Improving Architectural Workflow: A Grasshopper-Based Approach to Convert Rhino Models into Native Revit Elements

Paulin Faskel Tchoundie Tchuigwa[1*)], Aleš Marek[2)]

[1*)] Faculty of Architecture, Czech Technical University in Prague, Thákurova 9, 160 00 Praha 6, Czech Republic, faskel.tchoundie@fa.cvut.cz; https://orcid.org/0009-0003-1200-4631
[2)] Faculty of Architecture, Czech Technical University in Prague, Thákurova 9, 160 00 Praha 6, Czech Republic, ales.marek@fa.cvut.cz; https://orcid.org/0000-0002-9830-607X

https://doi.org/10.29227/IM-2025-02-02-064

Submission date: *2025-04-04* | Review date: *2025-05-09*

Abstract

The Architecture, Engineering, and Construction (AEC) industry has significantly advanced with the development of Building Information Modeling (BIM) software. However, AEC companies continue to face challenges related to interoperability and collaboration due to the heterogeneous nature of software tools. Various alternatives, such as Industry Foundation Classes (IFC), have been employed to address these issues, but their effectiveness remains limited. In current practice, many architecture firms use Rhino 3D for design, while BIM software such as Revit or Archicad is required for documentation and data management. This workflow presents significant limitations in data exchange, as Revit and Archicad often struggle to interpret Rhino models, importing them as direct shape that cannot be modified natively. This paper introduces a Grasshopper script designed to automate the creation of Revit elements such as walls, floors, windows, and doors from Rhino models using Rhino.Inside.Revit (RIR). While RIR allows the direct conversion of Rhino elements into Revit shapes, the proposed script enhances this process by generating native Revit elements from Rhino Brep geometry. The script functions in two key steps: first, it analyses the boundaries of wall elements in Rhino to automatically create Revit levels; second, it examines the boundaries of architectural components (walls, floors, windows, and doors) in Rhino to generate corresponding Revit family types and accurately place these elements in Revit at the exact coordinates from Rhino 3D using RIR nodes. This workflow improves architectural design efficiency and enables architects to maintain better control over their designs by leveraging BIM capabilities such as quantity take-offs and data management. Additionally, the script can function as a live synchronization tool, automatically updating any modifications made to Rhino elements within the Revit environment. Despite its effectiveness, the script has limitations when handling complex architectural geometries, such as double-curved surfaces. Addressing these challenges will be a focus of future research.

Keywords: *BIM, Interoperability, Automation, Workflow Optimization*

1. Introduction

Interoperability in architecture and BIM is crucial for effective collaboration and information management within the construction industry. BIM, as a shared knowledge resource, faces interoperability challenges due to the complexity of buildings and the diverse stakeholders involved [1, 2]. Ensuring seamless interoperability enhances communication, improves information management quality, and contributes to more sustainable project outcomes [3, 4]. However, current BIM applications face limitations in data consistency and model completeness when exchanging files through formats like IFC [5]. To overcome these challenges, researchers recommend knowledge-integrated, ontology-based, and IT-managed approaches to improve BIM interoperability [3]. Additionally, focusing on compatibility and interoperability throughout the project development process can enhance data exchange between models [6].

Interoperability issues between Rhino and Revit in architectural design workflows remain complex. El-Khouly and Abdelhalim [7] identified unproductive workflow integration and data loss as key barriers to interoperability between Rhino and Revit. As an alternative, Khan [8] suggested that visual scripting plug-ins like Grasshopper and Dynamo can be used together to create parametric models that respond to environmental factors. Furthermore, Ismail and Marin [9] highlighted that visual programming and API-based communication between software platforms enable real-time collaboration and data sharing, helping to preserve design integrity, improve workflow efficiency, and facilitate seamless hand-offs between conceptual design and BIM systems. These advancements benefit both professional practice and education in the AEC industry.

Recent research has explored interoperability between parametric modeling programs and BIM software for more efficient architectural design processes. According to Komarov and Kazarian [10, 11], the Rhino.Inside.Revit plug-in serves as a bridge between Rhino's freeform modeling capabilities and Revit's BIM environment, enabling seamless data transfer and enhanced design functionality. These integrations allow designers to leverage the strengths of both parametric and BIM software, improving accuracy and efficiency in the modeling process. Similarly, the Grasshopper-ARCHICAD Live Connection facilitates the direct generation of BIM models from Grasshopper's visual programming interface [12].

However, these approaches primarily focus on modeling directly within Grasshopper or converting geometries into direct shapes in Revit [13]. In contrast, this research aims to generate a Revit architectural model from an existing Rhino 3D model, utilizing the capabilities of Grasshopper and the Rhino.Inside.Revit plug-in to enhance interoperability and streamline the design-to-BIM workflow.

2. Interoperability between Rhino and Revit through Grasshopper and Rhino.Inside.Revit

The combination of Rhino and Revit, facilitated by tools like Grasshopper and Rhino.Inside.Revit, has become increasingly standard in architectural design and Building Information Modeling (BIM) workflows. This integration enables architects to leverage the strengths of both platforms Rhino's freeform and flexible modeling environment with Revit's structured, data-driven BIM system.

Grasshopper, Rhino's visual scripting tool, plays a key role by supporting parametric design. Meanwhile, Rhino.Inside.Revit serves as a direct interface plug-in, allowing Rhino to run inside the Revit environment. This setup enables architects to work within Rhino while gaining access to Revit's BIM functionality, facilitating a seamless and bidirectional workflow. This integration is particularly beneficial for maintaining design integrity, ensuring that the original design intent is preserved when transitioning from Rhino to Revit [7].

Through Grasshopper, users can create complex, parametric models that can then be exported to Revit using Rhino.Inside.Revit. Once in Revit, these models can be further analyzed and developed with BIM tools. This workflow is highly effective in automating repetitive processes and producing dynamic geometries that respond to environmental factors such as solar radiation, airflow, and thermal performance [8, 14].

One of the major challenges in interoperability between Rhino and Revit is the loss of design integrity during the model transfer. Rhino.Inside.Revit addresses this by ensuring that data is transferred accurately and efficiently, minimizing errors and saving valuable time [7]. Grasshopper's graphical scripting further enhances flexibility by allowing easy modification, optimization, and automation of designs.

Such interoperability is particularly powerful for environmental analysis, where Grasshopper scripts can automate the generation of climatic boundary conditions and extract meaningful data from simulation results [15]. In complex or large-scale projects involving multiple stakeholders, the integration of Rhino and Revit fosters collaboration, as it allows a unified modeling environment where multiple parties can contribute without compromising the overall design intent [16].

In summary, the interoperability offered by Grasshopper and Rhino.Inside.Revit provides a powerful, flexible, and accurate workflow for architects and engineers. It bridges the gap between conceptual design and BIM, enhances collaboration, and enables the automation of design processes, ultimately leading to improved efficiency and design quality.

3. Methodology

The proposed research aims to generate a Revit model based on a Rhino 3D model using Grasshopper and Rhino.Inside.Revit (RIR). This study focuses on architectural elements such as walls, floors, doors, and windows. Through Grasshopper, Rhino Brep elements are analysed to extract necessary information. Using RIR, a Revit model (with native Revit families) is then generated based on the analysed data. The research approach is structured into three main steps:

- Generating Revit levels and floors from the analysed Brep elements in Grasshopper.
- Generating walls based on the extracted data.
- Generating openings (windows and doors) in the appropriate locations.

This workflow ensures efficient, automated translation from Rhino to Revit. It is important to ensure that elements in the Rhino model are well-organized into layers. Elements of the same type (e.g., floors, walls, doors, and windows) should be organised in the same Rhino layer to ensure accurate selection in Grasshopper.

3.1 Creation of Revit Levels and Floors

To create Revit levels, the floor representations in the Rhino model were used as reference points. Using the Query Model Objects node in Grasshopper, we selected all floors by specifying their layer name in Rhino. Next, a custom node was used to determine the upper surface of each floor. The coordinates of these points were then deconstructed to extract their Z values. A list was created containing all the extracted Z values, and duplicates were removed to ensure that multiple levels were not created at the same height in Revit. After filtering the Z values, a corresponding list of names was generated for the Revit levels. Using the Add Level node in RIR, the filtered Z values were input as elevation data, while the generated names (created using the Concatenate node) were assigned as level names. This process automatically generated the Revit levels. The Fig. 1 illustrates the Grasshopper script used in this process.

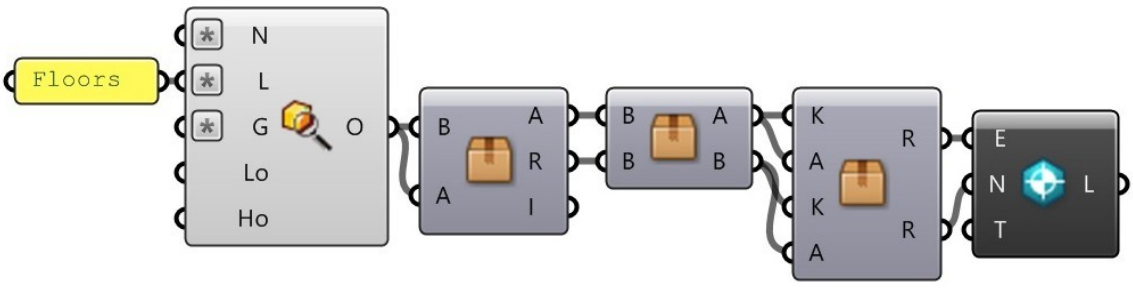

Fig. 1. Proposed script for Revit levels creation.

To create floors in Revit, the Query Model Objects node was used to select all Brep representations of floors in Rhino. The selected floors were then sorted to group those at the same height into separate branches within the Grasshopper list, facilitating data management. As shown in the Fig. 2, the center of the upper surface of each Brep was identified using a custom Grasshopper node. Then, XY and YZ planes were created, and their intersections with the corresponding Brep elements were determined.

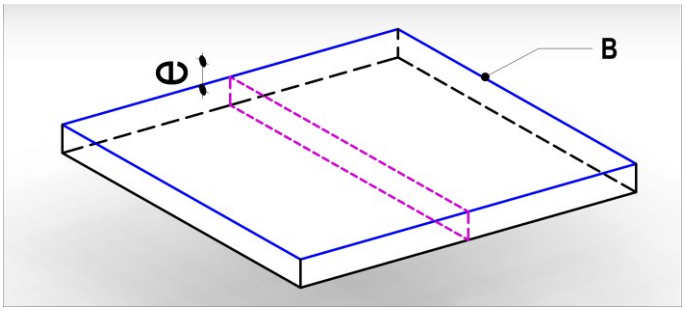

Fig. 2. Illustration of floor data extraction.

- The XY plane intersection (highlighted in blue) was used as the floor boundary in Revit;
- The YZ plane intersection (highlighted in pink) was used to determine the thickness of the floor.

All unique floor thicknesses were organized into a list, with duplicate values removed. Using the Concatenate node, a naming convention was established for the floor types in the format: "Floor_e mm", where "e" represents the floor thickness, and "mm" denotes the unit. Next, the Query Types node of RIR was used to select a predefined floor type that had been loaded into Revit. The selected floor type was duplicated and renamed according to the constructed list of floor names. Using the Construct Compound Structure Layer and Construct Compound Structure nodes, the floor layers were edited based on the material properties of each floor in the model. Finally, to generate floors in Revit, the Add Floor node of RIR was used:

- The list of boundaries obtained from the XY plane intersections was set as the boundary input;
- The edited list of floor types with their compound structures was used as the floor type input;
- The height of each boundary was used to query the corresponding Revit level for proper floor placement.

The Fig. 3 provides an illustration of the Grasshopper script used in this process.

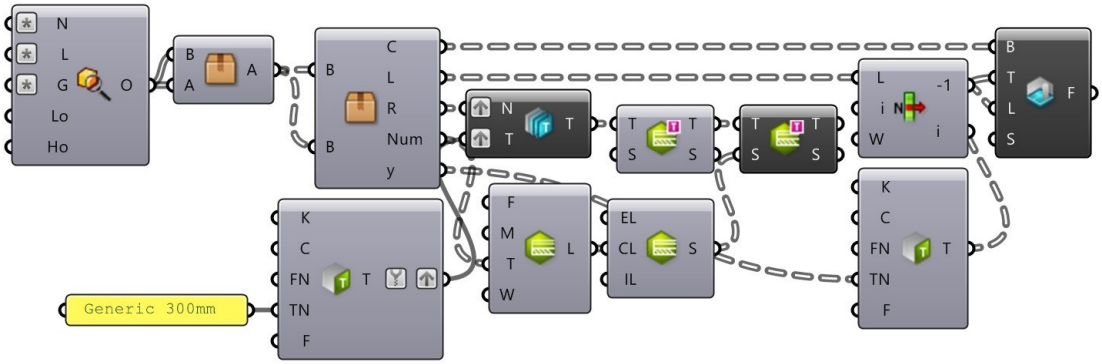

Fig. 3. Floors creation script.

3.2 Creation of walls

To create a wall using the Add Wall (Curve) node in RIR, four main inputs are required: the curve of the wall, the wall type, the level, and the height. The main challenge in this process was extracting the curve of the wall, which depends on how the Rhino model was constructed. The queried walls from Rhino, using the Query Model Objects node, were sorted into two main categories:

- The first category contains walls represented by a single line.
- The second category contains walls made by joining two or more lines.

The Fig. 4 highlights the two different categories: elements of the first category are shown on the left, while those of the second category are shown on the right.

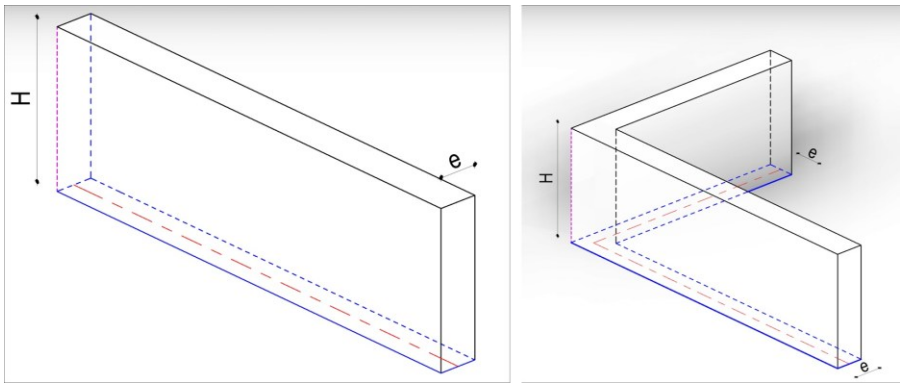

Fig. 4. Illustration of wall data extraction.

To sort these walls, an intersection was created between the Brep and the XY plane at the highest point of each Brep. The output was a list of polycurves, which were then exploded. The number of segments in each polycurve was counted, and the list

elements were compared to 4 using the “Larger Than” node in Grasshopper. The resulting Boolean values were then used to dispatch the elements into the two categories. For the first category (single-line walls), a custom node was used to obtain the curve from the intersection between the Brep and the XY plane. After exploding the curve, the two longest segments of the rectangle were identified. A vector was then constructed from the midpoint of one segment to the midpoint of the other, and one segment was moved using that vector. The distance moved was half the thickness of the wall. This process aimed to define the centreline of the wall. For the second category (multiple joined walls), the same process was applied, with the only difference being that a custom script analysed each wall in different directions.

For the height of the wall, each Brep was deconstructed to determine the minimum and maximum points. These points were deconstructed to extract the Z values. Using the expression H = Zmax - Zmin, we successfully calculated the height of each wall.

For the levels, the minimum Z value of each wall was used to query the corresponding elevation using the Query Level node.

For the thickness of the wall, the length of the shortest segment from the exploded curve (intersection of the XY plane and the Brep) was used as the thickness of the wall. To create Revit wall types, wall types were first sorted by thickness to facilitate data management in Grasshopper. We then applied a procedure similar to the one used for floors, querying a Revit wall type and setting up the material layers.

Finally, using the Add Wall node, we connected the values obtained from the previous steps to the corresponding inputs. The Fig. 5 below provides a visual representation of the Grasshopper script employed in this process.

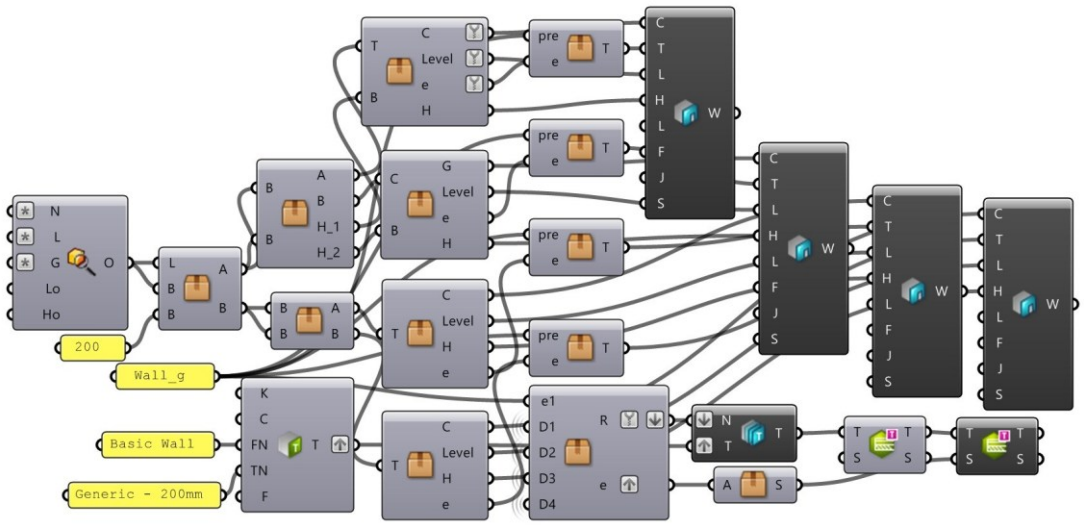

Fig. 5. Walls creation script.

3.3 Creation of Openings (Windows and Doors)

To create openings, we first selected all windows and doors in the model using the Query Model Objects node. Next, a bounding box was generated for each opening. Since the Add Component (By Location) node requires location, type, level, and host inputs, we extracted the necessary information from the bounding box to provide these inputs.

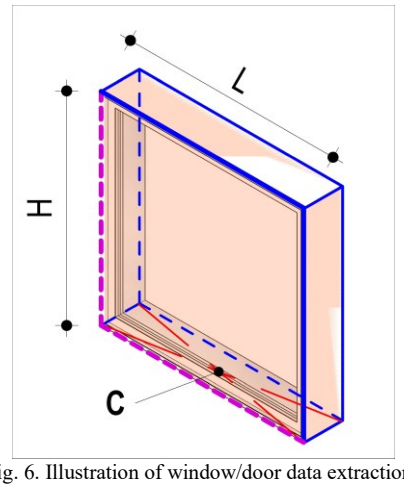

Fig. 6. Illustration of window/door data extraction.

As shown in the Fig. 6 above

- The centre of the bottom surface of each bounding box was extracted for the location input.
- To determine the height of the window, we extracted the minimum and maximum Z-coordinates from the bounding box, then calculated the height using the formula: H = Zmax - Zmin
- To determine the width, we performed a similar process in both the X and Y directions to account for different window orientations. Since one value corresponds to the window width and the other to the wall thickness, we created a comparison pattern to extract the larger value, ensuring the correct width measurement.

With the width and height extracted, the next step was to create the corresponding Revit window/door family based on the Rhino model. This issue can be addressed in two ways:

- Grouping similar opening types in the same Rhino layer.
- Using attributes that can be queried in Grasshopper.

Using the Query Types node, we selected a Revit family type and modified its dimensions according to the extracted values. The Element Parameter node was then used to set these parameters.

For the host element, the Check Brep Intersection function was used to identify intersections between each window/door and the wall geometry. The function produced a Boolean list, which was used to dispatch and establish a link between each opening and its corresponding wall.

Finally, by connecting all the extracted inputs to the Add Component (By Location) node, we successfully generated the windows and doors in the Revit model. The Grasshopper script used in this process is illustrated in the Fig. 7 below.

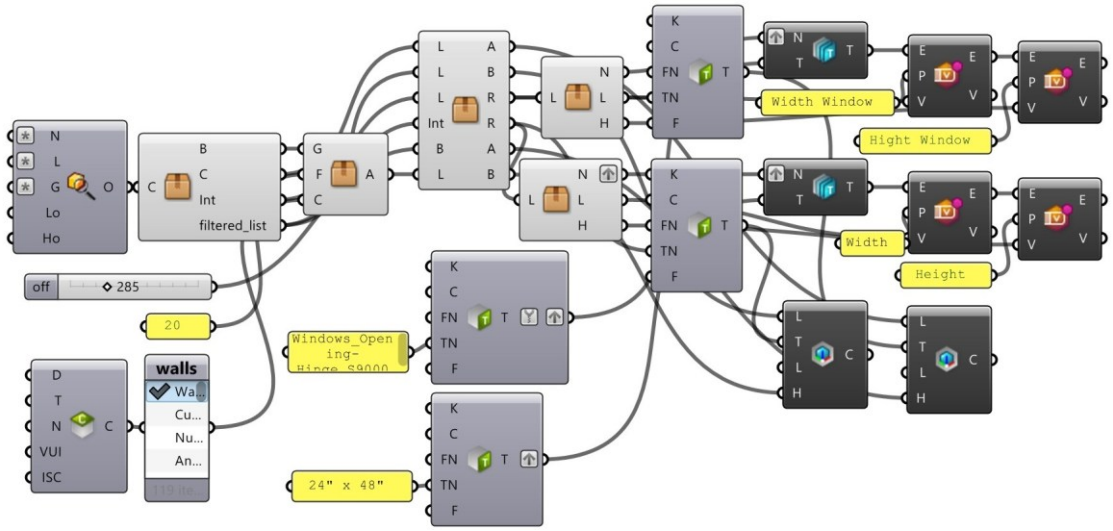

Fig. 7. Window/door creation script.

This script can be used to create both windows and doors, but it must be executed separately for each type.

4. Case study and discussion

To test the effectiveness of the script, a Rhino model of a building consisting of a ground floor and two storeys was used to generate the corresponding Revit model. To facilitate data processing in Grasshopper, the architectural elements were organized by layers in Rhino, with sublayers created for each window and door type. The Fig. 8 below shows a 3D view of the Rhino model.

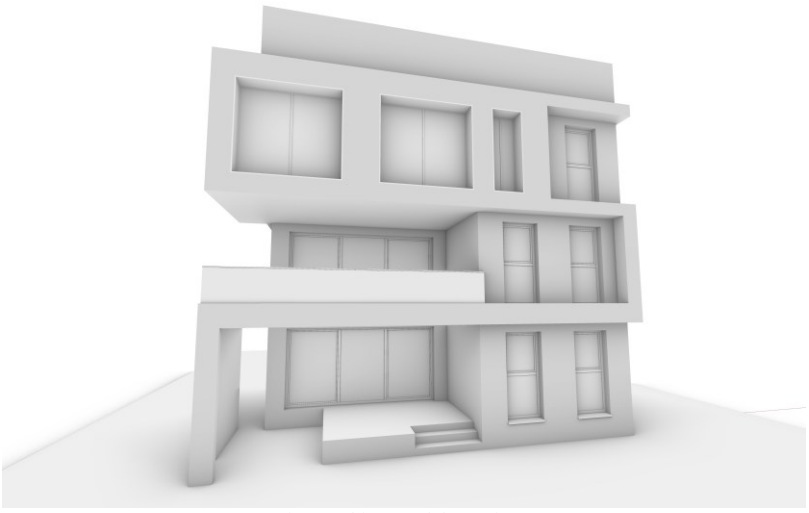

Fig. 8. Rhino model 3D view.

Using the proposed script, a Revit model was successfully generated, as illustrated in the Fig. 9 below, which depict the transformation from the Rhino model to the Revit model.

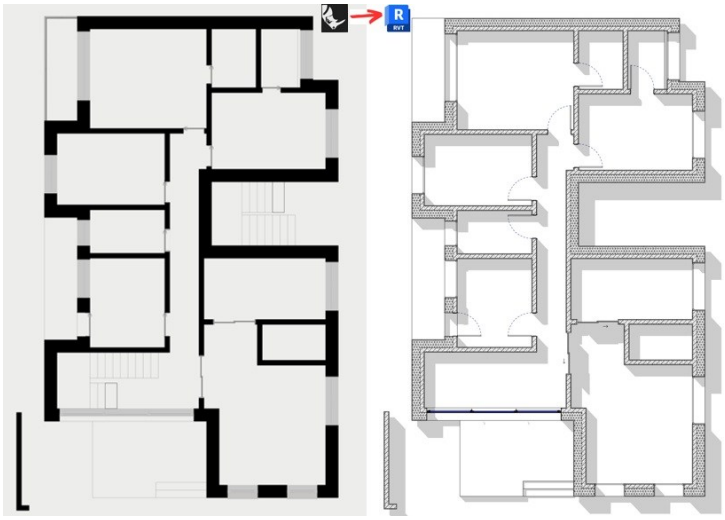

Fig. 9. Ground floor Rhino and Revit model.

As shown in the Fig. 9 above, all target elements (floors, walls, doors, and windows) were successfully generated in Revit. However, stairs were not generated.

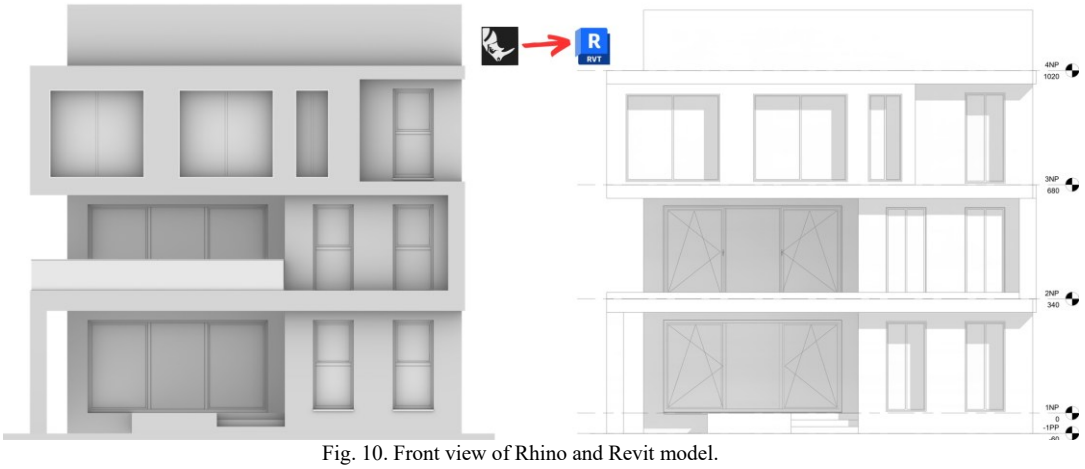

Fig. 10. Front view of Rhino and Revit model.

Additionally, the Revit levels were generated based on the floor references from the Rhino model, ensuring accurate level placement (see Fig. 10).

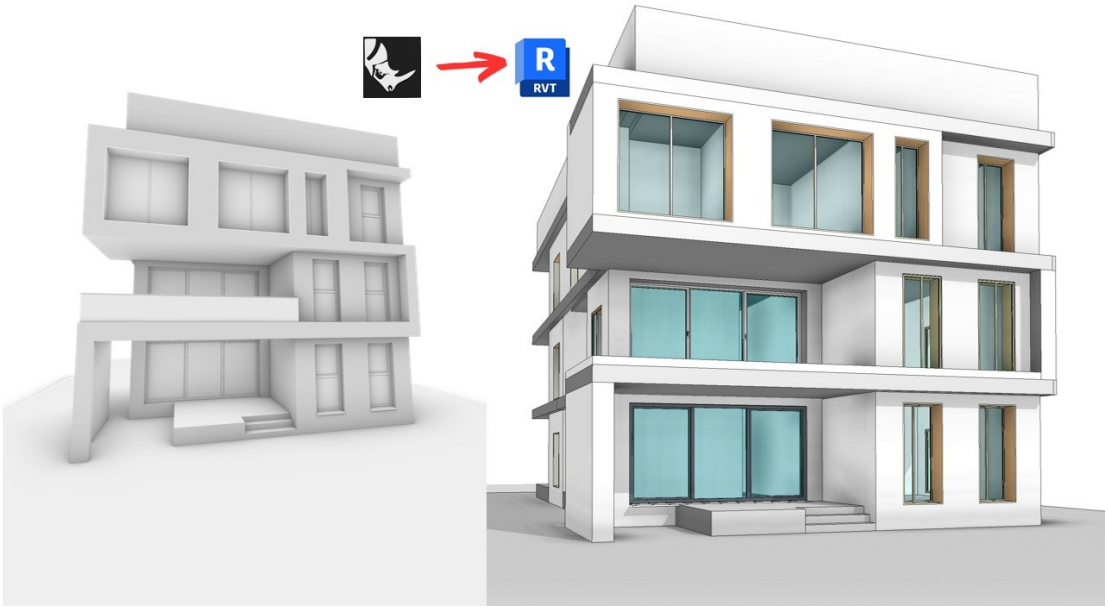

Fig. 11. 3D view of the Rhino and Revit model.

The Fig. 11 above presents a 3D comparison between the Rhino model and the generated Revit model. It is also observed that railings were not generated.

By generating the Revit model, the script demonstrates significant potential, as it allows architects to save time when exchanging data between Rhino and Revit. Additionally, the script can function as a bridge between Rhino and Revit, ensuring that any changes made in the Rhino model are automatically updated in the Revit model as long as the two remain linked through the script.

Despite its potential, the script has some limitations, such as its inability to generate certain elements like stairs and railings. Furthermore, at this stage of the research, we focused on simple geometric forms, and the script has not yet been tested on complex architectural geometries, such as double-curved surfaces.

5. Future Research

In future research, we aim to expand the script's capabilities and address its current limitations. Additionally, further improvements will focus on enhancing its ability to handle complex architectural geometries.

6. Conclusion

This research proposes a Grasshopper script to automate the exportation of architectural elements from Rhino to Revit. Utilizing RIR, the script successfully generates floors, walls, windows, and doors, enabling efficient data transfer between the two software programs. Additionally, it functions as a dynamic link, ensuring that modifications made in Rhino automatically update in the Revit model. While the script offers a valuable workflow for designers and architects, it has certain limitations, particularly in generating stairs and railings. Furthermore, the current version has been tested primarily on simple geometric shapes, and its ability to handle complex architectural geometries remains an area for future research. Future work will focus on improving the script's performance, addressing its limitations, and expanding its applicability to more advanced architectural designs. By refining this workflow, we aim to enhance Rhino and Revit's interoperability, making the process more seamless, robust, and automated for architects and designers.

References

1. Djuedja JFT, Karray MH, Foguem BK, Magniont C, Abanda FH. Interoperability Challenges in Building Information Modelling (BIM). In: Popplewell K, Thoben K-D, Knothe T, Poler R, editors. Enterprise Interoperability VIII. Cham: Springer International Publishing; 2019. p. 275–282. doi:10.1007/978-3-030-13693-2_23.
2. Shehzad HMF, Ibrahim RB, Yusof AF, Khaidzir KAM, Iqbal M, Razzaq S. The role of interoperability dimensions in building information modelling. Computers in Industry. 2021;129:103444. doi:10.1016/j.compind.2021.103444.
3. Ozturk GB. Interoperability in building information modeling for AECO/FM industry. Automation in Construction. 2020;113:103122. doi:10.1016/j.autcon.2020.103122.
4. Hu Z-Z, Zhang X-Y, Wang H-W, Kassem M. Improving interoperability between architectural and structural design models: An industry foundation classes-based approach with web-based tools. Automation in Construction. 2016;66:29–42. doi:10.1016/j.autcon.2016.02.001.
5. Andrade MLVXd, Ruschel RC. APPLICATION INTEROPERABILITY BIM USED IN ARCHITECTURE THROUGH THE FORMAT IF. GTP 2009. doi:10.4237/GTP.V4I2.102.
6. Martins JZ, Buzar MAR. Interoperabilidade no processo Building Information Modeling (BIM). Revista Científica Multidisciplinar Núcleo do Conhecimento. 2023:92–118. doi:10.32749/nucleodoconhecimento.com.br/arquitetura/building-information-modeling.
7. El-Khouly T, Abdelhalim O. Preserving conceptual design integrity: strategies for enhancing interoperability in architectural digital design workflows. Sci Rep. 2024;14:1–23. doi:10.1038/s41598-024-78640-8.
8. Khan H. Microclimatic architectural design by interfacing grasshoppers and Dynamo with Rhino and Revit. Measurement: Sensors. 2024;33:101143. doi:10.1016/j.measen.2024.101143.
9. Ismail AW, Marin P. Direct Interoperability between Industrial Engineering and AEC: Technical and Geometric Challenges. In: Pak B, Wurzer G, Stouffs R, editors. eCAADe 2022: Co-creating the Future - Inclusion in and through Design; 9/13/2022 - 9/16/2022; Ghent, Belgium: eCAADe; 2022. p. 537–546. doi:10.52842/conf.ecaade.2022.1.537.
10. Samadi S, Caffey S, Yan W. Parametric Modeling of Modular Structures for Designing Pemb Systems: Elsevier BV; 2024.
11. Komarov K, Kazarian B. Optimization Of The Development Of Student Architectural Projects With Help Of Technology Rhino.Inside®.REVIT. Ukrainian Academy of Arts. 2023:17–24. doi:10.32782/2411-3034-2023-33-2.
12. Karóczkai Á. Parametric BIM Models in ARCHICAD - The Grasshopper - ARCHICAD Live Connection. In: Kępczyńska-Walczak A, Białkowski S, editors. eCAADe 2018: Computing for a better tomorrow; 9/19/2018 - 9/21/2018; Łódź, Poland: eCAADe; 2018. p. 63–64. doi:10.52842/conf.ecaade.2018.1.063.
13. Chao Gao, Shengting Li, Zhenkai Wang. Research on Data Fusion Method of Revit and Non-BIM Software in Complex Shaped Buildings. J. Phys.: Conf. Ser. 2022;2202:12027. doi:10.1088/1742-6596/2202/1/012027.
14. Bedra KB, Zheng J, Li J, Sun Z, Zheng B. Automating Microclimate Evaluation and Optimization during Urban Design: A Rhino–Grasshopper Workflow. Sustainability. 2023;15:16613. doi:10.3390/su152416613.
15. Natalia Knyazeva, Vladislav Larin. Automation of repetitive tasks when designing an eco-friendly residential home. E3S Web Conf. 2021;244:5021. doi:10.1051/e3sconf/202124405021.
16. Lombardi A. Interoperability Challenges. Exploring Trends, Patterns, Practices and Possible Futures for Enhanced Collaboration and Efficiency in the AEC Industry. Coding Architecture. 2024:49–72. doi:10.1007/978-3-031-47913-7_3.