BIM software. This proposed workflow is then evaluated on two specific cases: Chapel XXIV and Chapel XXVIII of Sacro Monte Varallo.

The fourth chapter explains the procedure for creating Dynamo scripts using nodes and Python scripts to automate the modelling of various building elements, such as arches, columns, Mouldings, floors, doors, walls, windows, stairs, vaults, and roofs, for both levels of detail.

The fifth chapter presents and compares the results of this automated modelling process, demonstrating the precision and accuracy of the proposed workflow. But also, its flaws and requirements.

The final chapters summarize the findings and suggests potential avenues for future research, highlighting the importance of continued development in automated Scan-to-HBIM processes for the preservation of built heritage.

### 3 CASE STUDY: SACRO MONTE DI VARALLO

The Sacro Monte di Varallo, located in North-West Italy, stands as a one of the several examples of the country's rich architectural and cultural heritage. This UNESCO World Heritage site, renowned for its collection of chapels depicting scenes from the life of Christ, presents a unique opportunity to explore the intersection of historical preservation and modern digital technologies. This study focuses on the digitalization and preservation of data associated with the chapels at Sacro Monte di Varallo, proposed by the Main10ance project [2], which emphasizes the need for a digital transition in heritage conservation.

This chapter provides an in-depth analysis of chapels XIV and XVIII within the Sacro Monte di Varallo, offering details of their historical and architectural significance. By meticulously documenting and examining the architectural characteristics and historical contexts of these chapels, this research contributes to the broader discourse on heritage conservation.

#### 3.1 Sacro Monte di Varallo History

The purpose of this chapter is to present a case study of the architectural and cultural heritage sites located in Sacro Monte di Varallo, Italy. The selected case studies illustrate the integration automatization of the modelling process of Scan-to-HBIM by using DL frameworks in the semantic segmentation of point clouds for heritage preservation, as well as historical and artistic evaluations of specific chapels within the Sacro Monte di Varallo complex.

The Sacro Monte di Varallo was first built with statues, paintings, and images inside simple, rural buildings by Franciscan friar Bernardino Caimi in the late 15th century to create a “New Jerusalem” for pilgrims who could not travel to the Holy Land. In the first years of the 16th century, the complex was significantly expanded under the guidance of Gaudenzio Ferrari, who introduced realistic, life-sized sculptures (Figure 1) and frescoes to create a sort of realistic experience for visitors.

Not so many years later, expansion and remodelling works were executed by Galeazzo Alessi, a famous late-Renaissance architect. The subsequent centuries saw further developments by various artists and architects, transforming Sacro Monte di Varallo into a comprehensive spiritual and educational journey through Christ's life [3].



*Figure 1. Jesus before the court of Herod. [4]*

As of today, Sacro Monte di Varallo is renowned for its collection of chapels, each depicting scenes from the life of Christ. These chapels are masterpieces of architecture and art, reflecting the religious and cultural significance of their time. Due to its exceptional value, Sacro Monte di Varallo has been recognized as a UNESCO World Heritage site, underscoring its global cultural importance [4]



*Figure 2. Aerial view of Sacro Monte di Varallo. [5]*

### **3.2 Chapel XXIV**

This thesis specifically examines data from two Chapels, with Chapel XXIV, depicting "Jesus before the Tribunal of Anna," as the first case study. The current complex formed by Chapels 16 and 24 is the result of volumetric integration of two bodies different in size, yet linear and simple in both design and constructive concept. Chapel XVI, the smaller one, is dominated by Chapel XXIV, built more

than a century and a half later. Although they belong to different eras, they contrast ideologically with the ideal concept that led to their construction.



*Figure 3. Chapel XVI and Chapel XXIV Complex. [6]*

Chapel XVI, known as the "Son of the Widow," was built according to Alessi's design, reflecting the architectural code of the time. Meanwhile, Chapel XXIV, known as the "Tribunal of Anna," was planned by the same architect but eventually built by Bescapè in the early 1600s and completed in 1737, deviating significantly from all previous projects as (Figure 4). The original name of the Chapel XXIV was "Christ's Capture" [3].



*Figure 4. Hypothetical reconstruction of Galeazzo Alessi's project. [3]*



Chapel XVI consists of a single rectangular space, symmetrical in all openings, Mouldings, and external embellishments. It is covered by a barrel vault, arranged longitudinally, and the internal route certainly retains the central entrance already realized in the sixteenth century, although the exit was perhaps modified. The gable roof with two slopes on a rectangular plan is connected to the building by a simple cornice and classical triangular tympanums on the short sides, constructed in local stone.



Figure 5. Chapel XVI Facade. [6]

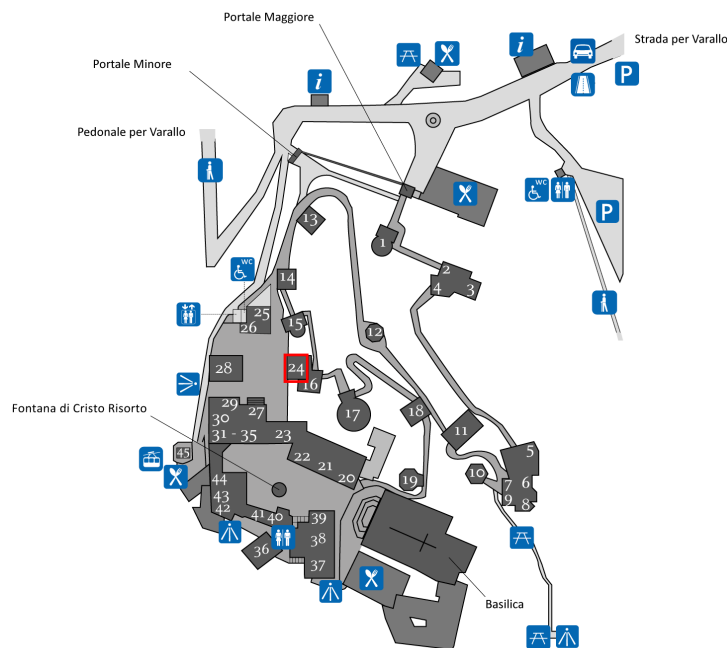


Figure 6. Location of Chapel XXIV. [4]

Chapel XXIV, the "Tribunal of Anna," was the last built on the Sacred Mount, and its proportions are perhaps not equally aligned with the prospect of the adjacent Piazza dei Tribunali. The main front, with three superimposed orders placed on a projecting plane, is resolved differently: baroque Mouldings on the upper orders, with false windows framed in stone, surmount a large portico with

three cross vaults enclosed by the perimeter walls, bordered on the front by stone balustrades and an external central staircase.

A slightly sloping partition wall internally determines an elliptical vaulted space of considerable dimensions, encompassing the plastic-pictorial scene of the Tribunal of Anna. The ambit's walls are all enclosed and secluded towards the back where a small opening perhaps allowed access to the adjacent space to the dome cover of the elliptical vault. The pavilion roof with four stone slopes is surrounded by a quite simple cornice and capitals terminating the corner pilasters.



*Figure 7. Outside view of Chapel XXIV: Jesus at the court of Anna. [6]*

Construction events and historical notes [3] indicate that both constructions should not have had any contact points, both appearing as isolated nuclei. The "Widow's Chapel" was commissioned between 1587 and 1590 by Matilde di Savoia, Marchioness of Pianezza, following designs by G. Alessi. It was completed between 1599-1606, as indicated in historical guides.

During the same period, Bescapè visited the Sacred Mount, imparting new guidelines for the location of the "Tribunal of Anna." The paintings, attributed to Galloni, and the stucco statues, recognized by Bordiga and Bartolomeo Revelli, were completed in 1737.

In 1969, Enes Pignoni, with Giorgio Perrone's collaboration, provided for the restoration. The core of Chapel XXIV - the Tribunal of Anna - according to designs already planned by Alessi in 1566, appears planned even in 1593 with an unsigned plan, now preserved in the Diocesan Archives of Milan. This design, perhaps reflecting Bescapè's intentions during his frequent episcopal visits, suggests that the chapel should be isolated, with its front portico perfectly aligned with the road

connection between Piazza della Basilica and Piazza dei Tribunali. The project was conducted by Giovanni Battista Morondi (1700-1770), who ensured architectural continuity despite stylistic inflections.



*Figure 8. First design of Chapel XXIV. [7]*

These buildings are volumetrically integrated, maintaining their independence in form and route. The "Widow's Chapel" appears linear and perpendicular to the other three chapels in Piazza dei Tribunali. The volumetric integration of these buildings forms an almost inconspicuous nucleus in shape, size, and environmental use. From a collected entrance, almost overshadowed by a high terminal blind, the pass to the symmetrical and scenically designed layout of Chapel XXIV on Piazza dei Tribunali is made.



*Figure 9. Interior of Chapel XXIV. [6]*

Regarding the constructive aspects, the plant comprises two irregular rectangular sections integrated together. The coverings are made of locally available, uneven loose stone. The vaults feature cross-shaped designs and a semi spherical dome. The stairs consist solely of external stone access steps to the buildings.

The wall techniques involve the use of stone and brick, while the floors are made of stone, featuring cut designs and regular arrangements for public routes, as well as mortars with brick and terracotta. Externally, there are decorative elements including squares and corner capitals made of stucco. Notably, there are no underground structures. As of June 30, 1980, Chapel XXIV was reported to be in a precarious condition. ([8], [6])

### 3.3 Chapel XXVIII

The second case study focuses on Chapel XXVIII, known as "Jesus before the Tribunal of Herod". This significant chapel, also designed by Giovanni D'Enrico, depicts Jesus being presented to Herod, emphasizing the dramatic interactions and expressions through life-sized statues and intricate frescoes.

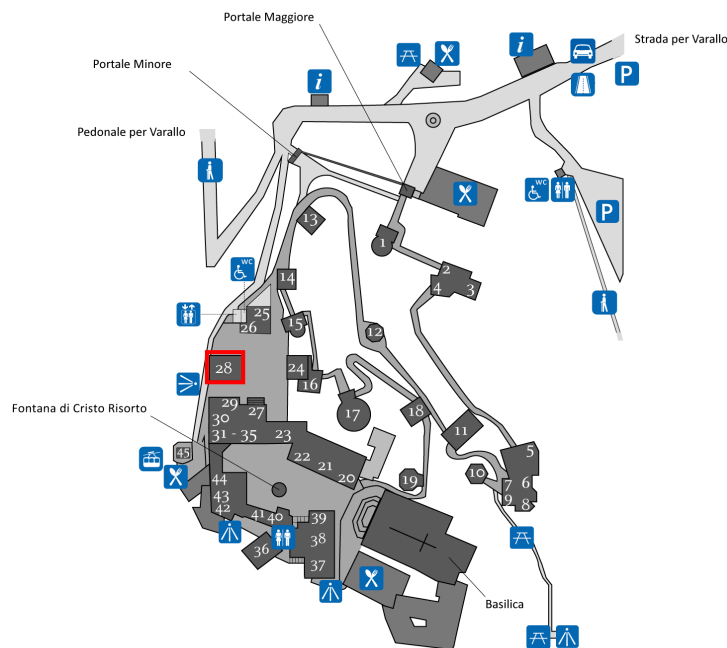


Figure 10. Location of Chapel XXVIII. [4]

The detailed artistic work captures the historical and theological essence of the scene, enhancing the educational and spiritual value of the site [8]. Located in the Piazza dei Tribunali, this chapel

constitutes the West Side, positioned centrally between chapels 25 and 27, and facing Chapel XXIV. Built as an isolated block, it stands prominently within the complex.

The building, with a rectangular layout, is composed of two rectangular bodies: a front porch, narrower, and a rear body that includes the chapel, wider and shorter in height. The porch, with four orders, rises on four columns supporting the architrave and a central arch, on which two-barrel vaults and full-height arches are set on the sides, and a large central cross vault all made of stone and plastered.

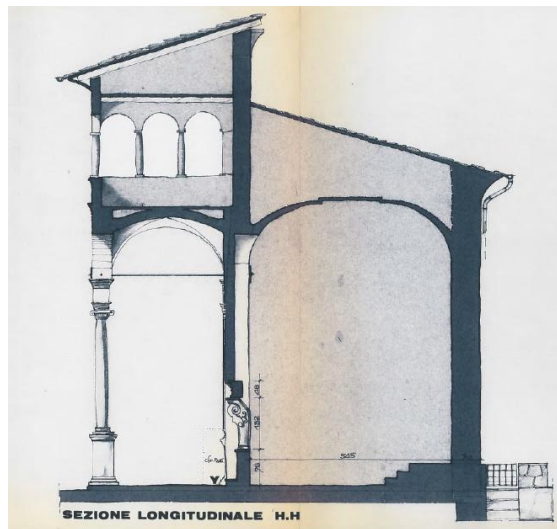


Figure 11. Longitudinal section of Chapel XXVIII. [9]

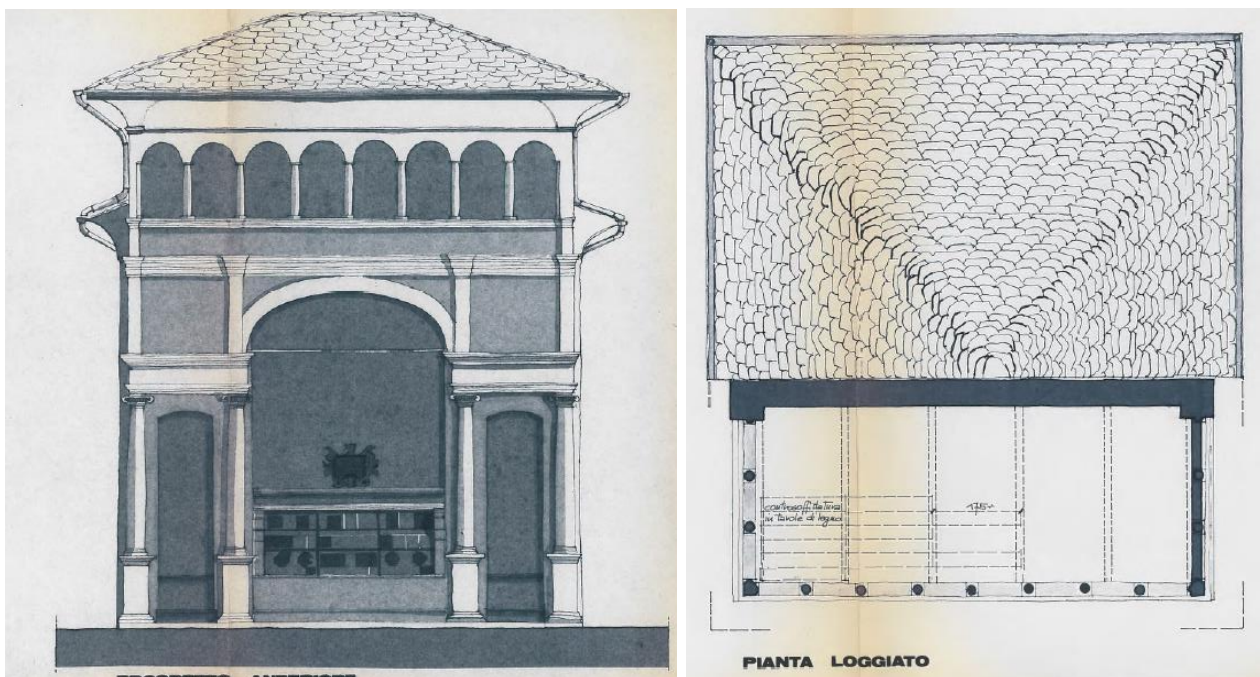


Figure 12. Plan View and Prospect of Chapel XXVIII. [9]

At the fourth order, there is an elevated loggia, formed by eight full-height arches (Figure 12) supported by stone columns and brick pilasters. The rear body consists of a single large bay, covered by a large pavilion vault. It contains the representation of Jesus before the Tribunal of Herod.

The floors are composed of regular stone slabs in the porch and irregular stone with mortar painted in diagonal squares inside (Figure 13). The stairs are limited to external stone access steps. The covers consist of two pitched roofs at different altitudes, supported by a wooden structure with a mantle in gneiss. There are no underground structures. The walls techniques involve stone masonry with smooth plaster and the internal vault is frescoed with an architectural and central perspective scene by Tanzio da Varallo, completed by Marziano Bernardi in 1628.

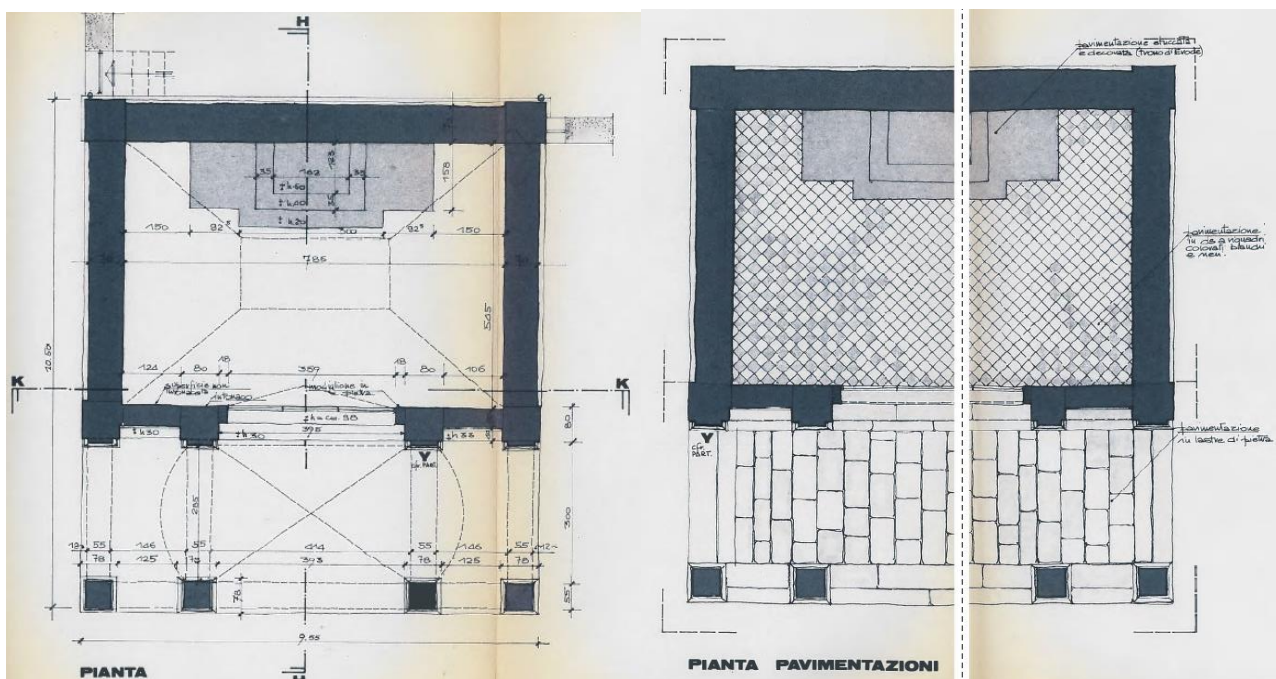


Figure 13. Plan Views of Chapel XXVIII. [9]

The 35 statues that compose the scene were made by Giovanni D'Enrico on painted plaster [9]. The scene is visible from the outside through a wooden grille with leaded windows located above a kneeling point. Significant restoration work was conducted around 1826, which involved replacing the columns and the local stone trabeation [10].

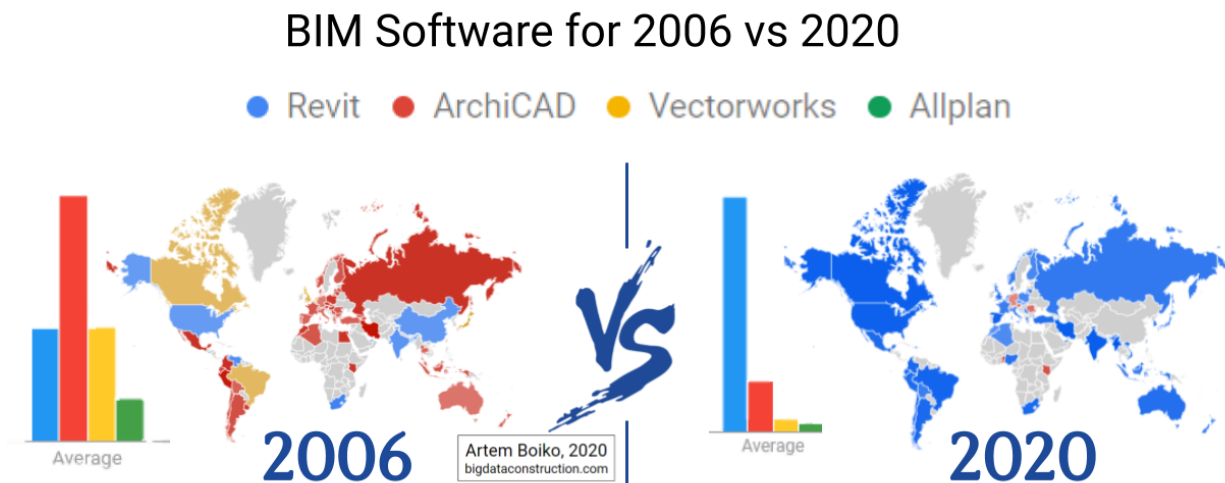


*Figure 14. Outside views of Chapel XXVIII: Jesus before the court of Herod. [9]*

## 4 STATE OF THE ART: SCAN-TO-BIM.

### 4.1 Current Technologies and Tools.

Scan-to-HBIM automation process uses advanced technology and software tools to transfer a physical structure into a digital model precisely. The most common tools for this transformation are laser scanners, photogrammetry software, and BIM software. For instance, some of the significant BIM software tools are Autodesk Revit, Graphisoft ArchiCAD, Trimble Tekla, and Nemetschek Allplan.



*Figure 15: BIM-CAD chosen software in the years. [11]*

Revit is among the most adopted BIM software tools and is prominent in its lead for architectural design, structural engineering, and construction. It enables multidisciplinary collaboration and advanced modelling in 3D. Critical features of Revit [12] include Parametric Modelling: Allows users to create intelligent models with parametric components. MEP Design: Facilitates the design of mechanical, electrical, and plumbing systems. Construction Simulation: Detailed simulation of the construction process. Multi-disciplined collaboration, advanced 3D visualization, rendering, performance analysis, and automated documentation creation make Revit an overall robust BIM tool.

Bentley Systems offers BIM software such as OpenBuildings Designer [13], which is tailored for infrastructure projects and large-scale construction. This software supports detailed 3D modelling and integrates seamlessly with other Bentley tools for project management and analysis. Bentley's software is known for its robust engineering capabilities and suitability for complex infrastructure projects.



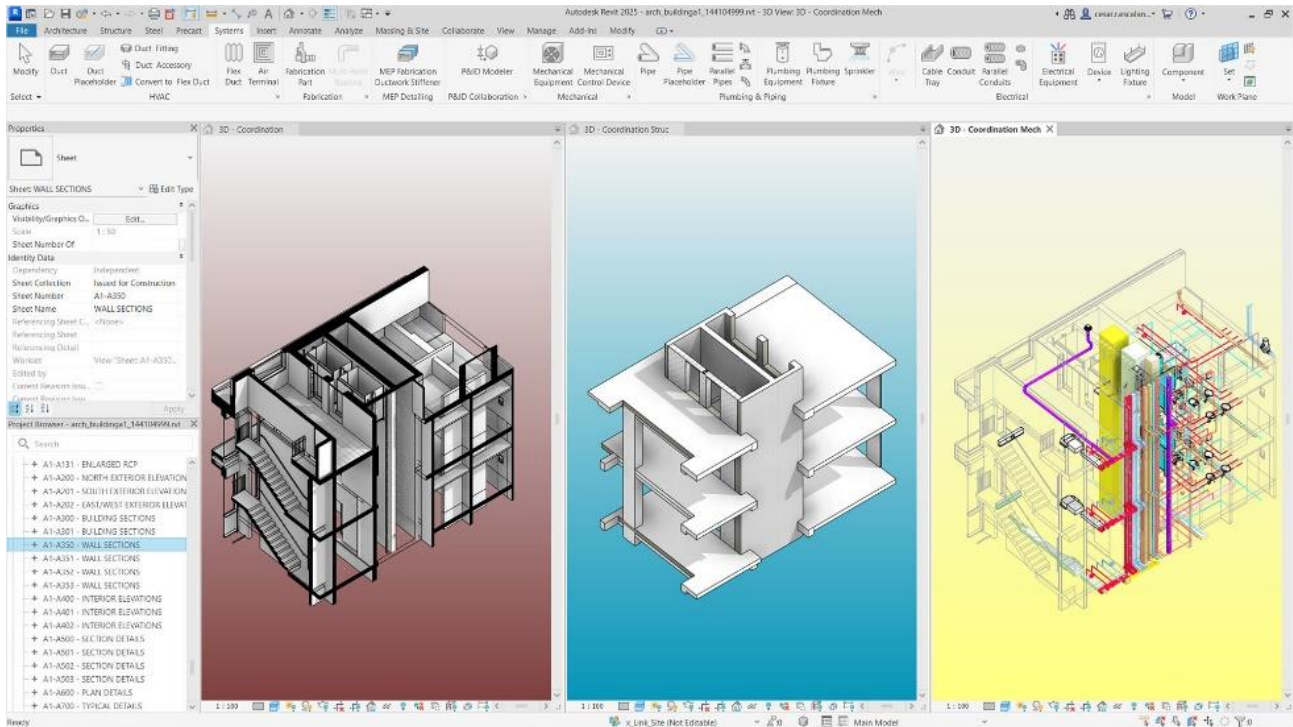


Figure 16. Autodesk Revit Architecture- Structure-MEP Disciplines. [12]

ArchiCAD by Graphisoft [14] is renowned for its user-friendly interface and powerful architectural design tools. It supports open BIM standards, enhancing interoperability with other software. ArchiCAD excels in providing detailed building models and is favoured by architects for its design-centric features and efficient workflow management.

Tekla Structures [15] is a BIM software known for its strength in structural engineering, particularly useful for steel and concrete detailing. It provides precise modelling tools that aid in fabricating and managing complex structures. Tekla is often used in conjunction with other software to enhance the detail and accuracy of structural designs.

Allplan [16] offers a comprehensive suite of tools for both architects and engineers. It provides powerful features for detailed design and supports 3D modelling and BIM workflows. Allplan's strength lies in its flexibility and ability to manage both architectural and engineering tasks within the same platform.

Additionally, visual programming languages such as Dynamo for Revit and Grasshopper for Rhino enhance the modelling process through automation and custom workflows. Dynamo allows for the creation of scripts to automate repetitive tasks, generate complex geometry, and manage data, integrating seamlessly with Revit. Grasshopper provides similar capabilities for Rhino, and through add-ins, it can also interface with Revit.

## 4.2 Methodologies and Standards.

BIM methodology employs different LOD to define the precision, completeness, and reliability of information at different stages of a project. The differences in these levels are standardized to ensure consistency and clear communication among all the parties involved. The understanding and utilization of LOD is crucial for the successful implementation of BIM across different countries all over the world, each having its own specific standards. This subchapter explores the differences between each LOD, their denominations, and the differences between the BIM standards in the principal countries like United States, the United Kingdom, and Italy, with a particular focus on the Italian regulation.

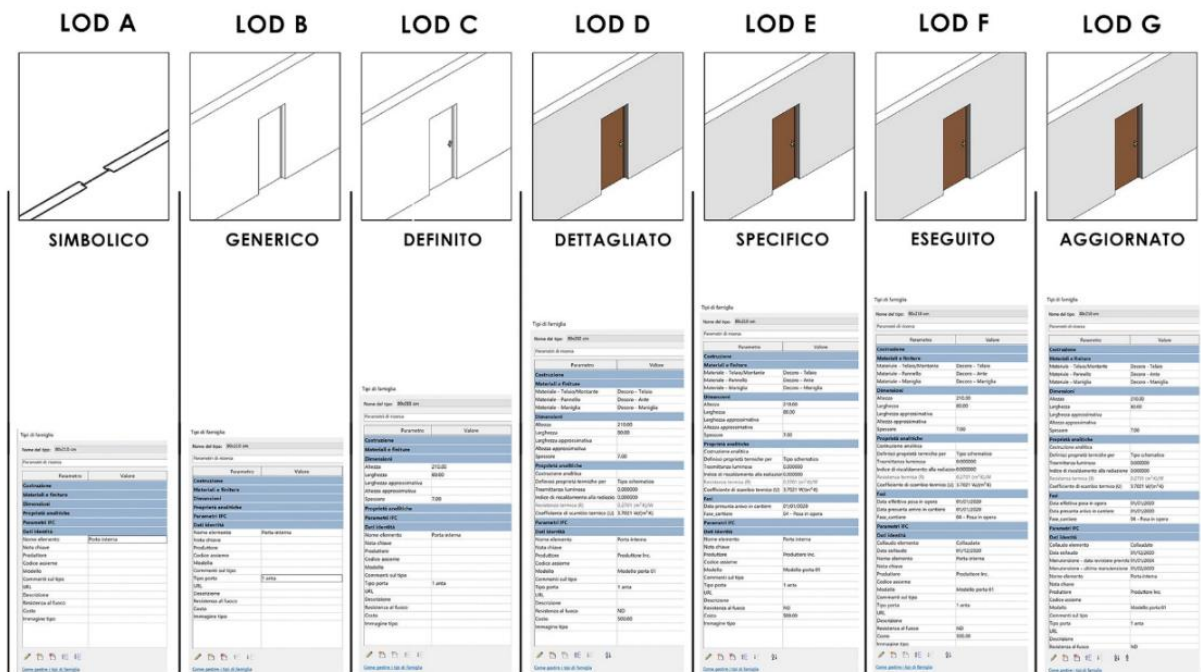
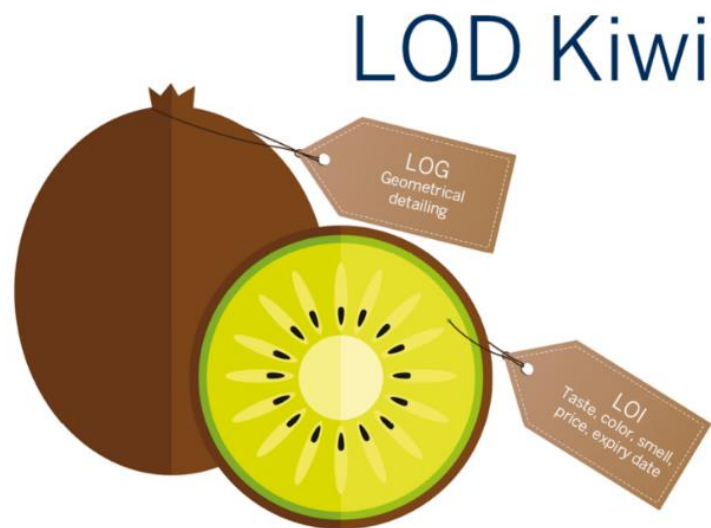


Figure 17. Differences in Levels of Details. [17]

The concept of LOD in BIM ranges from basic representations to highly detailed as-built models. Each LOD is composed by two key components: Level of Geometry (LOG) and Level of Information (LOI) [18].

The LOG refers to the graphical representation of elements within the BIM model, defining the accuracy and detail of geometric shapes, dimensions, and spatial positioning of the elements. The LOG evolves from simple shapes with approximate dimensions at the conceptual stage to highly detailed and accurate representations in the final as-built models.

The LOI encompasses the non-graphical data associated with BIM elements, including information such as materials, quantities, performance data, and other attributes necessary for construction, operation, and maintenance. The LOI evolves from basic information about materials and approximate quantities at initial stages to detailed specifications and as-built conditions in the final stage.



*Figure 18. LOD = LOG + LOI [18]*

In Italy, the UNI 11337 - 4 [19] standard outlines the LODs used in BIM. Being LOD A, or Conceptual, the most basic level used during the initial project phases. The LOG at this stage includes symbolic representations or approximate dimensions, while the LOI provides general data such as the overall shape and location. This level is crucial for initial feasibility studies and preliminary design concepts.

As the design progresses to LOD B, or Schematic Design, the LOG becomes more refined, including approximate size, shape, and location. The LOI at this stage includes basic material information and approximate quantities. This level is used for developing the design concept further and preparing for detailed design stages, providing a more tangible representation of the project's layout and primary components.

LOD C, or Detailed Design, represents a more advanced stage where the LOG includes elements with specific size, shape, location, and orientation. The LOI at this stage includes detailed materials, quantities, and assembly details. This level is crucial during the design development and construction documentation phases as it provides the necessary detail for accurate construction planning and

execution. It ensures that all elements are precisely defined and that their interactions and connections are well-documented.

In the Construction Documentation stage, known as LOD D, the LOG is further refined to include detailed connections and interfaces between components. The LOI provides comprehensive coordination information, such as assembly details and construction sequences. This level is essential for ensuring that all the parts of the project are accurately depicted, and that potential clashes or issues are identified and resolved before construction begins.

LOD E, or Fabrication & Assembly, is the stage where the LOG is highly detailed and ready for fabrication. The LOI includes detailed fabrication data such as exact dimensions, tolerances, and materials, making it suitable for actual construction processes. This level ensures that all components can be manufactured and assembled accurately, based on the detailed specifications provided.

The final level, LOD F, or As-Built, represents the completed construction as it is built. The LOG at this stage is fully accurate and precise, reflecting the constructed building, while the LOI includes comprehensive details necessary for operation and maintenance. This level captures the true state of the project after completion, including any changes made during construction, ensuring that the final model is an accurate representation of the built environment.

The United States, the United Kingdom, and Italy each have their own BIM standards, with specific denominations and focuses for LOD. In the United States, the National BIM Standard-United States (NBIMS-US), defined by the BuildingSMART Alliance and the American Institute of Architects (AIA), uses a straightforward numerical system.

The United Kingdom follows the UK BIM Framework [20], guided by the UK BIM Alliance and the Centre for Digital Built Britain (CDBB), which complements with the RIBA Plan of Work stages which focuses on the design stages [21]. Meanwhile, Italy's UNI 11337 [19] standard defines LODs that align with international practices while incorporating specific Italian regulations.

Despite the differences in terminology and numbering systems, the core concept of LOD remains consistent across these standards. The US standards focus on the geometries from symbolic representation finishing in an accurate as-built representation, the UK standards align LOD with project stages for better workflow integration, and the Italian standards combine global best practices with local regulations by adding a LOD considering that elements can be updated.

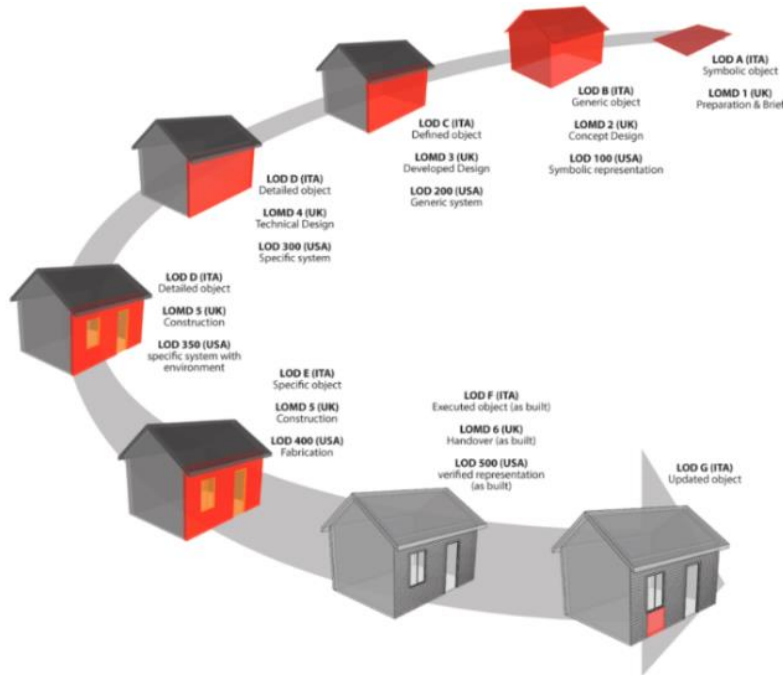


Figure 19. LOD differences in Italy, U.S. and U.K. [22]

Understanding and applying the appropriate LOD for each phase of a project is essential for efficient project delivery, cost and time savings, and enhanced collaboration. Standardized LOD definitions help reduce errors and rework, improving the overall quality and reliability of the construction process.

Additionally, adhering to national standards ensures regulatory compliance and adherence to industry best practices, leading to better project outcomes and more efficient building processes.

### 4.3 Recent developments in the Built Heritage domain.

The ArCH dataset is a comprehensive collection designed for the classification and semantic segmentation of architectural cultural heritage. It includes 17 annotated and 10 non-annotated point clouds from various historically significant sites, many of which are UNESCO World Heritage candidates. The semantic annotation follows the CityGML LOD 3/4 and incorporates the IFC standard and the Art and Architecture Thesaurus (AAT).

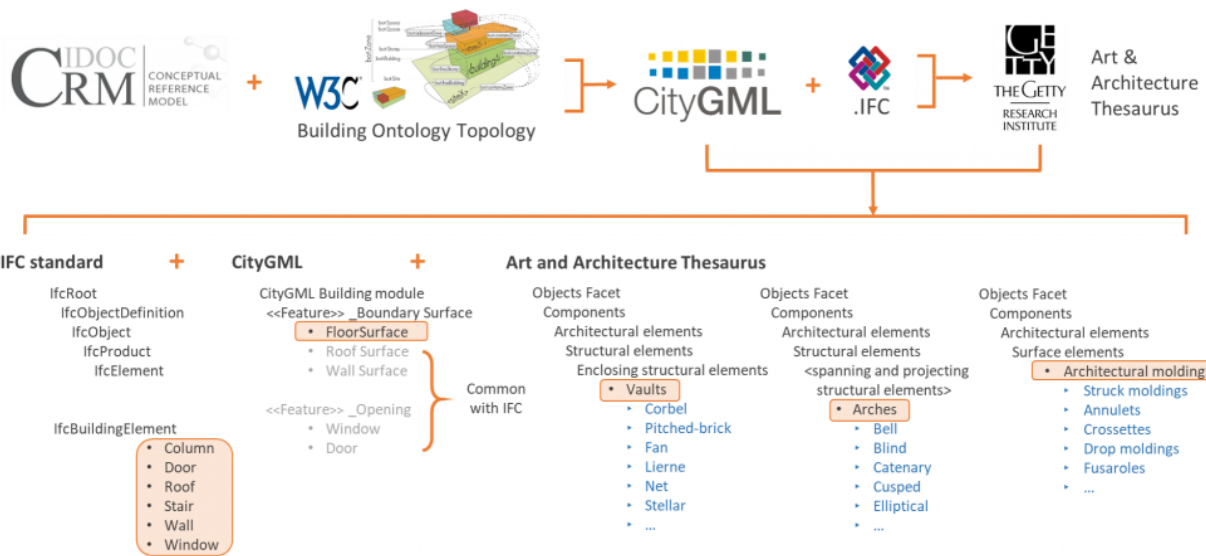


Figure 20. ArCH dataset Class Definition. [23]

The dataset categorizes elements into nine classes, such as arches, columns, and roofs, with an additional "other" category. This detailed annotation aids in the precise analysis and preservation of heritage structures, enhancing the dataset's value for developing advanced computational models for heritage conservation.



Figure 21. ArCH dataset Categories. [23]

Between the latest studies the Multiclass semantic segmentation for digitisation of movable heritage using DL techniques is studied [24], the application of DL for multiclass semantic segmentation of movable heritage is meticulously explored. Conducted at the Politecnico di Torino's Laboratory of Geomatics for Cultural Heritage, this research focuses on automating the digitization processes of museum artifacts.

Utilizing DL techniques, the researchers developed a methodology to automatically generate exclusion masks, thereby optimizing photogrammetric procedures. This approach not only accelerates the digitization process but also ensures high precision in the generated 3D models, facilitating superior documentation and preservation of cultural heritage.

Similarly, the research ML-Based Monitoring for Planning Climate-Resilient Conservation of Built Heritage [25] underscores the integration of ML with photogrammetry in the conservation of built heritage. Their study employed drones for photogrammetric surveys, creating detailed 3D models of the Palazzo Pitti. These models were instrumental in the automatic segmentation and classification

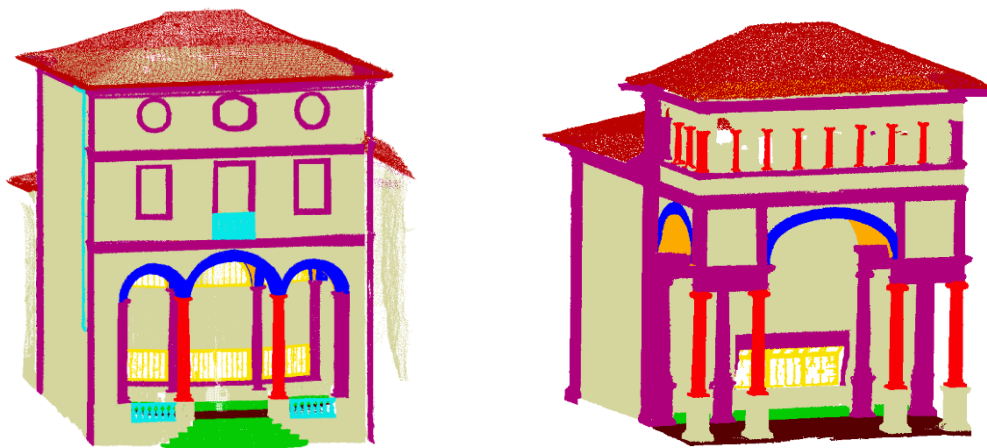
of architectural elements, allowing for the identification of structures most vulnerable to environmental degradation.

From the previous articles is observed that remote sensing and modelling techniques have, in these past years, become very significant contributors to the labour of preservation and restoration of cultural heritage sites.

One of the most recent research [26] focused in the Sacro Monte di Varallo case study, analyses point cloud semantic segmentation by using a DL framework to recognize the diverse types of historical architectural elements in the named ArCH dataset (Figure 22), which are segmented in 10 different categories: 0 - Arch, 1 - Column, 2 - Decoration, 3 - Floor, 4 - Door, 5 - Wall, 6 – Window, 7 – Stairs, 8 – Vault, 9 - Roof.

This then applied to a dataset containing 11 labelled point clouds from different historical and architectural contexts, followed using a modified Dynamic Graph CNN (DGCNN) for point cloud segmentation, modified to include additional features.

The network is then trained and evaluated using different scenes of the ArCH dataset giving as a result a remarkably high segmentation accuracy with the modified DGCNN outstanding when compared to other methods like PointNet, PointNet++ and PCNN. These results will be used later to improve modelling times for Building Information Modelling, in this case, referred to Historical BIM (HBIM) models.



*Figure 22. Chapel XIV and Chapel XVIII semantic segmentation. [23]*

Techniques such as Scan-to-HBIM can be effectively applied together with Finite Element Modelling (FEM) for analytical dynamic modelling of structure behaviour in heritage buildings [27] with

particular use in identifying and addressing structural issues in masonry and roofing systems (Figure 23). This by proposing a methodology that integrates a 3D metric survey into a 3D HBIM model using Rhinoceros 3D integrated with Autodesk Revit for the structural BIM model and followed by a dynamic simulation of the Finite Element Model (FEM) in PRO\_SAP.

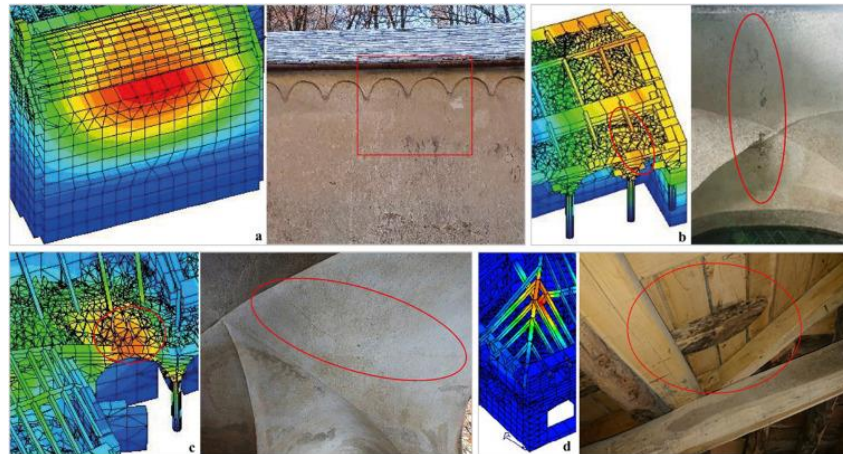


Figure 23. Comparison between analytical models and damages on wall structures. [27]

#### 4.4 Latest semi-automated Scan-to-HBIM approaches.

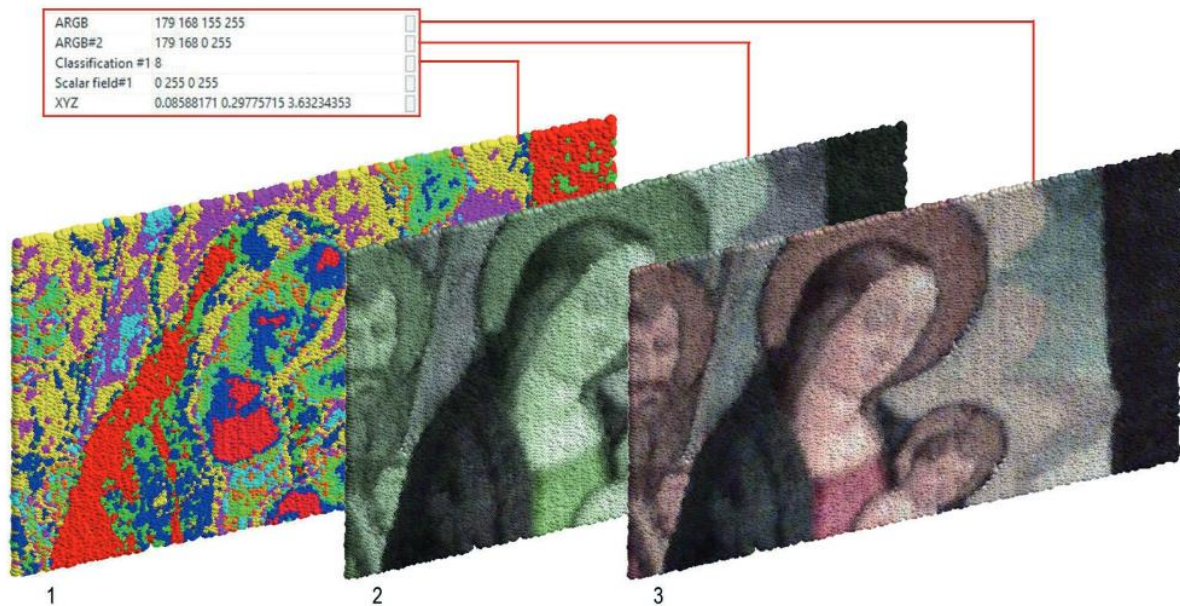
In the context of architectural heritage preservation, significant advancement has been made toward the digitization and modelling of historical buildings through automated Scan-to-HBIM methodologies. These approaches have been used for the conversion of a vast amount of 3D data into BIM models that are detailed and extremely precise for use in documentation, analysis, and preservation purposes.

Most recent Volumetric Modelling approach [28] focuses on the automated transformation of point clouds into HBIM models using volumetric modelling methods. This involves the creation of cubic-shaped Revit families containing relevant data such as colour and materials, enhancing the LOD and accuracy in representing historical buildings. By integrating diverse datasets, including GIS, BIM, and real-time sensor data, this methodology underscores the importance of managing and centralizing multifaceted information.

The research demonstrates the advantages of this automated process in improving the efficiency of data management and the fidelity of the resulting models. The programming for this process was done using Dynamo software, importing the point clouds from an ASCII file with a .txt extension to retain all relevant data. Since the elements are cubic-shaped, they simplify the reality significantly, and like meshes, they are essentially surface representations of reality.



For a better representation or smoother surface representation, the size of the cubes must be small, with 1 mm being the smallest size allowed in Revit. This requires the point cloud to contain enough points to cover all visible surfaces adequately. However, a denser point cloud demands more processing capacity from the computer, necessitating the segmentation of the point cloud to balance processing capabilities and data coverage.



*Figure 24. 3D Model representing the Sacristy Mayor Painting. [28]*

From a Parametric Modelling perspective [29] the exploration of the application of AI and ML techniques in the HBIM process. Their semi-automated approach employs AI for semantic segmentation and classification of architectural elements within point clouds.

Utilizing Random Forest classifiers, this methodology automates the recognition and annotation of complex structures and the manual modification of parametric template geometries, via Grasshopper with a linkage to Revit, which are replicated for each category, enhancing the efficiency and accuracy of HBIM creation.

This research emphasizes the potential of AI to streamline traditionally time-consuming manual processes of point cloud data segmentation and the replication of geometries within the model. However, the process is not fully automated due to the manual creation or modification of parametric geometries.

Additionally, there is a possibility of misclassification of elements during segmentation and the need for training the ML algorithm for other datasets. The accuracy of the elements in the model can vary from 0 to 0.5 m, which may require revision for high-precision needs.

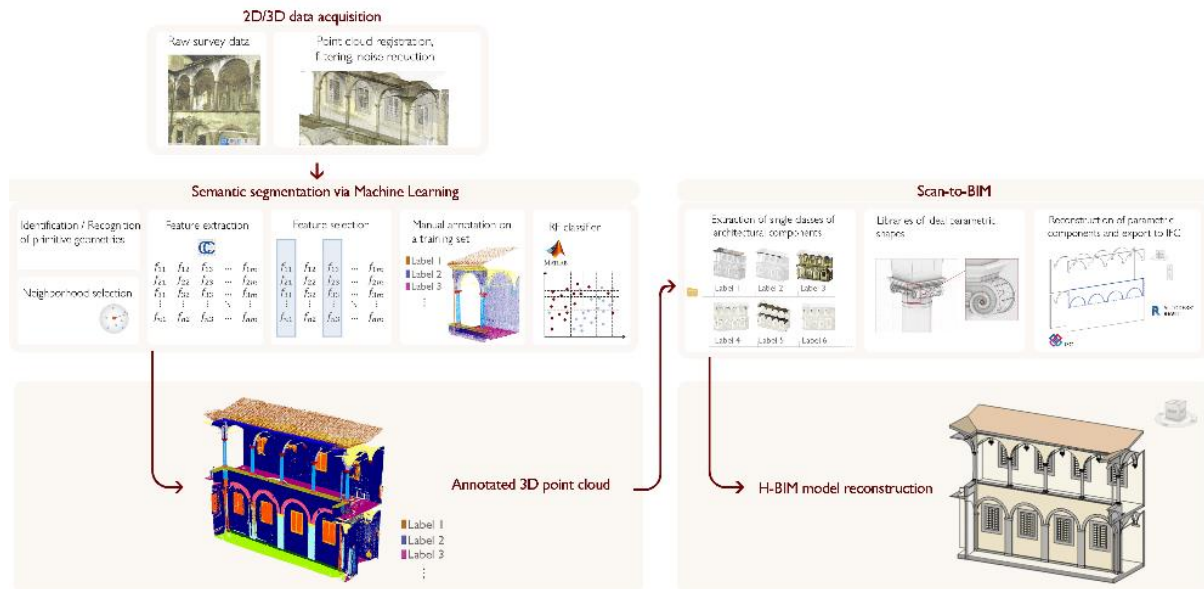


Figure 25. Croce's proposed methodology. [29]

Recent Surface Modelling Approach [30] propose a semi-automated workflow for converting point cloud data into BIM models using Surface Modelling methods. This approach leverages automatic segmentation and classification techniques to create 3D meshes that accurately represent architectural elements, significantly reducing the manual labour traditionally required in HBIM.

The utilization of advanced algorithms, such as Delaunay triangulation and the Ball-Pivoting algorithm, enables the generation of precise 3D surfaces from raw point cloud data. This process is conducted using Rhinoceros and its VPL software Grasshopper, and subsequently, the model is manually exported to ArchiCAD (see Figure 26) to obtain a BIM model.

A notable limitation of this method is the interoperability between Rhinoceros and ArchiCAD, which does not support automated export of the modelled elements. Additionally, the use of meshes results in objects that are primarily voids, lacking interior materials (for columns) or interior layers (for roofs, walls, floors). However, the accuracy of the data is remarkably high, with values ranging between 0.04m and 0.057m for the mesh elements.

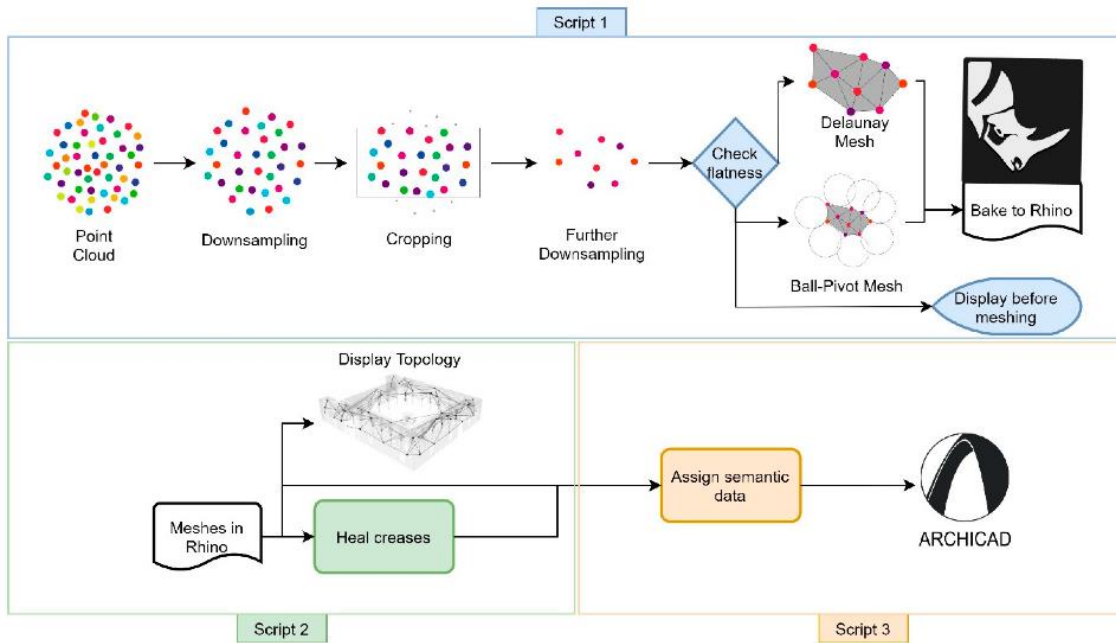


Figure 26. Automated Parametric Workflow. [30]

Latest Surface Modelling approach [31] propose a semi-automated Scan-to-HBIM process using the VPL Dynamo to employ Surface Modelling via Boundary-Representation (B-rep) methods. This approach bridges the gap between rapid 3D data acquisition and labour-intensive modelling. By manually conducting the segmentation and classification phases, the issue of element misclassification is addressed but lose the automated advantage [31]. Instead, the individual exportation of segments separately as .txt files, which are then reimported individually into families for reconstruction.

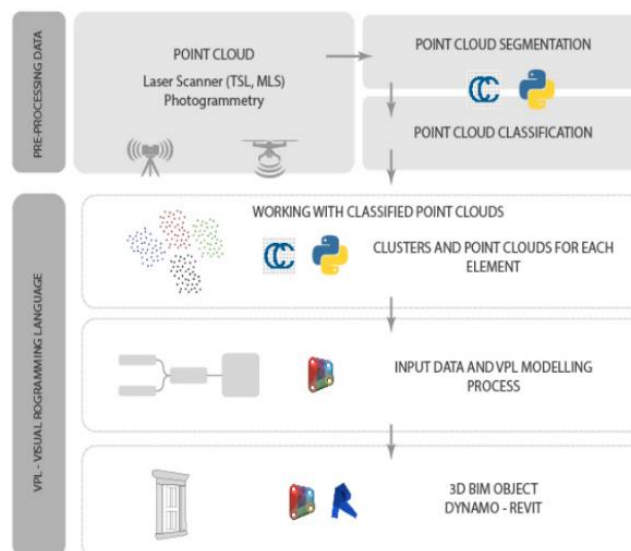


Figure 27. Automated Parametric Workflow. [30]

One notable procedure used is the clipping box, which allows for better interception of points inside the volume to create a NURBS curve and subsequently generate geometry through a sweep operation. A limitation of this clipping box procedure is that more points inside the box result in a higher LOG, but at a greater computational cost.

This procedure also relies on the precision and accuracy of the point cloud, with the average accuracy of the elements varying from 1 to 2 cm in cases where data gathering is performed well.

In comparison, these studies reveal a common goal of enhancing the Scan-to-HBIM process through automation and advanced computational techniques. **Escudero** [28] focuses on fully automated approaches leveraging diverse datasets and advanced algorithms, while **Croce et al.** [29], **Avena et al. (2023)**, and **Andriasyan et al.** [30] emphasize semi-automation to balance computational efficiency and practical usability.

Each methodology presents unique strengths and addresses specific challenges within the HBIM workflow, highlighting the importance of integrating AI, ML, and parametric modelling. Challenges remain in fully automating the Scan-to-HBIM process, including managing large datasets, ensuring data accuracy, and developing automatized protocols. A summary of the distinct aspects of each research can be found in the next table:

	AUTOMATION LEVEL	ACCURACY	SOFTWARE USED	LIMITATIONS
<b>ESCUADERO (2023)</b>	Automated	Up to 1 mm	Revit + Dynamo	High processing capacity.
<b>CROCE ET AL. (2023)</b>	Semi-Automated	0-0.5 m	Revit + Grasshopper	Manual creation templates.
<b>ANDRIASYAN ET AL. (2020)</b>	Semi-Automated	0.04 - 0.057 m	Rhinoceros + Grasshopper - > ArchiCAD	Interoperability between software.
<b>AVENA ET AL. (2023)</b>	Semi-Automated	0.01-0.02 m	Revit + Dynamo	Manual Segmentation.

*Table 1. Summary of Reference methodologies.*

After comparing the table, it can be seen that for the Volumetric Modelling approach [28] the volumetric elements don't represent the actual materials and layers of the real elements (Figure 28) meaning that families from whichever software is used, cannot be applied, plus this requires a high

processing capacity from the computers or a manual segmentation of the points clouds in order to be done.

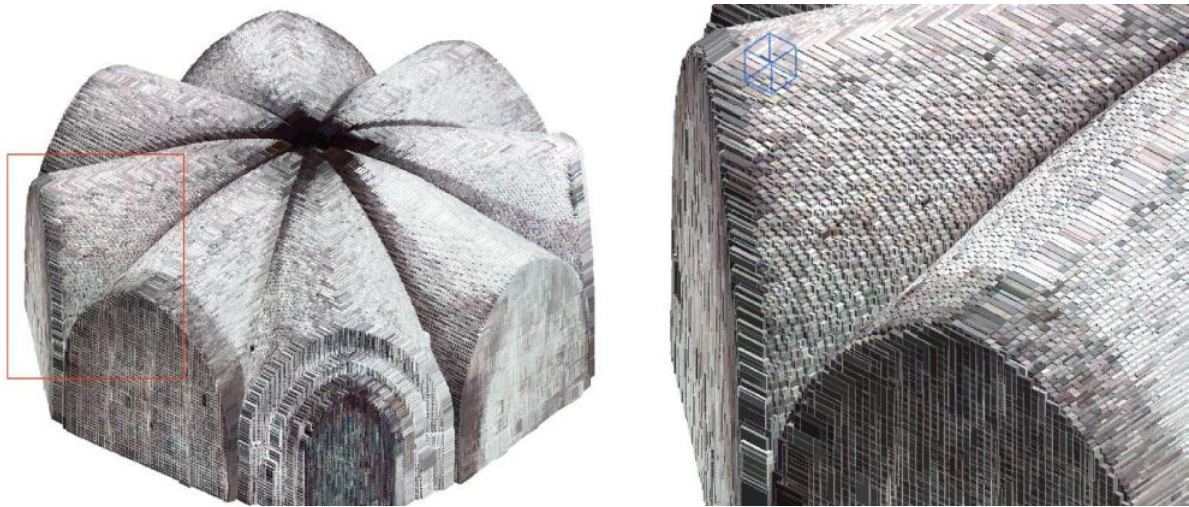


Figure 28. Generated model in Revit from Point Cloud. [28]

From the parametric modelling approach for Heritage Components [29], the whole method relies on a manual creation of template families that are later replied (Figure 29), the problem in here is that the only way to do parametric families is manually, neither dynamo nor grasshopper are allowed to render them parametric. At most what can be done is to create geometries and from that assign families to it.

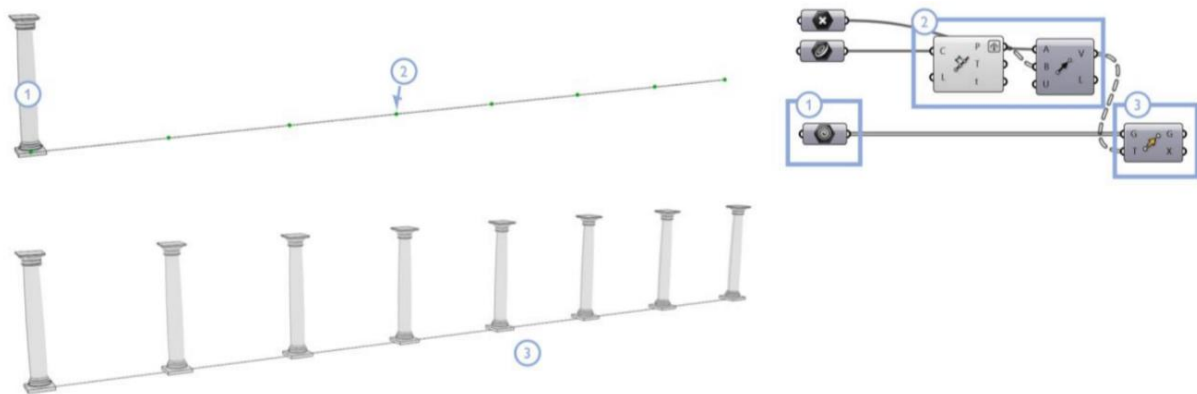


Figure 29. Replication of template families using Grasshopper. [29]

In the Surface Modelling approach by using meshes [30] the main limitations are the interoperability between diverse software, which can just be solved by the creation of plugins that allow the interoperability between Rhinoceros and ArchiCAD and the void elements that come from meshes.

As of July 2022, the interoperability problem was solved by “Robert McNeel & Associates“ [32], but having the surfaces of the elements (Figure 30) is just a part of what HBIM is, because also internal materials have to be considered, considering that is how they are really made.

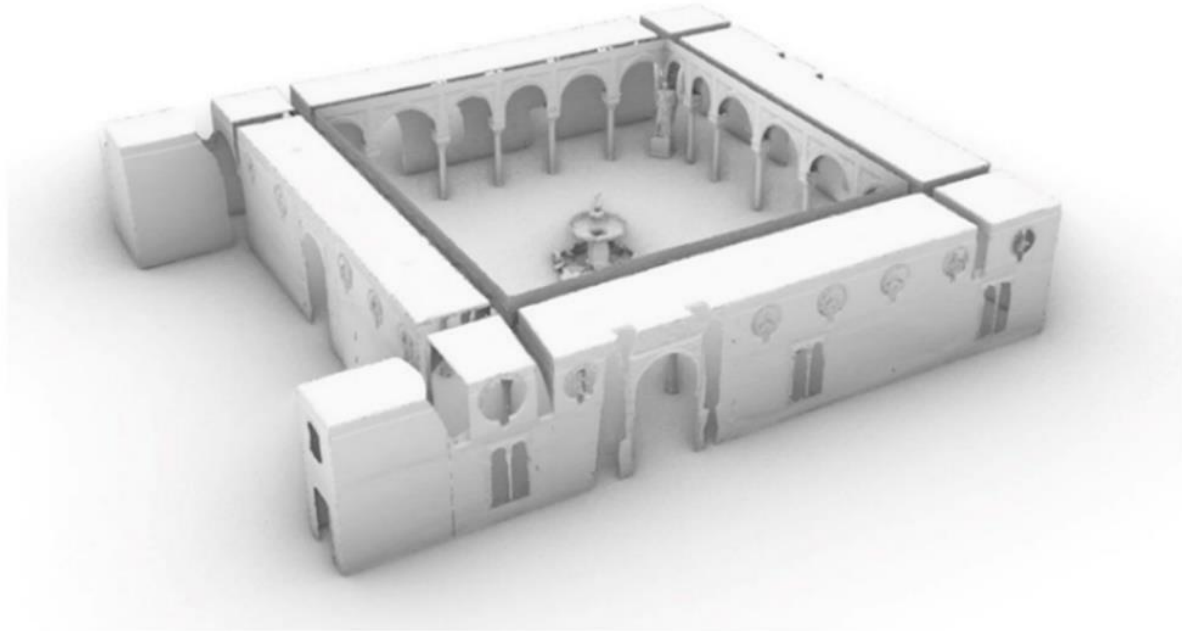


Figure 30. 3D Model of Casa di Pilatos in ArchiCAD by using meshes. [30]

At last, but not least, is considered that the surface modelling approach for Heritage Components [31] main problem is the manual segmentation of points by exporting separate .txt files in order to create the profiles of each one of the elements (Figure 31) that are going to compose the family.

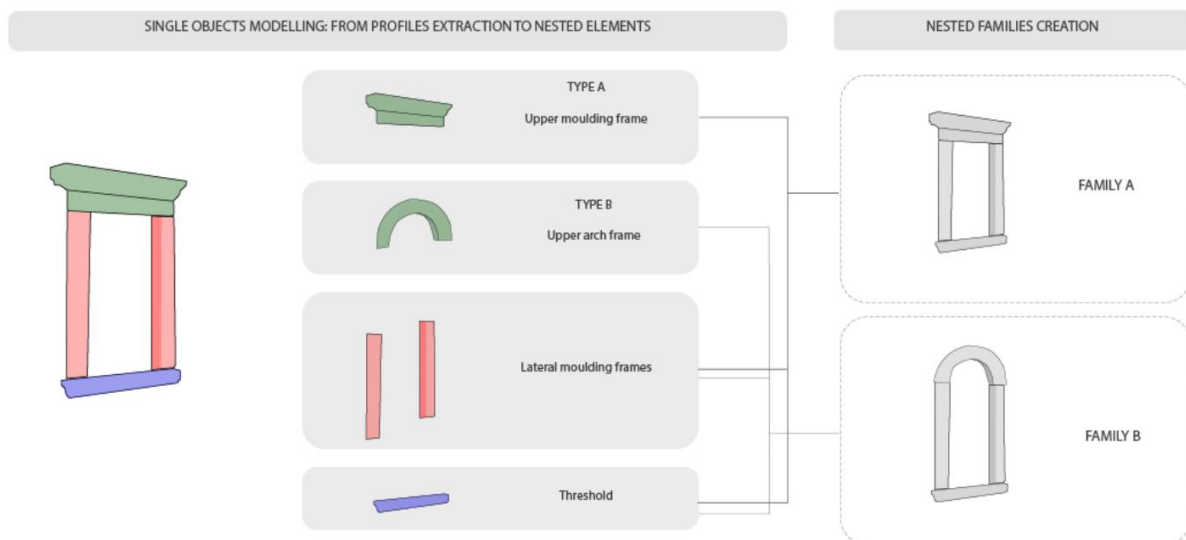


Figure 31. Manual segmentations of elements. [31]

After considering these 4 methodologies where the limitations are more difficult to sort in some cases, the problem to solve is the integration of Revit's integrated VPL, Dynamo and a solution for fitting methods for the detection of planar surface, this by applying a further instance segmentation for points performed by Croce et al. This considering that actual advancements in technology such as ML can help to automate this process.

**PROS**

**CONS**

	<b>PROS</b>	<b>CONS</b>
<b>ESCUDERO (2023)</b>	Reach high precision and visual representation.	Requires high computational cost. Based on Surfaces. Approximates reality by using volumes.
<b>CROCE ET AL. (2023)</b>	Reach high Level of Detail.	Requires manual creation of template families.
<b>ANDRIASYAN ET AL. (2020)</b>	Reach good representation of surfaces.	Based on surfaces, not filled elements.
<b>AVENA ET AL. (2023)</b>	Reach high Level of Detail	Requires manual segmentation of elements.

*Table 2. Pros and Cons of Reference methodologies.*

## 5 METHODOLOGY

This chapter refers to the methodologies for converting point cloud data into HBIM models, focusing on different LOD and utilizing Dynamo's capabilities. It details the process of reading input data, categorizing elements, and applying specific algorithms and nodes for efficient and accurate modelling. By employing a two-way approach—using both LOD B and LOD C—this method ensures flexibility in modelling while maintaining the capacity to refine and enhance the model as needed.

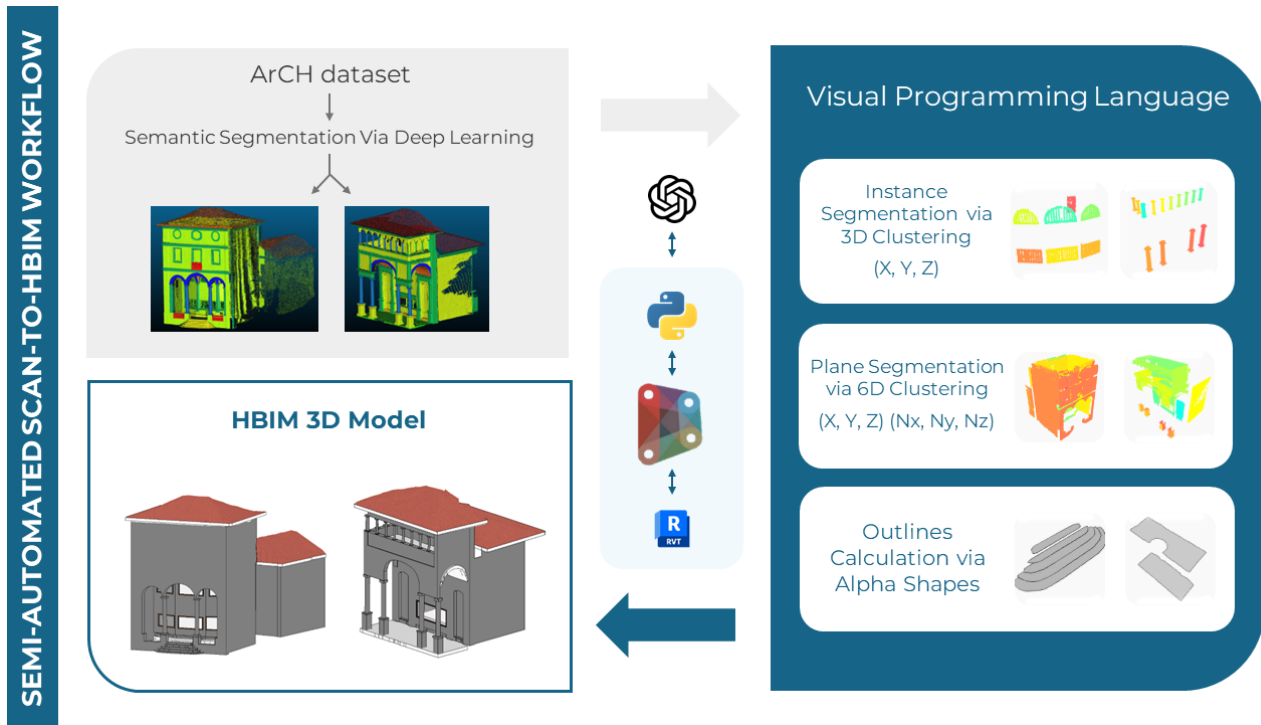
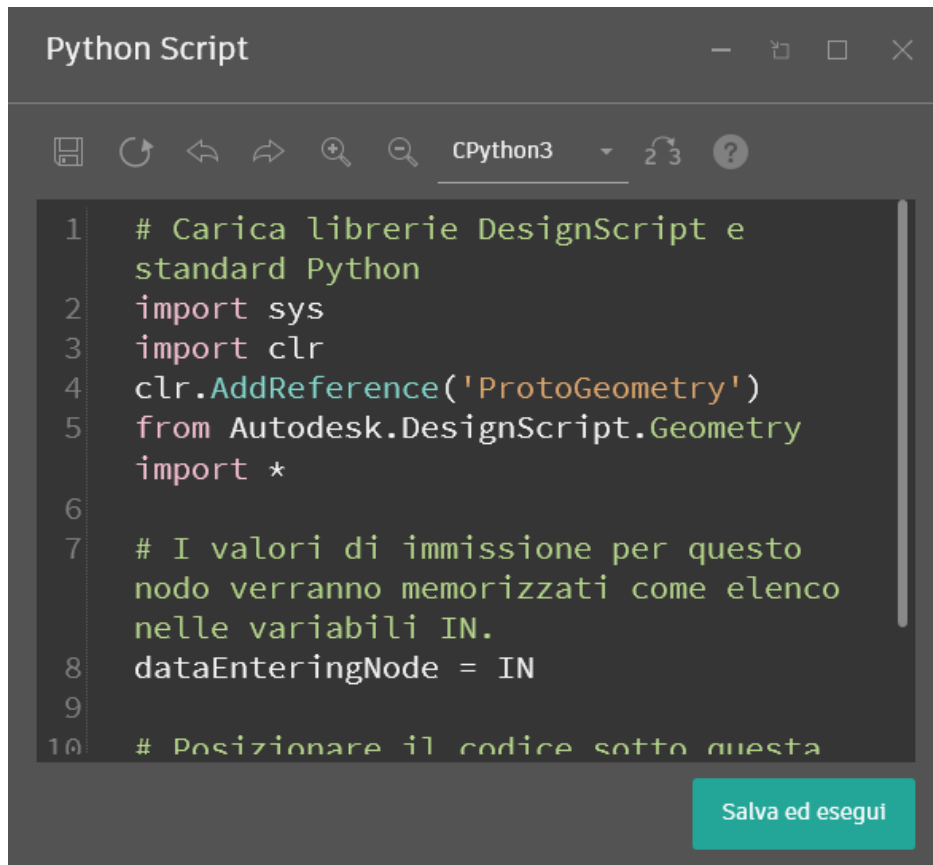


Figure 32. Semi-Automated Scan-to-HBIM proposed methodology.

As a first step, the AI capabilities are exploited by its facility of coding. The process is very repetitive each time the prompt is written, it starts with the language in which the code is written, the inputs the node is going to have, the process that is desired to do and what is required for the output.

The programming language used is C, implemented by CPython3 which is the standard native Python interpreter that Dynamo uses [33], this allowing now to import the libraries from Python differently than the previous interpreter Iron Python that didn't allow to import these.





```
Python Script
Python3
1 # Carica librerie DesignScript e
  standard Python
2 import sys
3 import clr
4 clr.AddReference('ProtoGeometry')
5 from Autodesk.DesignScript.Geometry
  import *
6
7 # I valori di immissione per questo
  nodo verranno memorizzati come elenco
  nelle variabili IN.
8 dataEnteringNode = IN
9
10 # Posizionare il codice sotto questa
```

Salva ed esegui

Figure 33. Default Python node content using CPython3.

For the inputs are considered the hierarchy of the lists coming from the previous node, since the lists have different levels [34], the code must create all the necessary iterations to reach the level where the points or the data is located. So, the input prompt must contain how many levels are in the output of the connecting node.

Regarding the algorithm development, the prompt must contain the process desired by giving a description of the problem. Also, the method desired must be specified or in other cases some simple explanation of the problem to be solved can be texted to make ChatGPT find an optimal solution as output.



Apply the 3D Hough-transform algorithm to detect geometric primitives such as planes. This method can be used in conjunction with RANSAC to improve detection accuracy.

Applying the 3D Hough-transform algorithm in conjunction with RANSAC can significantly enhance plane detection accuracy in point cloud data. Below is a step-by-step guide on how to implement this in Dynamo using Python. The process involves the following main steps:

1. **Convert Dynamo Points to Numpy Arrays:** This makes mathematical operations easier.
2. **Implement the 3D Hough-transform Algorithm:** This algorithm will help detect planes by transforming the point cloud data into a parameter space.
3. **Combine with RANSAC:** Use RANSAC to refine the detected planes and handle noise and outliers.

### Step-by-Step Implementation

#### Step 1: Prepare the Dynamo Environment

Ensure you have Dynamo installed with the Python scripting node available. Organize your point cloud data into a list of lists, where each sublist contains the points for a specific part of the cloud.

#### Step 2: Implement the Python Script

Use the following Python script to implement the 3D Hough-transform and RANSAC algorithms. This script should be placed inside a 'Python Script' node in Dynamo.

```
python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

import math
import random
import numpy as np

# Function to convert Dynamo points to numpy array
def points_to_np_array(points):
    return np.array([[p.X, p.Y, p.Z] for p in points])

# Hough-transform algorithm for plane detection
def hough_transform(points, theta_res=1, phi_res=1, rho_res=0.1):
    points = points_to_np_array(points)
    max_dist = np.sqrt((points**2).sum(axis=1)).max()
    theta = np.deg2rad(np.arange(-90, 90, theta_res))
    phi = np.deg2rad(np.arange(-180, 180, phi_res))
    rho = np.arange(-max_dist, max_dist, rho_res)

    accumulator = np.zeros((len(theta), len(phi), len(rho)))

    for point in points:
        for t_idx in range(len(theta)):
            for p_idx in range(len(phi)):
                rho_val = point[0] * np.cos(theta[t_idx]) * np.cos(phi[p_idx]) + \
                    point[1] * np.cos(theta[t_idx]) * np.sin(phi[p_idx]) + \
                    point[2] * np.sin(theta[t_idx])
                r_idx = int((rho_val + max_dist) / rho_res)
                accumulator[t_idx, p_idx, r_idx] += 1
```

Figure 35. Python code generated by ChatGPT.

## 5.1 From the Point Cloud to the HBIM Model.

Since the input points are given in a .txt file of ASCII format, these data points need to be converted into objects (points, colours, integer numbers, vectors) used in Dynamo. Python scripts make it possible to conduct this conversion.

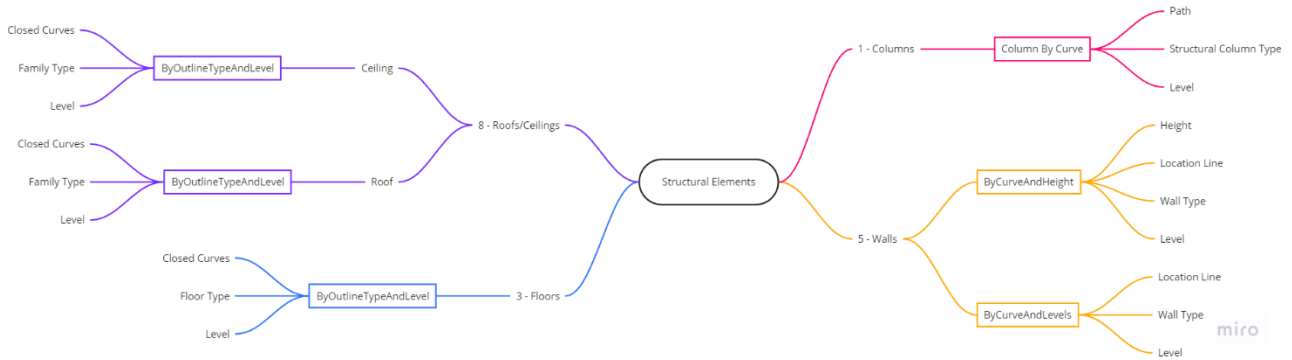


Figure 36. Classification of Structural elements.

The first entity that Revit is being dealt with is the Families. They are subdivided into three types: Structural Elements, Hosted Elements, and Non-structural Elements. More specifically, the families consist of Columns, Floors, Walls, and Roofs/Ceilings in the category of Structural Elements. The hosted elements cover other elements like Mouldings, Doors, and Windows. Finally, the non-structural elements are composed of Arches, Stairs, and Vaults. As detailed in Figure 37.

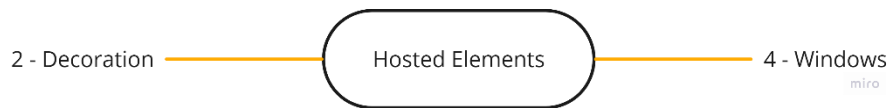


Figure 37. Classification of Hosted Elements.

Each category will require a different approach for modelling because some of its elements can be easily modelled (like floors and roofs), and others are more complex to model (vaults and mouldings). This is highly dependent on detail, form, or irregularity of the element. While it can simplify reality, BIM modelling should, on the one hand, include as much information as possible about each building component.

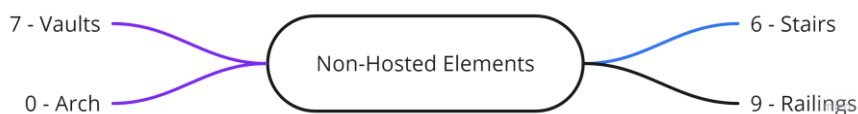


Figure 38. Classification of Non-Hosted elements.

The following two methods of approach will be employed for the thesis:

LOD B: The approach makes use of standard parametric families in Revit, which are suitable for more straightforward and quicker modelling, at the cost of detail at initial stages, but accessible for post model processing.

LOD C: This approach features more detailed modelling with mostly non-modifiable geometries. However, some members, such as floors and roofs, will be done using generic families and basic dynamo nodes, which can be adjusted to many family parameters, such as height offsetting from the base. Or for the creation of geometries for elements that have a higher complexity and cannot be created by using directly the Dynamo nodes.

In any case, whatever choice is made, the geometric names encode the most critical information, for instance, in the case of columns the type of name includes the length of base, radius of shaft, and length of capital.

Elements will then be modelled following a constructive approach; this approach is taken to replicate the actual construction process so that some sort of logic exists in the order of elements being constructed.

Project parameters must also be set and modified concerning the construction reference. For example, levels will be set concerning the floor and roof heights because, generally, the floor and roof elevations correspond to the levels. Additional parameters must be added to the geometries or families for a particular LOI. For example, hierarchies will be added as parameters for every type of element so that these types of families are always privy to their category.

Each category will be modelled using Dynamo capabilities. Python libraries will be imported into Python nodes within Dynamo environment despite being usually unsupported. As an example, methods developed by the Scikit-learn [35] library for clustering will be used to make it possible to identify points belonging to several elements. The use of ChatGPT in designing Python scripts will make this possible.

### ***5.1.1 Input reading.***

Considering that the point cloud is in a .txt file then is needed to read the data inside by considering that all the values are separated with blanks and that the structure of each lines has the next values: X, Y, Z, R, G, B, Element classification as a scalar value,  $N_x$ ,  $N_y$ ,  $N_z$ . This is realized by using a python node that needs to take the input which is the .txt file path, then should be capable of loading the file to read every value, this reading by considering spaces as delimiter value of each column. After that, it must extract the first ten columns of the data into separate lists, each one containing each of the values. Finally, the script outputs these lists as tuple values for a posterior processing.

### **5.1.2 ArCH dataset Categories**

Before proceeding into the modelling, the point clouds must be subsampled due to the high computational cost required. The subsampling is made by using a minimum space between points of 5 centimetres. This to obtain files around the 10 MB of weight, which require up to 8 GB of RAM when running the Dynamo Script.

#### **5.1.2.1 Floors**

The process of modelling floors is realized through the Floor.ByOutlineTypeAndLevel [36] node in Dynamo. This node has three necessary inputs: the Closed Curves defining each floor's boundary, the type of each Floor, and the Level at which would be liked to host each floor.

Beginning with the Closed Curves, the most convenient way to manage this is to use the Alpha Shapes algorithm [37]. The algorithm guarantees that the perimeter definition is precise. This is achieved by generating the minimum convex form, which could accommodate all the points that define the floor outline.

In the case of the Floor Type, if the dimension for the lowest floor is unknown, it is assumed to be of some generic type. The height difference is examined for the next floor, and the correct type of floor is determined according to a ceiling or vault. This method ensures that each floor is assigned a relevant and structurally appropriate type based on its position and the surrounding architectural features.

The lowest level is set to all floors to eliminate mistakenly assigned levels. When the floor family is placed, the level adapts by changing the Height Offset parameter. This means reassignment of the floor to the closest level based on its actual elevation within the building.

The height offset parameter is modified to guarantee that each floor is correctly positioned relative to the building levels and that it's consistent and accurate within the model, thereby allowing for the correct floors to be built with precision boundaries and proper type with exact level assignment.

#### **5.1.2.2 Columns**

For columns, the approach is more complex due to the nature of columns, so the process is a bit more involved. Two different methods based on the LOD, both beginning with HDBSCAN Instance Segmentation [35] are applied, which partitions the points defining each element into lists.

At LOD B, is used the Dynamo node Column.ByCurve [36]. This node requires input that is a path, and this is created using the node Line.ByBestFitThroughPoints [36]. The level is first set to the lowest level and further adjusted if the condition that the minimum z-value of points is greater than

the z-value of the level. This method only allows the creation of structural columns. For support, this type from the model [36] is chosen.

Coarser detail in this approach enables later manual modification of the family to allow the creation of highly detailed elements at a later stage. The main advantage of this technique is that it is simple, fast, and the family type can easily be changed, but this is at the expense of the initial detail.

For LOD C, the method consists in dividing each element into three parts consisting of the base, shaft, and capital [29], with these divisions based on the normal values, specifically the normal (Nz) value [31]. As the first assumption, it is considered that the points at the upper part of the base have a positive normal value around 1. The same consideration is done for the capital, in which the lower part should have normal values in a negative direction with values near -1.

Initially, each element is divided into two parts to calculate the normal (Nz) values for each division, wanting to ensure that the positive or negative normal (Nz) values are taken in the points where the shape change exists. Points below the mid-elevation with a positive normal (Nz) value are designated as base top points. These points are used to create a cuboid geometry representing the base of the column, especially if the base shape differs from the shaft.

For the upper part, points with a negative normal (Nz) value are considered. These points define the capital section of the column that is also represented as a cuboid, in contrast to points belonging to the shaft. The shaft points lie between the heights of the base and the capital and can be extruded to form the column shaft [38]. This is one way in which a more detailed representation of a column can be formed as its various parts are accurately modelled.

The three geometries of the base, shaft, and capital are assembled after being created [39]. The joint of these geometries is critical for a column to be adequately represented within the model. After ensuring that the geometries can be appropriately assembled, a family type is defined for these, and subsequently, they can be inserted into Revit. This method has a higher LOD than level B as the distinction in geometrical shapes of the column is marked.

These can be applied based on the applicable degree of detail, and it can satisfactorily model complex columns when the two meet the desired project requirements. The ordering of the points for efficient modelling based on the applicable degree of detail in both methods is guaranteed with HDBSCAN clustering [35] at the first stage.

### 5.1.2.3 Walls

In the case of the walls, the first step is to filter the elements that have a Normal value in the Z direction greater than 0.75 and lower than -0.75, this because walls are vertical elements so any point that has a value out of this range must be considered as noise.

After this filtering, another separation of the points is realized and this time is according to the levels present in the project, those ones coming from the floor points.

To create walls the Dynamo node `Wall.ByCurveAndHeights` [36] is used, this node contains 4 inputs which are: curve of the wall defining the curve in the middle axis of the wall, height of the wall, level in which is located the wall and `WallType` which is the type of family of the wall.

Since the walls are mainly straight and not so detailed elements, a down sampling can be realized to reduce memory usage. As next step the planes of each wall are calculated by using the Random Sample Consensus (RANSAC) + Hough Transform algorithm [40], [41], [42] that identifies the inliers within a dataset by iteratively selecting random subsets and then fitting the model to these subsets.

But since the algorithm is not so efficient due to the noise and irregularity of the data, the result are planes that contain points far from each other. Other alternative is the creation of a histogram with posterior calculation of each line inside it [43].

This is then solved by first creating clusters applying the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [44] algorithm to separate farther points inside of each plane, and then a re-joining of the planes is done by joining the planes that have a dot product between normal values less than 0.9 which is the cosine of 25 degrees, this meaning that walls that have an orientation difference less than 25 degrees are joined.

But also, another condition must be applied, and is that the distance between the centres of the walls in the normal direction must be less than 1 meter, this to not join walls that are far from each other but still be able to join both faces of a wall when present.

As a second solution is found an Instance Segmentation by using not just positioning but also normal values [45] [46], this adding another layer of filtering to these algorithms. The clustering algorithm chosen is HDBScan due to its spatial clustering as main capacity. The only parameter used is the main cluster size defined as a minimum of points in each cluster equal to 10. For this method is still needed the joining of the planes, which is done by considering the conditions that have a dot product between



plane normal values less than 0.9, a distance in the normal direction less than 0.75 which is considered for 110% of the greatest value of the wall widths, and as final parameter a maximum distance between clusters of 0.8 meters to not join farther planes.

The first step to find the lines is the creation of Bounding Boxes that are an abstract cuboid representation of the geometry. Then its faces are calculated and the faces with the highest areas are going to be the simple representation of the walls' faces. Then to get the middle line a middle point is calculated between the two centroids of the surfaces and moving one of the faces to this point.

But sometimes walls present just one face, so a conditional is applied that in the case of a distance between the faces less than 10 cm or bigger than 37.5 cm from the points, it does not make any translation, just considers the middle line as the exterior face. After this, the exterior lines are trimmed/extended to avoid overlapping of walls.

For the second parameter, the height of the wall, the difference between the maximum Z coordinate and the minimum Z coordinate is calculated and rounded. For the third parameter, the level, the split of the points for each level was already done and this corresponds to the list of levels coming from the roof and floor nodes.

For the Wall Type it is more complex since the family types must be created so the wall width is calculated by considering multiplying the average distance of the points from the face by 2 and by 100 so it is converted into centimetres. This is done just for the walls that have two faces. After this a family type is created for each one of the dimensions and the resultant name type is "Wall X cm", being X the thickness of the wall in meters.

Finally, a Base offset from the level is applied to the wall by considering the lowest elevation of each list of points.

#### ***5.1.2.4 Roofs and Ceilings***

Roofs, ceilings, and floors share similar input parameters. The creation of roofs and ceilings involves using the `Roof.ByOutlineTypeAndLevel` [36] and `Ceiling.ByOutlineTypeAndLevel` [36] nodes in Dynamo. Both nodes require three main inputs: the family type, the closed curves defining the perimeter, and the level at which they are located.

For the family type, a generic family in Revit is selected, with the understanding that roof and ceiling types differ. This ensures that each element is appropriately categorized while allowing for flexibility in later stages of the modelling process.

The level input follows a procedure like that used for floors. Initially, the lowest level is assigned to avoid misassignments. Once the elements are placed, the level is adjusted by modifying the height offset parameter, ensuring that each element is correctly positioned according to its actual elevation in the building.

The closed curves defining the perimeter of roofs and ceilings are determined using the Alpha Shapes algorithm [37]. This algorithm generates the smallest convex shape encompassing all points representing the perimeter, ensuring precise boundary definition.

In the case of roofs, after determining the closed curves and setting the initial level, an additional step is required to adapt the shape to the point cloud. This involves adding points to refine the roof's shape, making it more accurately conform to the actual structure. Due to computational limitations, subsampling must be performed to manage processing requirements effectively. This step ensures that the roof accurately reflects the detailed topography of the building while maintaining manageable data sizes for processing.

#### **5.1.2.5 Mouldings**

These elements are not modelled.

#### **5.1.2.6 Windows**

Since Windows are Hosted elements, the approach consists in finding dimensions, Host element and Location point. For the dimensions, a bounding box can be created and the greatest value between the width and length is the one defining the width of the window, for the height the value is taken from the Bounding Box. For the host, the closest wall element must be found, this by calculating the distance between the window points and the walls. For the Location Point, the mid location of the element is found by using the origin point of a plane that fits the best the points. At last, the elements are created by using the `FamilyInstance.ByHostAndPoint`

#### **5.1.2.7 Vaults**

For the Vaults, the approach is to get the points that are defining its boundaries. If the Length is different than the Width, then it is not a simple Cross vault, meaning that there are also Barrel Vaults in the shape. Considering this, the first step is to create a Barrel Vault that contains all the length of the cluster, then filter the points that are farther than at least 5 to 7 centimetres, this to get the Cross Vault points. To create the Cross Vault the first step is to calculate the Diagonal Arcs, which are defining the 4 external points and the middle point in which both Arcs intersect. Since some points

may have been lost in the separation from the Barrel Vault points, the 4 external points of the Cross Vault are translated to the minimum height of the cluster. The next step then is to get the Arcs defined by the Barrel Vault and then the 2 missing arcs have its start and end in the previous 4 external points and the middle point is defined by the height of the intersection point of the diagonal Arcs. With this 4 Arcs, 2 new Arcs are created in middle of each direction, this to create a surface that is constrained in passing at the middle point of the Cross Vault. Final steps consist in cutting these two surfaces to keep only the external parts and joining them with the Barrel Vault. From these surfaces an array of points at every 0.1 parameter is created, resulting in a matrix of points that are going to define the shape of the roof element. For the thickness of the Vault, a cutting region is created at the middle of each Arc, then the difference between the maximum and minimum heights is calculated and averaged. If there are no Arcs present on an element, this means that the element is not a vault but a ceiling, to which is applied a default thickness of 10 centimetres.

#### **5.1.2.8 Arc**

These elements are considered together with the Vaults. Its main function is to determine the thickness of the Vault.

#### **5.1.2.9 Stairs**

For the stairs, a simple approach is performed. First, a general Convex Hull of the stairs is created. Followed by this, the points are filtered by their normal value, so only the values that have a positive Nz value are used.

Now, the differentiation of each step as an individual instead of treating them altogether must be done. So is applied a clustering and computed the planes that best fit each list of points. This allows to calculate what the convex hull of each step would be and then extrude it down to the bottom. That leaves one full element composed by the union of the solids. After this, the family is assigned to each one of the elements.

#### **5.1.2.10 Others**

These elements are not modelled since do not contain enough information.

## 6 MODELLING PROCESS

This chapter describes the transformation of processed point cloud data into a detailed HBIM model. Utilizing Dynamo and Python scripts, the 10 element categories were modelled in two different LOD. The process begins with data importation and Instance Segmentation of elements such as floors, columns, and walls. Each section explains the specific routines and scripts used to automate the creation of these elements, ensuring both efficiency and accuracy. The modelling of simpler components at LOD B is covered first, followed by more detailed components at LOD C, addressing the unique challenges posed by each element type.

### 6.1 Import Data

To import the .txt into the dynamo environment a Python Node is used in both cases, where this code reads a dataset from a file specified by the file\_path variable, with the data being space delimited. The `numpy.loadtxt` [47] function is utilized to load the data into a NumPy array.

The dataset is expected to have ten columns, each representing different parameters. These columns are then individually assigned to separate lists: X\_list, Y\_list, Z\_list, R\_list, G\_list, B\_list, Scalar\_list, Nx\_list, Ny\_list, and Nz\_list. These lists respectively correspond to coordinates (X, Y, Z), colour components (Red, Green, Blue), a scalar value, and normal vectors (Nx, Ny, Nz). Finally, these lists are output together as a tuple.

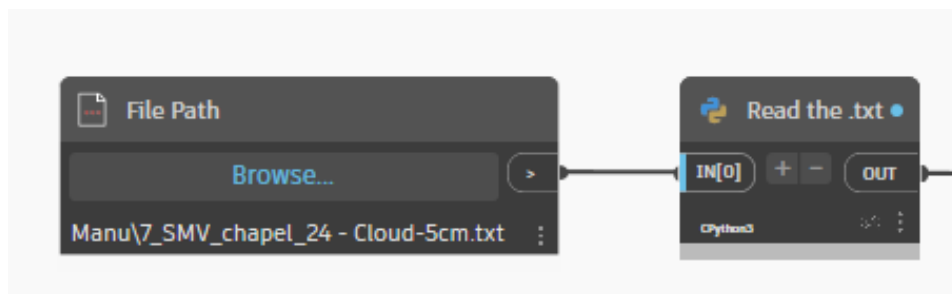


Figure 39. Reading Node for Point Cloud .txt file.

## 6.2 Dynamo Routine for LOD B

### 6.2.1 Floors

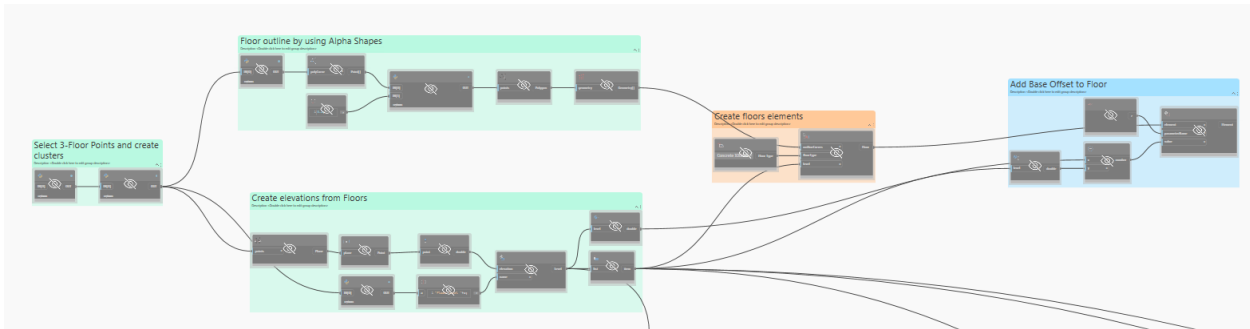


Figure 40. Overall script for Floors - LOD B.

Starting with the creation of the floors, the first thing to do is to separate the points that have a scalar value of 3 in the list of points of the input, to have a better organization of the data. This by taking as input a set of lists, referred to as Lists\_input, which contains data corresponding to coordinates (X, Y, Z), colour components (R, G, B), scalar values, and normal vectors (Nx, Ny, Nz). The objective is to filter these lists to extract elements where the scalar value equals 3, a condition specified by the variable "Tipo."

The next step is to identify each element by Instance Segmentation of 3D points using HDBSCAN and outputting them as Dynamo Point objects. It imports necessary libraries, combines the input X, Y, and Z coordinates into a NumPy array, and configures HDBSCAN with specified parameters (min\_cluster\_size and min\_samples). After performing Instance Segmentation, it filters points for each cluster label and creates Dynamo Point objects for clusters meeting the size criteria. The resulting clusters are output as a list for further use in Dynamo.

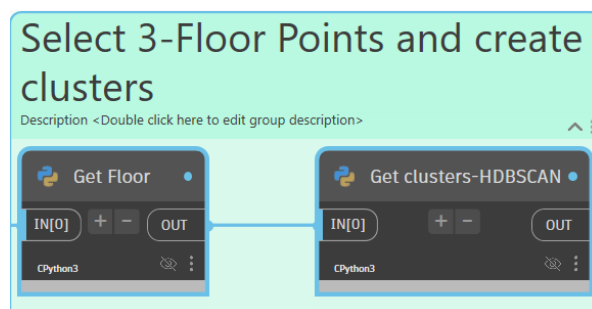


Figure 41. Floor Point filtering and clustering.

The next step is to create the Levels referred to the Floors. It starts by finding the best fit plane through given points using Plane.ByBestFitThroughPoints. The plane's origin Z coordinate is extracted and

used to determine the elevation. The elevation data is then indexed and labelled with floor level names using a code block that formats strings. Finally, these elevations are converted into Dynamo elevation objects with specific names using Level.ByElevationAndName.

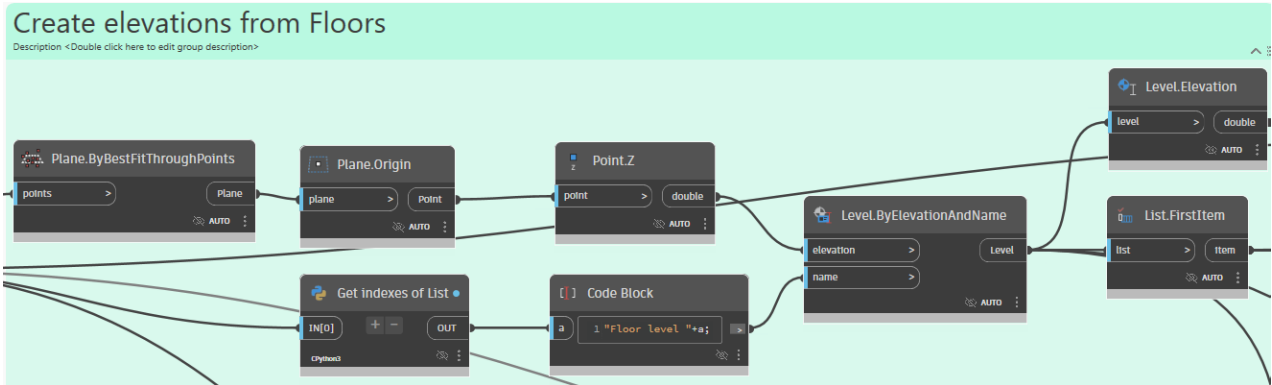


Figure 42. Levels creation from Floors.

After this, the script generates floor outlines using Alpha Shapes. It starts by calculating the alpha shapes from input data, which defines the floor outline as a polycurve. The points of this polycurve are extracted and simplified using the Ramer-Douglas-Peucker algorithm, which reduces the number of points based on a tolerance value specified as 0.1. The simplified points are then used to create a polygon, which is subsequently exploded into its constituent outline Curves.

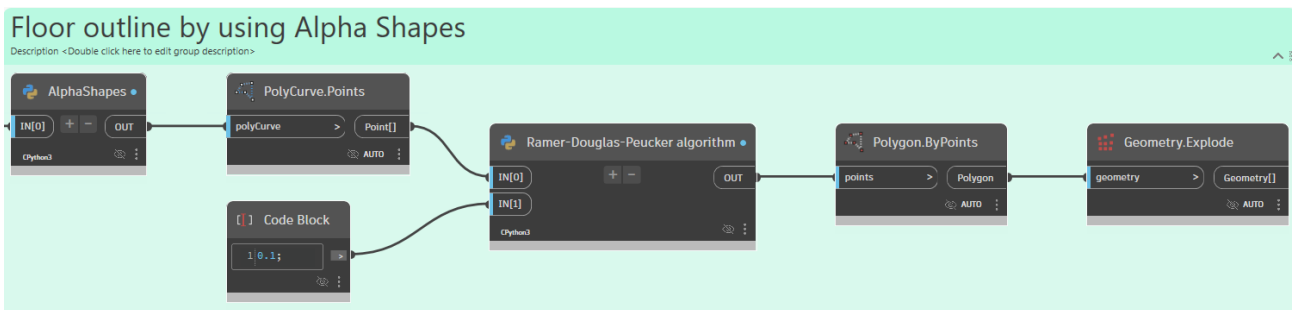


Figure 43. Floor outline by Alpha Shapes.

The next step is for the script to create floor elements by defining floor outlines, types, and levels. The floor outline curves are connected to the Floor.ByOutlineTypeAndLevel node, which takes these curves along with a specified floor type and level to generate the floor elements. The floor type is set to a random type in the Floor Types node, and this information, along with the outline curves and the previously calculated levels, is used to create the floor elements in the model.

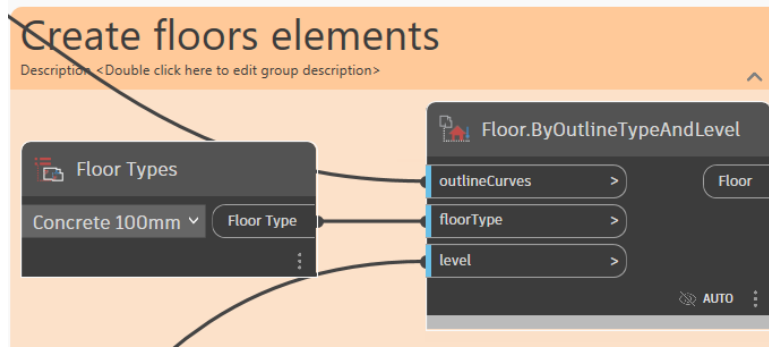


Figure 44. Floor elements creation.

After the creation of the elements, the next step is to modify the base offset for floor elements. It starts by retrieving the elevation level using the Level.Elevation node. The offset value is calculated by subtracting the lowest elevation in the points of each floor from the elevation. The parameter name for the offset is set using a string node labelled Offset di altezza da livello. Finally, the Element.SetParameterByName node applies the calculated offset value to the floor elements, adjusting their base elevation accordingly.



Figure 45. Add Base Offset to Floor.

## 6.2.2 Columns

For the columns, the first step is to identify and cluster each column points using the DBSCAN clustering algorithm. The Get Pillars node retrieves the points that have a Scalar Value = 1, which are then clustered using the DBSCAN algorithm. The List.Count node counts the number of points in each cluster. A condition to filter clusters with fewer than 10 points is set. The List.FilterByBoolMask node applies this condition to filter the clusters, and the List.Clean node removes any empty or null items from the list, preserving the indices.

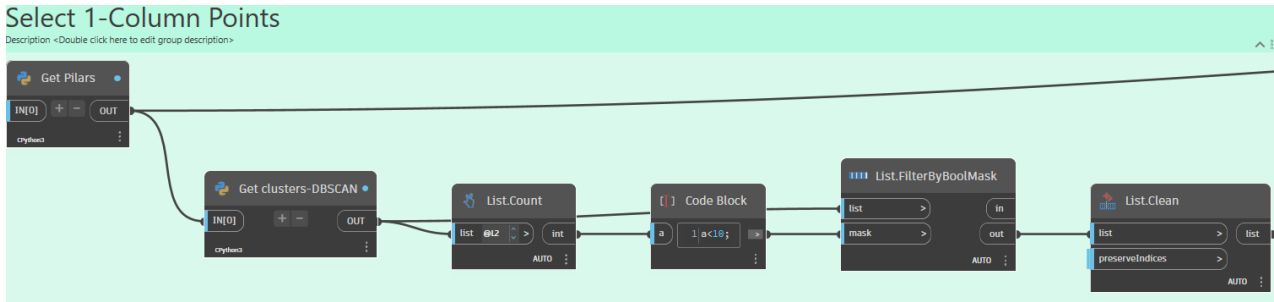


Figure 46. Column Point filtering and clustering.

Then the middle line is found by `Line.ByBestFitThroughPoints`, and a default family from Revit is applied to create the column instance.

### 6.2.3 Walls

The third elements to be modelled are then the Walls.



Figure 47. Overall script for Walls - LOD B.

The Dynamo script starts by deleting elements along the Z-axis for both mouldings and walls. It retrieves these points using custom Python nodes: `Get Moulding` for mouldings and `Get Walls` for walls. Each point's Z-axis index is identified using the `Index in List (Z)` set to 9.

The script extracts the Z-coordinate for each point, defining two numeric values, -0.1 and 0.1, to determine if the point's Normal along the Z-axis falls within this range. These points are then filtered using a Boolean mask in `Filter by Bool Mask`.



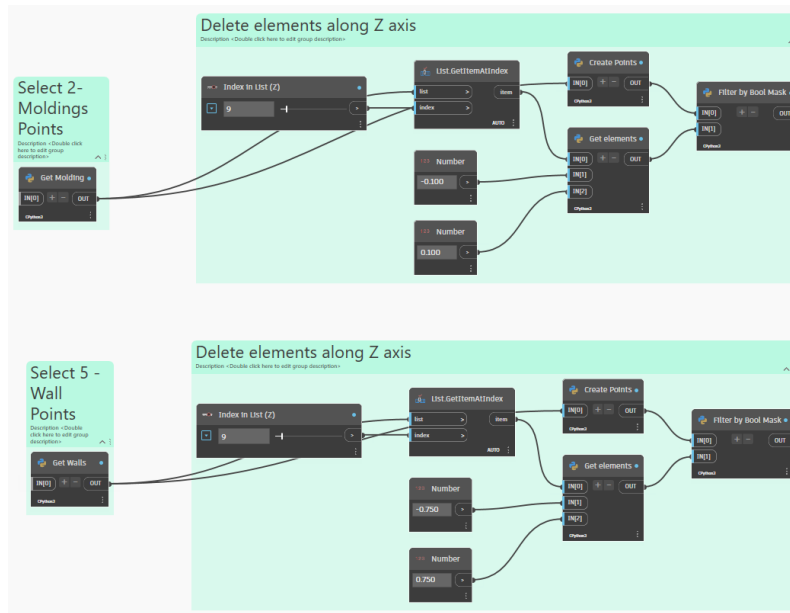


Figure 48. Filtering in Z axis for Wall points.

Next, points are filtered by level and clustered using the HDBSCAN algorithm. Elevation levels are obtained with the Level.Elevation node and sorted using List.Sort. These sorted levels and points are fed into a custom Python node (Point Filter by Level) to filter points based on their elevation levels. Duplicate points are removed using Point.PruneDuplicates with a tolerance of 0.05, and unwanted points are deleted using another custom Python node (Delete unwanted points). The cleaned points are clustered using HDBScan through the Get clusters-HDBSCAN python node.

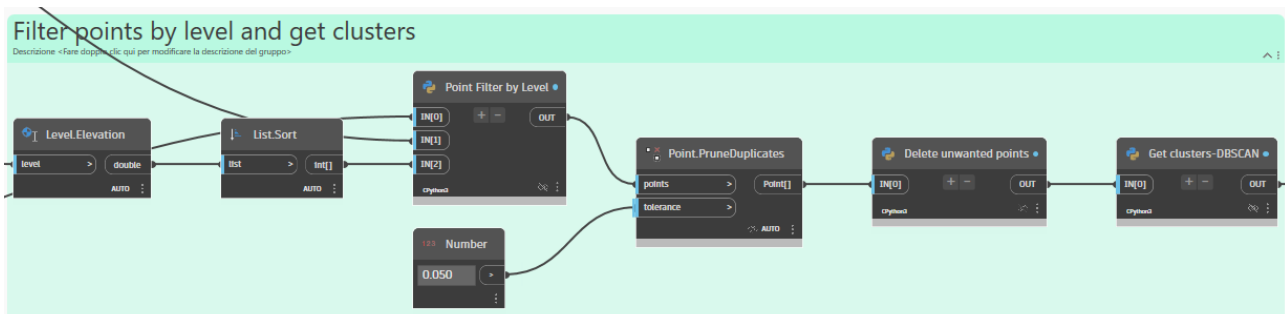


Figure 49. Level filtering and clustering for Wall points.

The script also removes horizontal planes from a set of points. Using the Get Inliers-RANSAC custom Python node, it retrieves inlier points, creating planes with the Plane.ByBestFitThroughPoints node. The normal vector of each plane is extracted using Plane.Normal, and the Z-component is compared to a threshold of 0.1 to identify horizontal planes. Comparisons check if the Z-component is between -0.1 and 0.1, combined with an And node to create a Boolean mask that identifies horizontal planes.

The List.FilterByBoolMask node filters out these planes, and List.Clean removes empty entries while preserving indices.

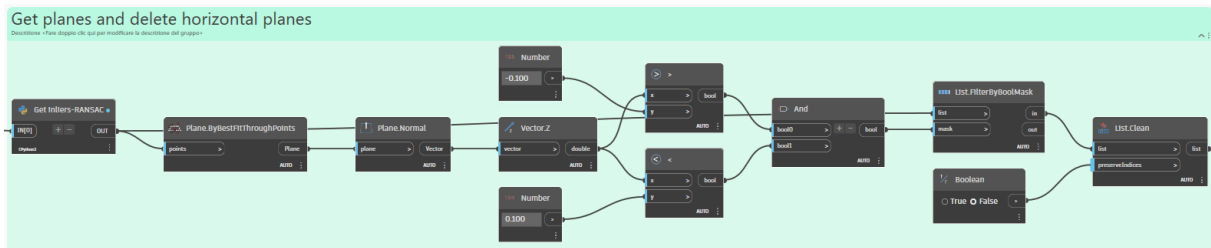


Figure 50. Planes calculation and filtering for Wall points.

The List.Count node counts elements in each list. A threshold of 10 is set, and the script compares the count of each list to this threshold using a greater-than comparison node. The result is a Boolean mask indicating lists with 10 or more points. The List.FilterByBoolMask node uses this mask to filter the lists.

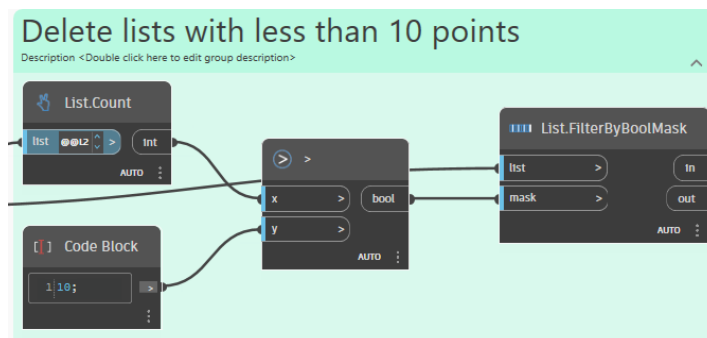


Figure 51. Lists filtering for Wall points.

For clustering points, the script sets DBSCAN algorithm parameters with an epsilon (Eps) of 1 and a minimum of 150 points per cluster, executed via a custom Python node (Get clusters-DBSCAN). Clusters are processed to remove empty lists using List.Clean, preserving original indices, and flattened into a single list with List.Flatten.

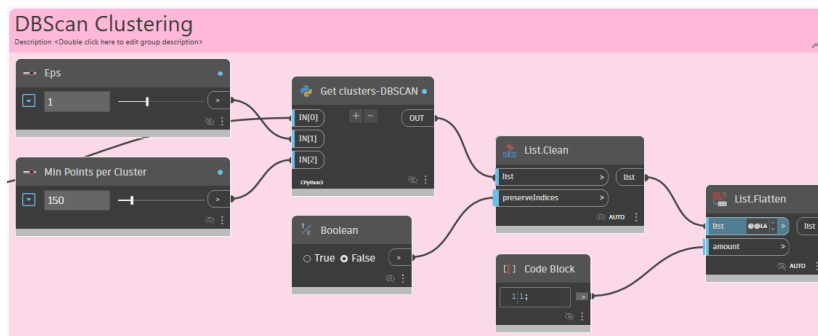


Figure 52. DBScan Clustering for Wall points.

In another instance, the script joins planes by filtering based on a minimum number of points. The Join planes custom Python node aggregates plane data, and List.Count counts points in each plane. A comparison node checks if the count is greater than 100. The resulting Boolean mask is used by List.FilterByBoolMask to filter planes.

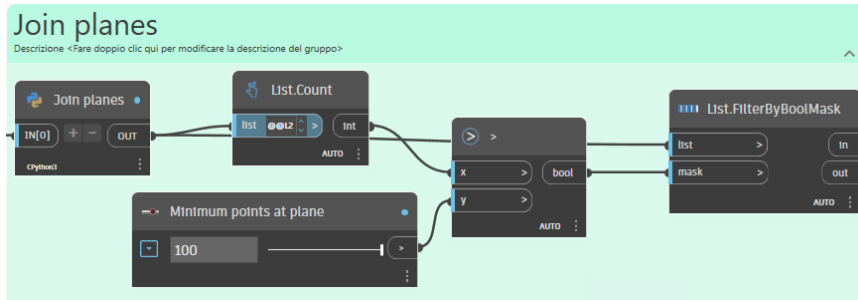


Figure 53. Join planes of Wall points.

the script performs DBSCAN clustering on points with parameters of Eps 5 and a minimum of 150 points per cluster, fed into Get clusters-DBSCAN. Clusters are cleaned with List.Clean and flattened with List.Flatten.

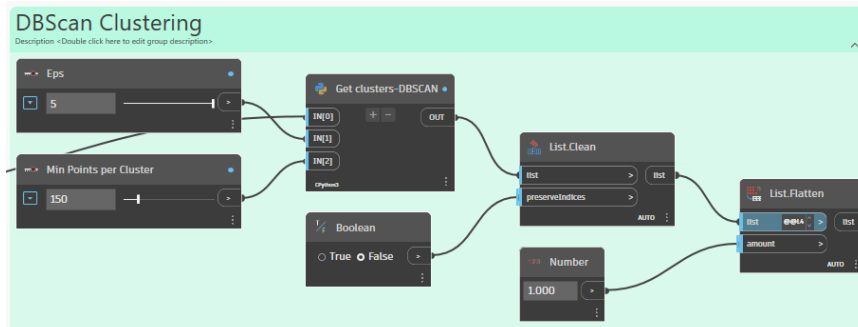


Figure 54. Third clustering for Wall points.

For the second plane detection method, the union of points from Walls and Mouldings categories begins with two List.Transpose nodes, which reorganize the rows and columns of the input nested lists, by then resulting in lists with the values of the parameters for each point. These transposed lists are then fed into a List Create node, which joins the individual lists into a single, unified list. This list is subsequently processed by a List.Flatten node, which reduces the nested lists to a single level, simplifying the data structure for clustering. Finally, the Create Clusters node, powered by a Python script, takes the lists, gets the X, Y, Z, Nx, Ny, Nz parameters from each point and use them as input for the HDBScan algorithm.

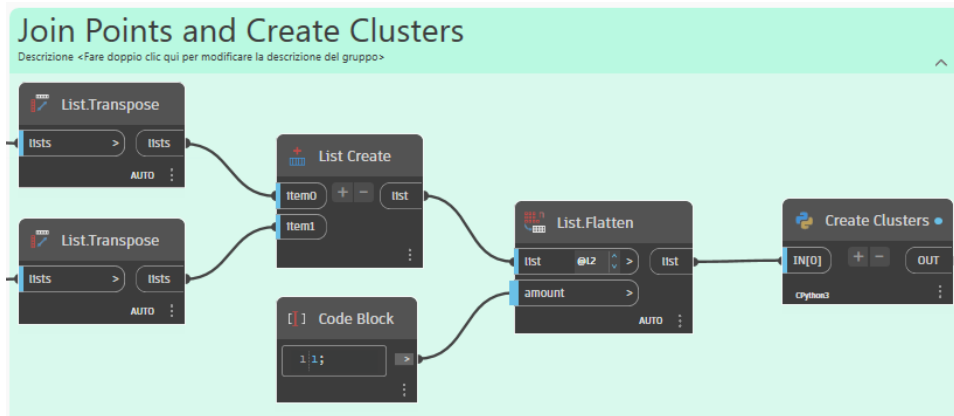


Figure 55. Creation of cluster by a 6D HDBScan algorithm.

To identify the centroid of the largest surface, List.DropItems skips smaller surfaces. Remaining items are flattened using List.Flatten. Surface.PointAtParameter calculates the centre point with U and V parameters set to 0.5.

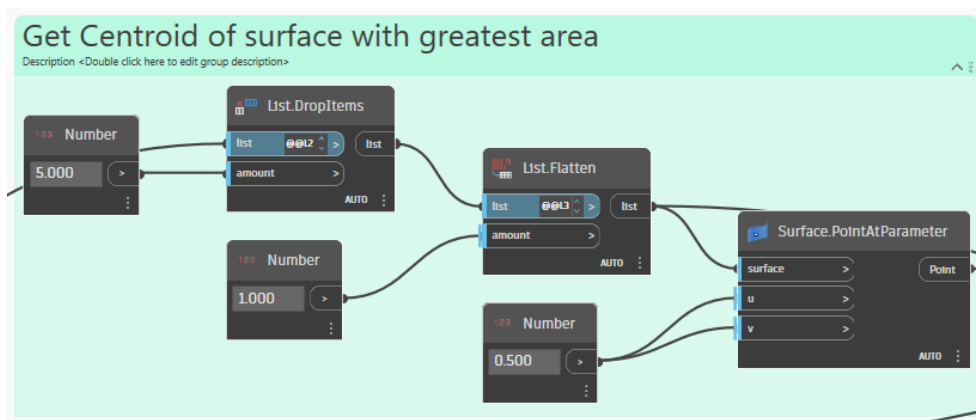


Figure 56. Centroid calculation for Wall faces.

The centroid of a wall surface is calculated by dropping specified items with List.DropItems and transposing the remaining items with List.Transpose. The first and last items are extracted using List.FirstItem and List.LastItem. Surface.PointAtParameter calculates midpoints, and Line.ByStartPointEndPoint creates a line between these midpoints. Curve.PointAtParameter finds the midpoint of this line.

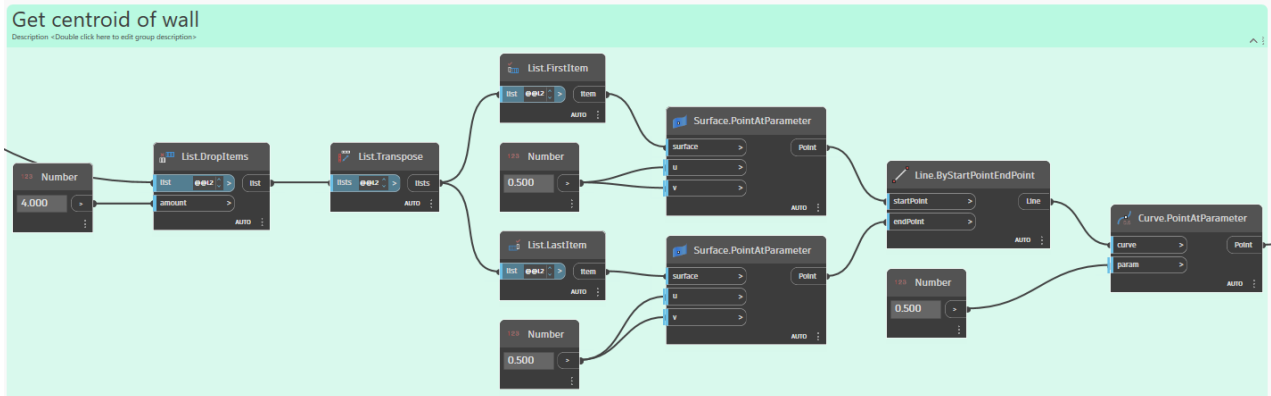


Figure 57. Centroid calculation for Wall points.

For calculating the average distance between two sets of points, the script defines a vector between two points using `Vector.ByTwoPoints`. This vector is used to translate the geometry with `Geometry.Translate`. The `Geometry.DistanceTo` node computes distances between the translated geometry and another set of points, which are averaged using the `Math.Average` node.

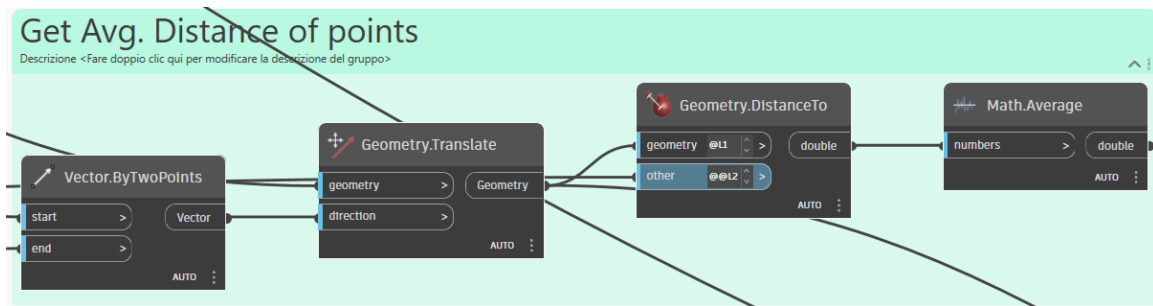


Figure 58. Average distance between Surface and Wall points.

In extracting wall lines, the script defines minimum and maximum wall thicknesses to filter relevant wall curves. The `Surface.PerimeterCurves` node retrieves perimeter curves of surfaces. Using `List.GetItemAtIndex`, a specific curve is selected, flattened with `List.Flatten`, and translated using a Python script node and conditional checks based on wall thickness. The `Curve.PointAtParameter` node identifies a point on the curve, and `Plane.ByOriginNormal` creates a plane at this point with a normal vector. The `Curve.PullOntoPlane` node projects the curve onto this plane.

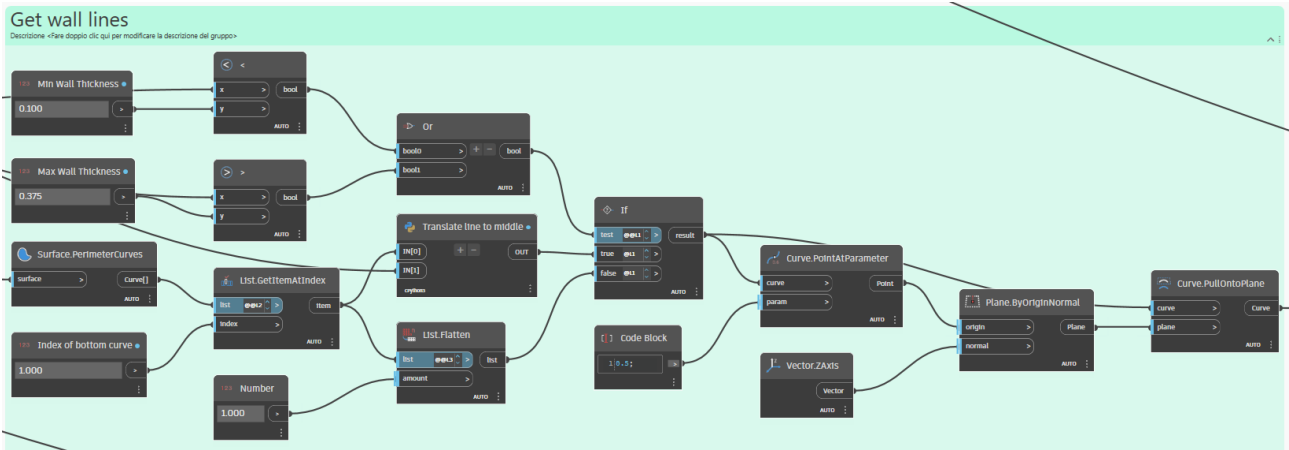


Figure 59. Middle line for Walls.

Wall height is calculated by finding the difference between the highest and lowest Z-coordinates using Point.Z. The highest and lowest Z-values are determined with List.MaximumItem and List.MinimumItem, respectively, and subtracted to find the height, rounded to two decimal places using Math.Round.

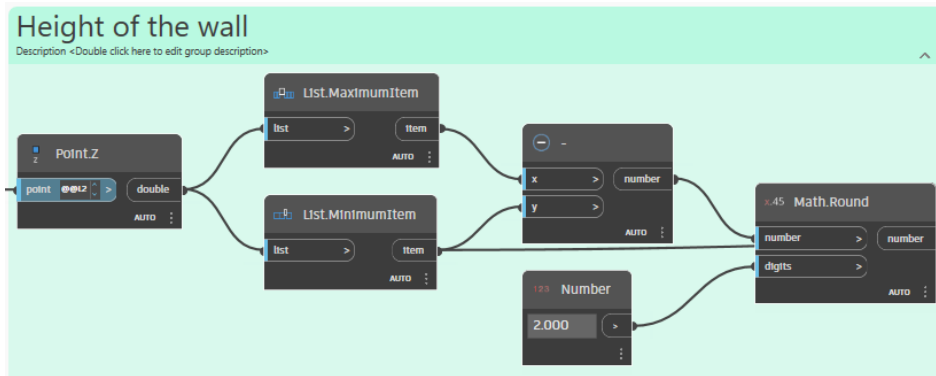


Figure 60. Wall height calculation.

To calculate the width of a wall and convert the measurement from meters to centimetres, the script processes a numeric input through a multiplication node and converts the unit using Convert by Units. The final wall width is rounded using Math.Round.

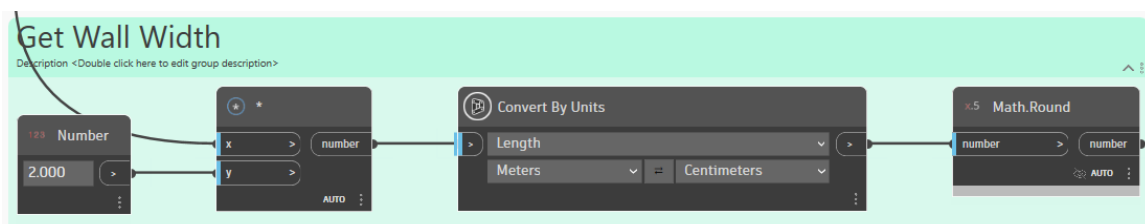


Figure 61. Wall width calculation.

A string describing the wall type with its measurement in centimetres is created by rounding down a numeric input with Math.Floor, converting to a string, and combining with other strings using Concatenate String (e.g., Wall 200 cm).

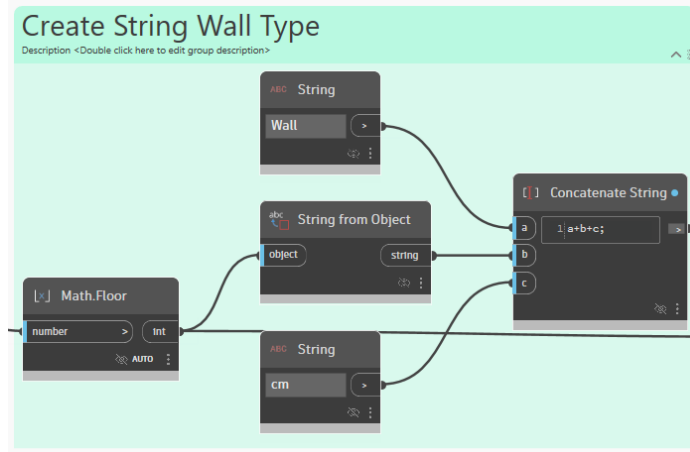


Figure 62. Wall width String concatenation.

For creating new wall types with specified properties, the script selects an existing wall type and defines a new wall name. It converts the wall thickness using Convert by Units and generates a new wall family type with Create Compound System Family Type. The FamilyType.SetCompoundLayerWidth node sets the width of compound layers, ensuring the desired wall structure and dimensions. The List.Flatten node prepares the list for further use.

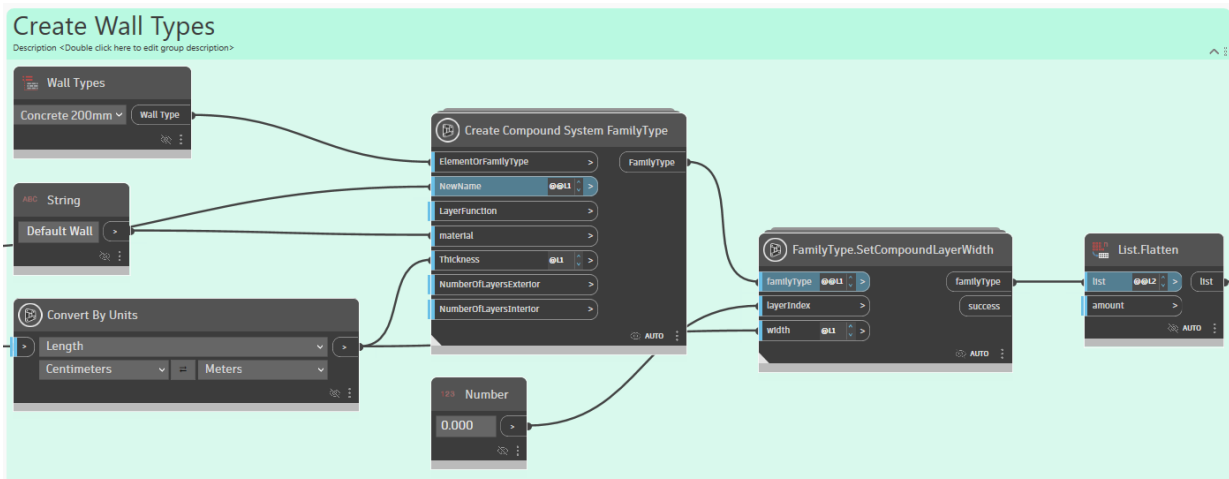


Figure 63. Wall family types creation.

The final node to create the wall elements.

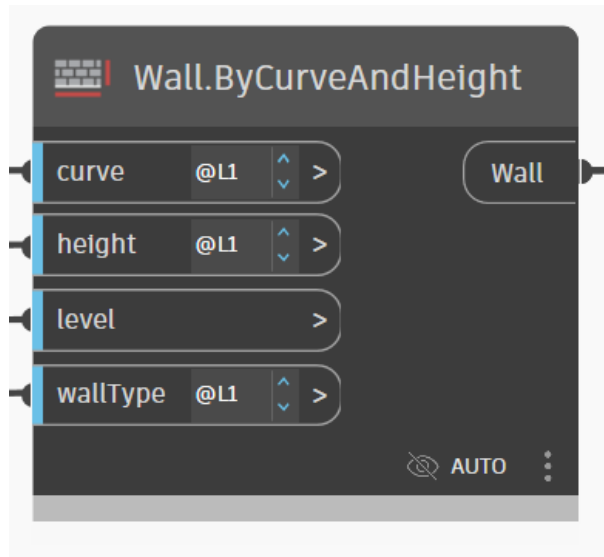


Figure 64. Wall creation by Curve and Height.

Finally, the script adjusts the base offset of a wall element by retrieving all parameters with `Element.Parameters`, selecting the base offset parameter with `List.GetItemAtIndex`, and converting to a string with `Parameter.Name`. `Level.Elevation` provides the level's elevation, used to calculate the new base offset. `Element.SetParameterByName` updates the wall's base offset parameter.

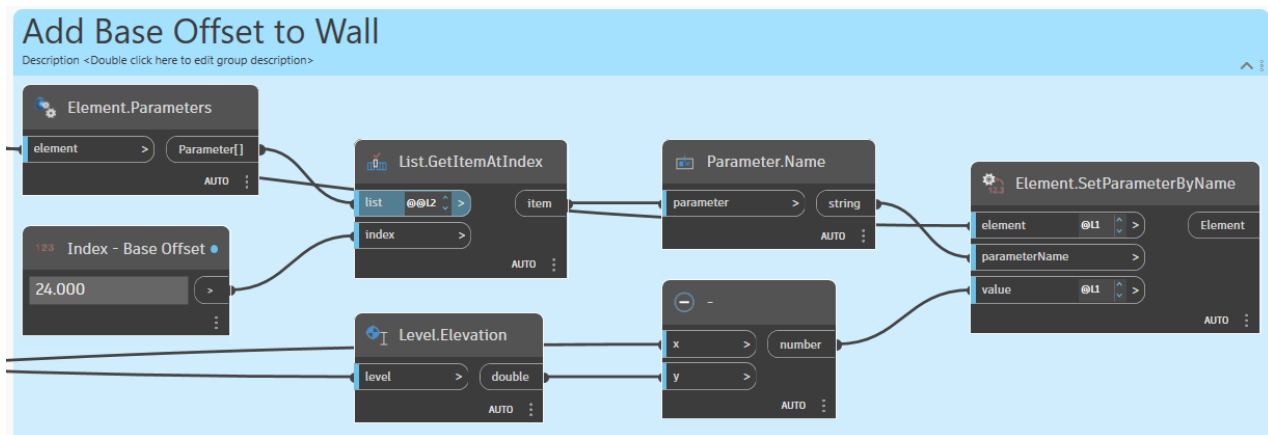


Figure 65. Base offset of Wall modification.

#### 6.2.4 Roofs and Ceilings

The fifth script is the script for Roofs and Ceilings.



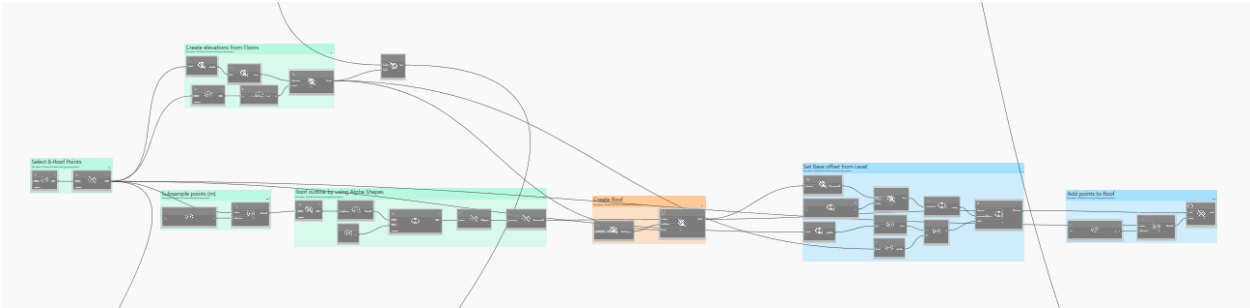


Figure 66. Overall script for Roofs and Ceilings - LOD B.

It begins by using a custom Python script node, Get Roof, to extract roof-related point data. These points are then passed to another Python script node Get clusters-HDBSCAN, which applies the HDBSCAN clustering algorithm to group the points into clusters.

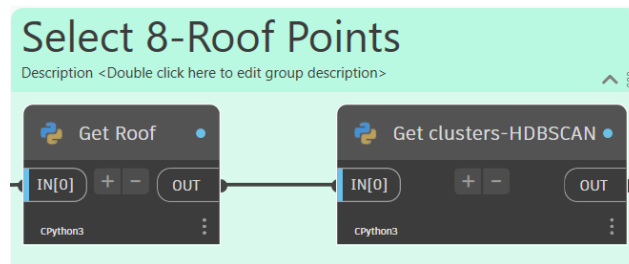


Figure 67. Roof Point filtering and clustering.

This Dynamo script subsamples a set of points to reduce duplicates based on a specified proximity tolerance. The Min Proximity (m) node sets a tolerance value, in this case, 0.2 meters. This tolerance value is then used by the Point.PruneDuplicates node, which removes duplicate points that fall within the specified proximity.

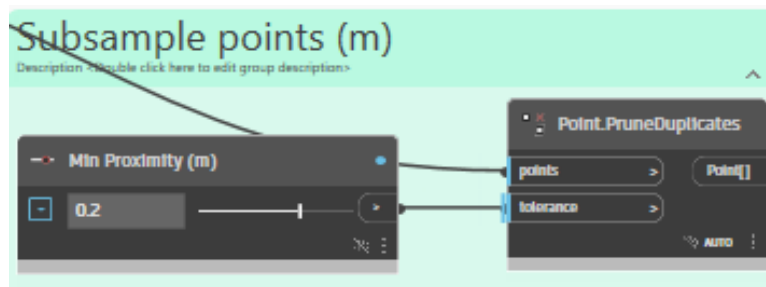


Figure 68. Roof points subsample node.

This Dynamo script generates a roof outline using Alpha Shapes. It begins with the Alpha Shapes node to create a polycurve representing the roof outline. The points of this polycurve are extracted using PolyCurve.Points. These points are then simplified using the Ramer-Douglas-Peucker algorithm, which reduces the number of points based on a specified tolerance of 0.1. The simplified

points are used to create a polygon with the Polygon.ByPoints node. Finally, the Geometry.Explode node breaks down the polygon into its lines.

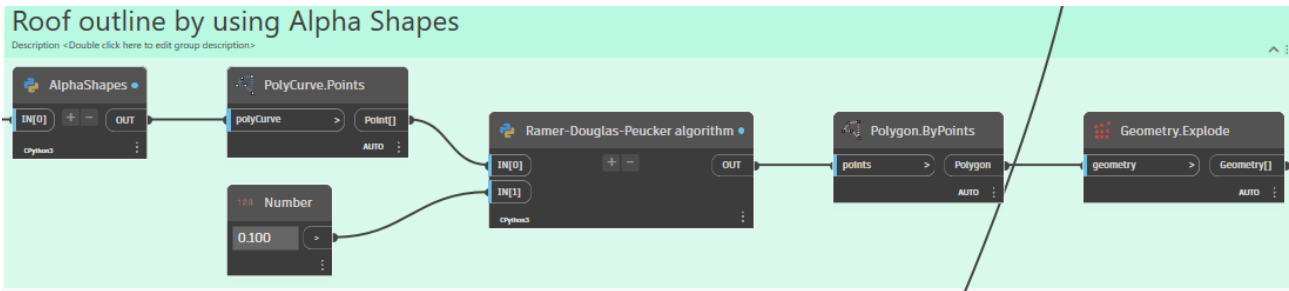


Figure 69. Roof outline by Alpha Shapes.

This Dynamo script creates a roof by defining its outline, type, and level. The Roof Types node specifies the type of roof, in this case, Generic - 225mm. The Roof.ByOutlineTypeAndLevel node then combines the roof outline, selected roof type, and specified level to generate the roof element.

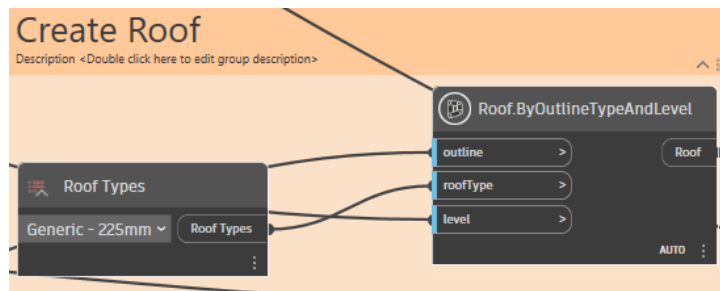


Figure 70. Floor elements creation.

This Dynamo script sets the base offset of elements from a specified level. It starts by retrieving the parameters of the selected elements using the Element.Parameters node. The index of the base offset parameter is specified with the Base Offset Index node. The List.GetItemAtIndex node retrieves the specific parameter name for the base offset. The Z-coordinates of points are obtained using the Point.Z node, and the minimum Z-coordinate is identified with List.MinimumItem. The elevation of the level is retrieved using the Level.Elevation node. A code block calculates the difference between the minimum Z-coordinate and the level elevation to determine the base offset value. Finally, the Element.SetParameterByName node sets the base offset for the elements using the calculated value and the parameter name, updating the elements accordingly.

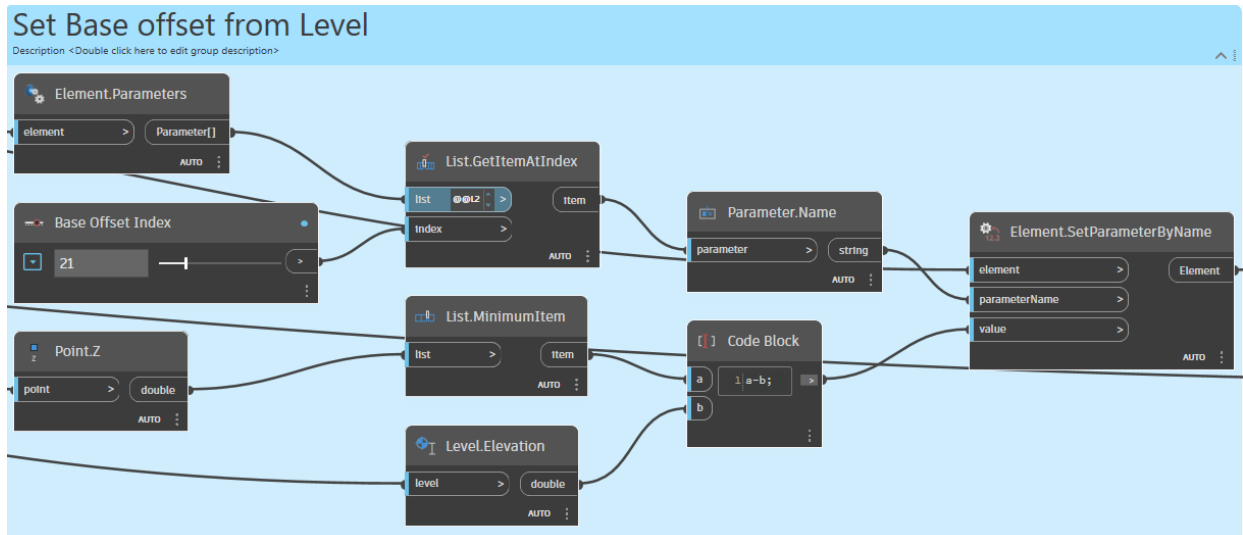


Figure 71 Set Base offset from level to Roofs.

This Dynamo script adds points to a roof while ensuring no duplicate points within a specified proximity. It starts by setting a minimum proximity tolerance of 0.3 meters using the Min Proximity (m) node. The Point.PruneDuplicates node then processes the input points, removing any that are closer to each other than the specified tolerance. Finally, the Roof.AddPoint node takes the pruned list of points and adds them to the roof element, ensuring that the points added to the roof are unique and appropriately spaced.

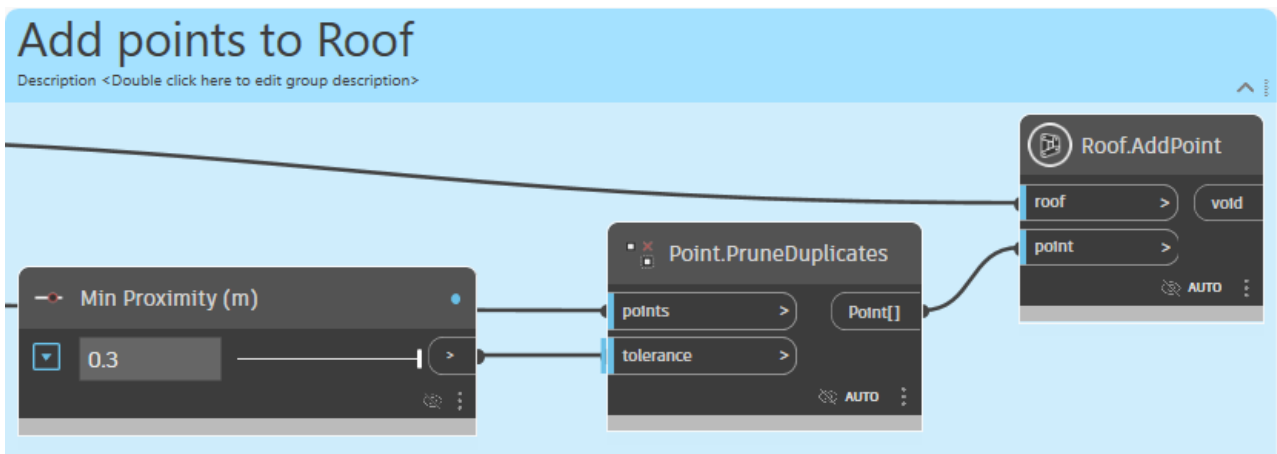


Figure 72. Points addition to the Roof elements.

### 6.2.5 Windows

The sixth script is the script for Windows.

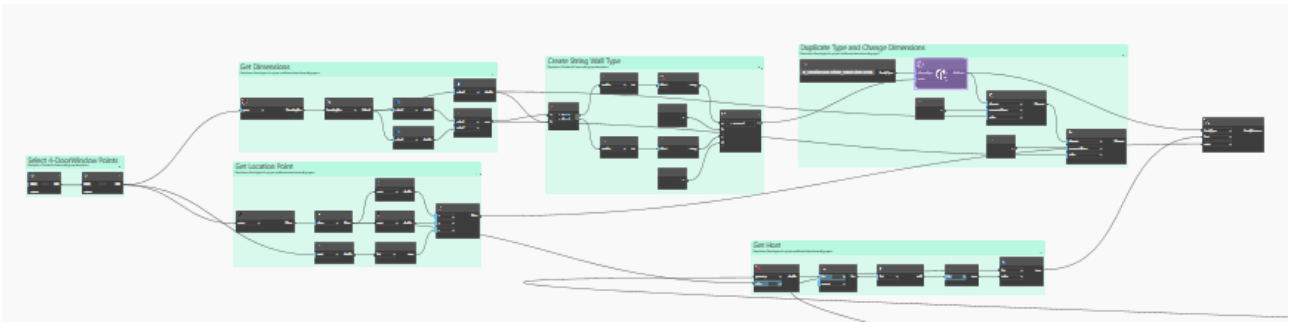


Figure 73. Overall script for Windows - LOD B.

The initial step in this workflow involves two custom Python nodes labelled as Get Windows and Get Clusters-Point. The Get Windows node is likely responsible for identifying and selecting window elements within the design model. The output from this node is then passed to the Get Clusters-Point node, which appears to cluster these window points, potentially for subsequent analysis or operations.

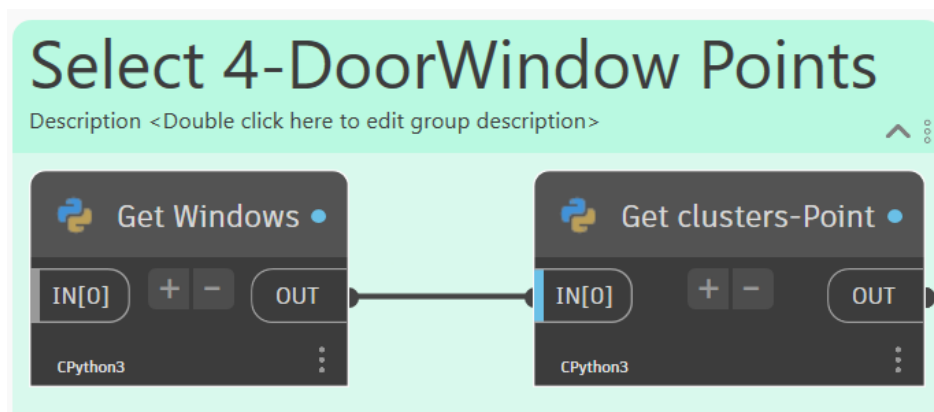


Figure 74. Windows Point filtering and clustering.

The second step focuses on determining a specific location point based on a set of input points. The `Plane.ByBestFitThroughPoints` node calculates a plane that best fits the given points, followed by extracting the plane's origin using the `Plane.Origin` node. The coordinates (x, y, z) of this origin point are retrieved through the `Point.X`, `Point.Y`, and `Point.Z` nodes. The z-coordinates of the points are evaluated using the `List.MinimumItem` node to find the minimum z-value, ensuring the point is aligned with the lowest elevation. Finally, the `Point.ByCoordinates` node constructs the location point using these coordinates.

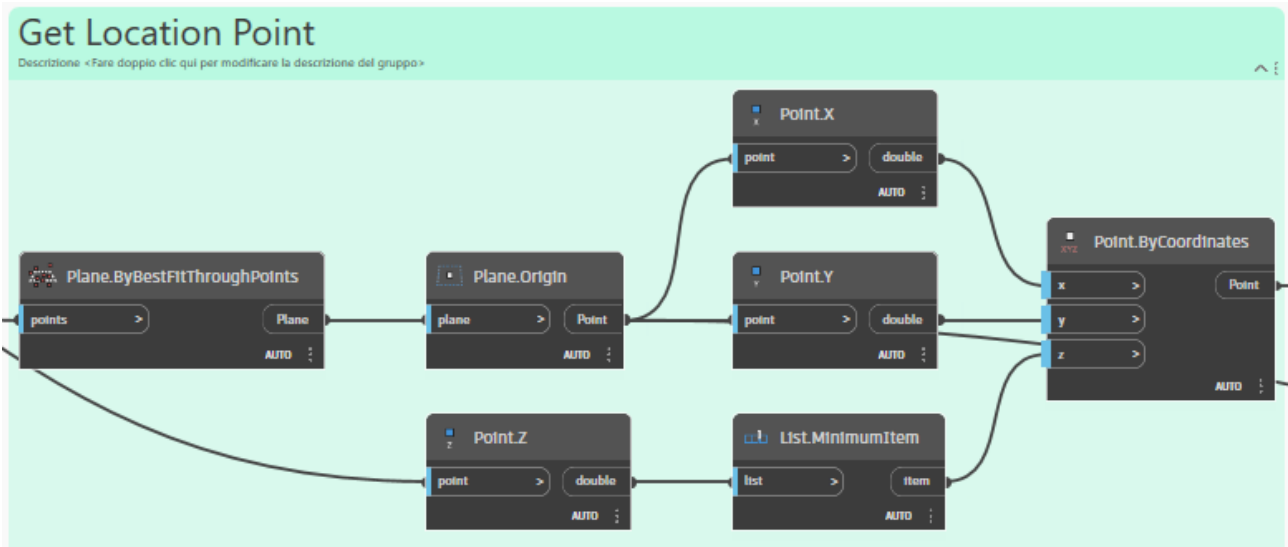


Figure 75. Windows element location point calculation.

In the third step, the script calculates the dimensions of the window element. Starting with the `BoundingBox.ByMinimumVolume` node, the smallest bounding box encapsulating the geometry is created. This bounding box is converted into a cuboid using the `BoundingBox.ToCuboid` node. The width, length, and height of the cuboid are extracted via the `Cuboid.Width`, `Cuboid.Length`, and `Cuboid.Height` nodes, respectively. The first two dimensions are then compared using the `Math.Max` node to determine the largest dimension of the cuboid.

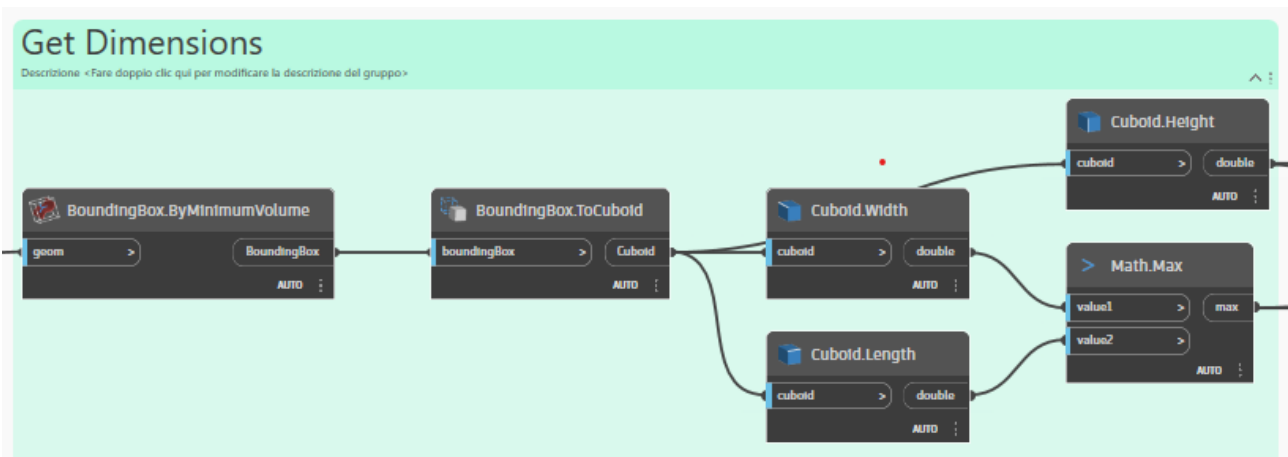


Figure 76. Windows element dimension calculation.

In the fourth step, the script generates a string representation for the window type. The previously calculated measures are converted to centimetres from metres. These variables are processed by `Math.Floor` nodes, ensuring they are integers. The `String from Object` nodes convert these integer

values into strings, which are then combined with other string elements using a Concatenate String node. This creates a descriptive string that specifies the window type, integrating numerical and textual data to generate a comprehensive label.

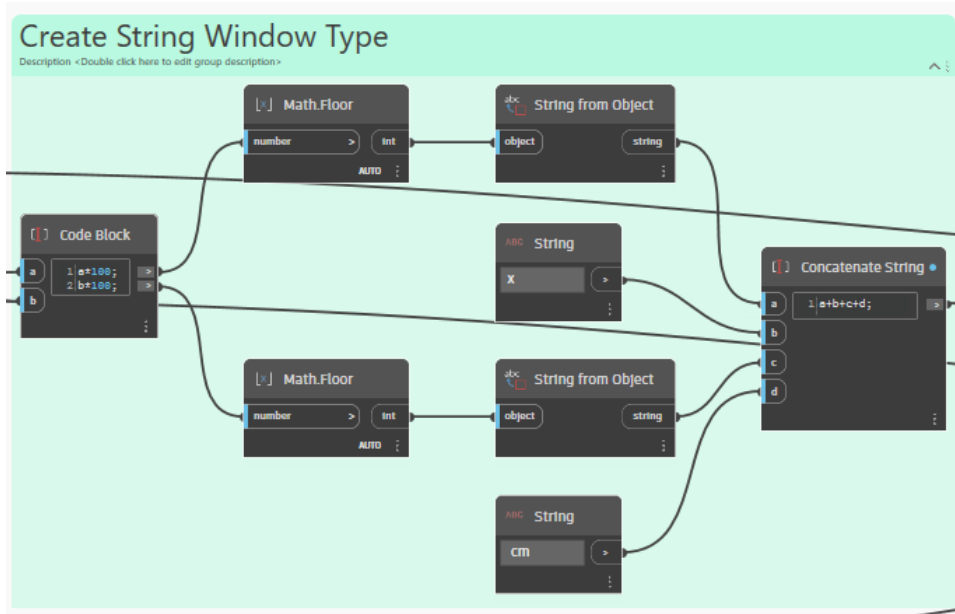


Figure 77. Windows element Window Type Name String creation.

The fifth step involves duplicating an element type and modifying its dimensions. The Family Types node selects a specific family type from the model. This type is duplicated using the FamilyType.Duplicate node. The new type's parameters are adjusted via Element.SetParameterByName nodes, where the String nodes provide the parameter names (e.g., "Altezza" for height and "Larghezza" for width). The Element.SetParameterByName nodes update these parameters with new values, effectively altering the dimensions of the duplicated element.

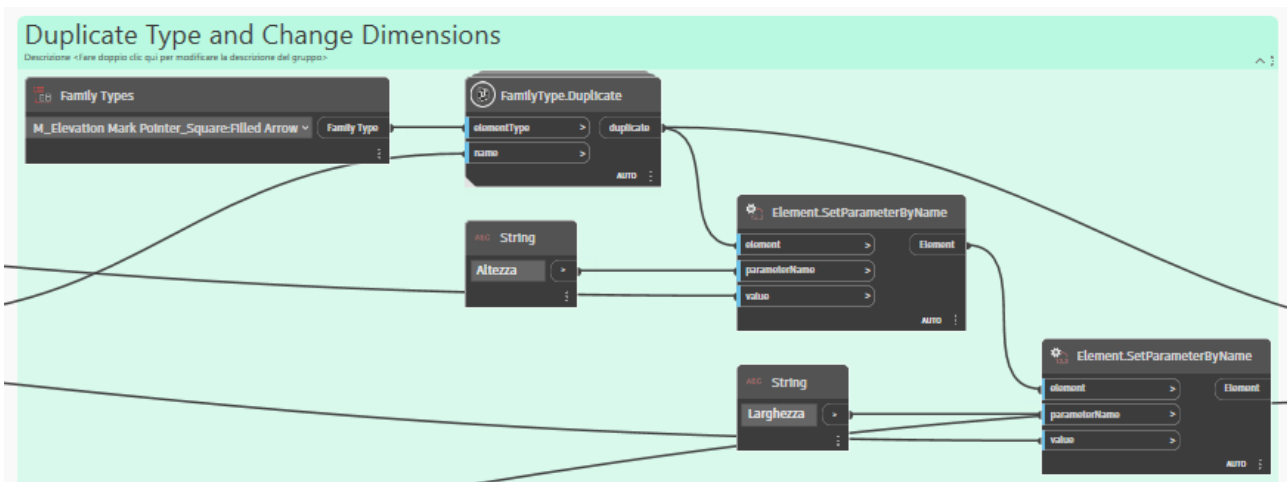


Figure 78. Windows element Window Type creation.

The sixth step identifies the host element for a given geometry. The Geometry.DistanceTo node calculates the distance between the geometry and other elements. The resulting list of distances is flattened using the List.Flatten node. The List.SortIndexByValue node sorts these distances, and the List.FirstItem node selects the nearest element. Finally, the List.GetItemAtIndex node retrieves the corresponding host element based on its index.

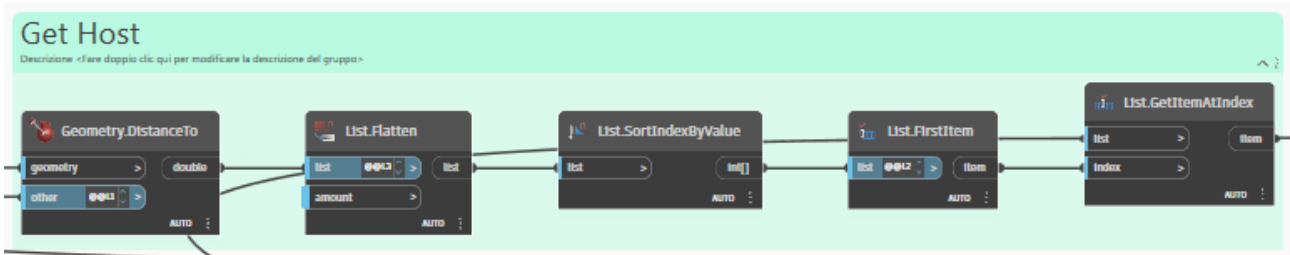


Figure 79. Windows element Hosting element calculation.

Finally, the elements are placed by its previously defined Family Type, Host element and Location Point.

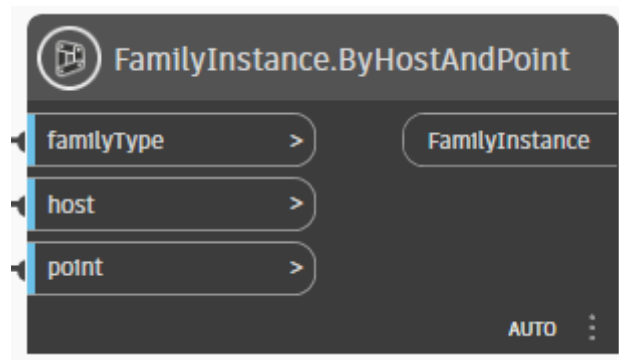


Figure 80. Window elements creation.

### 6.2.6 Vaults

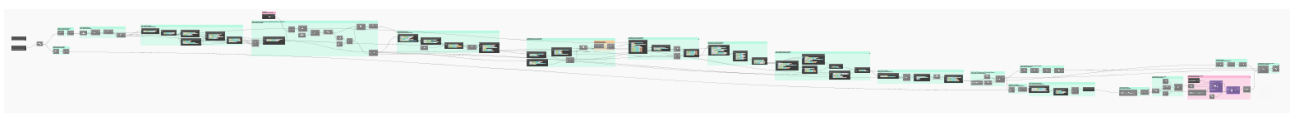


Figure 81. Overall script for Vaults.

The Vault creation procedure starts with the Get Vaults and Create Points nodes to select and create points for vault structures. A Boolean toggle determines whether to preserve indices during processing. The script filters the vault points using the Delete unwanted points node, which removes

unnecessary points. Finally, the Get clusters-DBSCAN node clusters the remaining points using the DBSCAN algorithm, organizing them for further analysis or operation.

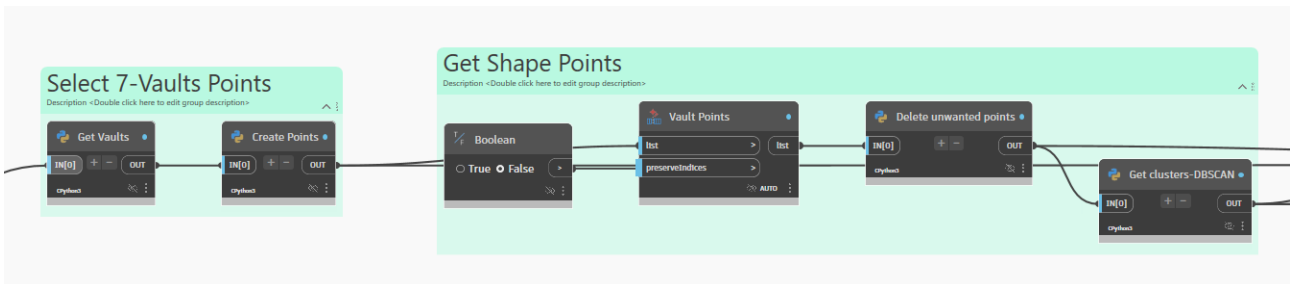


Figure 82. Filter Vault Points and Get Clusters.

The next section of the Dynamo script focuses on extracting and processing vault profiles, continuing from the previous step. It begins by identifying the outer surfaces of the vaults using custom Python scripts. Then, it selects the two smallest surfaces from these outer surfaces. Next, it extracts curve points from the surfaces to identify the Barrel Vault profiles. These points are pre-processed to align and filter them appropriately.



Figure 83. Calculate profiles of main Barrel Vault

In the third step, the Dynamo script filters point lists based on their distance from the Barrel Vault surface and the number of points in each list. The workflow begins with adjusting the distance parameter. Points are then compared to this distance using a custom Python script to create a surface and filter points accordingly. The filtered points are cleaned to remove duplicates or unwanted points and then clustered using the DBSCAN algorithm. Lists are further filtered by ensuring they meet a minimum count threshold before being processed into clusters.



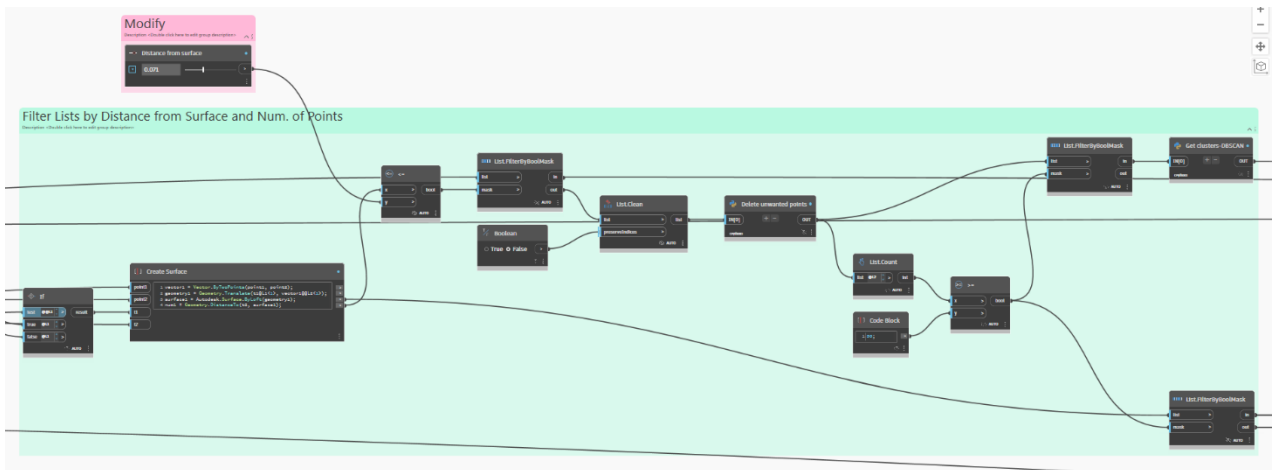


Figure 84. Get points of Cross Vault.

In the fourth step, the Dynamo script focuses on extracting diagonal arcs from vault profiles. It begins by identifying the lateral faces of the vault using a custom Python script “Get Lateral Faces”. These faces are processed to extract edge curves. Later, further refine and filter these curves based on their lengths is realized, ensuring only significant curves are considered. The script then creates diagonal surfaces using the filtered curves. Finally, the Python script “Get Diagonal Arcs” is employed to identify and generate diagonal arcs from these surfaces.

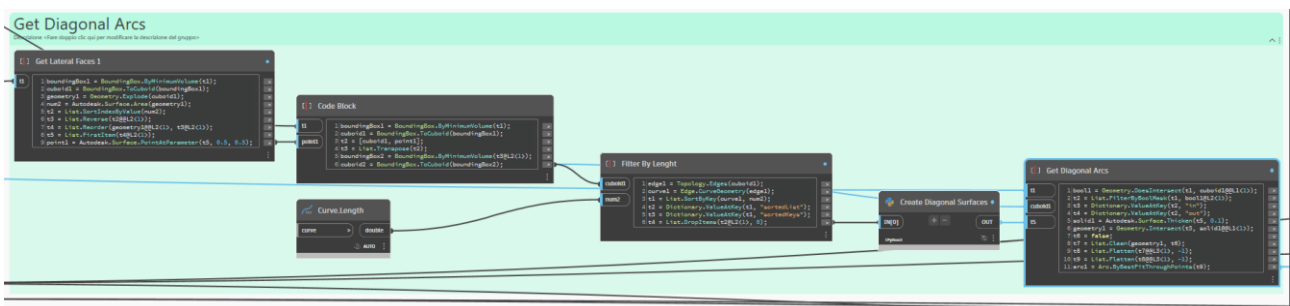


Figure 85. Get Diagonal Arcs of Cross Vault.

In the fifth step, the Dynamo script is designed to create the arcs of a cross vault. It starts by joining and flattening lists of points that represent the edges of the vault. The script then calculates the minimum height of each cluster of point. Using the gathered data, it identifies the middle point of the cross vault through a custom Python script “Points to Min Height”. These points are then used to form a polygon, which is subsequently exploded into individual geometries. Finally, these geometries are used to create surfaces that define the cross-vault structure.

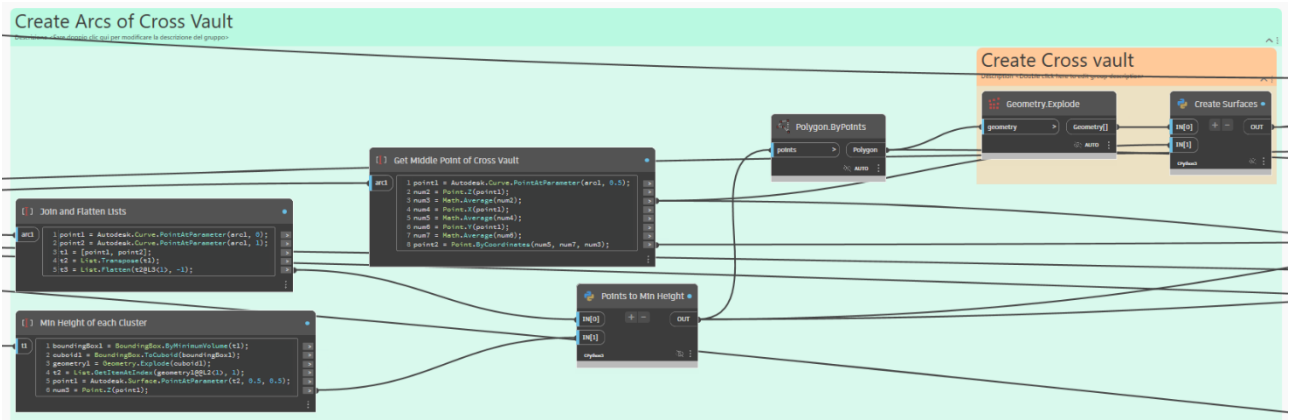


Figure 86. Get Perimetral Arcs of Cross Vault.

In the sixth step, the Dynamo script is focused on creating arcs for extrusion in the vault modelling process. It begins with a code block that defines the process of fitting lines through points and checking for intersections. This block also filters and organizes the points to ensure the correct geometry is used for the arcs. The script then calculates the height of the arc from the barrel vault using a Python script, which involves determining the average height of intersecting points. Two types of arcs are generated based on these calculations, which are then joined together and filtered to obtain the Arcs that start on the Barrel Vault and the 2 other missing arcs.

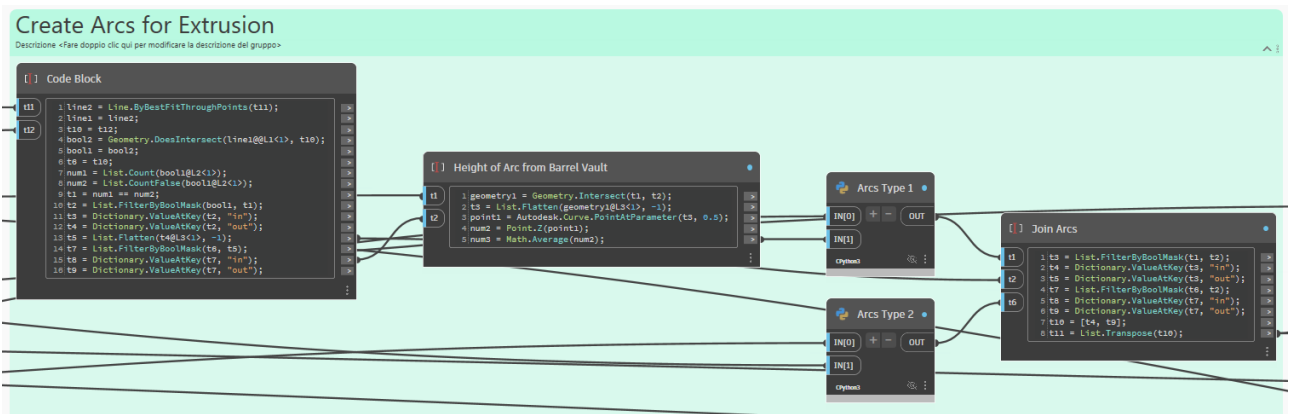


Figure 87. Filter Points.

In the seventh step, the Dynamo script is focused on creating the cross-vault surface. It begins with a code block that generates a new middle arc by identifying key points and using these to create curves. The arc creation process involves using parameters to define points along the curve and then transforming these points into the required geometry. This middle arc is added to the existing geometry. Finally, the script generates the cross-vault surface by sorting and reordering points,

calculating distances, and using these points to define the surface geometry. This results in the formation of the cross-vault surface.

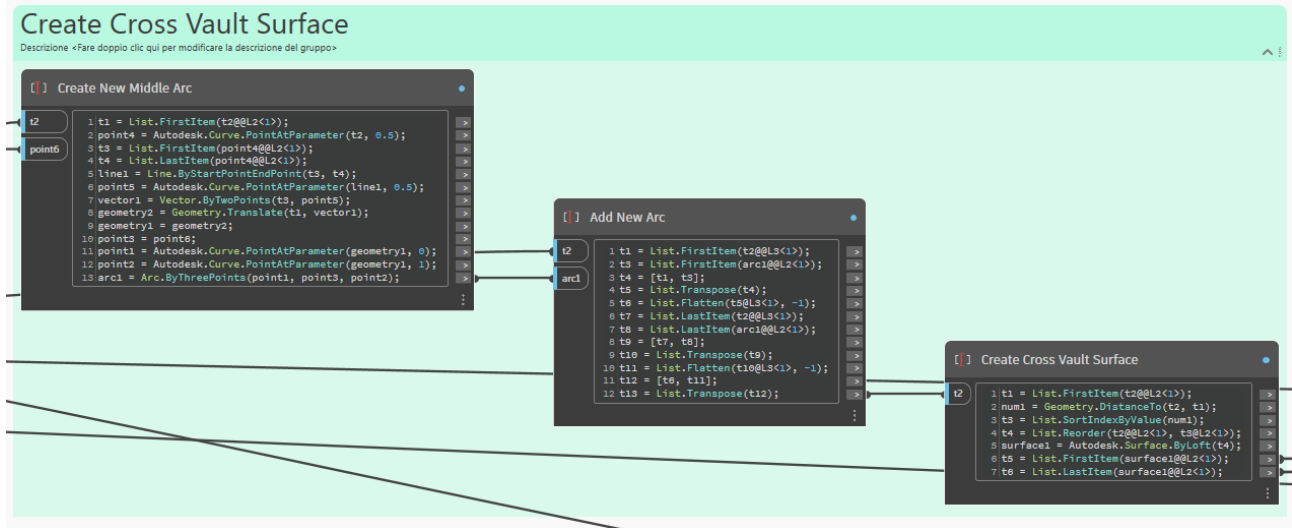


Figure 88. Create Cross Vaults Surfaces.

In the eighth step, the Dynamo script is aimed at generating the general vault surface by combining the surfaces of the cross-vault and the barrel vault. It begins by extracting arcs from the barrel vault using a code block that processes the geometry to define these arcs. Another code block gathers the remaining arcs. These arcs are then filtered by area to ensure only the relevant sections are used. The script joins the lists of these arcs, consolidating them into a comprehensive set. This set is then used to cut the barrel vault surface, incorporating the necessary geometrical transformations. Finally, the cross vault and barrel vault geometries are joined together, resulting in the creation of a unified general vault polysurface.



Figure 89. Cut Barrel Vault and Join to Cross Vault.

In the ninth step, the Dynamo script aims to create a grid of points on a vault surface. It starts with creating the surface by union, where vectors are defined to align the surface in the Z positive direction. An 'If' node then evaluates conditions to ensure the correct surface configuration. Following this, surfaces are joined together using a code block that manages the geometry. The script proceeds to generate a grid of points on these surfaces, ensuring an even distribution. Finally, another code block performs a subsample of the grid of points, which involves flattening the list of points, removing duplicate points, and filtering out those not inside the area of the vault.



Figure 90. Create Grid of Points in the Surface.

In the tenth step, the focus is on obtaining all the roof type thicknesses, translating points, defining the roof outline, and clustering arcs for analysis. Initially, the script retrieves a selected roof type and calculates its thickness. The grid points are then translated accordingly in the Z direction. The script continues by defining the roof outline using Alpha Shapes, which simplifies the boundary points into a manageable polygon and subsequently explodes the geometry into its constituent parts. In parallel, the arcs are clustered using the DBSCAN algorithm, which helps in identifying distinct arcs based on their spatial distribution. The thickness values of these arcs are then calculated using custom Python scripts that extract and compute these values, ensuring they meet specified conditions before flattening the list for further processing.

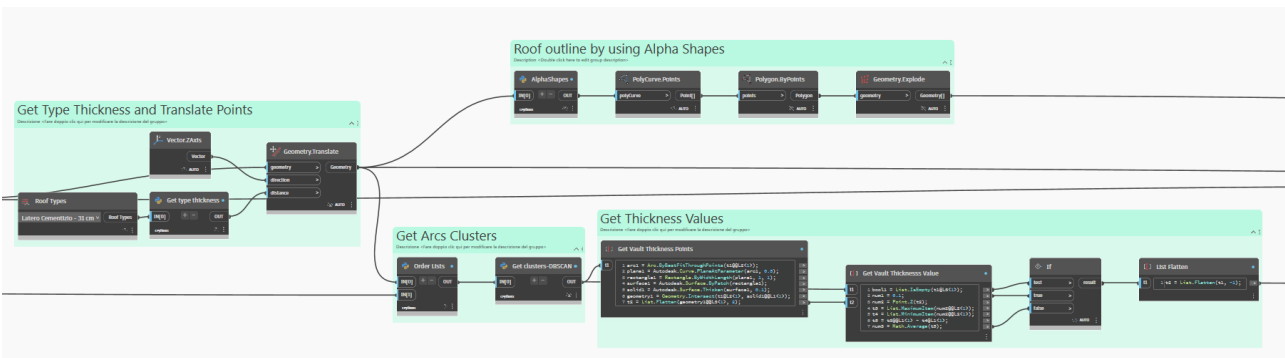


Figure 91. Calculate Thickness of Vault.

In the eleventh step, the Dynamo script focuses on defining vault thickness parameters and creating vault family types. The process begins by determining the vault width through unit conversion and

rounding operations to ensure consistency in measurements. This width is then integrated into a string format to create a specific vault type identifier, which includes concatenating various string elements representing different attributes of the vault, having as result that each Type name has the “Vault X cm” structure, where X is the thickness. Subsequently, the script proceeds to create compound system family types for the vault, using the previously defined attributes and measurements. These parameters include material, layer function, and thickness for each layer.

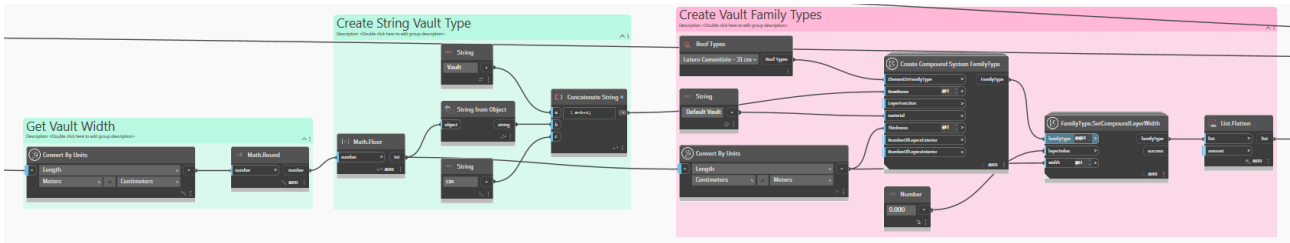


Figure 92. Create new Family Type.

In the last step, the process involves determining the level elevation using the Z-coordinate of each vault point and identifying the minimum item from each cluster. This minimum item is then used to define elevation levels in the Revit environment. Once the levels are established, the script proceeds to create roofs using the specified outline found with Alpha Shapes, each calculated roof type, and the recently created levels. The roof creation is executed through the “Roof.ByOutlineTypeAndLevel” node. Following the roof creation, specific points are added to the roof using the “Roof.AddPoint” node, to align the shape of the roof elements to its real shape.



Figure 93. Create the Roof element and Add Points from Grid.

### 6.2.7 Arch

These points are considered in the vault procedure. Are used to calculate the thickness of the vault element.

## 6.2.8 Stairs

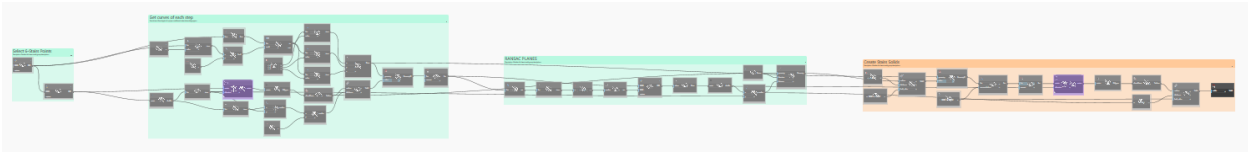


Figure 94. Overall script for Stairs - LOD B.

This Dynamo script identifies and selects points related to stairs for further analysis. The Get Stairs node, a custom Python script, retrieves point data associated with stairs. These points are then input into the Get clusters-HDBSCAN node, which applies the HDBSCAN clustering algorithm to group the points into clusters.

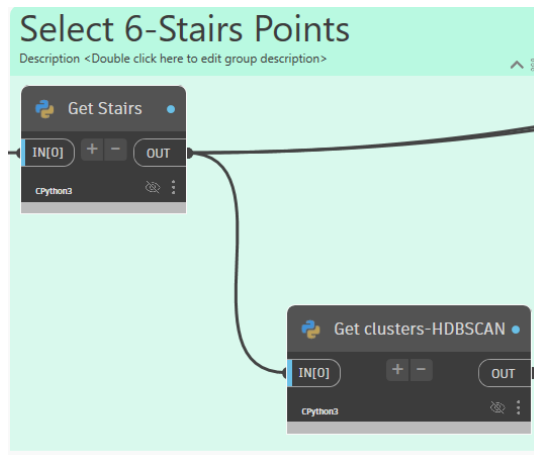


Figure 95. Stair Point filtering and clustering.

The script then extracts the curves of each step in a stairway. It begins by identifying the index of a specific normal vector component (Index of Nz), used to filter the points based on their Z-coordinates. The filtered points are further processed to find minimum and maximum Z-values, helping to determine the elevation of each step.

Convex hulls of the points are calculated to define the boundary of each step, creating polygons using Polygon.ByPoints. These polygons are then converted to surfaces using Surface.ByPatch. The surfaces are thickened to solids with Surface.Thicken, and intersections between these solids and another geometry are computed using Geometry.Intersect.

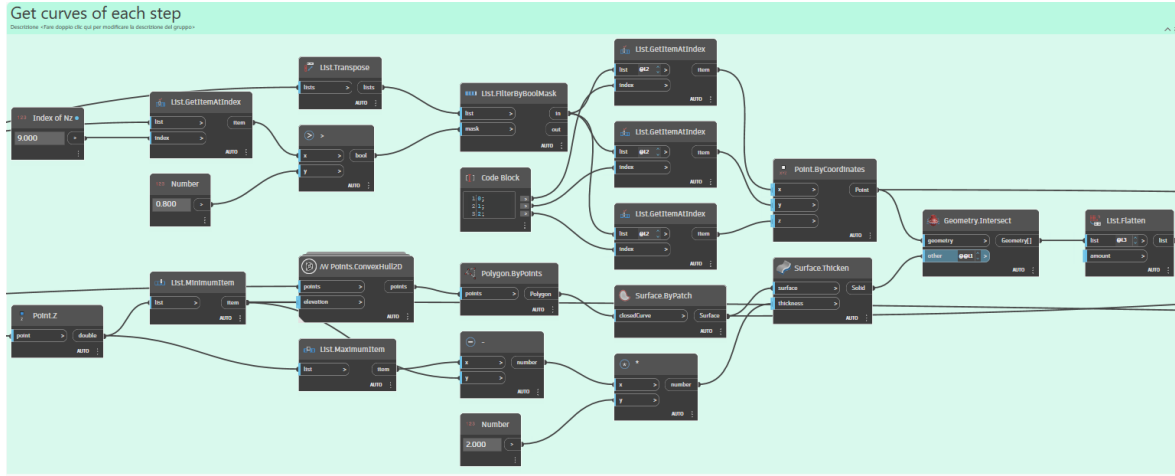


Figure 96. Filter points in Z coordinates and get Convex Hull of whole element.

The script also uses the RANSAC algorithm to identify planes from a set of points. It begins by counting the elements in the list with List.Count and determining the minimum item using List.MinimumItem. A code block and Math.Floor node calculate a threshold value, which is then input into a custom Python node “RANSAC PLANES” to perform the RANSAC plane fitting.

The resulting planes are processed to extract their origins with Plane.Origin, and the Z-coordinates of points are obtained using Point.Z. These coordinates are used to create a translation vector by using the node Vector.ZAxis, which creates a vector with 0, 0, 1 values, for translating geometry along the Z-axis. The Geometry.Translate node then applies this translation to the geometry.

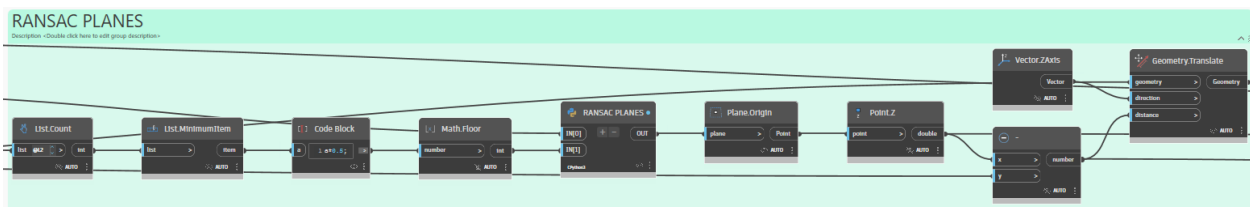


Figure 97. Get the planes of each step.

To create solid representations of stair steps, the script identifies the convex hull of points on each step using AW Points.ConvexHull2D found in the AW Points package, followed by generating polygons with Polygon.ByPoints. These polygons are converted into surfaces using Surface.ByPatch. The surfaces are then thickened into solids on both sides with Surface.Thicken.

The thickened surfaces are intersected with another geometry using Geometry.Intersect. The resulting intersected geometries are cleaned with List.Clean and flattened with List.Flatten to remove nested lists. These cleaned geometries are then converted into solids using Surface.Thicken and combined

into a single solid with Solid.ByUnion. Boolean operations and code blocks control the flow and ensure the operations are applied correctly.



Figure 98. Steps extrusion to the bottom and solid creation.

### 6.3 Dynamo Routine for LOD C

#### 6.3.1 Floors

The overall procedure in this LOD is the same as in the previous one.

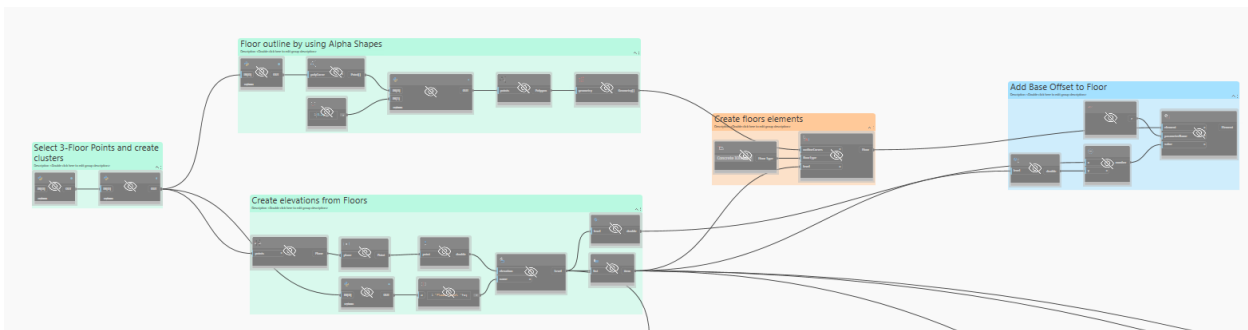


Figure 99. Overall script for Floors - LOD C.

#### 6.3.2 Columns

The second elements to be modelled are the columns.

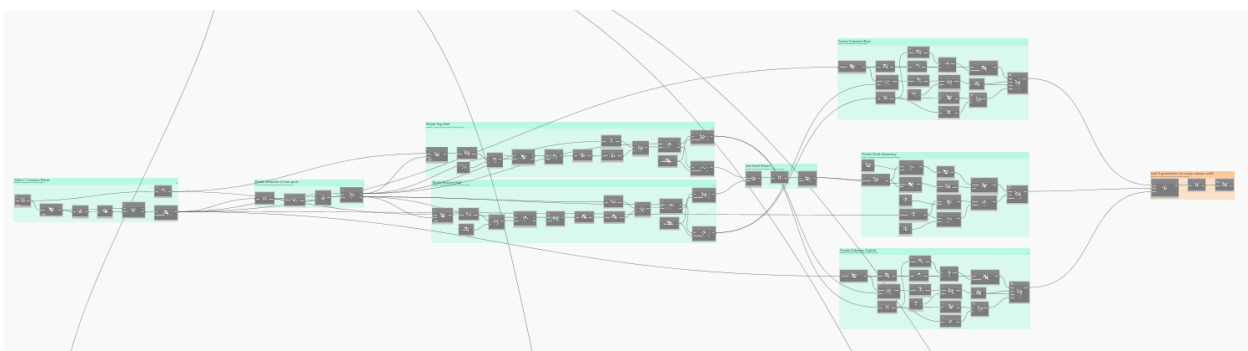


Figure 100. Overall script for Columns - LOD C.

The first step is the exact same as the previous LOD.



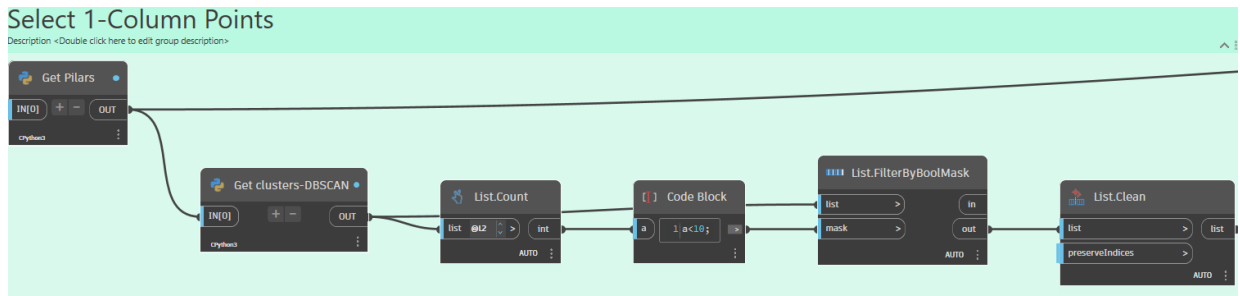


Figure 101. Column Point filtering and clustering.

The next step is to divide each cluster into two parts based on their Z-coordinate. The Point.Z node extracts the Z-coordinates of the points. The Math.Average node calculates the average Z-coordinate value. A comparison node checks if each Z-coordinate is greater than the average. This Boolean mask is then used in the List.FilterByBoolMask node to filter the list into two parts: the first containing elements with Z-coordinates above the average and the second with elements below or equal to the average.

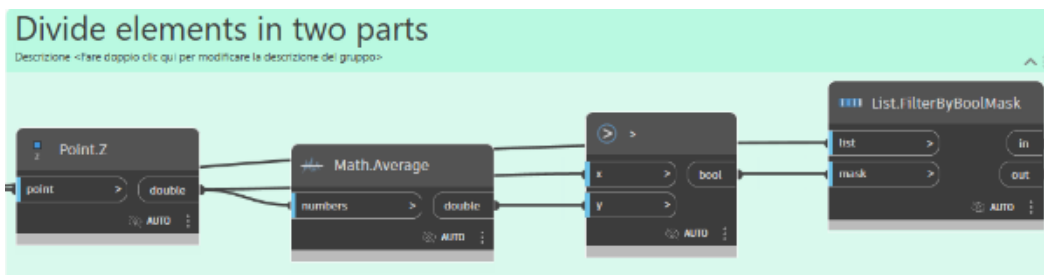


Figure 102. Column division in two groups.

To create the base, the next step is to get the bottom half of elements based on their normal vectors and Z-coordinates. The process begins by obtaining normal vectors with the Get Normal Vector node and extracting their Z-components using the Vector.Z node. A threshold of 0.7 is set in a code block to filter these vectors. The List.FilterByBoolMask node applies this threshold, and the resulting list is flattened using List.Flatten.

Next, the Z-coordinates of the points are extracted with the Point.Z node, and the average Z-coordinate is calculated using Math.Average. Another List.FilterByBoolMask node uses this average to filter the list into elements below the average Z-coordinate. The List.Clean nodes remove any empty or null items from both lists while preserving indices, resulting in two clean lists of elements for further analysis or processing.

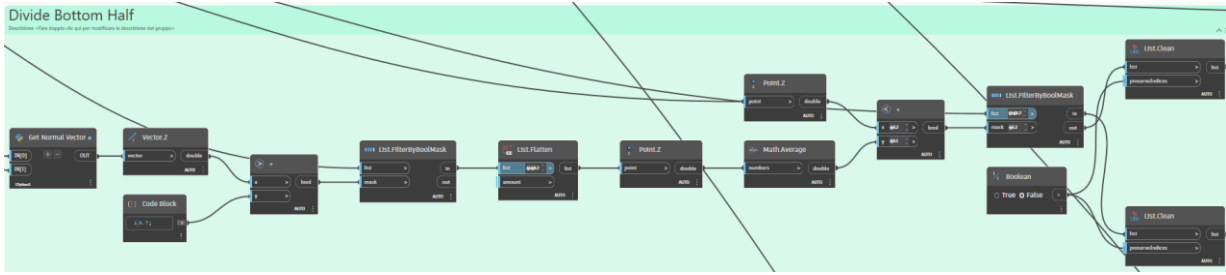


Figure 103. Selection of bottom half of column points.

After the base points are selected, the creation base columns by calculating various geometric parameters and constructing cuboids is performed. The process starts with determining a best-fit line through given points using `Line.ByBestFitThroughPoints`. The start point of this line is identified, and distances between points and this line are computed using `Geometry.DistanceTo`. The Z-coordinates of points are extracted with `Point.Z` and averaged. These averages are used in several calculations, including finding the segment length on the curve and rounding the results. The script also determines the maximum and minimum items in the lists. Finally, cuboids are created at specified segment points on the curve, with dimensions calculated using the average values and predefined multipliers, resulting in the placement of base columns in the model.

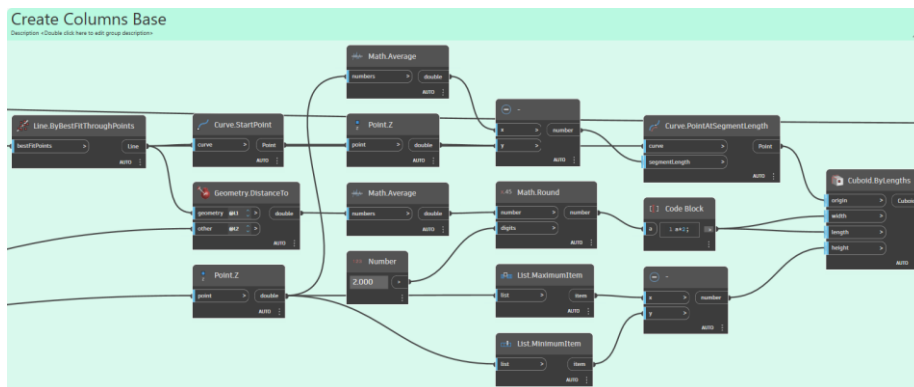


Figure 104. Creation of Column Base.

Now, on the other hand, each cluster is divided into the top half of elements based on their normal vectors and Z-coordinates. It begins by obtaining normal vectors using the `Get Normal Vectors` node and extracting their Z-components with `Vector.Z`. A threshold value of `-0.7` is set to filter these vectors using the `List.FilterByBoolMask` node. The filtered list is flattened using `List.Flatten`.

Next, the Z-coordinates of the points are extracted using `Point.Z`, and the average Z-coordinate is calculated with `Math.Average`. This average is used in a comparison to determine which points are above the average Z-coordinate. The `List.FilterByBoolMask` node filters the list based on this

comparison. The List.Clean nodes then remove any empty or null items from both lists while preserving indices, resulting in two clean lists of elements for further analysis or processing.

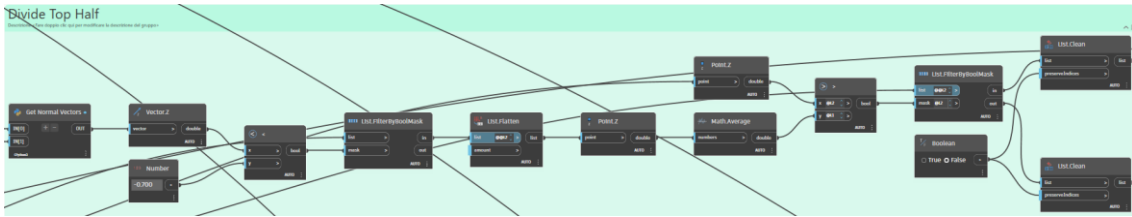


Figure 105. Selection of top half of column points.

After the capital points are selected, the column capitals are generated by calculating various geometric properties and constructing cuboids. It starts with a best-fit line through the given points using Line.ByBestFitThroughPoints. The start point of the line is determined, and distances from this line to other points are calculated using Geometry.DistanceTo. Z-coordinates of the points are extracted and averaged using Math.Average. These averages are used to find specific segment lengths on the curve, which are then rounded. The script identifies maximum and minimum items in the lists and uses these dimensions to create the cuboids referring to the capital section.

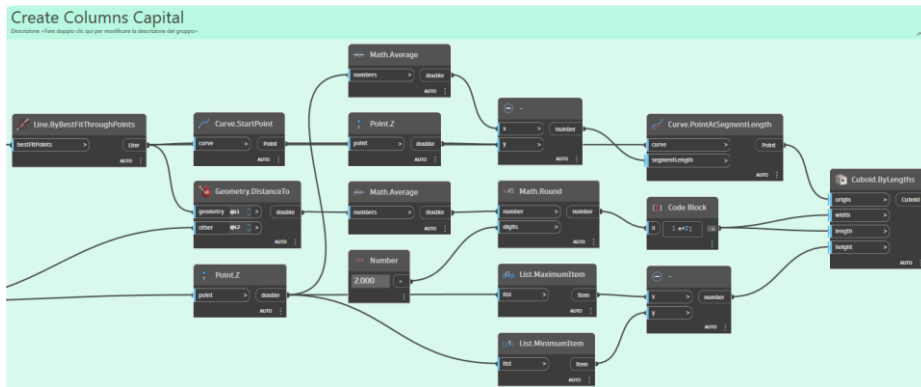


Figure 106. Creation of Column Capital.

After the creation of Base and Capital, the script is responsible for preparing shaft points by combining and flattening lists. The List.Create node aggregates multiple lists into a single list of lists. The List.Transpose node then switches rows and columns, effectively reorganizing the data structure. Finally, the List.Flatten node reduces the nested list structure into a single, flat list.



Figure 107. Selection of shaft section of column points.

The next step is to create a shaft geometry by defining and sweeping a profile along a path. It starts by generating a best-fit line through a set of points using `Line.ByBestFitThroughPoints`. Points at specific parameters (0.01, 0.5, and 0.99) along this curve are found using `Curve.PointAtParameter`. These points are transposed into a list for further operations. The distances between these points and the geometry are calculated with `Geometry.DistanceTo`, and the average distance is computed using `Math.Average`. This average distance is used to define the radius for circles created at the specified points with `Circle.ByCenterPointRadius`.

A path for the sweep is created using `Line.ByStartPointEndPoint`, connecting the start and end points on the curve. Finally, the shaft geometry is formed by sweeping the circular profile along the defined path using `Solid.BySweep`.

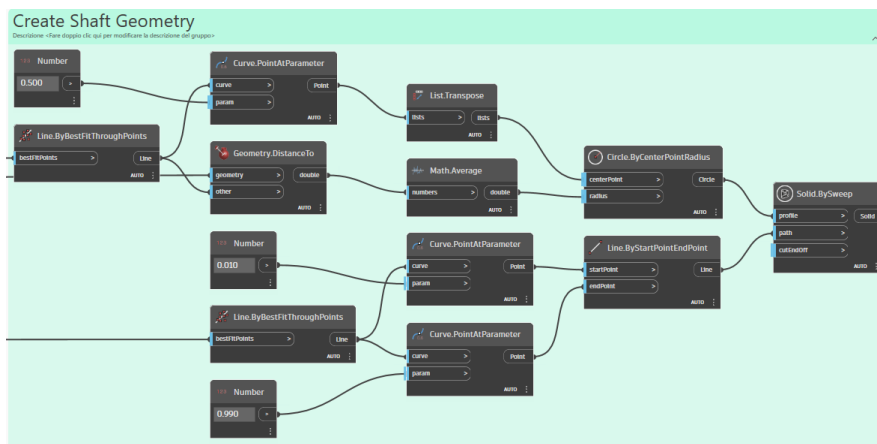


Figure 108. Creation of Column Shaft.

The last step for geometry creation is to combine the base, shaft, and capital geometries to create a single column solid. The `Join Base, Shaft, and Capital` node aggregates these three geometries into a list. This list is then reorganized using the `List.Transpose` node, which ensures the geometries are correctly aligned for union. Finally, the `Solid.ByUnion` node merges these geometries into one cohesive solid, resulting in a complete column object.

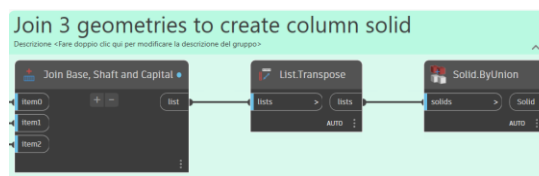


Figure 109. Joint of 3 different elements into a solid.

### 6.3.3 Walls

The third elements to be modelled are then the Walls. The script is the same as before.



Figure 110. Overall script for Walls - LOD C.

### 6.3.4 Roofs

The fifth script is the script for Roofs and Ceilings. The script is the same as before.

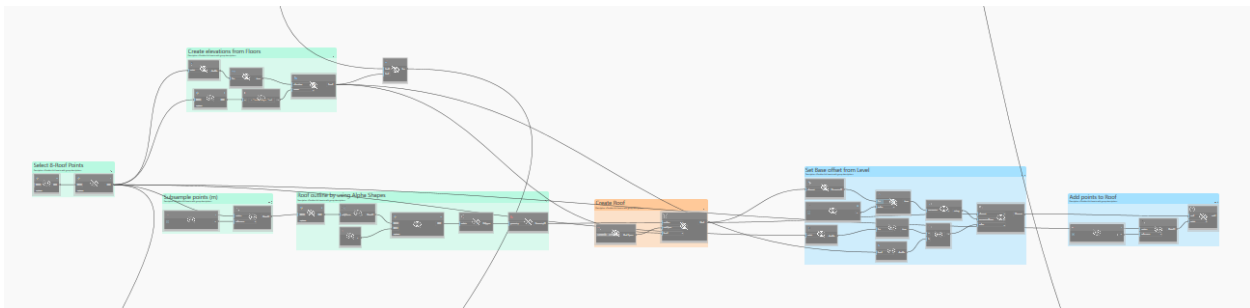


Figure 111. Overall script for Roofs and Ceilings - LOD C.

### 6.3.5 Windows

Same as previous LOD.

### 6.3.6 Vaults

Same as previous LOD.

### 6.3.7 Arch

Same as previous LOD.

### 6.3.8 Stairs

The last script is the script for Stairs. The script is the same as before.

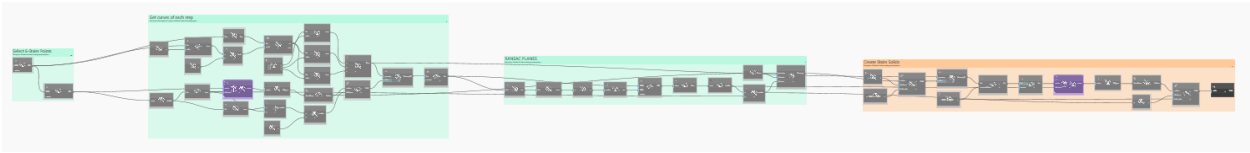


Figure 112 Overall script for Stairs - LOD C.

## 7 RESULTS

In this chapter, the results of the semi-automated Scan-to-HBIM workflow applied to Chapels XXIV and XXVIII of the Sacro Monte di Varallo are presented. The objective of this study was to develop a detailed and accurate HBIM model using advanced Instance Segmentation and modelling techniques. The results are analysed through various metrics, including average error values, Instance Segmentation accuracy, and visual error representations, which are essential for evaluating the effectiveness of the proposed methodology.

### 7.1 Chapel XXIV: Category Based Levels of Details Comparison.

- **Floors:**

Due to missing points in the Point Cloud, the Instance Segmentation algorithm is not so effective and outputs two clusters. Also, the floor elements do not cover the whole area of the Chapel but just a small portion of the Façade. In this case the results are again under 5 cm.

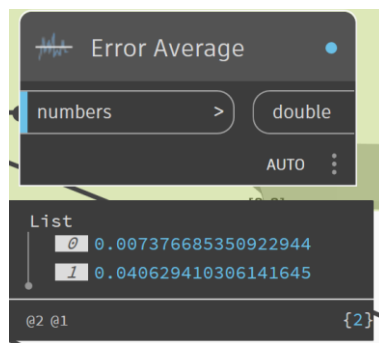


Figure 113. Average Error Value per Floor – Chapel XXIV.

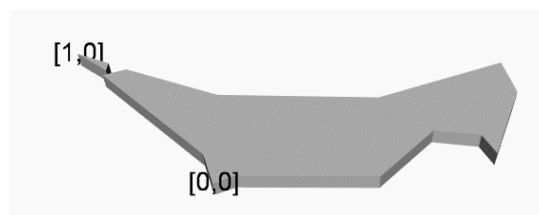


Figure 114. Identification of Floor Elements – Chapel XXIV.

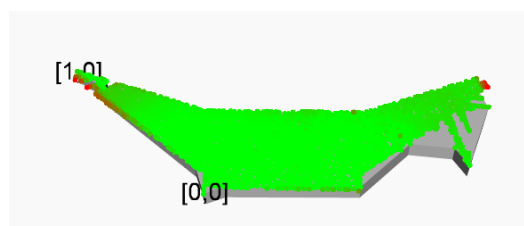


Figure 115. Visual Error Representation for Floors – Chapel XXIV.

- **Columns:**

For both LOD the column elements are still basic geometries. For LOD B a generic family of 30cm diameter was used. The error average is under 5 cm in both cases, being more precise in the LOD C due to the addition of the base and capital geometries.

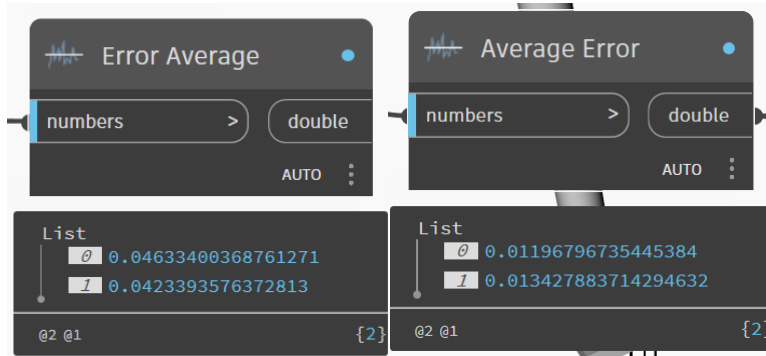


Figure 116. Average Error Value per Column – Chapel XXIV.



Figure 117. Identification of Column Elements – Chapel XXIV.

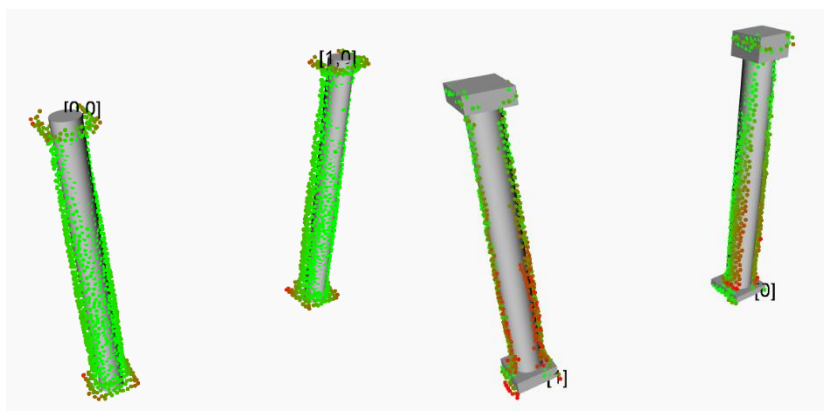


Figure 118. Visual Error Representation for Columns – Chapel XXIV.

- **Walls:**



After trying the two Instance Segmentation procedures, the Instance Segmentation comparison presented in Figure 120 show that the RANSAC + Clustering works better in cases where the wall points are segmented by levels, and the Clustering by Positioning and Normal Values works better if the data has all the information. In the cases where the points do not contain data in the Normal Values the second process misinterprets these points as a new cluster even if they are in the same position of other clusters. In both cases further processing must be done to calculate the openings of the walls, where one approach could be the use of Alpha Shapes, or by manual modification of the profiles of the wall elements.

List		List	
0	0.007193037350860749	0	0.002667285656227223
1	0.01233871817443924	1	1.5762808596179214
2	0.0035008572219232155	2	0.040716079084377584
3	0.03970473583437362	3	0.027728207087977134
4	0.016504488540949467	4	0.01066747010045504
5	0.015175375239366408	5	0.0034084937985776886
6	0.10813108715118779	6	0.011236052309263247
7	0.03051775678619665	7	0.007435545187209564
8	0.014293879439342684	8	0.01644135204518231
9	0.046777000520465885	9	0.025769311491578586
10	0.010453271063296982	10	0.39393504824527437
11	0.04152818021853524	11	0.005361956716843107
12	0.5791273496244328	12	0.03316634030579302
13	0.044085892003082945	13	0.009483696305648555
14	0.034467760806612326	14	0.0008961891192961061
15	0.020074971176554195	15	1.6717372050740915
16	0.012995480108885162	16	0.000303286068442457
17	0.005142914901590842	17	0.011861581987258785
18	0.011184701947450613	18	0.007665445145603587

Figure 119. Average Error Value per Wall – Chapel XXIV.

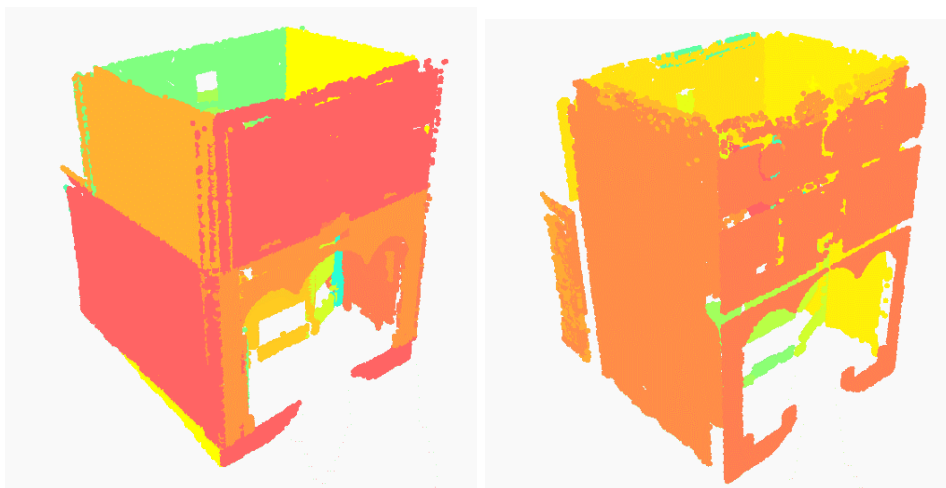
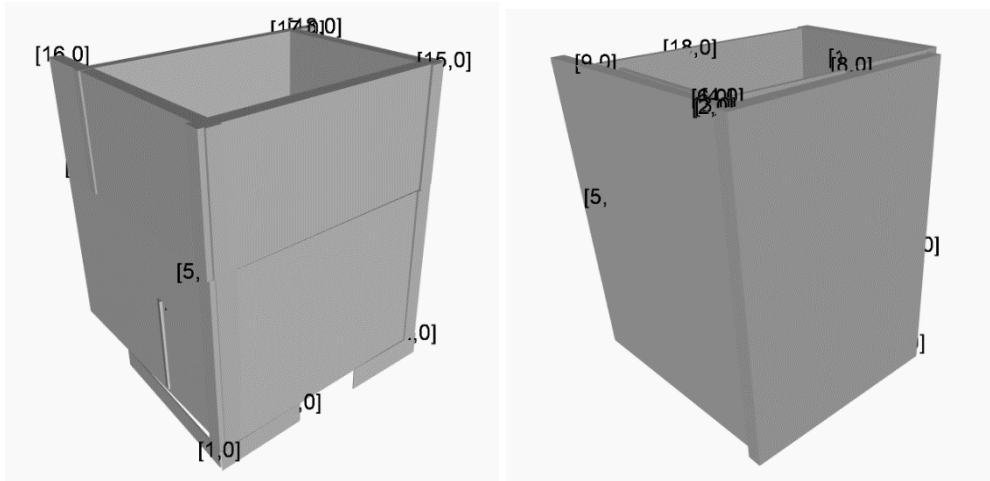
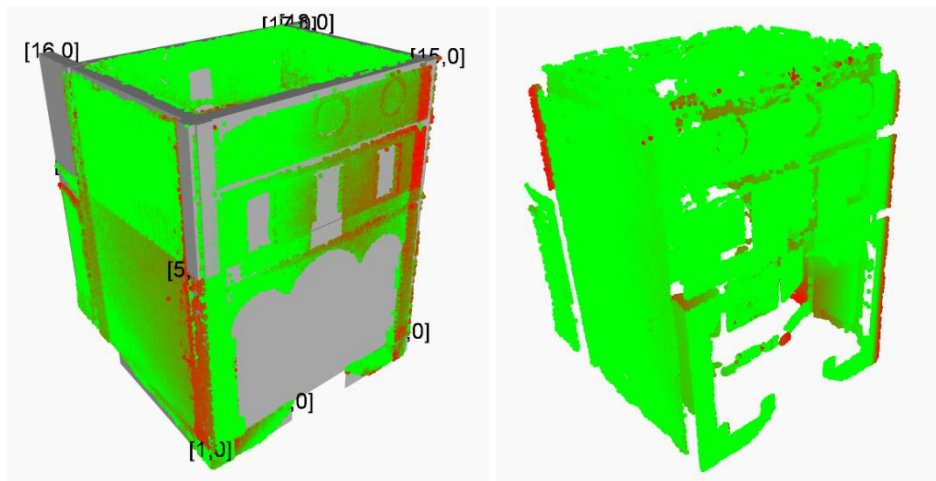


Figure 120. Wall Instance Segmentation by using RANSAC + Clustering vs 6D Clustering – Chapel XXIV.



*Figure 121. Identification of Wall Elements – Chapel XXIV.*



*Figure 122. Visual Error Representation for Walls – Chapel XXIV.*

- **Roofs:**

For the Roof elements the use of Alpha Shapes outputs a very precise geometry. Since the elements created are flat surfaces and points are added in a posterior phase, the procedure was tried by subsampling the points by minimum spacing at 30 centimetres, but after 15 minutes passed the routine had to be stopped. This then giving as result average error for each element around 50-100 centimetres.

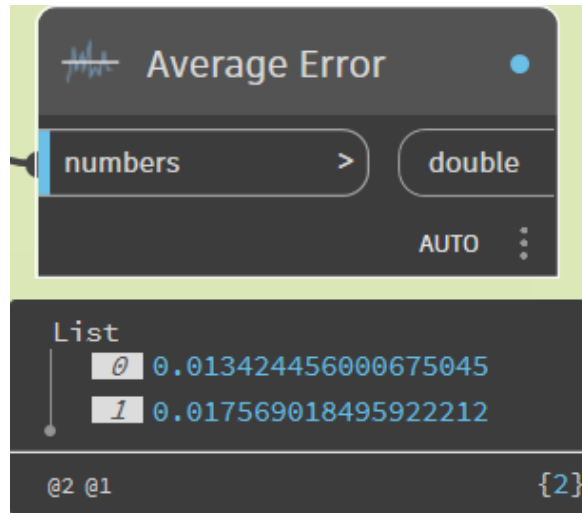


Figure 123. Average Error Value per Roof – Chapel XXIV.

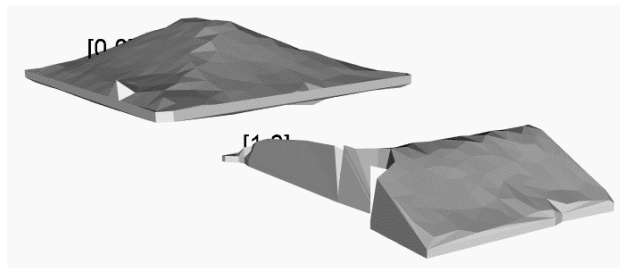


Figure 124. Identification of Roof Elements – Chapel XXIV.

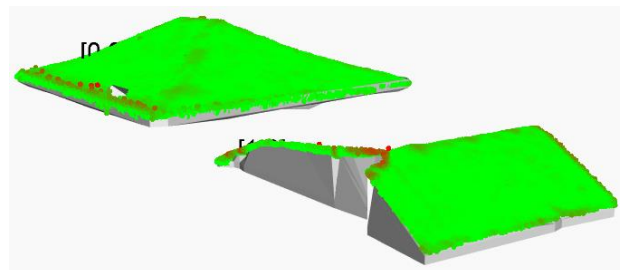


Figure 125. Visual Error Representation for Roofs – Chapel XXIV.

- **Windows:**

Due to error in the data caused by the glass present in the windows elements, the Instance Segmentation by 6D Clustering cannot to be done. This then forcing to use the simple 3D clustering and then create new types from default template families present in the Revit Library. Since the windows depend on the position of its hosts, walls, the error is mainly related to the position of those.

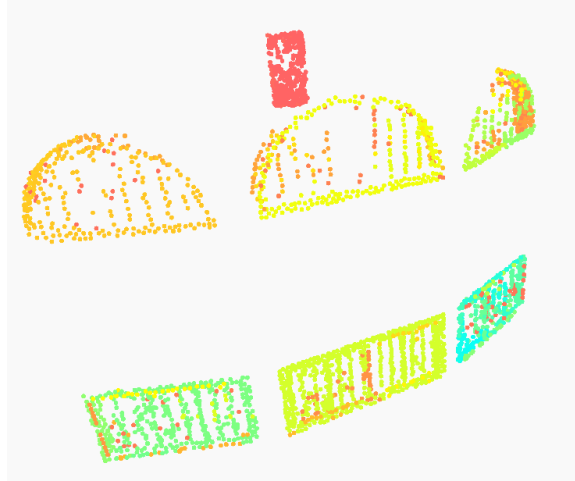


Figure 126. Window Instance Segmentation by using 6D Clustering – Chapel XXIV.

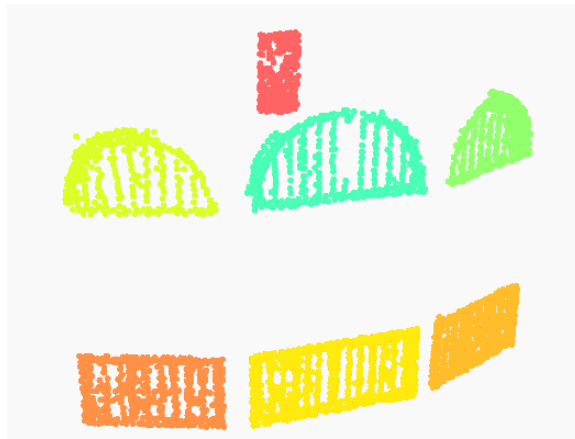


Figure 127. Window Instance Segmentation by using 3D Clustering – Chapel XXIV.

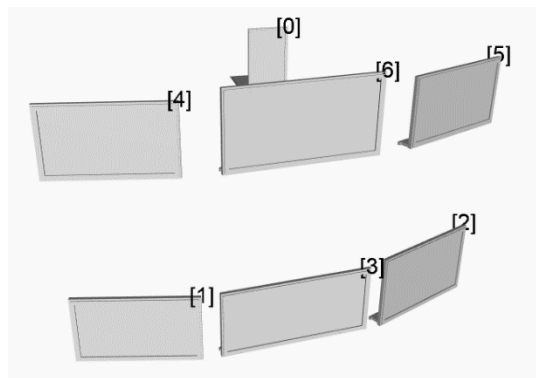
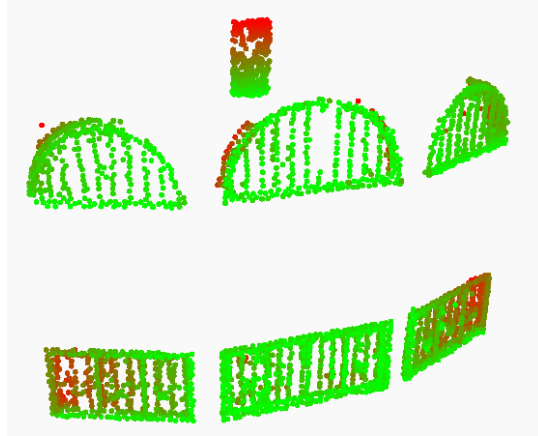


Figure 128. Identification of Window Elements – Chapel XXIV.



*Figure 129. Visual Error Representation for Windows – Chapel XXIV.*

- **Vaults-Arcs:**

Due to voids in the Point Cloud regarding these elements, the routine did not work for this case study. Several approaches were tried, for example calculating the points of a mesh and adding those points to a roof element, a second approach was to join the Arcs points with the Vaults to attempt in the filling of the geometries but still the routine was not able to find a suitable geometry. As seen in Figure 130, the 6D clustering is not precise with the actual data.



*Figure 130. Vault Instance Segmentation by using 6D Clustering – Chapel XXIV.*

- **Stairs:**

In the case of the Stairs, the Convex Hull algorithm works properly to determine the outline of the elements. The clustering algorithm also has satisfactory results by considering there are two stair elements present in the Chapel. The errors are in these cases, in the range of millimetres.

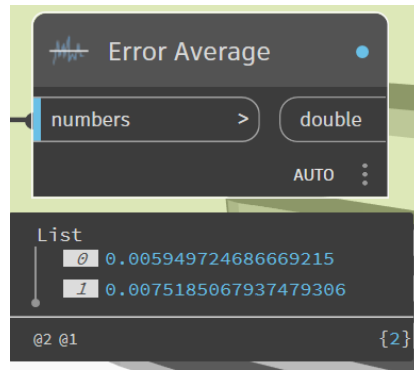


Figure 131. Average Error Value per Stair – Chapel XXIV.

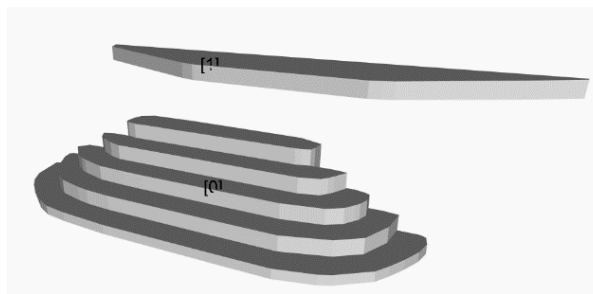


Figure 132. Identification of Stair Elements – Chapel XXIV.

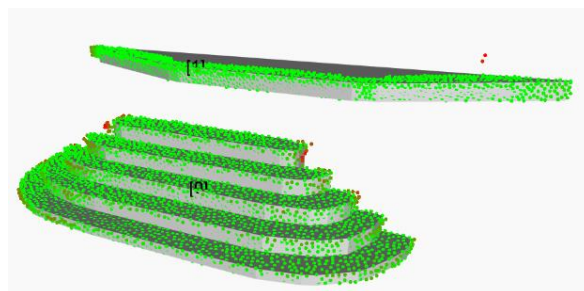


Figure 133. Visual Error Representation for Stairs – Chapel XXIV.

- **Point Cloud to HBIM model comparison:**

For the LOD C the average processing time is 25 to 30 minutes, having an input .txt file of 13 MB, meaning a subsampling of the Point Cloud by minimum spacing of 5 cm. In the next figure is presented a comparison of the Point Cloud with the postprocessed 3D HBIM model. In the postprocessing stage just the profiles of the walls are modified.

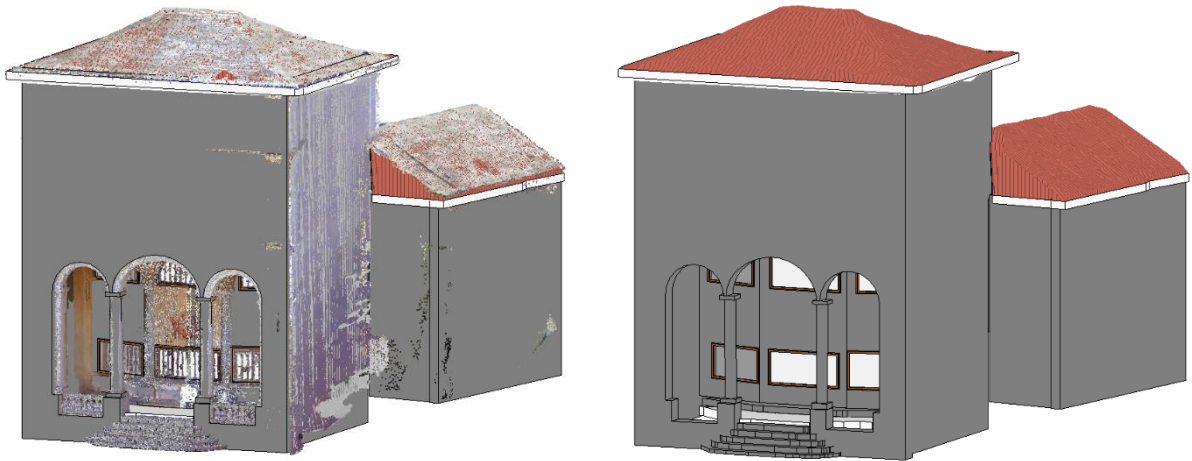


Figure 134. Comparison between Point Cloud and HBIM model – Chapel XXIV.

## 7.2 Chapel XXVIII: Category Based Levels of Details Comparison.

- **Floors:**

In this case, the main problem is again the presence of points only in the exterior areas. In comparison with the previous Chapel, this time the number of clusters is correct, but for the cluster on the first floor the data is precise. This then results in errors around 20 centimetres, but the element is similar to the real one.

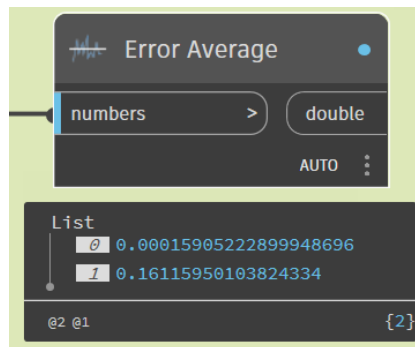
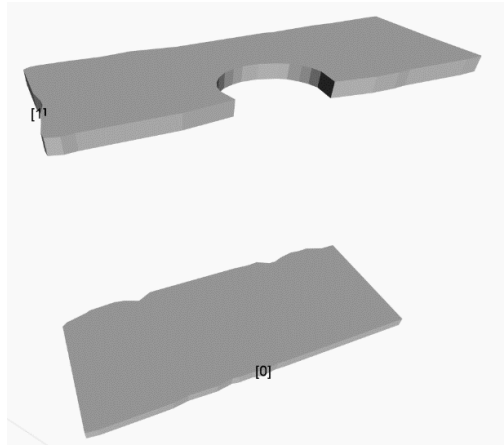
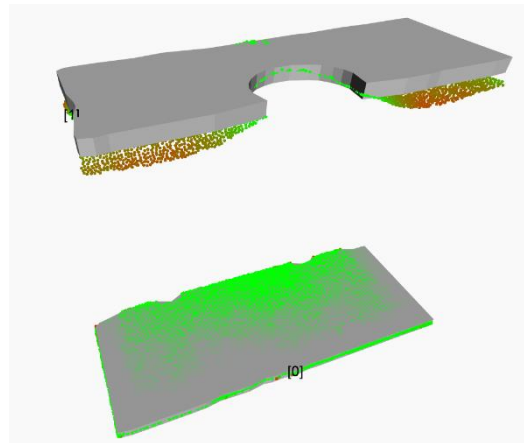


Figure 135. Average Error Value per Floor – Chapel XXVIII.



*Figure 136. Identification of Floor Elements – Chapel XXVIII.*



*Figure 137. Visual Error Representation for Floors – Chapel XXVIII.*

- **Columns:**

The errors in the left correspond to the LOD B elements and Right the LOD C elements. In both cases the errors are one-digit centimetres. For the LOD B the errors are higher due to the absence of the base and capitol geometries. This method provides a fast, precise and accurate positioning of the column elements.



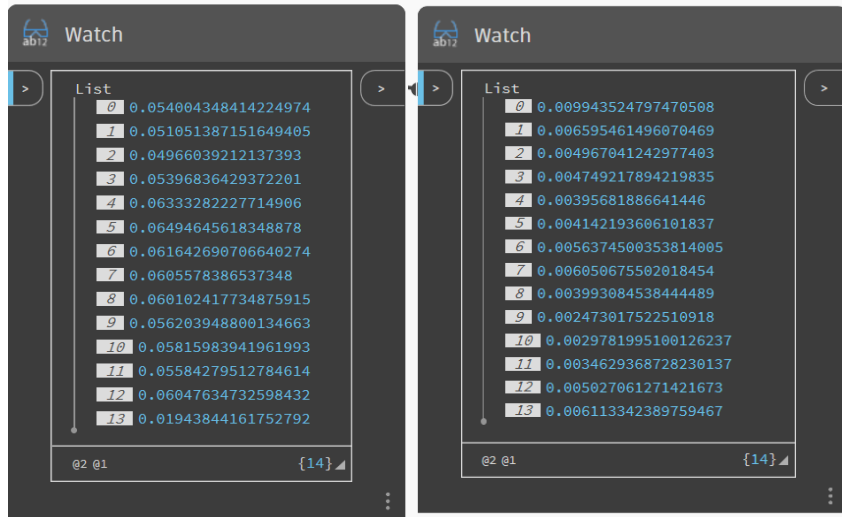


Figure 138. Average Error Value per Column – Chapel XXVIII.

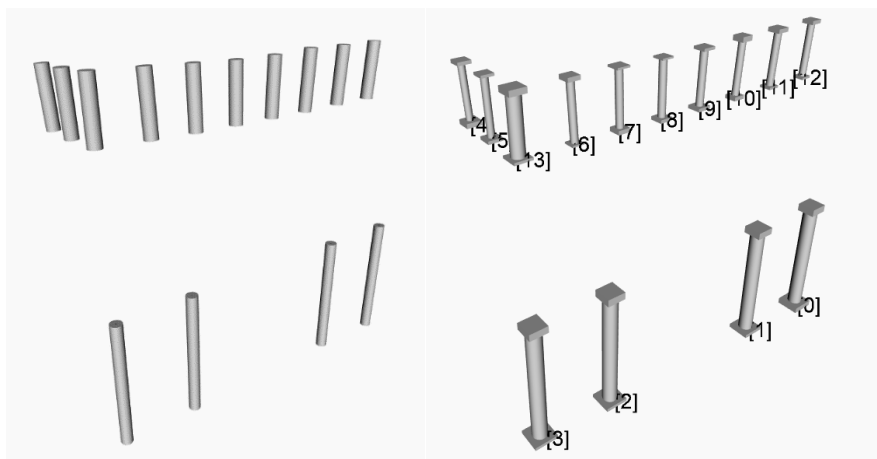


Figure 139. Identification of Column Elements – Chapel XXVIII.

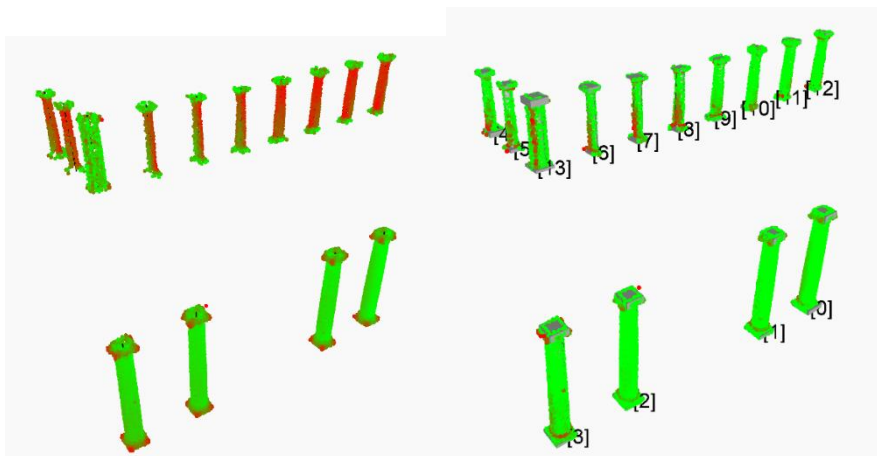


Figure 140. Visual Error Representation for Columns – Chapel XXVIII.

- **Walls:**

In this case study, again, after trying the two Instance Segmentation procedures, the Instance Segmentation comparison presented in Figure 142 show that the RANSAC + Clustering works better in cases where the wall points are segmented by levels, and the Clustering by Positioning and Normal Values works better if the data has all the information. Since in this case the Normal Values were present in almost all the points of the Point Cloud, the results are much better than in Chapel XIV when using the Instance Segmentation algorithm. In both cases further processing must be done to calculate the openings of the walls, where one approach could be the use of Alpha Shapes, or by manually changing the profiles of the wall elements.

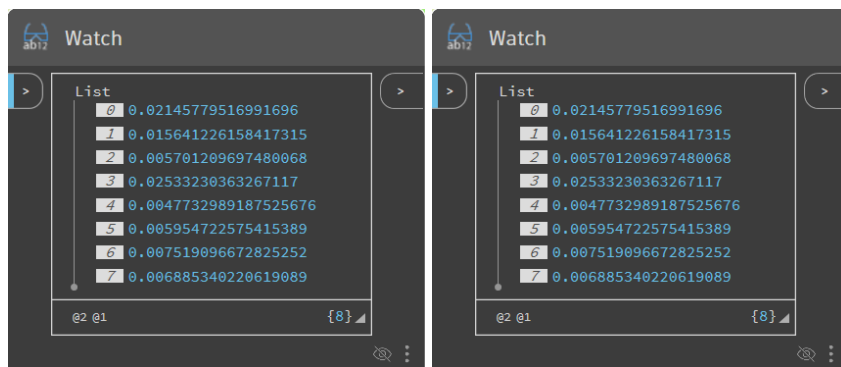


Figure 141. Average Error Value per Wall – Chapel XXVIII.

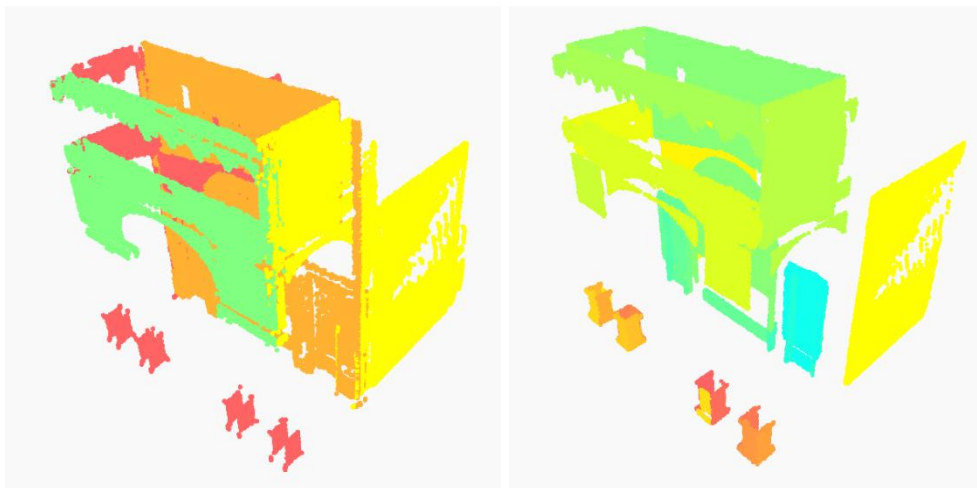


Figure 142. Wall Instance Segmentation by using RANSAC + Clustering vs 6D Clustering – Chapel XXVIII.

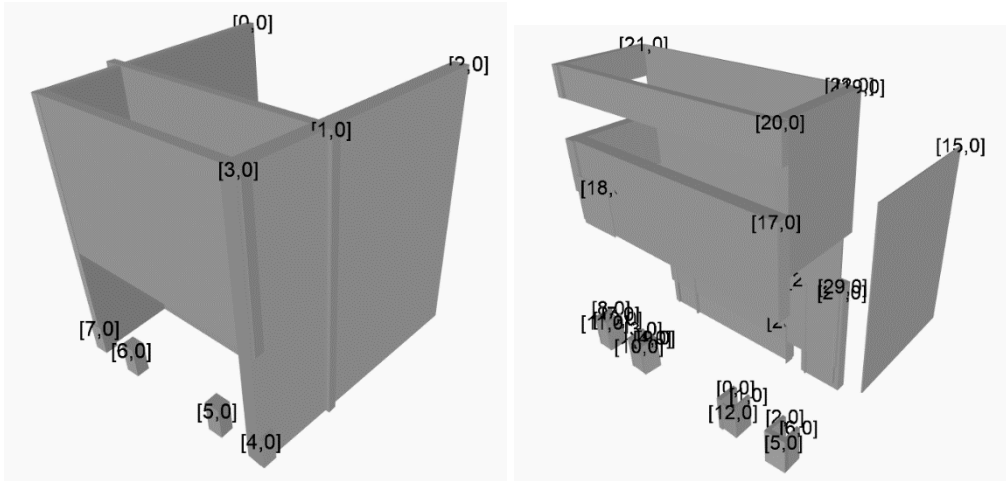


Figure 143. Identification of Wall Elements – Chapel XXVIII.

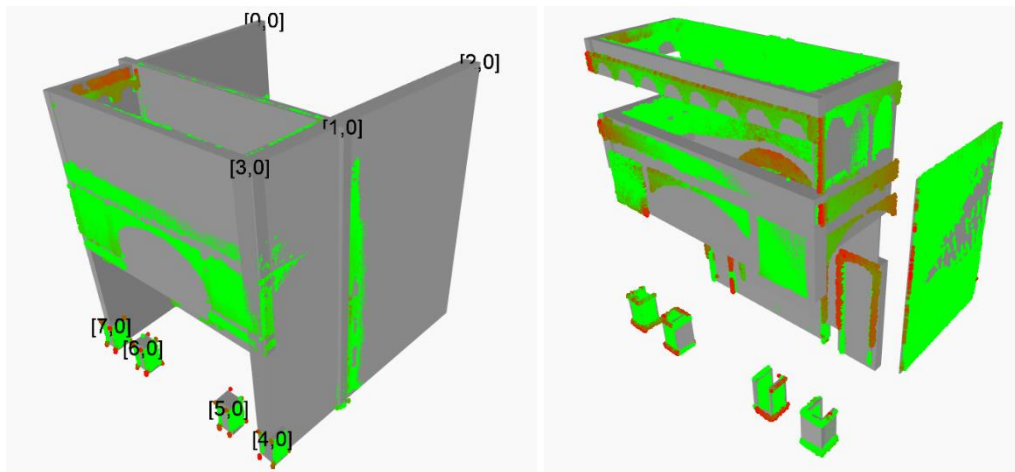


Figure 144. Visual Error Representation for Walls – Chapel XXVIII.

- **Roofs:**

The Addition of the points to the roof element require nearly 5 minutes for its calculation.

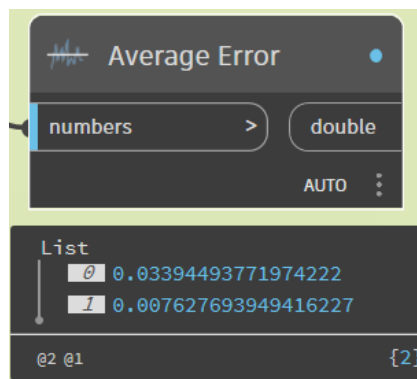
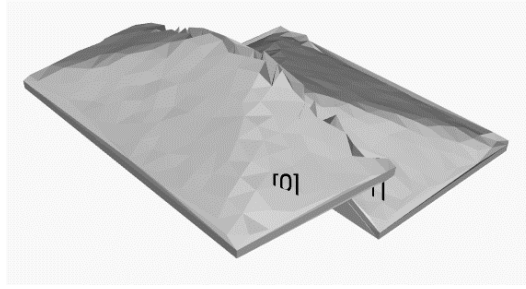
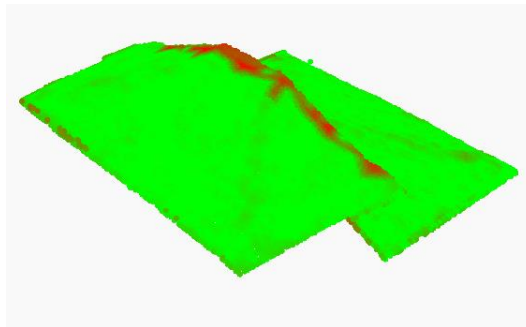


Figure 145. Average Error Value per Roof – Chapel XXVIII.



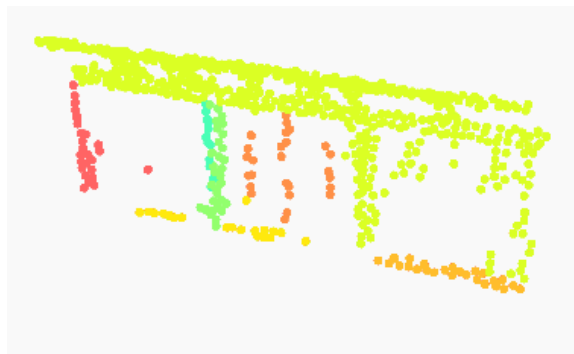
*Figure 146. Identification of Roof Elements – Chapel XXVIII.*



*Figure 147. Visual Error Representation for Roofs – Chapel XXVIII.*

- **Windows:**

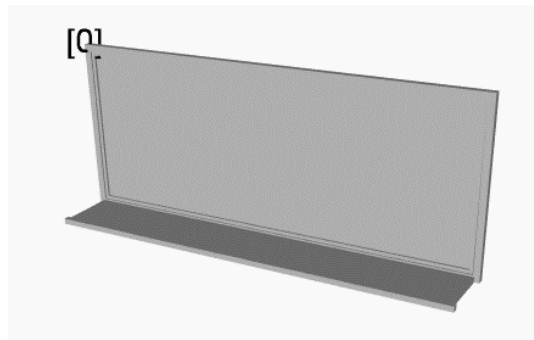
Due to error in the data caused by the glass present in the windows elements, the Instance Segmentation by 6D Clustering cannot be done. This then forcing to use the simple 3D clustering and then create new types from default template families present in the Revit Library.



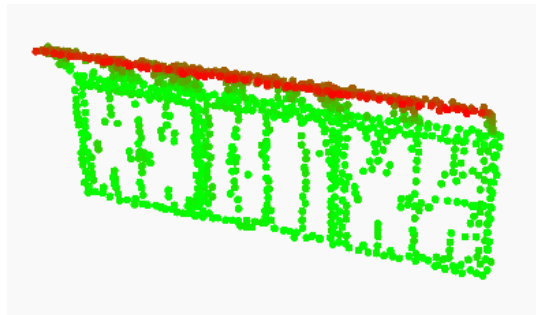
*Figure 148. Window Instance Segmentation by using 6D Clustering – Chapel XXVIII.*



*Figure 149. Window Instance Segmentation by using 3D Clustering – Chapel XXVIII.*



*Figure 150. Identification of Window Elements – Chapel XXVIII.*



*Figure 151. Visual Error Representation for Windows – Chapel XXVIII.*

- **Vaults-Arcs:**

In this Case Study, the data is more complete than in Chapel XIV. This allows to create the Vaults present in the Chapel without major problems. The separation of vaults and arc points, by taking the vault points for the outline calculation and the arcs points for the thickness of the Roof element was the approach used. The errors are around 1 cm which could be said to be more precise than any manual method can reach in the same amount of time.

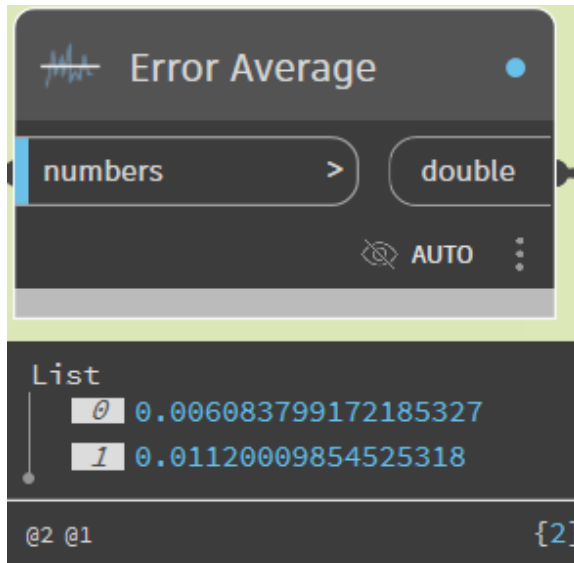


Figure 152. Average Error Value per Vault – Chapel XXVIII.

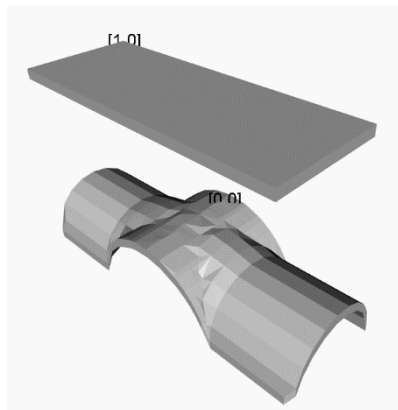


Figure 153. Identification of Vault Elements – Chapel XXVIII.

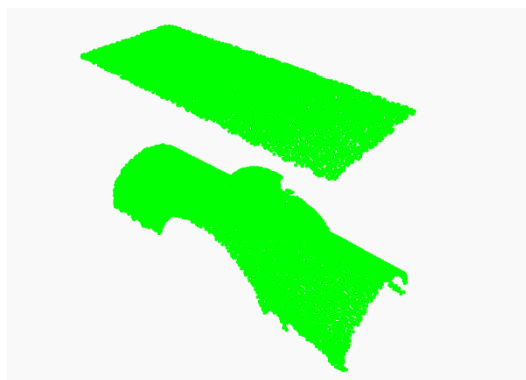


Figure 154. Visual Error Representation for Vault – Chapel XXVIII.

- Stairs:

For this Case Study, the stairs are simpler than in the previous one. With the existence of just one step, it is demonstrated that the routine can generate diverse complexities of stairs. The errors in this case are in the magnitude of centimetres, but always less than 4 centimetres.

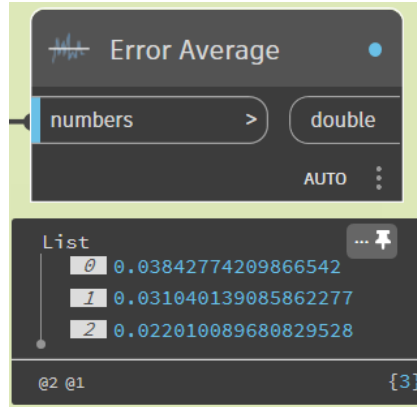


Figure 155. Average Error Value per Stair – Chapel XXVIII.

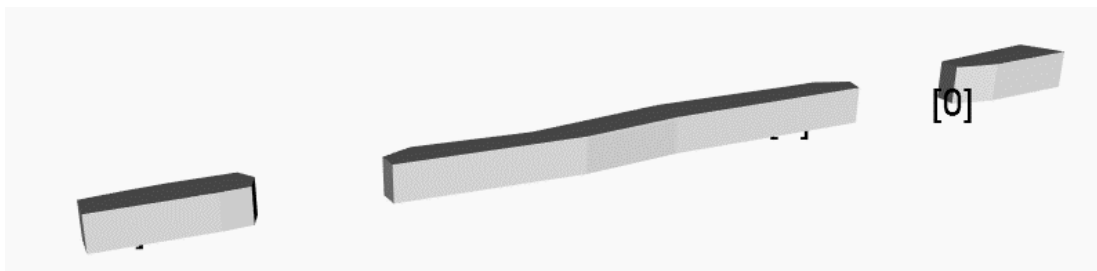


Figure 156. Identification of Stair Elements – Chapel XXVIII.

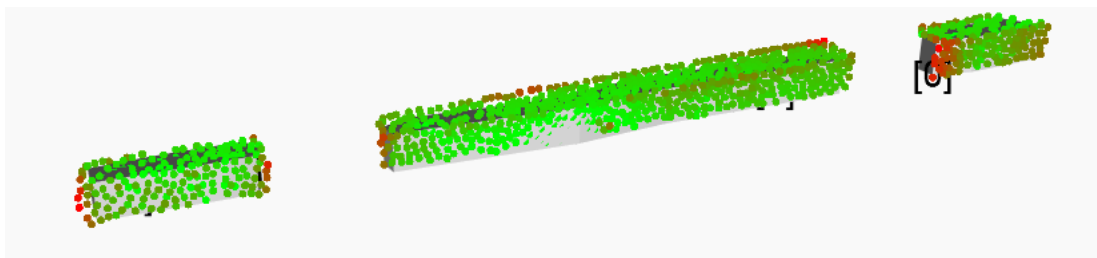
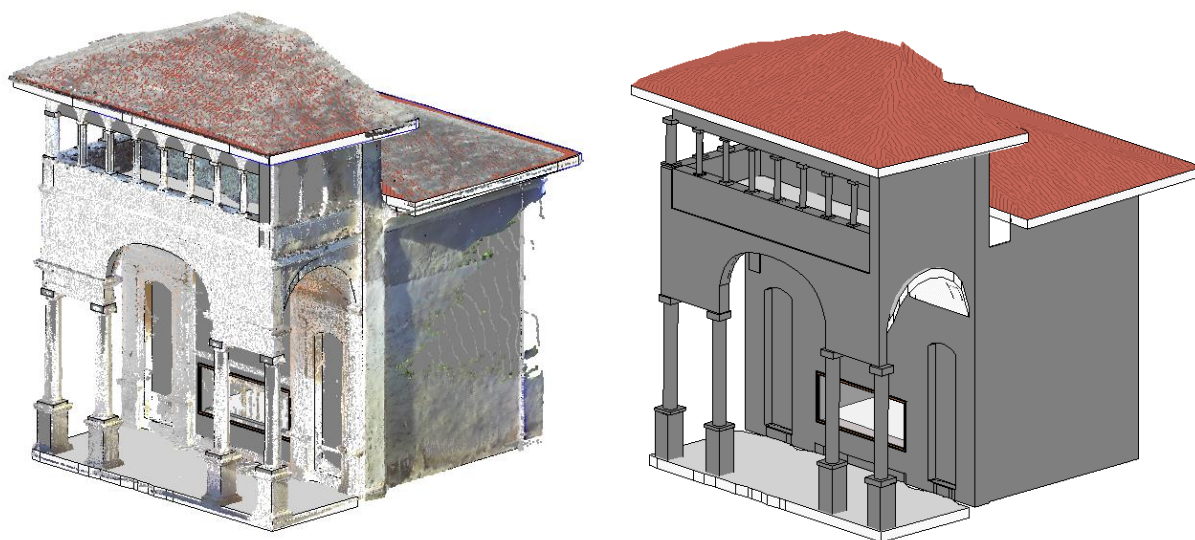


Figure 157. Visual Error Representation for Stairs – Chapel XXVIII.

- **Point Cloud to HBIM model comparison:**

For the LOD C the average processing time is 20 minutes, having an input .txt file of 10 MB, meaning a subsampling of the Point Cloud by minimum spacing of 5 cm. In the next figure is presented a comparison of the Point Cloud with the postprocessed 3D HBIM model. In the postprocessing stage just the profiles of the walls are modified.



*Figure 158. Comparison between Point Cloud and HBIM model – Chapel XXVIII.*



## 8 DISCUSSIONS

The integration of advanced technologies and methodologies in the Scan-to-HBIM workflow for built heritage presents both significant opportunities and challenges. This chapter provides an analysis of the results obtained from implementing Dynamo, included within Revit, and leveraging ChatGPT's Python programming capabilities to automate the modelling process. By examining the pros and cons of the proposed workflow, the benefits of precision and efficiency in element placement and modelling time reduction are highlighted. Additionally, the dependencies on high-quality input data, such as comprehensive point clouds and precise semantic segmentation are addressed. Furthermore, this chapter explores potential future improvements, including the incorporation of advanced Instance Segmentation techniques and non-geometric parameters, to enhance the accuracy and detail of HBIM models.

### 8.1 Pros and Cons of the proposed workflow.

The implementation of Dynamo, included within the Revit software suite, presents several advantages over alternative tools such as Grasshopper. One of the primary benefits of using Dynamo is its ability to integrate seamlessly with Revit, thereby providing a robust environment for parametric design and automated element placement. This integration allows for precise and accurate positioning of architectural elements automatically, which significantly reduces the potential for human error and enhances the overall accuracy of the HBIM model.

Furthermore, the use of Dynamo contributes to a substantial reduction in modelling times. This efficiency is primarily due to Python's advanced algorithmic capabilities incorporation in Dynamo, which semi-automates the primitive fitting methods and the Instance Segmentation through ML algorithms. This aspect is particularly beneficial in large-scale projects or complex architectural structures where manual modelling is time consuming.

Incorporating ChatGPT's programming skills in Python further optimizes the scripting process within Dynamo. ChatGPT can assist in generating Python scripts more quickly and efficiently, reducing the time required for script development and enhancing overall script performance. This ability to rapidly produce and refine scripts not only accelerates the modelling process but also ensures that the scripts are robust and error-free, thereby improving the reliability of the automated workflow.

However, the effectiveness of this semi-automated approach is highly dependent on the quality of input data, specifically the point clouds. These point clouds must be comprehensive and precise, as any inaccuracies or gaps can lead to significant errors in the resulting HBIM model. The requirement

for high-quality point clouds underscores the importance of utilizing advanced scanning technologies and methodologies to ensure the accuracy and completeness of the input data. As seen in the case of the walls for both Chapels, where Chapel XVIII had better results than Chapel XIV due to a better quality of the point cloud.

Additionally, the success of the automated modelling process hinges on the precision of semantic segmentation. Semantic segmentation, which involves categorizing each point in the point cloud according to its corresponding architectural element, is a critical step in the workflow. Any errors in this process can lead to misidentification of elements, resulting in inaccurate or incomplete models.

Despite these advancements, point clouds generally provide a Low LOI, capturing only the exterior surfaces of elements. This limitation means that while the exterior geometry is accurately represented, interior details and other critical information may not be included. As a result, additional data collection and integration may be necessary to achieve the desired LOD in the HBIM model.

Moreover, as the LOD increases, so does the demand on computational resources. Higher levels of detail require more complex computations and larger datasets, which can strain available computer processing capacity. This increase in computational demand necessitates robust hardware and optimized software solutions to manage and process the detailed data efficiently.

PROS	CONS
Cost reduction using Revit Dynamo's integrated graphical programming interface.	Highly dependent on quality of input data.
Time reduction of Scan-to-HBIM process via ML.	Low LOI reached.
Capacity of leveraging ChatGPT programming skills.	The higher the LOD, the higher the Computational requirements.

*Table 3. Pros and Cons of Proposed workflow.*

## 8.2 Future Work

Future work should focus on several key areas to further enhance the proposed semi-automated Scan-to-HBIM workflow. One area of improvement is the further semantic segmentation to increase the

output LOD. By leveraging DL techniques, it may be possible to improve the accuracy and efficiency of the Instance Segmentation process, thereby reducing the potential for errors and enhancing the overall quality of the HBIM model.

Additionally, efforts should be made to increase the LOI captured in point clouds. This could involve the use of more advanced scanning technologies or the integration of supplementary data sources to provide a more comprehensive representation of both the exterior and interior elements of the structure. Achieving a higher LOI would enable the creation of more detailed and accurate HBIM models, which are essential for advanced documentation, analysis and planning.

A promising avenue for future research is the use of advanced Instance Segmentation techniques that consider multiple factors such as position, normal direction, colour, and other attributes. Traditional Instance Segmentation methods primarily rely on spatial coordinates, which may not capture the full complexity of architectural elements. By incorporating additional data dimensions, such as the normal vectors that indicate the orientation of surfaces, the RGB colour values that can help differentiate materials, and other relevant attributes, the Instance Segmentation process can become more accurate and robust.

The implementation of enhanced computational techniques and optimizations will also be crucial as the complexity and detail of HBIM models continue to increase. Developing more efficient algorithms and leveraging high-performance computing resources can help manage the growing data and processing demands, ensuring that the workflow remains practical and efficient for large-scale projects.

Furthermore, the integration of non-geometric parameters such as material properties, historical data, and maintenance records into the HBIM model could provide additional value. These parameters can enhance the utility of the HBIM model for various stakeholders, including preservationists, architects, and engineers, by providing a more comprehensive dataset for decision-making and analysis.

## 9 CONCLUSIONS

The semi-automated Scan-to-HBIM workflow developed in this study for the Sacro Monte di Varallo case provides an innovative approach in the digital preservation of built heritage. By leveraging advanced Instance Segmentation techniques, ML algorithms, and VPL tools such as Dynamo.

Key findings from this research highlight notable improvements in both the efficiency and accuracy of the Scan-to-HBIM process. The use of Dynamo and Python scripts allowed for the automation of many repetitive tasks, significantly reducing modelling time and minimizing human error.

The relationship between input data and output is extremely related; high-quality input data is essential for generating accurate and reliable HBIM models. Any deficiencies in the input data directly affect the precision of the resulting models, highlighting the need for high-resolution and well-processed point clouds.

The ability to produce models at various Levels of Detail (LOD B and LOD C) ensures flexibility and precision in representing architectural elements, crucial for different stages of heritage conservation projects.

The integration of DL techniques for semantic segmentation proved effective in accurately categorizing architectural elements, reducing the manual effort required and enhancing the overall quality of the HBIM models.

This research contributes to the field of digital heritage preservation by providing a scalable and efficient methodology for the Scan-to-HBIM process. The semi-automated workflow not only improves the accuracy and efficiency of creating HBIM models but also lays another point of view for future research in integrating advanced technologies such as AI and ML for heritage conservation.

Future research should focus on improving semantic segmentation, enhancing the LOI, exploring advanced Instance Segmentation techniques, and incorporating non-geometric parameters such as material properties and historical data.

## 10 Bibliography

- [1] U. W. H. Centre, “Italy - UNESCO World Heritage Convention,” [Online]. Available: <https://whc.unesco.org/en/statesparties/it>. [Accessed 6 June 2024].
- [2] “PROGETTO – MAIN10ANCE,” [Online]. Available: <https://main10ance.eu/progetto/>. [Accessed 1 Luglio 2024].
- [3] L. Fecchio, “Galeazzo Alessi e il libro dei misteri (1565-1572). un architetto e il progetto per una nuova Gerusalemme sul Sacro Monte di Varallo.,” 31 May 2021. [Online]. Available: <https://hdl.handle.net/11583/2903498>. [Accessed 01 June 2024].
- [4] “Sacro Monte di Varallo,” [Online]. Available: <https://www.sacrimonti.org/sacro-monte-di-varallo>. [Accessed 26 May 2024].
- [5] “Itinerario turistico Sacro Monte, Lago d’Orta, Isola di San Giulio,” [Online]. Available: <https://www.ortaeoltre.it/it/itinerario-turistico-Tour-Orta-Sacro-Monte-Varallo.php>. [Accessed 25 May 2024].
- [6] “Cappelle nn 16/24 “Il figlio della Vedova di Naim” “Il Tribunale di Anna”,” [Online]. Available: <http://catalogo.beniculturali.it/detail/ArchitecturalOrLandscapeHeritage/0100022344>. [Accessed 25 May 2024].
- [7] L. Fecchio, “Galeazzo Alessi e il Libro dei Misteri (1565-1572). Un architetto e il progetto per una Nuova Gerusalemme sul Sacro Monte di Varallo. - Attachments.,” 2021. [Online]. Available: <https://hdl.handle.net/11583/2903498>. [Accessed 2 June 2024].
- [8] “Cappella XXIV - Gesù davanti al tribunale di Anna, Sacro Monte di Varallo,” [Online]. Available: <https://sacromontedivarallo.com/cappella-xxiv-gesu-al-tribunale-anna/>. [Accessed 26 May 2024].

- [9] “Cappella n 28 - Gesù al Tribunale di Erode Comune Di Varallo, Ammi,” 25 May 2024. [Online]. Available: <http://catalogo.beniculturali.it/detail/ArchitecturalOrLandscapeHeritage/0100015810>.
- [10] “Cappella XXVIII - Gesù al tribunale di Erode, Sacro Monte di Varallo,” [Online]. Available: <https://sacromontedivarallo.com/cappella-xxviii-gesu-al-tribunale-erode/>. [Accessed 26 May 2024].
- [11] A. Boiko, “Lobbyist Wars and the Development of BIM. Part 2: open BIM VS closed BIM. Revit vs ArchiCAD and Europe VS the Rest of the World,” 17 December 2020. [Online]. Available: <https://www.linkedin.com/pulse/lobbyist-wars-development-bim-part-2-open-vs-closed-revit-boiko-artem>. [Accessed 26 May 2024].
- [12] Autodesk, “Autodesk Revit,” 2024. [Online]. Available: <https://www.autodesk.com/products/revit/overview>.
- [13] Bentley Systems, “Building Design: OpenBuildings: BIM Software,” 2024. [Online]. Available: <https://www.bentley.com/software/openbuildings-designer/>.
- [14] Graphisoft, “ArchiCAD,” 2024. [Online]. Available: <https://graphisoft.com/solutions/archicad>.
- [15] Tekla, “Tekla Structures - Structural BIM Software,” 2024. [Online]. Available: <https://www.tekla.com/products/tekla-structures>.
- [16] Allplan, “Allplan AEC,” 2024. [Online]. Available: <https://www.allplan.com/products/allplan-aec/>.
- [17] 4. group., “I LOD (livelli di dettaglio) nel BIM. Spiegati bene,” 19 May 2020. [Online]. Available: <https://4mgroup.it/blog/i-lod-del-bim-spiegati-bene>. [Accessed 26 June 2024].
- [18] V. Maggiani, “LOD Simply Explained: The LOD Kiwi,” UK Construction Online, 15 April 2019. [Online]. Available: <https://www.ukconstructionmedia.co.uk/features/lod-simply-explained-the-lod-kiwi/>. [Accessed 16 June 2024].

- [19] UNI, *UNI 11337 – 4: Definizione dei livelli informativi.*, 2017.
- [20] “UK BIM Framework – BIM Standards, Guides & Resources,” UK BIM Framework, December 2019. [Online]. Available: <https://www.ukbimframework.org/>. [Accessed 21 June 2024].
- [21] S. Blundell, “RIBA and BIM: How design is fundamental to the process,” *Planning, Building & Construction Today*, 20 12 2017. [Online]. Available: <https://www.pbctoday.co.uk/news/digital-construction-news/bim-news/riba-and-bim-how-design-is-fundamental-to-the-process/37465/>. [Accessed 21 June 2024].
- [22] M. Ellis, “Level of Detail or Development: LOD in BIM,” Advenser, 2020. [Online]. Available: <https://rebim.io/level-of-detail-or-development-lod-in-bim/>.
- [23] “Dataset – ArCH dataset.” [Online]. Available: <https://archdataset.polito.it/dataset/>. [Accessed 29 June 2024].
- [24] G. Patrucco and F. Setragno, “Multiclass semantic segmentation for digitisation of movable heritage using deep learning techniques,” *Virtual Archaeology Review*, vol. 12, no. 25, p. 85, 2021.
- [25] L. Fiorini, A. Conti, E. Pellis, V. Bonora, A. Masiero and G. Tucci, “Machine Learning-Based Monitoring for Planning Climate-Resilient Conservation of Built Heritage,” *Drones*, vol. 8, no. 6, p. 249, 2024.
- [26] F. Matrone, R. Pierdicca, M. Paolanti, C. Morbidoni, E. S. Malinverni, E. Frontoni and A. M. Lingua, “Point Cloud Semantic Segmentation Using a Deep Learning Framework for Cultural Heritage,” *Remote Sensing*, vol. 12, no. 6, p. 1005, 2020.
- [27] A. Ursini, F. Matrone, A. Grazzini and M. Zerbinatti, “From scan-to-BIM to a structural finite elements model of built heritage for dynamic simulation,” *Automation in Construction*, vol. 142, p. 104518, 2022.

- [28] P. A. Escudero, “Scan-to-HBIM: automated transformation of point clouds into 3D BIM models for the digitization and preservation of historic buildings,” *VITRUVIO - International Journal of Architectural Technology and Sustainability*, vol. 8, no. 2, pp. 52-63, 2023.
- [29] V. Croce, G. Caroti, A. Piemonte, L. De Luca and P. Véron, “H-BIM and Artificial Intelligence: Classification of Architectural Heritage for Semi-Automatic Scan-to-BIM Reconstruction,” *Sensors*, vol. 23, no. 5, p. 2497, 2023.
- [30] M. Andriasyan, J. Moyano, J. E. Nieto-Julián and D. Antón, “From Point Cloud Data to Building Information Modelling: An Automatic Parametric Workflow for Heritage,” *Remote Sensing*, vol. 12, no. 7, p. 1094, 2020.
- [31] M. Avena, O. Roman, E. M. Farella, F. Remondino and A. Spanò, “A SEMI-AUTOMATED APPROACH TO MODEL ARCHITECTURAL ELEMENTS IN SCAN-TO-BIM PROCESSES,” *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vols. XLVIII-M-2-2023, pp. 1345-1352, 2023.
- [32] Graphisoft, “Rhino – Grasshopper – Archicad Toolset,” Graphisoft, [Online]. Available: <https://graphisoft.com/downloads/addons/interoperability/rhino>. [Accessed 29 May 2024].
- [33] P. APARAJIT, “Why has Dynamo Switched to Python 3, Should I Update Too?,” 17 May 2021. [Online]. Available: <https://dynamobim.org/why-has-dynamo-switched-to-python-3-should-i-update-too/>. [Accessed 9 June 2024].
- [34] “Lists of Lists | The Dynamo Primer,” DynamoBIM, [Online]. Available: [https://primer.dynamobim.org/06\\_Designing-with-Lists/6-3\\_lists-of-lists.html](https://primer.dynamobim.org/06_Designing-with-Lists/6-3_lists-of-lists.html). [Accessed 9 June 2024].
- [35] scikit-learn, “HDBSCAN,” [Online]. Available: <https://scikit-learn/stable/modules/generated/sklearn.cluster.HDBSCAN.html>. [Accessed 2 December 2023].
- [36] “Dynamo Dictionary,” [Online]. Available: <https://dictionary.dynamobim.com/>.



- [37] M. Mahmoud, W. Chen, Y. Yang and Y. Li, "Automated BIM generation for large-scale indoor complex environments based on deep learning," *Automation in Construction*, vol. 162, p. 105376, 2024.
- [38] G. Qin, Y. Zhou, K. Hu, D. Han, C. Ying and H. Liu, "Automated Reconstruction of Parametric BIM for Bridge Based on Terrestrial Laser Scanning Data," *Advances in Civil Engineering*, vol. 2021, pp. 1-17, 2021.
- [39] M. Lo Turco, E. C. Giovannini and A. Tomalini, "Parametric and Visual Programming BIM Applied to Museums, Linking Container and Content," *ISPRS International Journal of Geo-Information*, vol. 11, no. 7, p. 411, 2022.
- [40] H. Macher, T. Landes and P. Grussenmeyer, "From Point Clouds to Building Information Models: 3D Semi-Automatic Reconstruction of Indoors of Existing Buildings," *Applied Sciences*, vol. 7, no. 10, p. 1030, 2017.
- [41] V. A. Cotella, "From 3D point clouds to HBIM: Application of Artificial Intelligence in Cultural Heritage," *Automation in Construction*, vol. 152, p. 104936, 2023.
- [42] M. Mahmoud, W. Chen, Y. Yang and Y. Li, "Automated BIM generation for large-scale indoor complex environments based on deep learning," *Automation in Construction*, vol. 162, p. 105376, 2024.
- [43] Z. Chen and S. Gentes, "From Point Clouds to as-built BIM: Semi-automated Wall Reconstruction for Dismantling of Nuclear Power Plants," in *32nd Forum Construction Informatics*, Darmstadt, 2021.
- [44] "DBSCAN," scikit-learn, [Online]. Available: <https://scikit-learn/stable/modules/generated/sklearn.cluster.DBSCAN.html>. [Accessed 14 1 2024].
- [45] T. Xie, H. Chen, W. Liu, R. Zhou and Q. Li, "3D surface segmentation from point clouds via quadric fits based on DBSCAN clustering," *Pattern Recognition*, vol. 154, p. 110589, 2024.

- [46] T. Czerniawski, B. Sankaran, M. Nahangi, C. Haas and F. Leite, “6D DBSCAN-based segmentation of building point clouds for planar object classification,” *Automation in Construction*, vol. 88, pp. 44-58, 2018.
- [47] “numpy.loadtxt — NumPy v1.26 Manual,” [Online]. Available: <https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html>. [Accessed 27 11 2023].