

POLITECNICO DI TORINO

Master of Science
in Building Engineering

Master's Degree Thesis

**Development of an automated workflow
integrating BIM and geological models for
geotechnical finite element analysis in tunnel
design**



Supervisor:

Prof. Marco Barla

Co-supervisors:

Ing. Giovanni Quaglio

Dr. Francesco Campana

Candidate:

Seyed Amir Mohammad Taghvaei

July 2026

*“Everything must be made as simple as possible,
but not simpler.”*

-Albert Einstein

Abstract

The architecture, engineering, and construction (AEC) industry is undergoing a major digital transformation through the widespread adoption of Building Information Modelling (BIM). However, the geotechnical and underground engineering sectors have struggled to keep pace. This delay is largely due to the highly irregular and complex nature of subsurface soil and rock layers, which are difficult to map and represent in standard BIM environments. Consequently, geotechnical data and structural designs often remain disconnected in "information islands." A primary hurdle is software interoperability—the inability to transfer complex 3D geological data into numerical simulation programs without losing critical parameters or experiencing geometric failures. This forces engineers to spend countless hours manually redrawing cross-sections for analysis.

This thesis addresses these challenges by developing an automated BIM-to-FEM interoperability workflow that directly links Revit-based tunnel models with Rocscience RS2 finite element analysis (FEA) software. The core of this research is an "algorithmic interoperability engine" built using Dynamo visual programming and Python scripting. Starting with an implicit 3D geological model (originally developed in Leapfrog) and a 3D structural tunnel alignment in Revit, the programmed workflow automatically extracts precise, perfectly orthogonal 2D cross-sections at critical chainages. To bridge the gap between BIM geometry and the strict topological requirements of FEA solvers, the pipeline utilizes Autodesk Civil 3D as an intermediate processing platform. Here, custom scripts automatically clean, trim, and convert the raw, fragmented CAD blocks into mathematically closed polylines while preserving the semantic geological data through color-coded layer mapping. Building upon this, the automation extends directly into the RS2 environment, where customized scripts handle the geometry importation, material property assignment, and the sequential assignment of composite liners.

To validate the framework, the automated workflow was applied to a real-world twin-tube mined tunnel project. The extracted and optimized cross-sections were successfully imported into RS2 without any of the topological or meshing errors typically associated with BIM-to-FEM transfers. A full 14-stage finite element simulation, utilizing the Generalized Hoek-Brown failure criterion and load relaxation methods, was then executed to verify the structural integrity

of the rock mass and the composite support lining. The results confirm that this automated pipeline eliminates the need for manual geometric recreation, prevents data loss, and establishes a highly efficient, practical foundation for advanced numerical modelling in underground infrastructure.

Acknowledgment

As this chapter of my life comes to an end, I feel the need to thank the people who have supported me throughout these years and helped shape who I am today.

First and foremost, I would like to express my deepest gratitude to Professor Marco Barla from Politecnico di Torino, who has supported me throughout the development of my thesis. Every meeting we had was a booster for my work, both personal and academic. It was also thanks to his reputation that I was able to do my thesis at TYLin.

Next, I would like to thank Ing. Giovanni Quaglio and the entire TYLin team for providing me with the resources and the opportunity to conduct my master's research with them. Looking back, it hardly feels like I spent 396 coffee breaks (6 months) in the office with you guys, working on this thesis. Despite the immense amount of work going on in the office, Giovanni and the team were always available whenever I had a question.

I would also like to thank Dr. Francesco Campana for his patience and support in navigating the technical challenges I encountered during my research.

It would be unfair of me not to mention my friends both in Italy and around the world and the endless support they had for me during the hard times. I also want to give a special mention to my friends on the third floor of my university residence who I met at the same time I started my thesis and the memories we created together.

Finally, none of this would have been possible without my family: my mother, my father, and my brother. Their unwavering support was the fuel that kept me moving forward. I now deeply understand the hard work my parents had to put up with when they were pursuing their PhDs outside their home country, raising my older brother and me while we were completely unaware of the hardships they faced. I will never forget the days when my mom would take my brother and I to school before traveling a long distance to her university.

Table of Contents

Abstract.....	i
Acknowledgment.....	iii
Table of Contents	iv
Table of Figures.....	vi
List of Tables	xiii
Chapter 1: Introduction	1
1.1 Background and problem statement.....	1
1.2 Research objectives and scope	1
1.3 Key research questions.....	2
Chapter 2: From GeoBIM to numerical analysis for underground infrastructure	3
2.1 Building Information Modelling (BIM) in geotechnical engineering.....	3
2.2 The concept of GeoBIM.....	5
2.3 The BIM to numerical modelling integration for automated workflows (BIM – FEM) 10	
2.3.1 Challenges in data standardization: The role of IFC in geotechnical engineering	15
2.4 Visual programming languages in BIM	18
2.4.1 Challenges of IFC and neutral formats in VPL workflows	22
2.4.2 Python-driven API interoperability for 2D geotechnical analysis.....	23
2.5 From digital models to digital twins	25
Chapter 3: Tunnel case study.....	28
3.1 Introduction and project overview	28
3.2 3D model of the tunnel.....	28
3.2.1 Tunnel cross-sectional configuration.....	30
3.3 Geological soil model.....	31
3.3.1 Stratigraphic discretization and color-coding schema	31
Chapter 4: Architecting the BIM – FEM workflow	33

4.1 Overview and methodological framework.....	33
4.2 Phase 1: Geometric & geological baseline.....	35
4.3 Phase 2: Algorithmic interoperability engine.....	40
4.3.1 Automated 2D section generation	41
4.3.2 Algorithmic discretization and mathematical setup.....	43
4.3.3 Algorithmic execution of the geological radar	46
4.3.4 Generation of orthogonal section views and boundary conditions.....	51
4.3.5 DXF export and data transfer from Revit to Civil 3D	58
4.3.6 Automated cleaning in Civil 3D	61
4.3.7 External boundary generation for RS2 import.....	64
4.3.8 Geological layer classification and naming standardization.....	68
4.3.9 Dynamic legend generation and visualization	70
4.3.10 Excavation boundary tunnel isolation	72
.....	74
4.3.11 Final export for RS2 interoperability (DXF 2013)	74
4.4 Phase 3: Analytical extraction & simulation.....	75
4.4.1 The finite element framework and Rocscience RS2.....	75
4.4.2 Interoperability validation and automated model initialization	78
4.4.3 Parameter injection and model initialization	83
4.4.4 Programmatic formulation of structural support systems.....	86
4.5 Summary of the interoperability framework.....	89
Chapter 5: Workflow validation and numerical results	91
5.1 Automated section extraction across varying chainages.....	91
5.2 Finite element simulation and mesh generation in RS2.....	94
5.2.1 Geometry importation and material assignment.....	94
5.3 Mesh generation and staged excavation analysis of the 635.0-meter chainage	96
5.3.1 Project settings and stress analysis	98
5.4 Boundary conditions, discretization, and numerical computation	101

5.4.1 Implementation of the load relaxation method	103
5.4.2 Sequential support installation and composite liners.....	105
5.5 Analysis of computational results	108
5.5.1 Verification of the initial geostatic state.....	108
5.5.2 Evolution of ground displacements and surface settlement.....	109
5.5.3 Analysis of rock mass yielding and plastic zone development.....	111
Chapter 6: Conclusion and future development.....	113
References	117
Appendix A.....	121
A.1 Contents.....	121
A.2 Revit and Dynamo extraction script.....	121
A.3 Civil 3D geometric optimization script	125
A.4 Automated finite element configuration script.....	131

Table of Figures

Figure 1. The top surfaces of each stratum considered in the study (Menaka & Sivakumar Babu, 2025)	4
Figure 2. The 3D view of the geotechnical BIM model (Menaka & Sivakumar Babu, 2025).....	4
Figure 3. A typical sectional view from the 3D subsurface model (Menaka & Sivakumar Babu, 2025)	5
Figure 4. Workflow N.1 (Fonsati, Cosentini, Tundo, & Osello, 2023).....	6
Figure 5. Workflow N.2 (Fonsati, Cosentini, Tundo, & Osello, 2023).....	7

Figure 6. Workflow N.3 (Fonsati, Cosentini, Tundo, & Osello, 2023).....	8
Figure 7. Workflow N.4 (Fonsati, Cosentini, Tundo, & Osello, 2023).....	8
Figure 8. Leapfrog model (Geological interpretation and BIM). (Lai, 2023).....	9
Figure 9. Initial and optimised design of Pymont Station adits (Lai, 2023).	10
Figure 10. Interoperability of BIM-FEM-model updating (Talebi, Potenza, & Gattulli, 2023)	11
Figure 11. Data mapping schemas for manual geometry-only transfer (DXF) vs. automated parametric data exchange (Dynamo/MATLAB). (Talebi, Potenza, & Gattulli, 2023).....	11
Figure 12. Automated BIM-to-FEM integration workflow using python scripting (Shi, Jin, Lu , & Shi , 2023)	12
Figure 13. BIM - FEM framework based on automated data transfer (Huang, Ruan, Wu, & Qin, 2024)	13
Figure 14. Fully automated BIM-FEM conversion framework (Huang, Ruan, Wu, & Qin, 2024).....	14
Figure 15. Comparison of BIM model, mesh model, and numerical simulation model: (a) BIM model; (b) mesh model in Hypermesh; (c) numerical simulation model.....	14
Figure 16. IFC extended EXPRESS-G diagram of the geotechnical information model (Wu, et al., 2021)	16
Figure 17. The EXPRESS-G diagram of the geological information model based on extended IFC (Wu, et al., 2021).....	17
Figure 18. A Geological model diagram done in Dynamo (Huang, Ruan, Wu, & Qin, 2024).....	18
Figure 19. Parameterized modelling process of 3D geological BIM model: (a) Detailed borehole geology information; (b) Generating stratigraphic surface models; (c) Creating geometric entities using terrain boundaries; (d) Geological models after excavation and partition. (Huang, Ruan, Wu, & Qin, 2024)	19
Figure 20. A shield tunnel diagram in Dynamo (Huang, Ruan, Wu, & Qin, 2024).....	19
Figure 21. Overall model of shield tunnel and details of the geometry (Huang, Ruan, Wu, & Qin, 2024)	20
Figure 22. Layers of the project created in Revit using Dynamo (Alaei, 2023).....	21
Figure 23. with visual coding techniques in Dynamo open-source software, the profile of the tunnel is assigned to the tunnel alignment to be exported to the Revit software (Al Furjani, et al., 2020)	21

Figure 24. Python application for retrieving the ECI value based on the localID of a Revit element (on the left). Thematic plan for visualising environmental assessment results in the IFC model (on the right) (Marmo, Polverino, Nicoletta, & Tibaut, 2020).....	23
Figure 25. DT development timeline (Wang, Rezaei, & Hicks, 2025)	25
Figure 26. Conceptual representation of (a) digital model, (b) digital shadow, and (c) digital twin. (Babanagar, Sheil, Nini’c, Zhang, & Hardy, 2024)	26
Figure 27. Keywords co-occurrence network (Wang, Rezaei, & Hicks, 2025)	26
Figure 28. Challenges and causes for DT adoption in geotechnical engineering (Wang, Rezaei, & Hicks, 2025)	27
Figure 29. Three-dimensional integrated IFC Tunnel model.....	29
Figure 30. 3D DWG model of the tunnel.....	29
Figure 31. Structural Cross-Section of the mined Tunnel.....	30
Figure 32. 3D Geological Case study model.....	31
Figure 33. West elevation discretization illustrating the preserved color-coded stratigraphic units imported via IFC.	32
Figure 34. Methodological framework illustrating the automated Pipeline.....	33
Figure 35. Default metric template from Revit	35
Figure 36. Setting project units in Revit.....	36
Figure 37. Utilizing the Link IFC tool.	37
Figure 38. Georeferencing using “Specifying Coordinates at Point” function	37
Figure 39. Identification of the localized reference point within the geological model.....	38
Figure 40. Selection of the "IFC Origin to Project Base Point" positioning	38
Figure 41. Tunnel + Soil (Open-BIM) Model.	39
Figure 42. Underlay Soil+Tunnel Model	39
Figure 43. Dynamo tab in Revit.....	42
Figure 44. The default blank Dynamo workspace.....	42
Figure 45. 3D Tunnel Curve of the Tunnel IFC model.....	43
Figure 46. Tunnel curve correctly loaded in Dynamo.....	44

Figure 47. Dynamo node sequence for defining the spatial discretization intervals along the tunnel curve.	44
Figure 48. “Curves.ParameterAtSegmentLength” node converting real-world meter intervals into relative mathematical parameters.....	45
Figure 49. Dynamo inputs connected to Python	46
Figure 50. Importing required Revit and Dynamo libraries.	47
Figure 51. Data collection and virtual raycasting.....	48
Figure 52. Python script for the simulated geological radar part 1	48
Figure 53. Python script for the simulated geological radar part 2	49
Figure 54. Final script phase identifying and labelling critical geological sections.....	50
Figure 55. Math behind the 2D sections in Dynamo.....	50
Figure 56. 2D Section Creation Group in Dynamo.....	51
Figure 57. 2D Section Curve Parameter Node Inputs	51
Figure 58. Integration of the calculated origin point and custom axes to generate the dynamic local coordinate system.....	52
Figure 59. Code Block for creating the bounding box	53
Figure 60. Wiring configuration for the final generation of the section views	54
Figure 61. Automated application of the view name and view template properties	55
Figure 62. Complete structural layout of the Dynamo script for automated section generation.....	55
Figure 63. Automated 2D cross-section in Revit representing the maximum soil overhead at the 375.0- meter mark	56
Figure 64. Sequential organization of the automated cross-sections along the tunnel alignment within the project browser.....	56
Figure 65. The Dynamo player interface displaying the adjustable discretization step	57
Figure 66. Configuration of the rule-based filters using the unique IfcGUID of the imported soil solids	59
Figure 67. Application of the original geological colors to the soil layers in the View Template.....	59
Figure 68. Export Setups DWG/DXF	60
Figure 69. DXF export layers configuration	61

Figure 70. The exported DXF section successfully opened in Civil 3D, displaying the preserved layer organization and color coding.....	61
Figure 71. Opening a python script node in Dynamo	63
Figure 72. The complete Dynamo automation workflow in Civil3D.....	64
Figure 73. LINEWORKSHRINKWRAP command in Python for Dynamo in Civil3D	65
Figure 74. EXPLODE and HATCHGENERATEBOUNDARY Command in Python	65
Figure 75. Analytical CAD polylines (highlighted) automatically generated from the visual soil hatch patterns	66
Figure 76. OVERKILL and PEDIT commands in Python for the external boundary	67
Figure 77. Isolated view of the generated external soil boundary polyline, with internal soil layers hidden for clarity.....	67
Figure 78. The second stage of the Dynamo workflow, utilizing the 'Layer Naming' Python node.....	68
Figure 79. The Python dictionary array utilized to map extracted RGB color values to standardized geological layer names.....	69
Figure 80. The updated Civil 3D Layer Properties Manager demonstrating the successfully renamed geological layers.....	69
Figure 81. The third stage of the Dynamo workflow, using the 'Legend' Python node to dynamically draft a visual reference guide.	70
Figure 82. Bounding box python script for the legend creation.....	71
Figure 83. Final output of the 'Legend' script, showing the active soil types.....	71
Figure 84. The final stage of the Dynamo workflow, utilizing the 'Tunnel_OuterLayer' Python node	72
Figure 85. Double EXPLODE and JOIN Commands in python for "Tunnel Outer Layer"	72
Figure 86. AutoLISP script utilizing perimeter length evaluation	73
Figure 87. Final output of the Dynamo workflow, displaying the isolated outer tunnel boundaries, standardized geological layers, and dynamic legend ready for RS2 import.	74
Figure 88. Exporting the optimized Civil 3D cross-section to 2013 DXF format	74
Figure 89. Rocscience RS2 user interface.....	78
Figure 90. Rocscience RS2 import formats.....	78
Figure 91. RS2 displaying raw 2D cross section import from Revit.....	79
Figure 92. Flawless DXF geometry import inside RS2	80

Figure 93. The RS2 DXF import dialog, demonstrating the direct assignment of optimized CAD layers to their corresponding geotechnical boundary types	81
Figure 94. Initial DXF import in RS2, displaying the recognised material, external, and excavation boundaries	82
Figure 95. Scripting server port connector	83
Figure 96. Python script illustrating the connection to RS2 via a server port.....	83
Figure 97. Python script for extracting raw geomechanical parameters from an external CSV database	84
Figure 98. Generalized Hoek-Brown parameters script.....	85
Figure 99. Python script demonstrating the insertion of Generalized Hoek-Brown parameters directly into their fields	85
Figure 100. Python script showing the automated setup of the tunnel liners	86
Figure 101. RS2 property dialog for the primary Shotcrete liner support. RS2 property dialog for the primary Shotcrete liner support.....	87
Figure 102. Python script demonstrating the automated setup of a two-layer composite liner.....	88
Figure 103. The RS2 Define Composite Properties dialog	89
Figure 104. Automated 2D cross section at the 235.0-meter chainage	92
Figure 105. Automated 2D cross section at 60-meter chainage	92
Figure 106. Automated cross-section at 600.0m chainage.....	93
Figure 107. Automated cross-section at 635.0m chainage.....	94
Figure 108. RS2 interface displaying the automated initialization of the 375.0-meter cross-section	95
Figure 109. Strength material properties tab	95
Figure 110. Composite liner properties panel in RS2 configured automatically	96
Figure 111. The 635m chainage cross-section imported into RS2 after being processed in Civil 3D ...	97
Figure 112. The RS2 material properties dialog	97
Figure 113. The RS2 Material Properties dialog, illustrating the strength parameters based on the Generalized Hoek-Brown failure criterion.....	98
Figure 114. Project Settings dialog for Stress Analysis	98
Figure 115. The 14 construction stages listed in RS2	99

Figure 116. The imported cross-section with standard displacement boundary conditions applied	101
Figure 117. The RS2 mesh setup dialog.....	102
Figure 118. The generated finite element mesh	102
Figure 119. Stage 2, showing the geostatic factor 1 with 100% internal pressure	103
Figure 120. Tunnel and rock mass parameters defined in RocSupport.....	104
Figure 121. Support parameters in RocSupport, detailing the shotcrete properties at the distance of 1m from the face.	104
Figure 122. The ground reaction curve generated by RocSupport.....	105
Figure 123. The Define Liner Properties dialog.....	106
Figure 124. The “Define Composite Properties” dialog along with the crown and invert liner shown	106
Figure 125. Stage 5 (Crown Liner), showing the activation of the support elements along the top heading	107
Figure 126. Computing (Run) the numerical analysis.....	108
Figure 127. Stage 1 Major effective stress (σ_1) contour plot.....	109
Figure 128. Stage 1 Minor effective stress (σ_3) contour plot	109
Figure 129. Total displacement contour map after the completion and final lining of Tunnel 1 (Stage 7).	110
Figure 130. Final total displacement contour map of the twin-tunnel system at the completion of construction (Stage 14).....	110
Figure 131. Distribution of yielded elements around Tunnel 1 at the completion of Stage 7.....	111
Figure 132. Final distribution of yielded elements across the twin-tunnel system at Stage 14.....	112
Figure A.1 The complete Dynamo visual programming graph utilized for automated geometric extraction and cross-section generation within the Revit environment.....	122
Figure A.2. Python source code executed within the Revit Dynamo environment.....	123
Figure A.3. External boundary script in Dynamo	125
Figure A.4. Layer naming in Civil 3D	128
Figure A.5. Legend in Civil3D.....	129
Figure A.6. Tunnel outer layer	130

Figure A.7. The RS2 Python API automation script	132
--	-----

List of Tables

Table 1.Comparative utility of exchange formats (Colajanni, et al., Architectural and Structural Interoperability in the BIM Design Workflow, 2025)	24
Table 2. Geomechanical Soil Parameters.....	32
Table 3. Summary of the 14-stage finite element calculation sequence	100
Table A.1 Summary of automation scripts developed for the workflow	121
Table A.1 Summary of automation scripts developed for the workflow	121

Chapter 1: Introduction

1.1 Background and problem statement

The architecture, engineering, and construction (AEC) industry is currently experiencing a major digital transformation, moving away from traditional two-dimensional drawings toward more advanced three-dimensional digital models. Today, more than 70% of construction projects in North America make use of Building Information Modelling (BIM), highlighting how widely these technologies have been adopted across the industry. (Szwarkowski & Pilecka, 2017)

However, the geotechnical and underground engineering sectors have not progressed at the same pace. One of the main challenges lies in subsurface conditions: soil layers are often irregular, complex, and difficult to accurately represent within digital modelling environments. As a result, achieving a reliable understanding of the ground remains a critical issue. This limitation can have serious implications for construction projects, as inadequate site characterization is a major factor behind unexpected costs, schedule overruns, and project delays. In fact, such issues continue to affect more than 70% of public infrastructure projects. (Szwarkowski & Pilecka, 2017)

One of the main obstacles preventing further progress in underground engineering is the challenge of “interoperability”, which dictates the ability of different software platforms to exchange data efficiently without losing important information.

1.2 Research objectives and scope

Underground engineering projects frequently face inefficiencies because geological data, structural designs, and numerical analyses are often disconnected. This separation typically leads to manual rework and inconsistent data across different project disciplines. To address this gap, this thesis aims to develop an open BIM – FEM workflow that links Revit-based tunnel models directly with numerical analysis simulations. The primary goal is to support design and

performance simulation during the planning and design phases, ensuring that any structural or geological changes are efficiently coordinated.

The core of this research focuses on automating the generation of geometry and parametric data using Revit and Dynamo. Starting with a predefined 3D geological model prepared in Leapfrog as the foundational input, the BIM model will act as a coordination layer that simplifies complex geological information into practical, design-relevant zones. To ensure seamless interoperability between the BIM environment and advanced numerical tools like Rocscience (RS2), the research will explore various data mapping strategies, including custom Dynamo and Python interfaces, as well as potential IFC-based extensions.

Furthermore, this research clearly defines its position on the interoperability workflow. While advanced infrastructure systems require bidirectional, real-time data exchange, the current industry bottleneck lies in the initial translation of the design model into an analysis-ready state. Therefore, this thesis focuses on establishing an automated, unidirectional data pipeline from the BIM environment to the numerical simulation tool. By replacing manual, static model recreation with a scripted, forward-feeding workflow, this research provides the foundational architecture necessary for advanced BIM-to-FEM development.

To validate this workflow and ensure it meets practical industry needs, the framework will be applied to a real-world infrastructure case study. This case study will supply the necessary tunnel alignment, general design information, and boundary conditions, all of which will be handled with strict academic confidentiality. Ultimately, this thesis will deliver a parametric pipeline and a prototype interoperability schema that can feed 2D geological sections into numerical modelling software for interactive analysis. By establishing clear guidelines for managing geological data within BIM, this research seeks to provide a practical, reusable workflow that reduces duplicated modelling efforts and enhances design iteration compared to traditional tunnel analysis methods."

1.3 Key research questions

To guide the development and evaluation of this framework, this thesis addresses the following primary questions:

1. How can Revit and Dynamo be used to automate the generation of geometry and parametric data for numerical analyses?
2. What data mapping strategies (and possible IFC-based extensions) can ensure interoperability between BIM software and numerical analysis tools like Rocscience RS2?
3. How can a BIM-based interoperability framework improve coordination, traceability, and design iteration compared to traditional tunnel analysis workflows?

Chapter 2: From GeoBIM to numerical analysis for underground infrastructure

This chapter examines the shift in underground construction from traditional 2D drawings toward modern 3D digital models. It looks at how the transition of Building Information Modelling (BIM) into "GeoBIM" helps engineers map unpredictable soil layers and share this data clearly across project teams. A primary focus is the challenge of "interoperability," which ensures different software platforms can exchange data without losing critical information. Establishing this connection is what allows a static design to evolve into a dynamic, analysis-ready model that updates automatically as new data is received. Finally, the chapter discusses how visual programming language, such as Dynamo, serve as a bridge to automate manual modelling tasks, increasing both the speed and accuracy of the engineering process.

2.1 Building Information Modelling (BIM) in geotechnical engineering

The use of Building Information Modelling (BIM) in geotechnical engineering is a major change from old 2D methods to modern 3D modelling. While over 70% of construction projects in North America already use BIM, the geotechnical industry has struggled to keep up because soil layers are often irregular and difficult to map. In 1989, a standard called AGS was created to share digital data, but it was limited because it could not easily show 3D surfaces of the ground. Modern BIM solves this by letting engineers attach specific data—like soil strength and water flow—directly to 3D shapes. By combining ground data with building designs into one "global model," teams can avoid unexpected costs and delays, which affect over 70% of public projects when the ground is not properly understood. (Szwarkowski & Pilecka, 2017)

According to (Erharter, Weil, Bacher, Heil, & Kompolschek, 2023), the BIM method has seen significant application in infrastructure projects within the "DACH countries," (Germany, Austria, Switzerland) driven largely by government agencies and major infrastructure owners. This transition has moved the fields of engineering geology and geotechnics toward digital 3D modelling as a common practice.

While BIM technology is advancing globally, it isn't moving at the same pace everywhere. In India, for example, using BIM to manage soil and ground data is still very rare. This is largely because the ground itself is so difficult to map; tropical soils are shaped by heavy weathering, making them incredibly unpredictable. Since the soil's strength and moisture can change drastically from one spot to the next, it is hard to create a digital model that feels accurate. To make matters worse, India lacks a central database for soil records. Because the information is scattered and stored in different ways, engineers must spend a lot of time just cleaning up the data before they can even use it (Menaka & Sivakumar Babu, 2025).

In the case study of deep excavation in south India, (Menaka & Sivakumar Babu, 2025) demonstrate a Geotechnical BIM workflow for a 20-meter-deep excavation in an urban area of South India. The project was done in Autodesk Civil 3D 2026 with the Geotechnical Modeller extension to process data from 18 boreholes as seen in Figure 1. The top surfaces of each stratum considered in the study (Menaka & Sivakumar Babu, 2025)

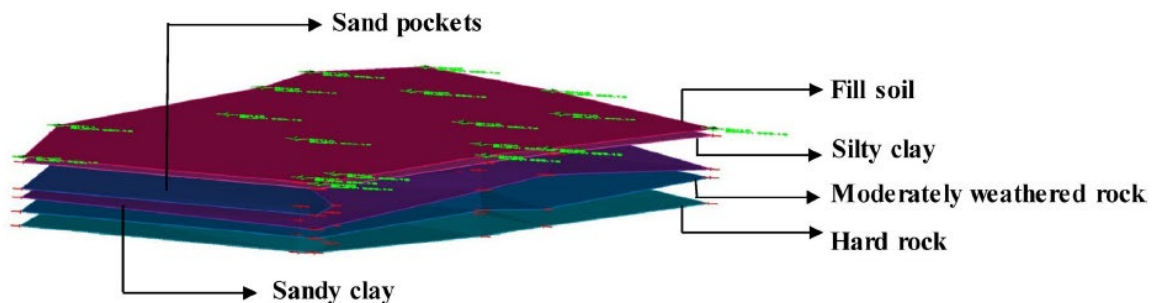


Figure 1. The top surfaces of each stratum considered in the study (Menaka & Sivakumar Babu, 2025)

Furthermore, the geotechnical model was combined with a structural model in Autodesk Revit 2026. The diaphragm wall and the neighbouring buildings with foundations were modelled using Revit software. The model provides a more reliable database for future reference and minimises the risk of unforeseen subsurface information during the design phase. The model is shown in Figure 2.

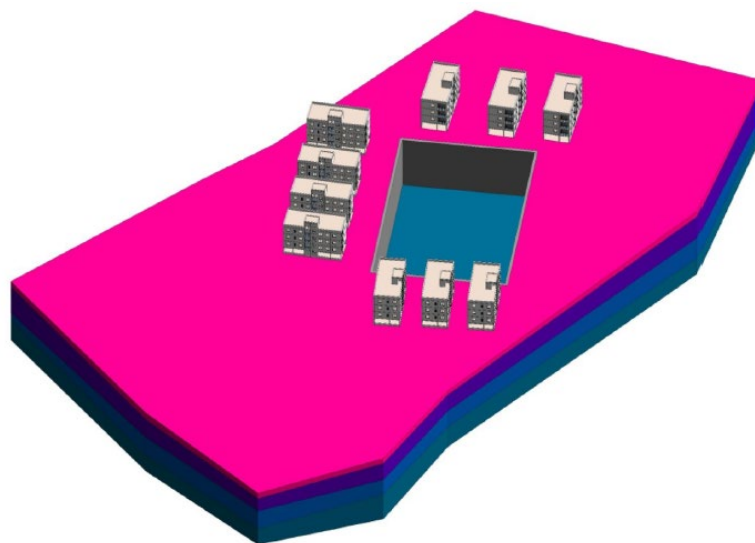


Figure 2. The 3D view of the geotechnical BIM model (Menaka & Sivakumar Babu, 2025)

A few 2D sections considered in Figure 3, demonstrating the developed geotechnical BIM model to be viewed through different sections to better understand the actual field conditions. (Menaka & Sivakumar Babu, 2025)

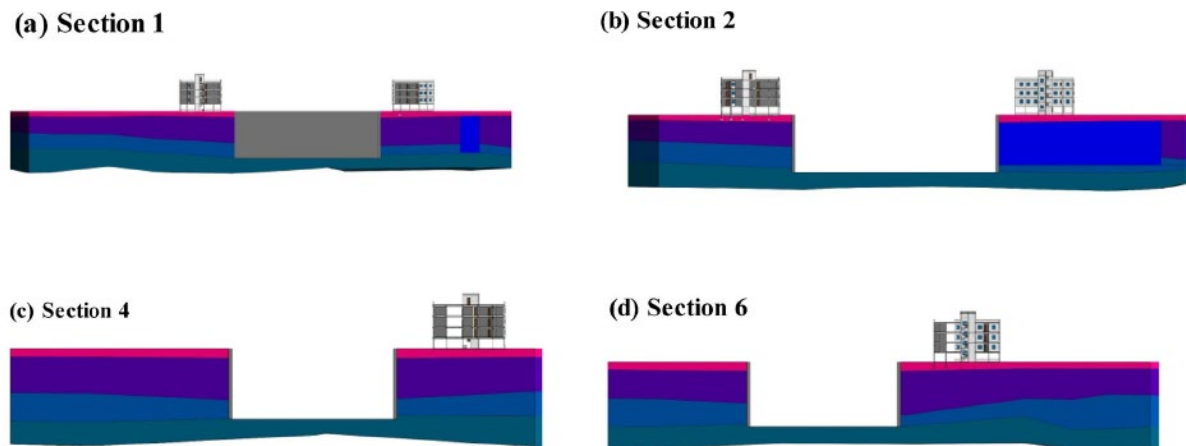


Figure 3. A typical sectional view from the 3D subsurface model (Menaka & Sivakumar Babu, 2025)

Furthermore, the use of BIM in geotechnical engineering can enhance collaboration and information sharing among the various stakeholders involved, including engineers, contractors, and project managers. By creating a centralized repository of project data, BIM enables seamless communication and coordination, reducing the potential for errors and delays. (Menaka & Sivakumar Babu, Building information modelling in geotechnical engineering, 2025)

Building Information Modelling (BIM) for the ground, often called "geotechnical BIM," is an important way to lower construction risks from unknown soil conditions. While standard BIM is mostly used for buildings above the surface, applying it to geotechnics allows teams to save and share complex underground data in a single 3D file. This technology makes it possible to visualize the entire soil profile, which helps engineers find features like geological faults or specific earth masses that are often missed when looking at traditional 1D borehole reports. By using this collaborative approach, project teams can better control costs and understand the site. However, combining this data is still difficult because soil information is often stored in many different file formats, and in some regions, it is still shared only on paper or as PDFs, which requires slow and difficult manual work to digitize. (LeeLee, et al., 2021)

While existing research has successfully modelled soil strata in specific regions, there is still a significant challenge in maintaining an "unbroken digital information supply chain" when moving models into a BIM environment like Revit.

2.2 The concept of GeoBIM

GeoBIM has developed along two main paths. At first, past researchers explored how to combine Building Information Modelling (BIM) with Geographic Information Systems (GIS), looking at how a building or asset fits into its surrounding city or landscape. More recently, the

focus has started to shift towards what's underground, with GeoBIM now often describing a system where detailed information about subsoil layers is used to create a digital model that works directly with BIM processes. (Fonsati, Cosentini, Tundo, & Osello, 2023)

The application of GeoBIM concepts used in (Fonsati, Cosentini, Tundo, & Osello, 2023) were demonstrated as part of the New Headquarters of the Piedmont Region in Turin, Italy. By bringing together geological, structural, and monitoring data in one place, the project team was able to build an "interpreted model." This made it possible to coordinate the deep foundation design with the challenging subsoil conditions at the Turin site in real time, ensuring the plans were always in sync with what's happening underground.

In (Fonsati, Cosentini, Tundo, & Osello, 2023) research, four different workflows were used to integrate geotechnical data into the BIM environment.

The first workflow follows a GIS-based strategy, with the Geographic Information System serving as the main hub for storing all ground data, such as borehole locations and site topography. While the goal is to move this information into a BIM environment to create 3D surfaces, a major issue is that no standard exchange file format exists to seamlessly link these two systems. Because there is no easy way to export and import the data, the digital information chain often breaks, making it difficult for engineers to maintain a smooth flow of information between the GIS database and the BIM design software. (Fonsati, Cosentini, Tundo, & Osello, 2023)

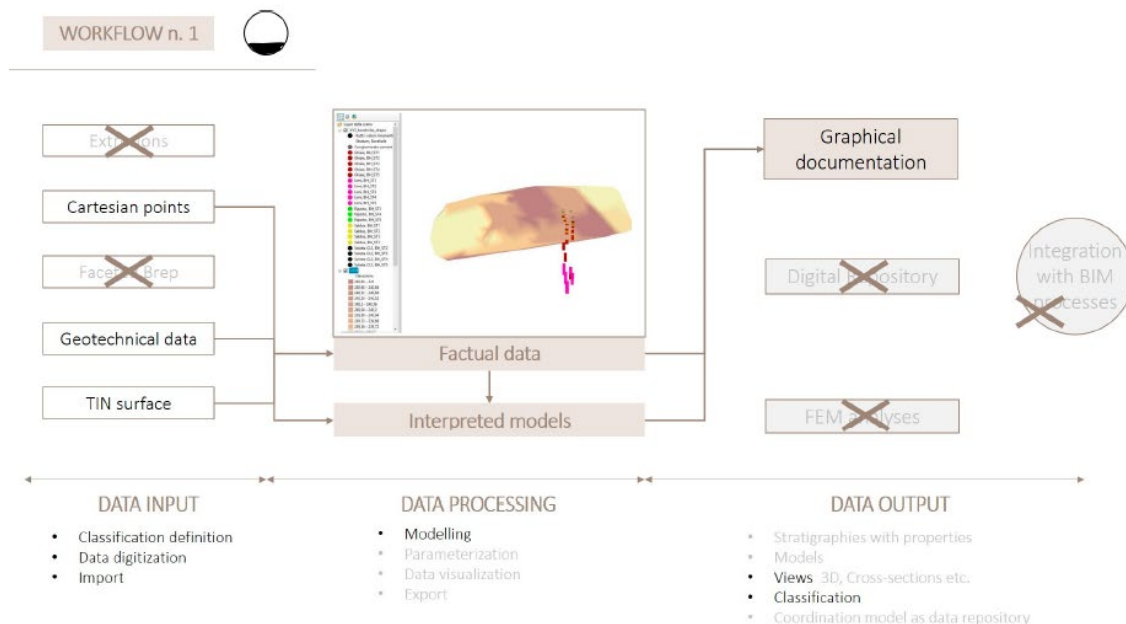


Figure 4. Workflow N.1 (Fonsati, Cosentini, Tundo, & Osello, 2023)

The second practice shown in Figure 5 focuses on a BIM-based strategy that utilises Rockworks 2023 for modelling underground stratigraphy. This software allows for the three-dimensional visualization of boreholes, topographic surfaces, and subsoil layers, resulting in a geo-model with distinct stratigraphic units. A significant limitation of this specific workflow is its static nature, as the modelling process cannot be easily altered or updated once the 3D model is generated. To move this data into the BIM environment, the model is exported using DXF and DAE file formats. While importing the DXF file into Autodesk Civil 3D successfully preserves both the geometry and the associated geotechnical attributes, the DAE format is more limited, as it only maintains the geometric shape without preserving the underlying data attributes (Fonsati, Cosentini, Tundo, & Osello, 2023).

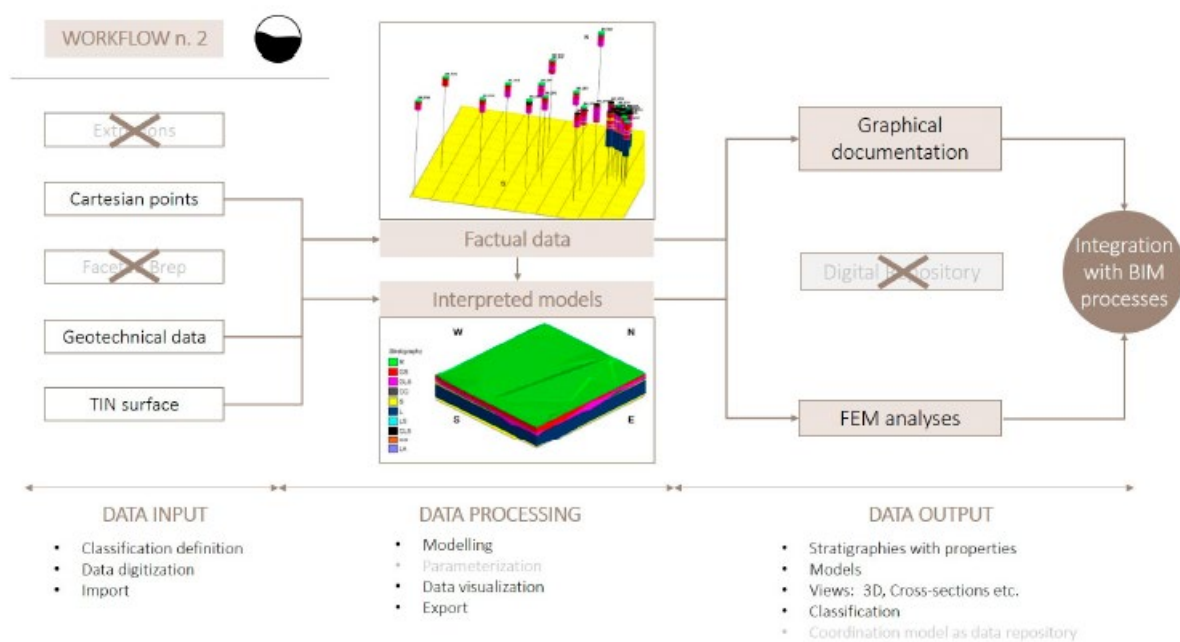


Figure 5. Workflow N.2 (Fonsati, Cosentini, Tundo, & Osello, 2023)

The third method used by (Fonsati, Cosentini, Tundo, & Osello, 2023), involved the “Geotechnical Module” plugin, designed for Autodesk Civil 3D 2023 exclusively for geological /geotechnical soil modelling using in situ borehole data. This method is different because it allows engineers to import borehole data directly from Microsoft Excel or specialized AGS files right into the design software.

The big advantage here is that the software can automatically create 3D surfaces and "sticks" that show the different soil layers based on the actual borehole data. Unlike the second method, this workflow is much more flexible because if the borehole information changes, the 3D model can be updated easily. However, one small downside mentioned in Fonsati’s paper is that this tool is specifically designed for Civil 3D, so the 3D models it creates are mostly surfaces rather than solid 3D volumes.

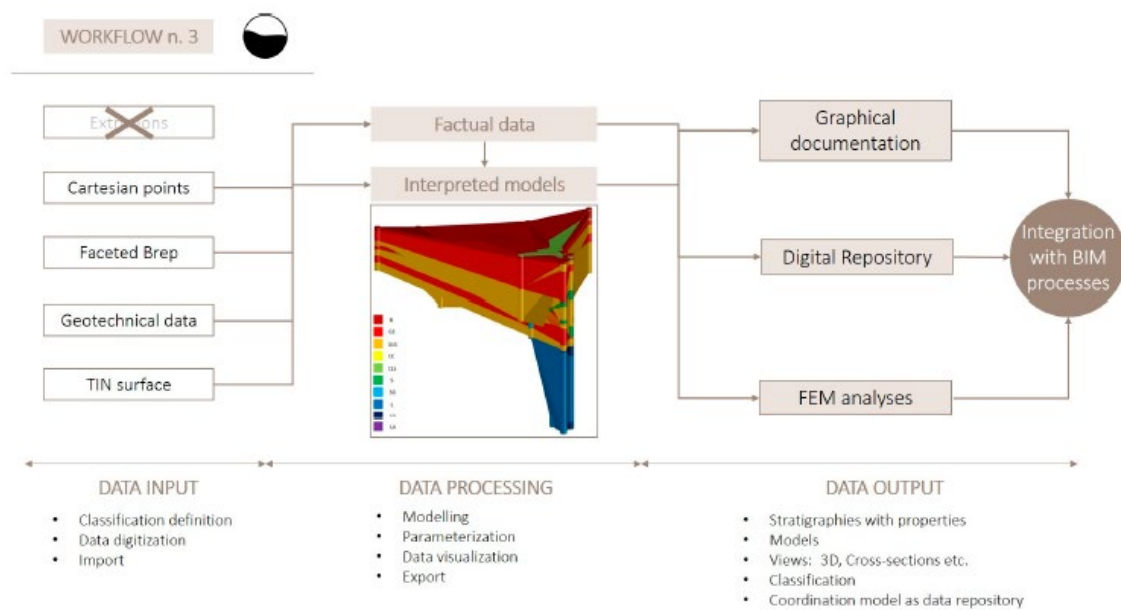


Figure 6. Workflow N.3 (Fonsati, Cosentini, Tundo, & Osello, 2023)

The fourth and final strategy utilized the Novapoint 21 software by Trimble to establish a collaborative environment that allows multiple professionals to work simultaneously on the same project as seen in Figure 7.

In contrast to the previous approaches, this method aims to simulate and manage geotechnical models directly within a BIM-integrated platform. To facilitate this, a specialized Civil 3D Connector was employed as the interchange tool between Novapoint and AutoCAD Civil 3D. This connection ensures that geotechnical data is not just a static import but an integrated part of the multidisciplinary design process, allowing for better synchronization between the subsoil model and the infrastructure elements.

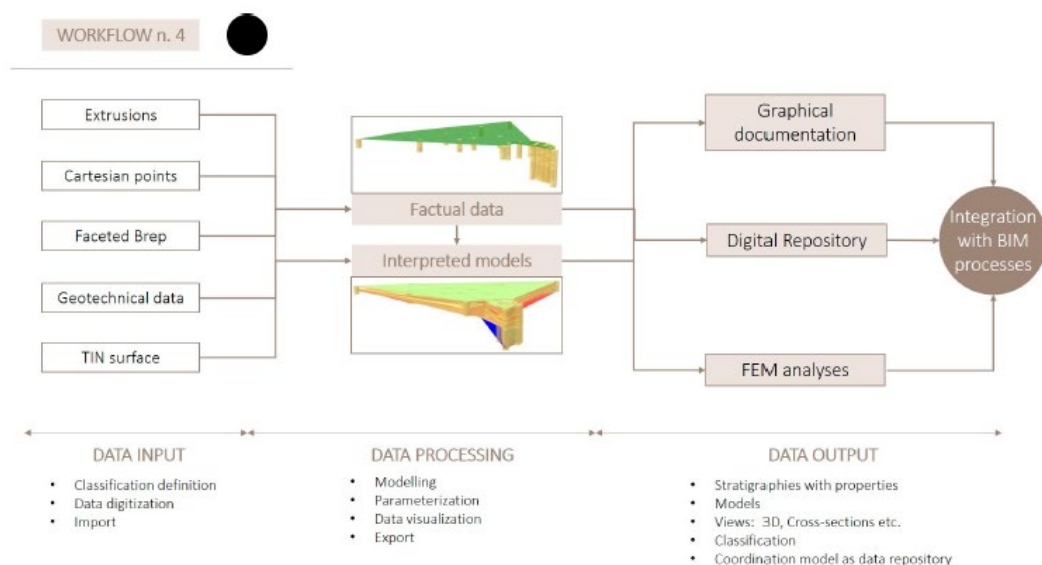


Figure 7. Workflow N.4 (Fonsati, Cosentini, Tundo, & Osello, 2023)

It was not possible to distinguish volumes based on materials, and it was allowed only for two distinctions; cut volumes and fill volumes. Only through the attribute tables they were visible, also the 3D boreholes were imported into the geo-model from Rockworks.

Based on these strategies, Fonsati's paper offers varying degrees of effectiveness for digital project management. The first method, which relies on a GIS-based approach, lacks providing a truly integrated environment because its primary output is limited to static graphic documentation rather than a functional, data-rich model. On the other hand, the other three workflows: external modelling, integrated BIM, and collaborative platforms, demonstrate a significant improvement in connectivity. These workflows review that geotechnical information is more than just a vision; they allow the data to become a dynamic part of the design process, providing higher levels of technical integration that bridge the gap between subsurface data and structural modelling (Fonsati, Cosentini, Tundo, & Osello, 2023).

An interesting work by (Lai, 2023) uses Leapfrog and BIM to carry out the interoperability between them for the Sydney Metro West (SMW) line concept design developed by Sydney Metro and their Technical partner Mott MacDonald. The SMW line comprises 24 km long twin-underground metro tunnels linking Westmead to Sydney's CBD and is due to open in 2030.

Lai describes a circular loop of Leapfrog to BIM and BIM to leapfrog. They first sent the tunnel design (BIM) into Leapfrog. They did this so the geologist could see the tunnel's location while drawing the soil layers, ensuring the geological interpretation was relevant to the excavation area as seen in Figure 8.

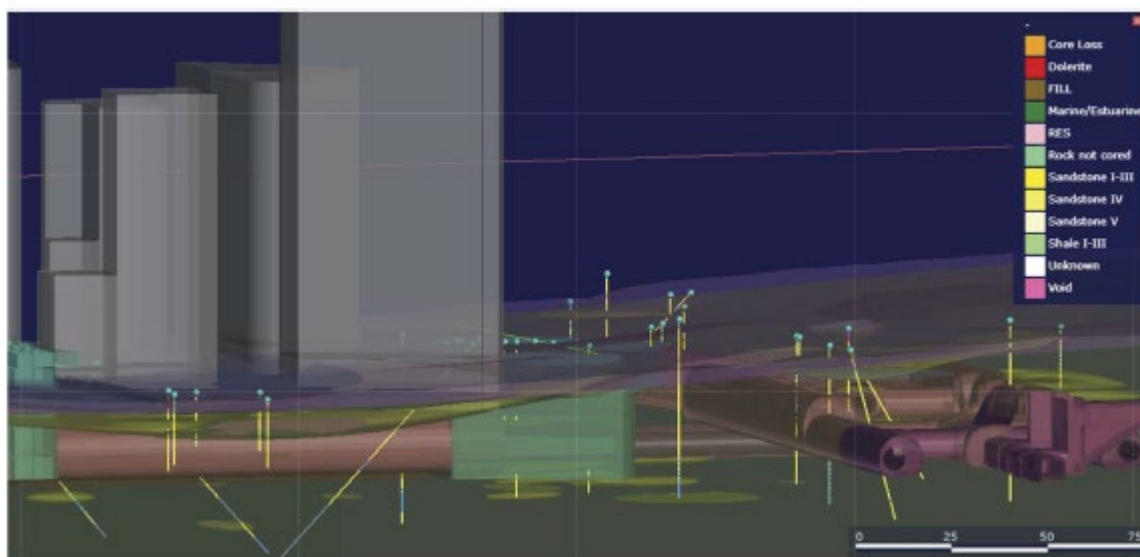


Figure 8. Leapfrog model (Geological interpretation and BIM). (Lai, 2023)

Furthermore, the integration of Leapfrog and BIM allowed for a detailed review of the nearest fault zone at the cavern being 50 meters away from a service adit; this resulted in the adoption of a twin's adit profile with long pillar in between the twins as shown in Figure 9 (Lai, 2023).

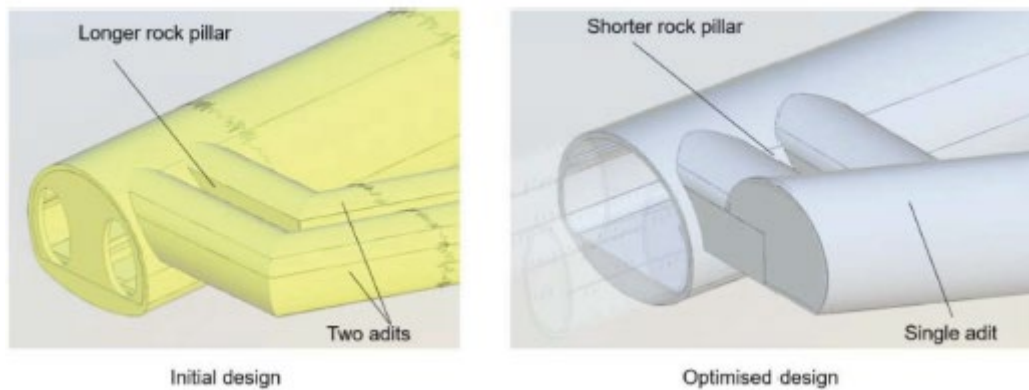


Figure 9. Initial and optimised design of Pyrmont Station adits (Lai, 2023).

While the Sydney Metro West project emphasized a circular exchange of data for visualization and assessment, it highlights the growing industry need for automated links between geological models and BIM environments. My research builds on this foundation by developing a direct "Leapfrog-to-BIM" pipeline. By using Dynamo as the integration tool.

2.3 The BIM to numerical modelling integration for automated workflows (BIM – FEM)

The transition from a static design model to a data-driven system requires not only a seamless connection between Building Information Modelling (BIM) and Finite Element Method (FEM) analysis, but also the integration of a monitoring platform and continuous geotechnical data. However, as (Talebi, Potenza, & Gattulli, 2023) identify, establishing that foundational BIM - FEM link presents a primary challenge of "interoperability"—the ability of different software platforms to exchange complex geometric and mechanical data without information loss.

(Talebi, Potenza, & Gattulli, 2023) also mentions that the BIM model could be used to quickly visualize possible negative trends related to structural behaviour. The lack of interoperability of the BIM models (switching between BIM and FEM) remains the issue. This integration with the incoming vibrational measurements could help the overall management of a building.

A more advanced approach involves file-based exchange using neutral formats like IFC or DXF to transfer geometry between platforms. While this method improves the speed of the transition, it frequently fails to preserve the complex geotechnical attributes and non-geometric data required for sophisticated finite element analysis. To address these limitations, the most robust level of integration utilizes API-based connections where Visual Programming Languages

(VPL), such as Dynamo for Revit, serve as the technical bridge between the design environment and numerical tools like PLAXIS 3D (Talebi, Potenza, & Gattulli, 2023).

In their study, (Talebi, Potenza, & Gattulli, 2023) evaluate their interoperability framework through the case study of the Annibaldi bridge, a slender pedestrian walkway in the historic centre of Rome characterized by a box-girder longitudinal beam and arch-shaped lateral wings.

Below, Figure 10 shows the interoperability framework proposed by (Talebi, Potenza, & Gattulli, 2023), where moving from traditional modelling to a fully integrated, dynamic system depends on automating data exchange between BIM and numerical analysis. While a "Refined FEM" approach still relies on slow, manual model updates, a "Simplified" path allows the model to update automatically as new sensor data or geological information arrives. This setup forms the basis of my thesis, which develops an automated pipeline by connecting Revit and Rocscience through Dynamo.

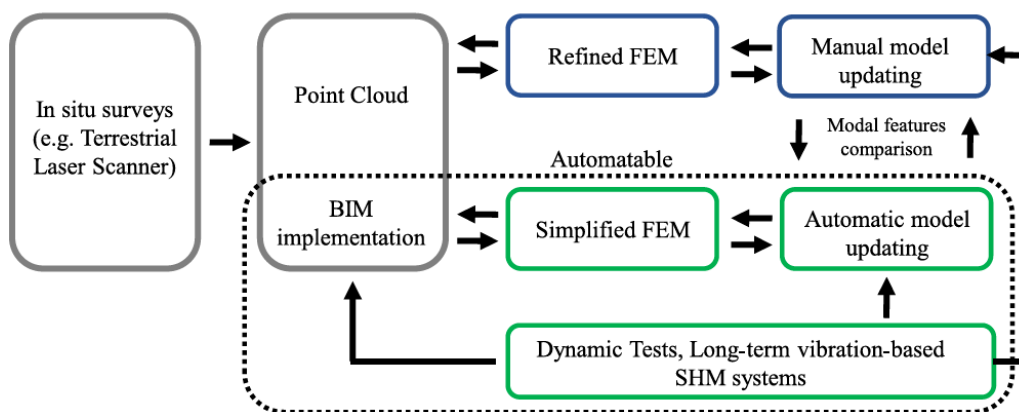


Figure 10. Interoperability of BIM-FEM-model updating (Talebi, Potenza, & Gattulli, 2023)

Figure 11 illustrates an advanced interoperability framework proposed by (Talebi, Potenza, & Gattulli, 2023), where a BIM implementation model serves as the central coordination layer for two distinct FE paths. While the traditional "Refined FEM" path relies on manual model

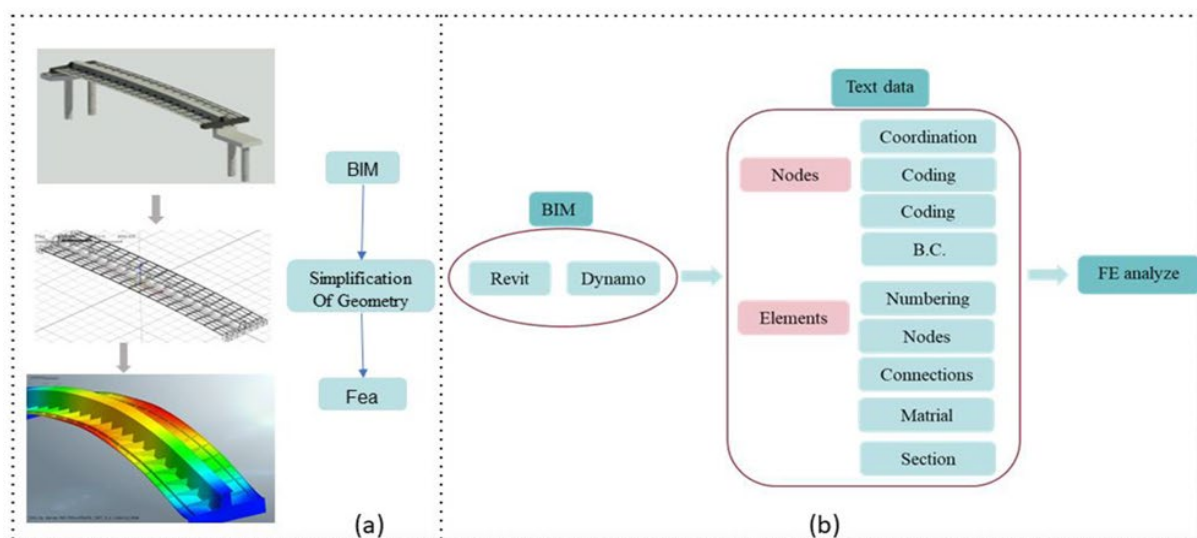


Figure 11. Data mapping schemas for manual geometry-only transfer (DXF) vs. automated parametric data exchange (Dynamo/MATLAB). (Talebi, Potenza, & Gattulli, 2023)

updating, the "Simplified FEM" path is designed to be fully automatable, enabling to synchronize with real-world conditions through continuous data from Structural Health Monitoring (SHM) systems.

Another search by (Shi, Jin, Lu , & Shi , 2023) proposes a BIM-based framework to overcome the information barrier between structural and geotechnical models in a large-scale excavation site with a total plan area of 8.7 million square metres. The excavation pit is designed to be the basement for multiple high-rise buildings. As shown in Figure 12, structural elements like piles and columns are first grouped by type and exported from BIM platforms as .dxf files, which decompose solids into basic geometric properties like points and lines. A Python script then extracts these control point coordinates and dimensions to create interoperable files, enabling the automatic generation of thousands of structural members within the PLAXIS 3D environment.

While this workflow may not be useful for small projects which could be managed manually, it helps the modelling of thousands of structural components in large-scale projects. It highlights a reliance on intermediate file formats, a limitation that this thesis aims to address by developing a direct API-based link through Dynamo.

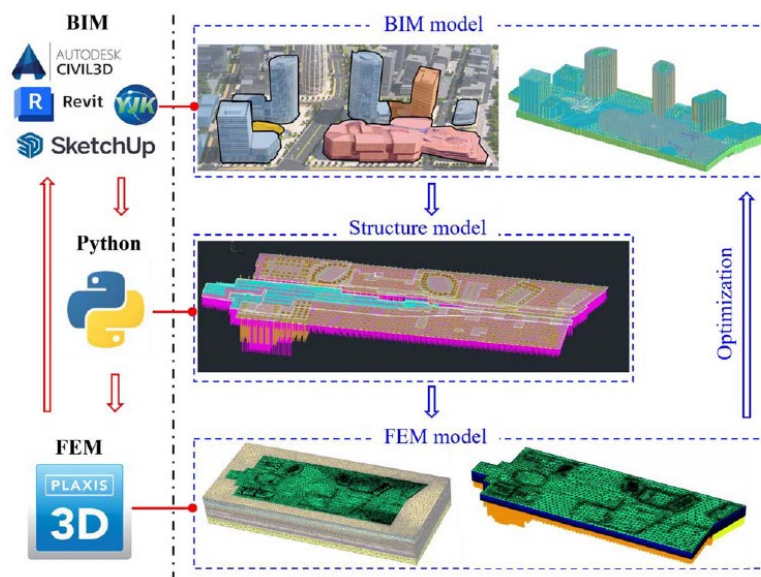


Figure 12. Automated BIM-to-FEM integration workflow using python scripting (Shi, Jin, Lu , & Shi , 2023)

In another study done by (Huang, Ruan, Wu, & Qin, 2024), they present a framework aimed at improving the integration of Building Information Modelling (BIM) with advanced Finite Element Simulations (FEM). The study addresses the “information gap” where the data in BIM environment are often lost or simplified when transferred to analysis software’s. Huang has established a complete automated pipeline as seen in Figure 13, from start to finish by creating

the geological model, modelling the tunnel using Dynamo and transferring the model to FEM software and analysing them.

To overcome the difficulties of interoperability between BIM - FEM, (Huang, Ruan, Wu, & Qin, 2024) proposed a bridge using Hypermesh software to translate BIM data into complex finite element models. This approach is particularly significant for large-scale infrastructures where geometric accuracy is critical; The researchers showed that by using Hypermesh as a link, they could create high-quality computer models automatically. This process keeps important structural details that usually disappear when you move files directly from one program to another. Their work proves that using a specialized "middleman" software is an effective way to keep engineering data accurate throughout the design and analysis process.

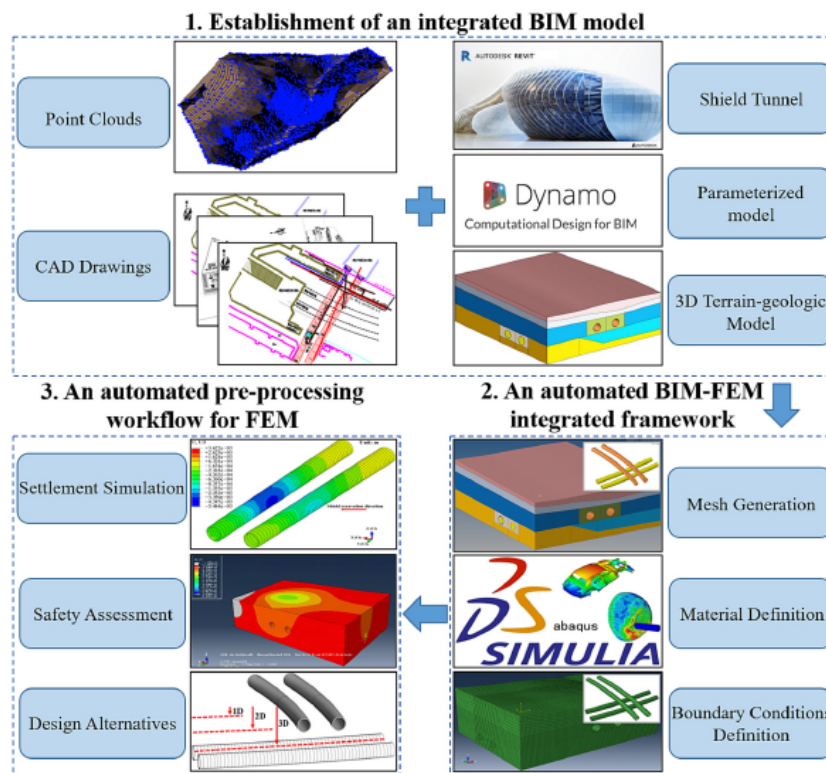


Figure 13. BIM - FEM framework based on automated data transfer (Huang, Ruan, Wu, & Qin, 2024)

The framework by (Huang, Ruan, Wu, & Qin, 2024) automates the transition from design to analysis through a four-step cycle presented in Figure 14. After the geometric data are modelled in a BIM environment (Step 1), the model is imported using the .sat (Standard ACIS Text) format to preserve mathematical precision for Hypermesh software, then defining the mesh size criteria and control standards for mesh shape based on geometric features (Step 2). In Step 3, the validated and optimized complete proximity engineering mesh model created in exported as an .inp file. Afterwards, the .inp file is imported into ABAQUS finite element software. A python code is generated for ABAQUS to define properties, simulate excavation processes and perform analysis. In the last step (step 4) Finite Element Analysis (FEA) is done using ABAQUS to evaluate the deformation patterns of the existing tunnel during the excavation of the newly built tunnel.

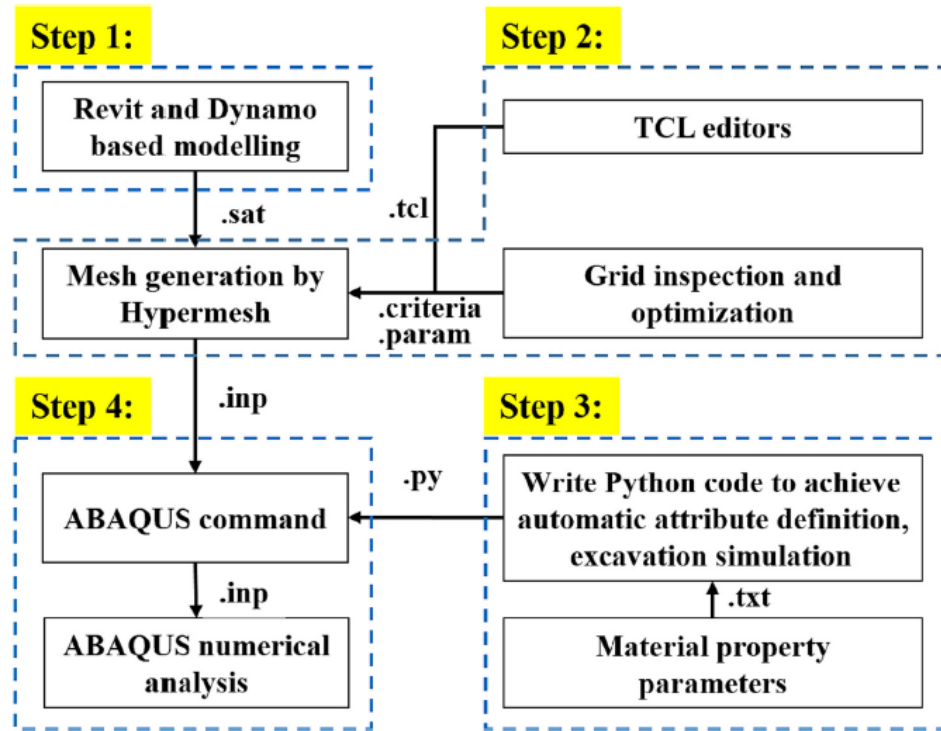


Figure 14. Fully automated BIM-FEM conversion framework (Huang, Ruan, Wu, & Qin, 2024)

A comparison between the BIM model, mesh model, and numerical model is shown in Figure 15.

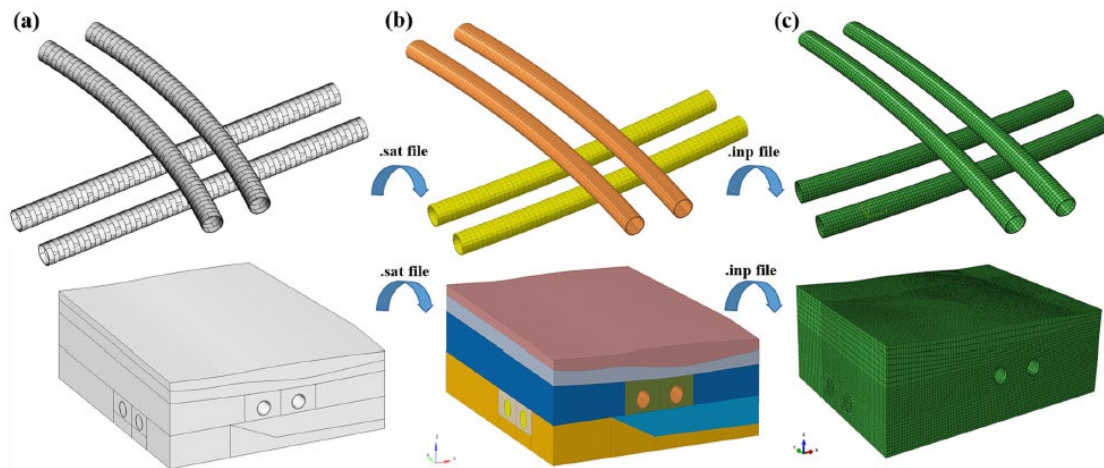


Figure 15. Comparison of BIM model, mesh model, and numerical simulation model: (a) BIM model; (b) mesh model in Hypermesh; (c) numerical simulation model

2.3.1 Challenges in data standardization: The role of IFC in geotechnical engineering

While Building Information Modelling (BIM) is now standard for buildings above ground, it is much harder to use for geotechnical work because we lack a single, unified way to share data. A major problem is what researchers call "information islands," where soil data and structural designs are stuck in different programs and cannot easily talk to each other. To fix this, (Wu, et al., 2021) suggested using Industry Foundation Classes (IFC) as a neutral "bridge" to link everything together.

Their research breaks the modelling down into two paths:

- **Built-In Generation Schema:** This is for man-made parts like piles or diaphragm walls. Since these are like normal building parts, they can "borrow" or inherit the properties of existing BIM elements. (Wu, et al., 2021)
- **Plug-In Extension Schema:** This is for the natural ground. Because soil layers are irregular and do not fit into standard BIM categories, new digital definitions had to be created from scratch to describe things like soil strata and underground cavities. (Wu, et al., 2021)

Figure 16 illustrates the logical structure of the "Built-In Generation Schema" used for man-made geotechnical elements. In this framework, structural components are integrated into the BIM environment by inheriting properties from existing classes within the standard IFC hierarchy. Some examples would be `IfcPile`, `IfcBeam` and `IfcWall` which are all positioned under the `IfcBuildingElement` block (Wu, et al., 2021).

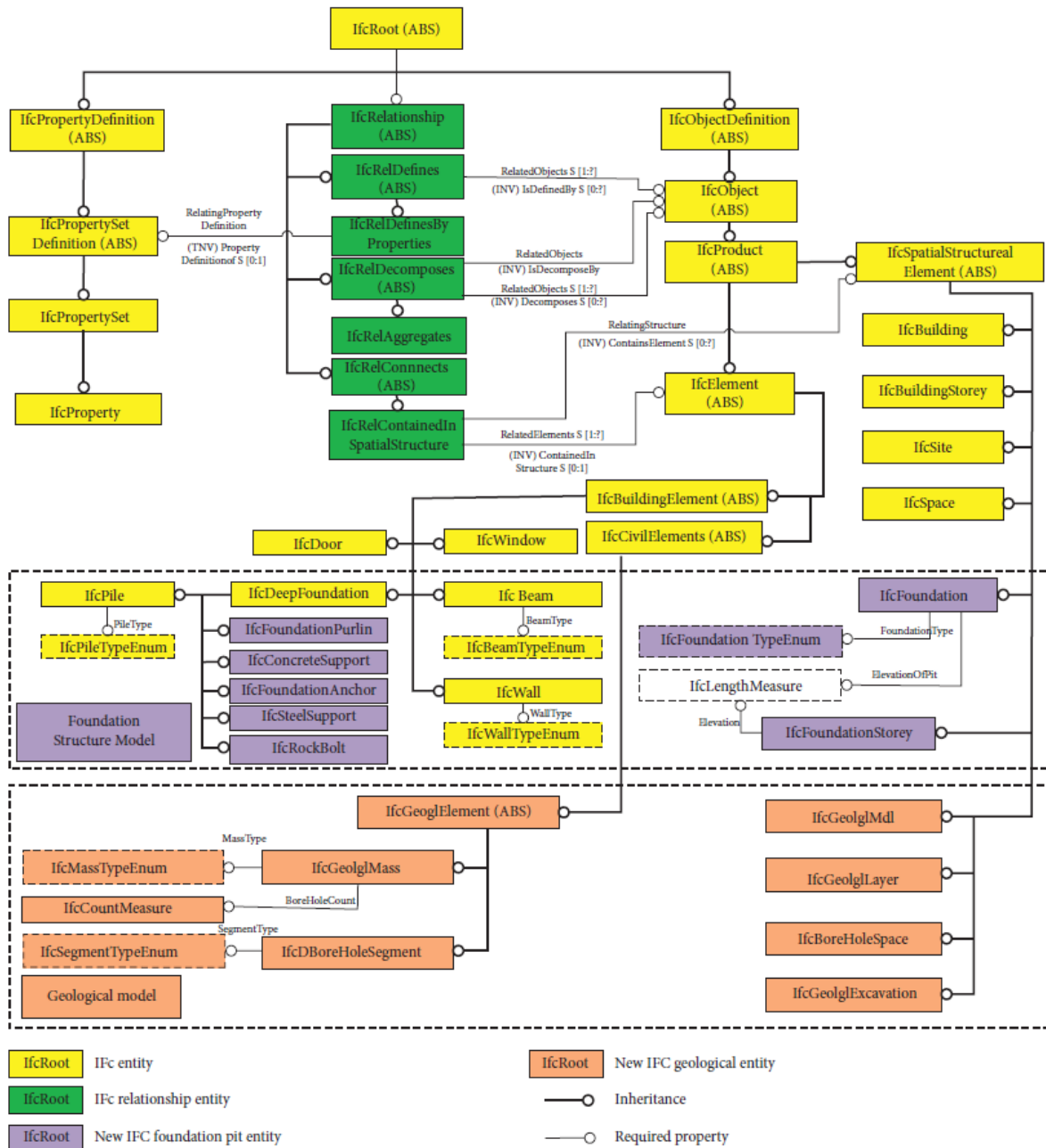


Figure 16. IFC extended EXPRESS-G diagram of the geotechnical information model (Wu, et al., 2021)

Since the natural world does not come in neat, straight lines like a building's columns or walls, new categories were created by (Wu, et al., 2021) from scratch to help computers understand the soil. The centre of the diagram in Figure 17 introduces a main category for all things underground, which then uses special 3D blocks to represent actual shapes like soil layers, rock formations, or even empty spaces like caves and boulders. These individual blocks are grouped into larger sections that eventually combine to form a complete 3D model of the projects subsurface. The schema also allows engineers to attach specific properties like how much water the soil holds or its exact density directly to the 3D shape so that the model contains actual engineering data rather than just being a simple drawing.

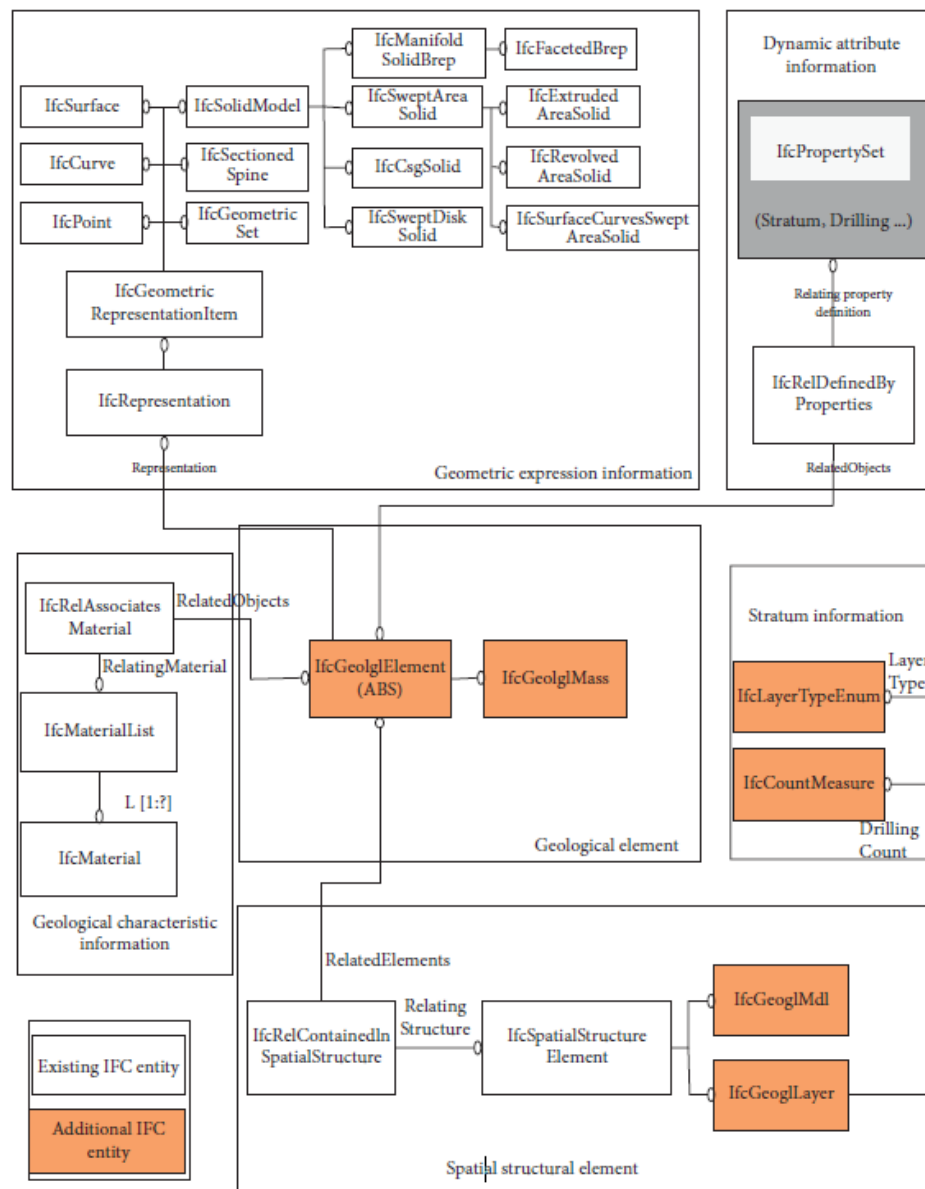


Figure 17. The EXPRESS-G diagram of the geological information model based on extended IFC (Wu, et al., 2021)

While the new digital schemas help organize underground information, a major problem remains: most 3D geological models are currently used only for looking at them in 3D. Many engineers describe these models as "good-looking but useless" because they do not contain the deep technical data and attributes needed for actual construction and design. This creates an "information island" where the soil data is separated from the building design, making it hard for different teams to work together (Wu, et al., 2021).

2.4 Visual programming languages in BIM

In the previous sections it has been seen that the interoperability between BIM and Numerical software's is done using visual programming language. Visual Programming Languages (VPL), like Dynamo for Revit and Grasshopper for Rhino, are tools that help engineers automate difficult design tasks without needing to write complex computer code. Instead of typing lines of text, users connect "nodes" (instruction blocks) with "wires" to create a visual flowchart that tells the software what to do. This allows the computer to handle repetitive work, such as building thousands of tunnel segments or mapping out complex soil layers, much faster than a person could do manually.

VPL is especially useful because it can talk directly to a program's API (its internal brain), allowing it to pull out important data like material strengths or exact coordinates and send them straight to analysis software. While many researchers use VPL to create middleman files (like .sat or .dxf) to move data between programs (Huang, Ruan, Wu, & Qin, 2024), it also has the power to create a direct, "live" link between BIM and simulation tools. By using these visual scripts, engineers can ensure that if the design changes, the analysis model updates automatically, saving time and preventing human errors.

To provide a foundation for numerical simulation, Huang developed a specialized workflow in Dynamo to automate the creation of 3D geological solid models. "Through the secondary development of the Dynamo plugin and leveraging its rich graphical programming capabilities, the automation of geological model generation and editing has been achieved in Figure 18. This method improves the efficiency of geological modelling while maintaining the accuracy of the models. The specific process is shown in" (Huang, Ruan, Wu, & Qin, 2024),

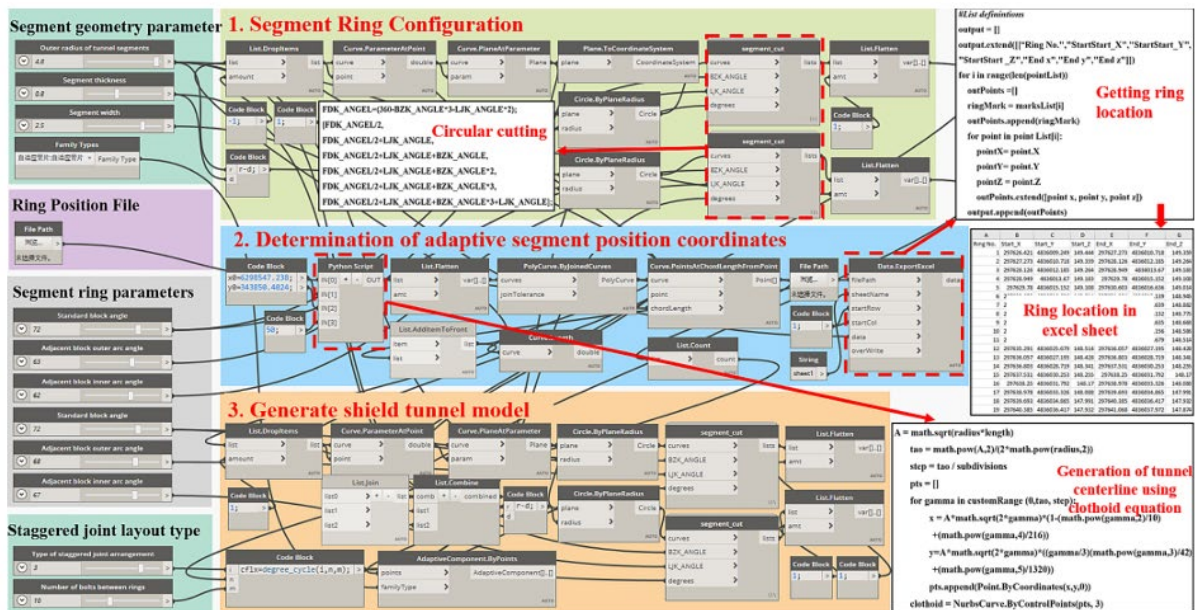


Figure 18. A Geological model diagram done in Dynamo (Huang, Ruan, Wu, & Qin, 2024)

In addition to the geological model, a shield tunnel workflow was developed in Revit by Dynamo as seen in Figure 20. Along with the 3D model shown in Figure 19.

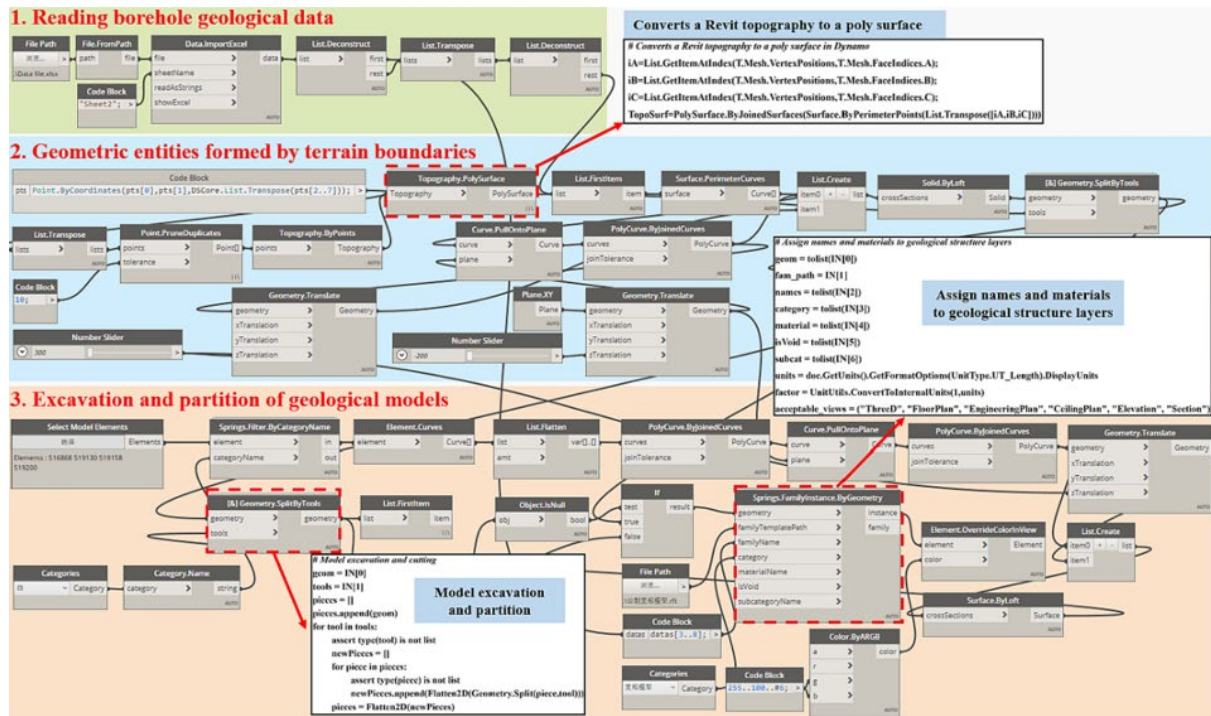


Figure 20. A shield tunnel diagram in Dynamo (Huang, Ruan, Wu, & Qin, 2024)

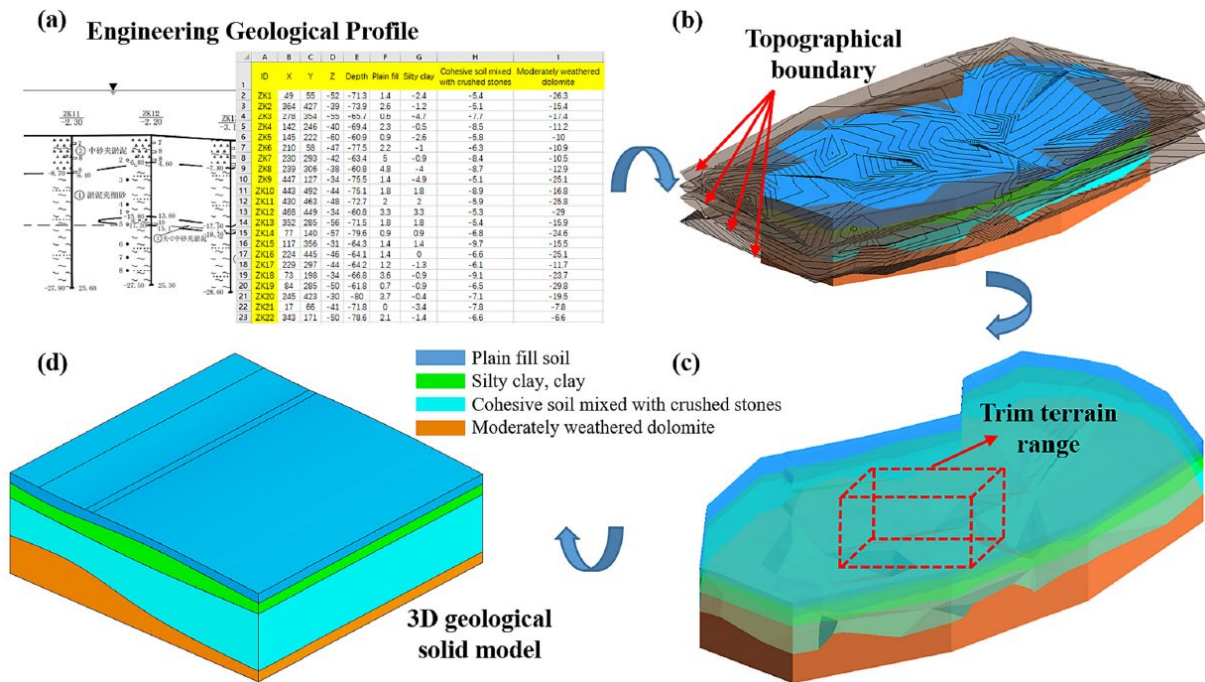


Figure 19. Parameterized modelling process of 3D geological BIM model: (a) Detailed borehole geology information; (b) Generating stratigraphic surface models; (c) Creating geometric entities using terrain boundaries; (d) Geological models after excavation and partition. (Huang, Ruan, Wu, & Qin, 2024)

This approach enables the creation of a detailed and accurate model of the segment ring as seen in Figure 21, with the ability to modify parameters to adjust for specific project requirements. By automating the process of generating adaptive components, the modelling efficiency can be significantly improved, reducing the time and effort required to create the model (Huang, Ruan, Wu, & Qin, 2024).

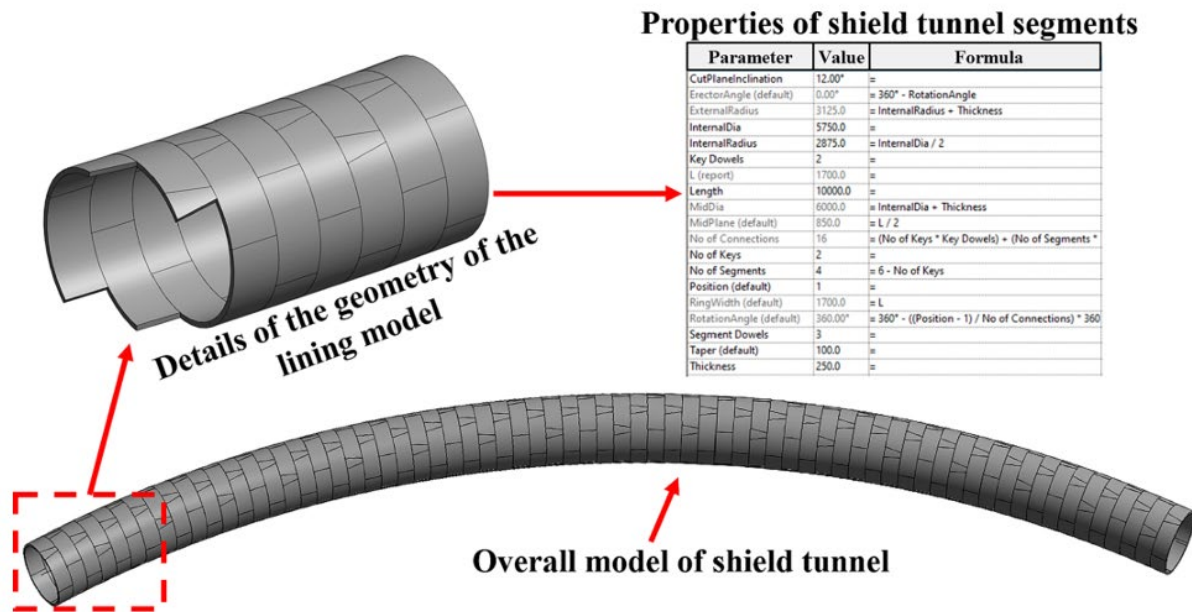


Figure 21. Overall model of shield tunnel and details of the geometry (Huang, Ruan, Wu, & Qin, 2024)

Building on the idea of using Visual Programming Languages (VPL) as a "live" link between data and design, recent research by (Alaei, 2023) shows how Dynamo can take the headache out of building complex underground models. A great example of this is the Västlänken Korsvägen II project, where the team used Dynamo to turn raw site data into a smart, 3D models. The process starts by taking 400 data points from ArcGIS and putting them into an Excel file, which Dynamo then reads to build the foundation of the model. Ten specific borehole locations are chosen to make sure the soil layers are mapped out exactly where they should be. To make the model as realistic as possible, the script calculates the distance between the ground surface and the borehole data, using "Cuboid" nodes to build the soil layers based on their highest and lowest points. To pull everything into Revit, as seen in Figure 22 the script uses a "DirectShape.ByGeometry" node, which creates the 3D shapes and automatically labels them with the right materials, like sand or silt. Finally, a "Color Range" node gives each soil type its own color, so the model is easy to read. Because the whole system is connected, if the borehole data in the Excel file changes, the 3D model in Revit updates on its own. This kind of automation keeps the data moving smoothly and saves engineers from the slow, boring task of typing in data by hand.



Figure 22. Layers of the project created in Revit using Dynamo (Alaei, 2023)

Beyond generating soil layers, visual coding techniques provide a flexible way to model actual infrastructure components. For example, parametric modeling can be used to create tunnel profiles with detailed geometry, material properties, and semantic information. This makes it possible to develop adaptive tunnel cross-sections that comply with BIM requirements and IFC standards. Using visual programming tools in open-source environments such as Dynamo, these tunnel profiles can be linked dynamically to the tunnel alignment and exported directly into Revit. This script-based workflow seen in Figure 23 done by (Al Furjani, et al., 2020), helps ensure that complex structural geometries are represented accurately and prepared for subsequent analysis stages.

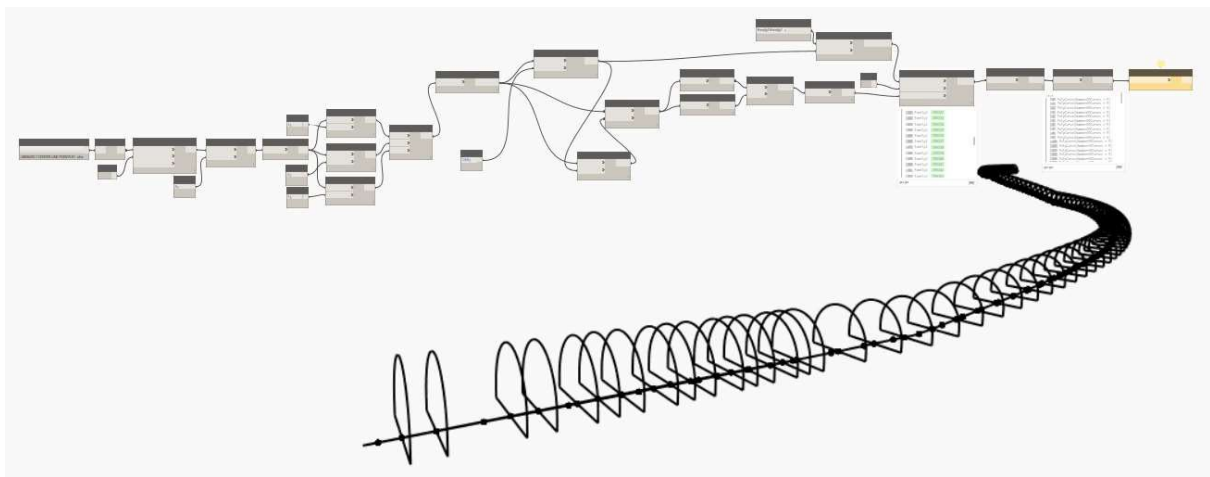


Figure 23. with visual coding techniques in Dynamo open-source software, the profile of the tunnel is assigned to the tunnel alignment to be exported to the Revit software (Al Furjani, et al., 2020)

2.4.1 Challenges of IFC and neutral formats in VPL workflows

While Dynamo in Revit serves as a powerful engine for automating design tasks, it often hits a major roadblock when trying to handle industry foundation classes (IFC's) or other neutral file formats. The industry frequently struggles with what researchers call information islands (Wu, et al., 2021), where critical soil data and structural designs and attributes are essentially trapped and unable to communicate. Even though industry foundation classes were designed to be a neutral bridge to solve this problem, the actual process of switching between different programs remains a major headache for engineers. The real issue is that while these file transfers might move the geometry across, they often fail to bring along the complex geotechnical attributes and non-geometric data that are vital for numerical analysis.

This leads to a real disconnect in the modelling process. Things we build, like piles or concrete walls, fit nicely into standard BIM categories because they have predictable shapes and follow clear rules. However, the natural ground is a completely different story. Since soil layers are so irregular and don't follow straight lines, they simply don't fit into the normal boxes that BIM software uses. This often leads to 3D models that people describe as good-looking but useless, because they look nice on a screen but don't have the deep technical data needed for real engineering work.

Even though tools like Dynamo are great for making work faster, they often hit a wall when trying to use IFC files or other basic formats. One of the biggest problems is that soil information and design plans usually end up stuck in their own separate worlds, which experts call information islands. This makes it difficult for different software programs to talk to each other, and moving data between them is still a major struggle for engineers. Even though we use IFC to try and connect everything, despite the new versions available now, these file transfers often lose important details like how strong the soil is or how much water it holds.

This creates a real disconnect in the process because while it is easy to categorize man-made structures like concrete piles, the messy and irregular shapes of natural ground layers simply do not fit into the standard boxes used in building models. Because of this, we often end up with 3D models that look nice on a screen but are useless for real engineering. Standard IFC files are also quite static, meaning they do not update automatically when we find new information at the construction site.

To fix these gaps, we can use custom Python scripts inside Dynamo to act as a live bridge between the database and the model. As shown in research by (Marmo, Polverino, Nicolella, & Tibaut, 2020), a script can use unique IDs to reach into the model and automatically update it with the right technical parameters. This method is much more flexible because it bypasses the errors found in basic file transfers and ensures the digital model stays in sync with real-world data.

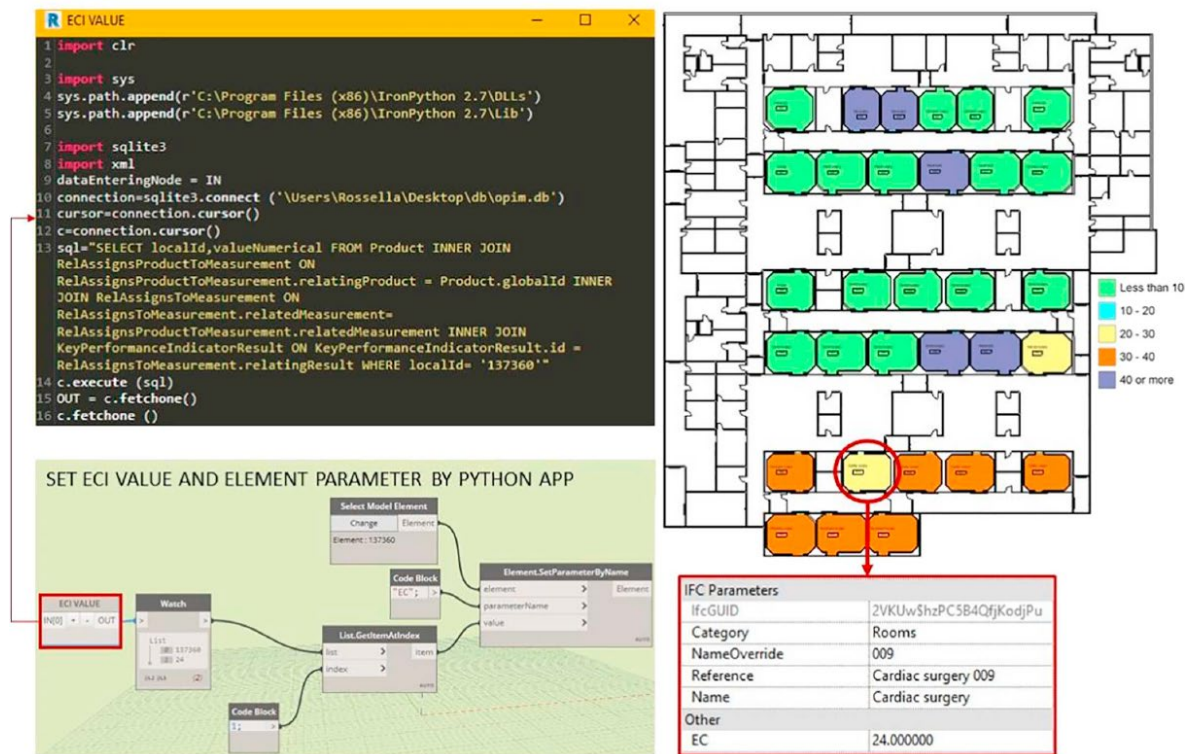


Figure 24. Python application for retrieving the ECI value based on the localID of a Revit element (on the left). Thematic plan for visualising environmental assessment results in the IFC model (on the right) (Marmo, Polverino, Nicolella, & Tibaut, 2020).

Figure 24 shows how using a Python application lets us tackle the kind of tough data work that regular visual nodes just aren't built for. Instead of trying to do everything manually, we can let the script pull technical info straight from the database and update the IFC model. This is really the key to finally bringing together those disconnected "information islands" that often keep project data separated (Marmo, Polverino, Nicolella, & Tibaut, 2020).

These results have been used to set ECI parameter values in the IFC model through the `Element.SetParameterByName` node. Environmental condition results are visualised in a thematic plan through a colour scheme as seen in Figure 24.

2.4.2 Python-driven API interoperability for 2D geotechnical analysis

Python has emerged as the preeminent "glue" language for geotechnical automation, primarily because it serves as a common scripting interface for both BIM authoring tools (e.g., Revit, Civil 3D) and numerical solvers (e.g., PLAXIS, RS2, OPTUM GX) (DartisTech, 2026). Python-driven API interoperability acts as a "semantic middleware" that interprets the structured data within a BIM environment and reformulates it into a language understood by the numerical analysis engine (Nugraha, Iarasati, & D ng, 2025).

The technical core of this integration lies in the Application Programming Interface (API), which allows external scripts to access a software's internal "brain". For instance, the Revit API

provides access to every element's parameter, geometry, and location within the 3D space (Fahmy, 2025). By utilizing Python within visual programming environments like Dynamo, engineers can move beyond the limitations of pre-built nodes to author custom workflows that programmatically control walls, tunnels, and geological surfaces.

Interoperability through APIs can be categorized into unidirectional and bidirectional workflows. Most current industrial research focuses on a unidirectional "forward-feeding" pipeline, where the BIM model acts as a coordination layer that simplifies complex geological information into design-relevant zones for numerical analysis.

The implementation of these workflows requires deep familiarity with transaction management and geometry conversion libraries. For example, extracting a 2D section from a 3D Revit model involves the "ViewSection" class and the "FilteredElementCollector" to identify every structural and geological element intersected by the section plane (Fahmy, 2025).

The design of underground structures, such as tunnels and deep excavations, typically involves the selection of a limited number of critical cross-sections. Traditionally, this process is manual, requiring the engineer to cut sections in CAD, export them to DXF, and manually assign material properties in the simulation software. This manual workflow is not only time-consuming but also prone to human error, particularly in large-scale projects with hundreds of potential section locations (HS2, 2020).

Research into BIM-to-FEM interoperability highlights that both native and IFC-based workflows frequently result in inconsistencies. Indirect transfers via IFC exhibit significant instability; for example, IFC2x3 and IFC4 Reference Views often fail to correctly interpret structural relationships, leading to unconnected nodes or missing constraint information as shown in Table 1. (Colajanni, et al., Architectural and Structural Interoperability in the BIM Design Workflow, 2025)

Format	Geotechnical Utility	Limitations
DXF	High for geometry	Lacks all semantic data and material parameters.
IFC2x3	Low	No native classes for soil layers or complex underground structures.
IFC4.3	Moderate	Improved infrastructure support; inconsistent implementation in analysis tools.
Python API	Very High	Full data retention; requires programming expertise.

Table 1. Comparative utility of exchange formats (Colajanni, et al., Architectural and Structural Interoperability in the BIM Design Workflow, 2025)

2.5 From digital models to digital twins

To better understand the integration of BIM and numerical analysis, it is important to define the concept of a digital twin (DT). Originally introduced in the aerospace industry by Michael Grieves in 2002 as shown in Figure 25, a digital twin is essentially a virtual representation of a physical asset that continuously exchanges data with its real-world counterpart. (Singh, et al., 2021)

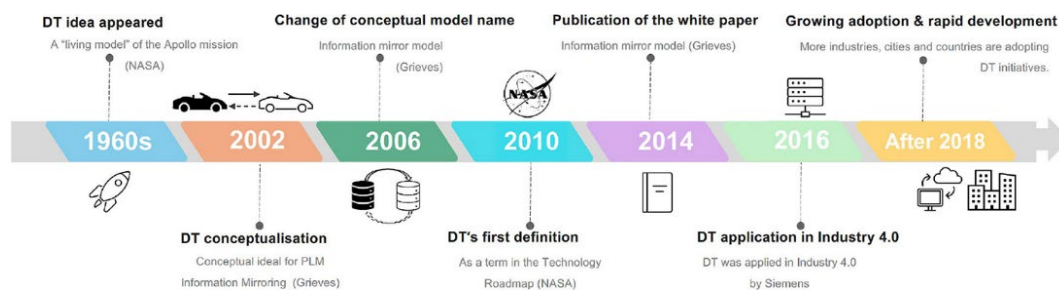


Figure 25. DT development timeline (Wang, Rezaei, & Hicks, 2025)

In the AEC industry, the term is often confused with standard 3D CAD or BIM models. However, a simple geometric model represents only a small part of a true Digital Twin. Traditional models are generally static and require manual updates, whereas a Digital Twin is dynamic, continuously evolving through automated and real-time data exchange. (Singh, et al., 2021)

The distinction between these digital representations as argued by Babanagar et al. (2025), is defined by the direction and automation of data flow between the physical and virtual entities. A digital model serves as a static geometric representation where data exchange with the physical counterpart is performed manually. In contrast, a digital shadow features an automated one-way data flow from the physical object to the digital representation, allowing for insights into historical behaviour without providing feedback to the physical twin. A complete digital twin is characterized by bidirectional and near-real-time data exchange enabled by data acquisition and processing technologies, which supports predictive analysis and optimized decision-making as shown in Figure 26.

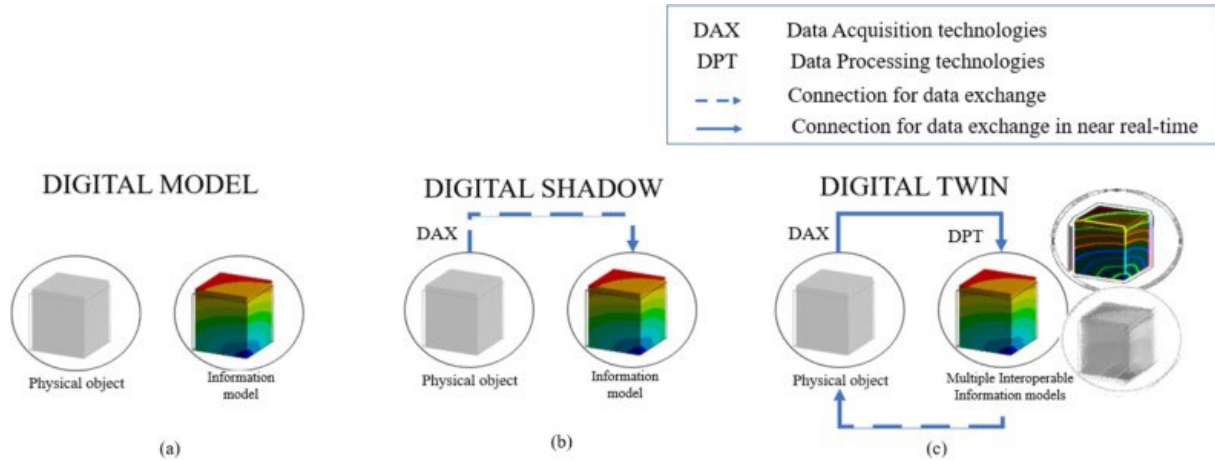


Figure 26. Conceptual representation of (a) digital model, (b) digital shadow, and (c) digital twin. (Babanagar, Sheil, Nini'c, Zhang, & Hardy, 2024)

In the context of underground engineering, transforming a tunnel BIM model into a Digital Twin means moving beyond static geometry and creating an integrated system where BIM models and numerical simulation tools can communicate and update automatically. Figure 27 shows specifically network illustrating the integration of digital twins with computational techniques and geotechnical applications. (Wang, Rezaei, & Hicks, 2025)

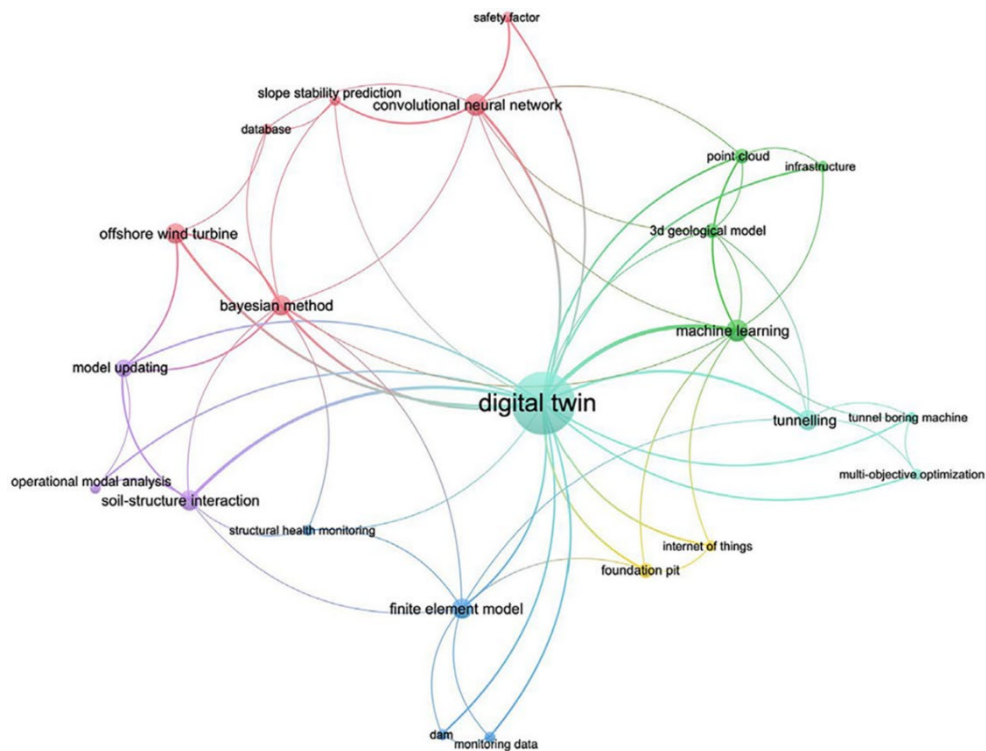


Figure 27. Keywords co-occurrence network (Wang, Rezaei, & Hicks, 2025)

Despite its static limitations, BIM is widely recognized as the essential foundational layer for developing a Digital Twin. According to (Nguyen & Adhikari, 2023), BIM provides the most efficient mechanism for creating the highly accurate, semantic 3D representations required to build a DT.

Adopting geotechnical Digital Twins remains difficult due to geological uncertainty and high computational demands. Incompatible software protocols continue to hinder the seamless integration and scalability of these underground systems as shown in Figure 28.

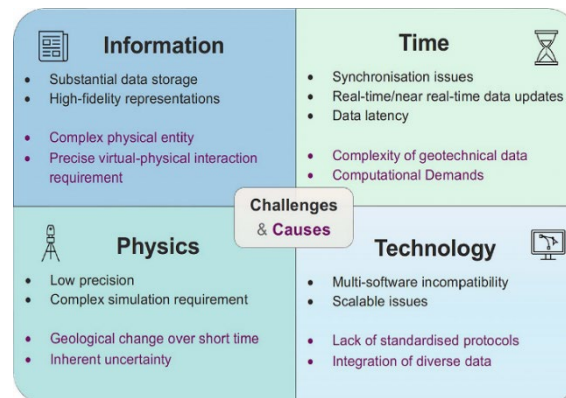


Figure 28. Challenges and causes for DT adoption in geotechnical engineering (Wang, Rezaei, & Hicks, 2025)

Chapter 3: Tunnel case study

3.1 Introduction and project overview

To validate the proposed BIM-to-FEM automated workflow, this research applies the developed framework to a real-world infrastructure tunnelling project provided by TYLin Italy. For reasons of confidentiality and data security, the specific identifying details of the client, precise location, contractors, and the exact project chainage have been anonymized.

The case study for this research is situated in a location defined by its geologically complex and hilly topography. This primary focus of this section is to describe the physical parameters of the mined tunnel structure together with the characteristics of the subsurface soil model. By integrating the structural components and geological formations within an open-BIM platform like Revit, the framework establishes a structured digital environment that enables the verification, management, and extraction of spatial data for numerical simulations.

3.2 3D model of the tunnel

As a primary geometric input for this research, the 3D digital model of the tunnel structure was provided, seen in Figure 29 to establish the foundational baseline from which analytical sections are extracted. This model functions as a coordinated open-BIM environment inside Autodesk Revit, combining the spatial design of the twin-tube alignment with localized subsurface information. The primary purpose of this digital asset within the automated workflow is to provide a highly accurate structural trajectory that can be systematically sliced at critical locations along the path. A 3D DWG model is also utilized (Figure 30), which isolates the pure geometric trajectory.

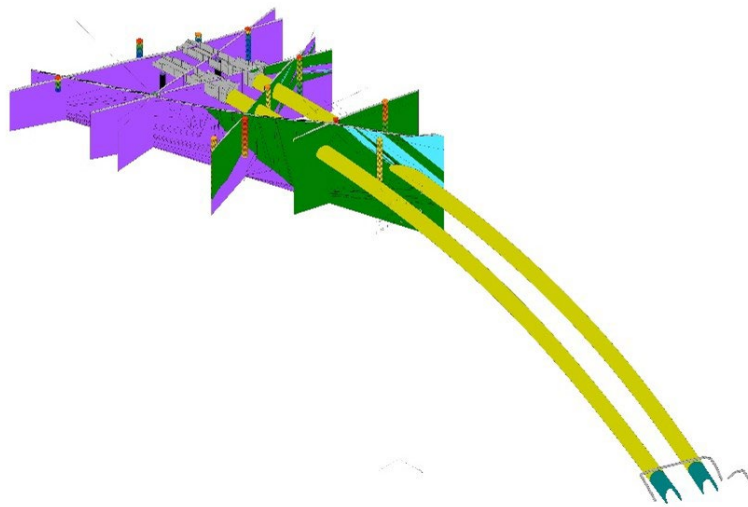


Figure 29. Three-dimensional integrated IFC Tunnel model

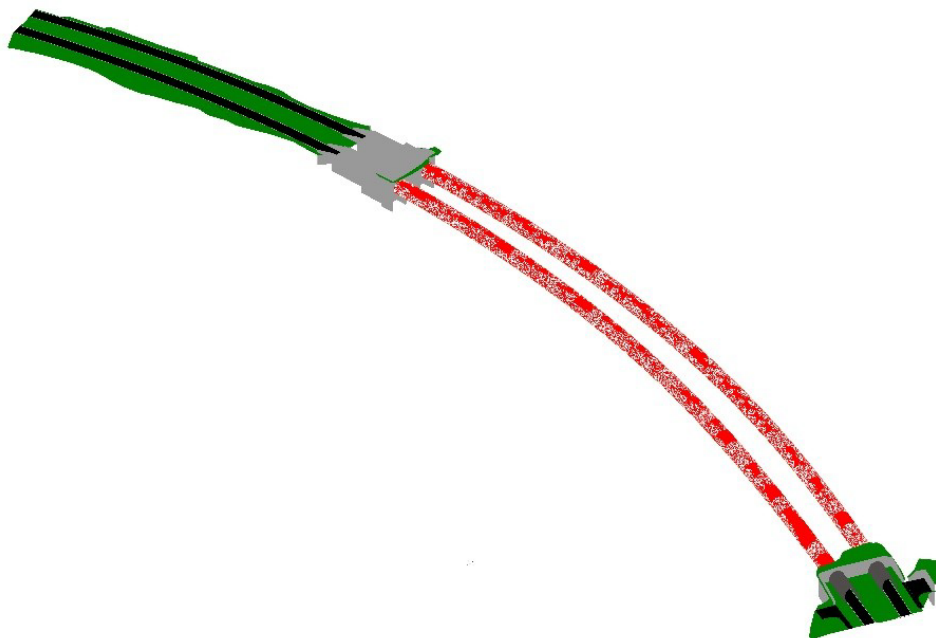


Figure 30. 3D DWG model of the tunnel

3.3 Geological soil model

To safely and efficiently manage the extreme spatial variability of the flysch formation and the overlying colluvial landslide deposits, a highly sophisticated, three-dimensional geological model is required. In the context of the present case study, this spatial representation was constructed utilizing Leapfrog Works, a software platform engineered specifically for implicit geological modelling in complex infrastructure environments. The model was then exported in IFC 4.3 format to be imported and utilized within Autodesk Revit 2026, as illustrated in Figure 32.

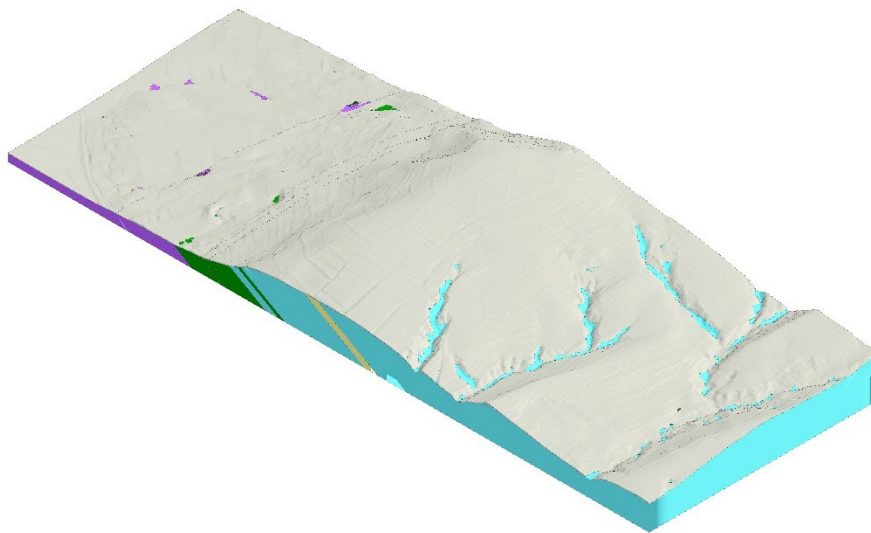


Figure 32. 3D Geological Case study model

3.3.1 Stratigraphic discretization and color-coding schema

To successfully bridge the gap between the three-dimensional implicit geo-model and finite element analysis (FEA), the geological profile along the tunnel alignment was discretized into six distinct stratigraphic units based on material behaviour, weathering grade, and lithological properties. This deterministic zoning translates complex lithological transitions into homogenous regions suitable for boundary value problem simulations.

As illustrated in Figure 33 the extracted west elevation discretization, the distinct color-coding for each stratigraphic unit was natively assigned within the Leapfrog geological model. Upon importing the IFC file into the BIM environment, these material colors were successfully

preserved and inherited, ensuring seamless semantic consistency throughout the interoperability cycle. The inherited color schema and the corresponding strata are defined as follows, with their specific geomechanical parameters detailed in Table 2.

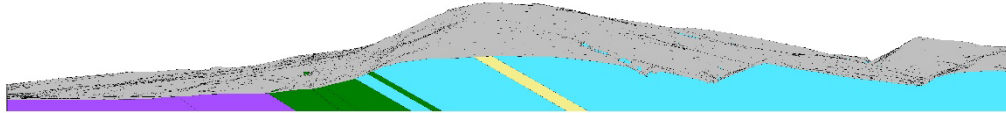


Figure 33. West elevation discretization illustrating the preserved color-coded stratigraphic units imported via IFC.

Layer Name	Lithology	Unit Weight (γ) [kN/m ³]	Young's Modulus (E/Ei) [MPa]
Ia	Colluvium	22	40
Ib	Colluvium	22	60
IXb1	Sandstone	26	9,625
IXb2	Sandstone	26	11,000
IXd3	Sandstone	26	13,750
VIIIId1	Sandstone	26	19,250

Table 2. Geomechanical Soil Parameters

Establishing a uniform naming and color-coding convention is a distinct advantage for ensuring semantic consistency across automated BIM-to-FEM workflows. Since each geological layer corresponds to a specific RGB code, this visual data can be directly utilized as a programmatic filter to isolate and extract essential stratigraphic information for the next stages of analysis.

Chapter 4: Architecting the BIM – FEM workflow

4.1 Overview and methodological framework

This chapter details the computational procedures developed to establish an automated data pipeline between the BIM environment and FEM numerical analysis tools. As a starting point, a comprehensive workflow was designed to map the transition of data from initial geological models to the final analysis-ready state, overcoming the "information islands" traditionally found in underground engineering.

As seen in Figure 34 the framework integrates data across three phases: the Geometric & Geological Baseline, the Algorithmic Interoperability Engine, and Analytical Extraction & Simulation.

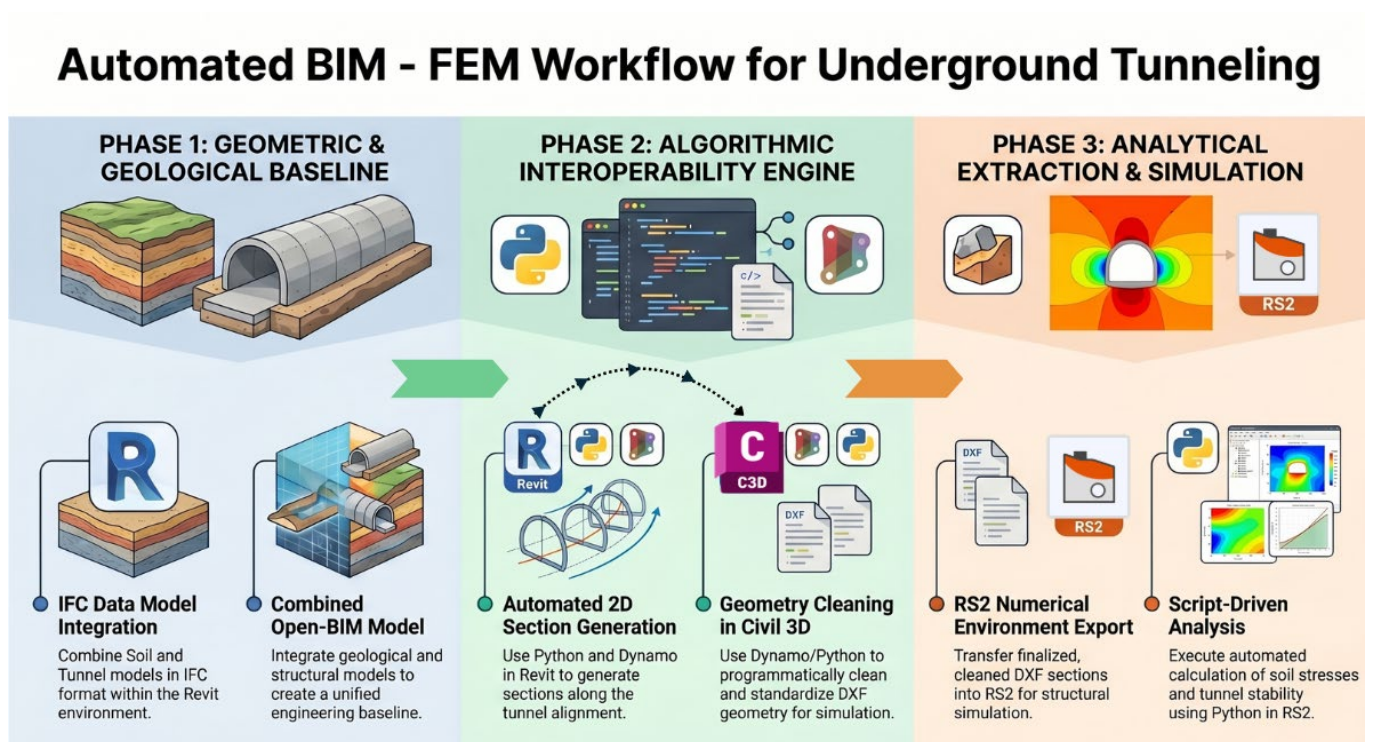


Figure 34. Methodological framework illustrating the automated Pipeline

Phase 1: Geometric & geological baseline

This first phase focuses on establishing a coordinated shared digital model that consolidates the project's 3D and geological data.

- **IFC Data Model Integration:** The workflow commences by importing the predefined soil strata model and the structural tunnel model, both utilizing the Industry Foundation Classes (IFC) format, into the Autodesk Revit environment.
- **Combined Open-BIM Model:** By seamlessly integrating these geological and structural models, a unified baseline model is established. This open-BIM model acts as the single source for the following parametric extraction.

Phase 2: Algorithmic interoperability engine

The second phase serves as the brain of the whole cycle, utilizing visual programming language and python scripting to extract and process data from the BIM environment.

- **Automated 2D Section Generation:** Utilizing Dynamo and Python scripting inside Dynamo, precise 2D cross-sections are automatically generated at targeted intervals along the specific tunnel alignment.
- **Geometry Cleaning in Civil 3D:** Once the sections are extracted, they are processed in Autodesk Civil 3D. Here, Dynamo and Python are stationed to programmatically clean, filter, and standardize the exported DXF geometry, ensuring the linework is fully optimized and structurally sound for RS2 import.

Phase 3: Analytical extraction & simulation

The final phase handles the transition of the processed geometry into the finite element analysis software (RS2) to evaluate the tunnel's structural behaviour.

- **RS2 Numerical Environment Import:** The finalized and cleaned DXF sections are systematically transferred into the Rocscience RS2 environment, effectively mapping the geometric data into the simulation space.
- **Script-Driven Analysis:** Utilizing the internal Python scripting editor within RS2, an automated analysis procedure is executed. This script drives the computational solver to evaluate complex geotechnical interactions, automating the calculation of soil stresses, displacements, and overall tunnel stability.

A more detailed breakdown of each phase and section, including the specific scripting logic, parameter mapping strategies, and step-by-step procedures utilized to achieve this automated pipeline, will be fully discussed and demonstrated in the subsequent sections of this chapter.

4.2 Phase 1: Geometric & geological baseline

As outlined in the methodological framework, the first critical step in developing the automated pipeline is establishing a unified digital environment. This phase, defined as the Geometric & Geological Baseline, focuses on establishing a coordinated shared digital model that consolidates the project's 3D tunnel model and the geological data together as a unified model.

To achieve this, Autodesk Revit serves as the central open-BIM platform. The workflow starts by importing the predefined soil strata model and the structural tunnel model, both utilizing the Industry Foundation Classes (IFC) format, into the Autodesk Revit environment. The first component is the 3D implicit geological model (originally generated in Leapfrog), which contains the color-coded stratigraphic layers. The second component is the structural model of the twin-tube tunnel alignment provided for the case study.

To properly set up the digital environment before importing any structural or geological models, the project was started using the default metric template in Autodesk Revit shown in Figure 35. Choosing a standard metric template provides a completely clean workspace that naturally follows European engineering practices. This ensures that the software is fully prepared to handle the complex 3D geometries that will be brought in from other platforms.

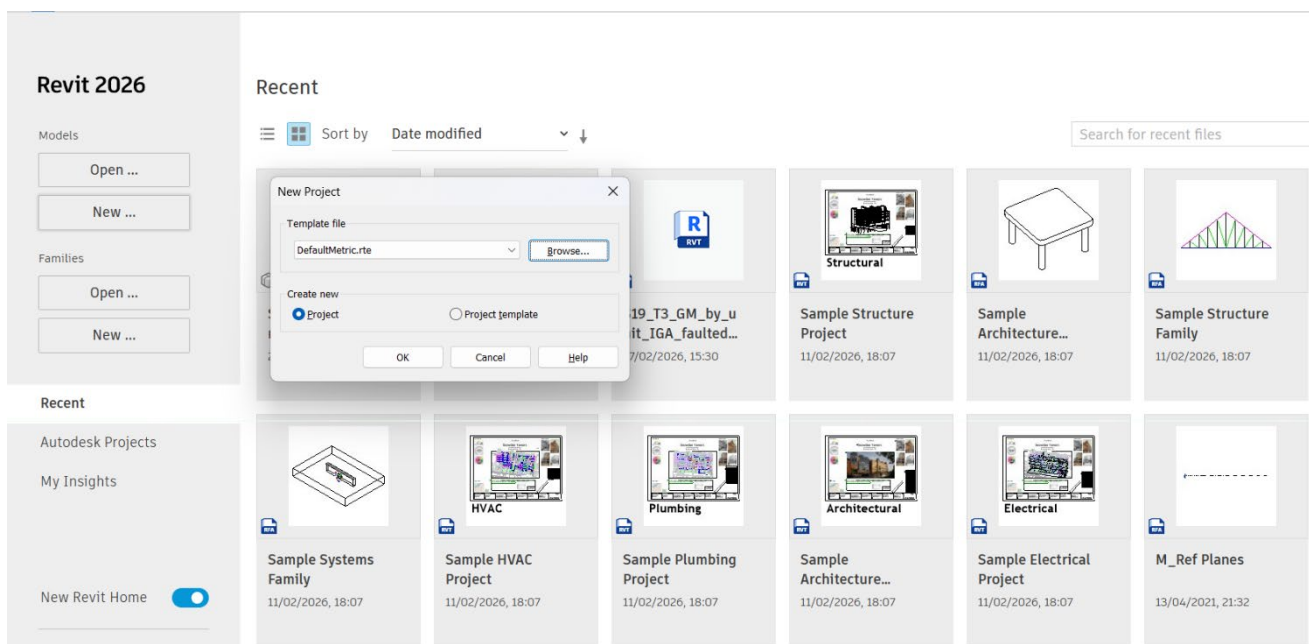


Figure 35. Default metric template from Revit

However, before linking any external files, the absolute most important step is configuring the project units correctly. Since all the survey information, geological dimensions, and the tunnel's alignment are measured in meters, the Revit workspace had to match this exactly. By accessing the Project Units menu (Figure 36), the formats for length, distance, and coordinates were all locked to meters. Getting this dimension system right from the very beginning is crucial. If the base units were left unchecked or mismatched, it could cause massive scaling issues later.

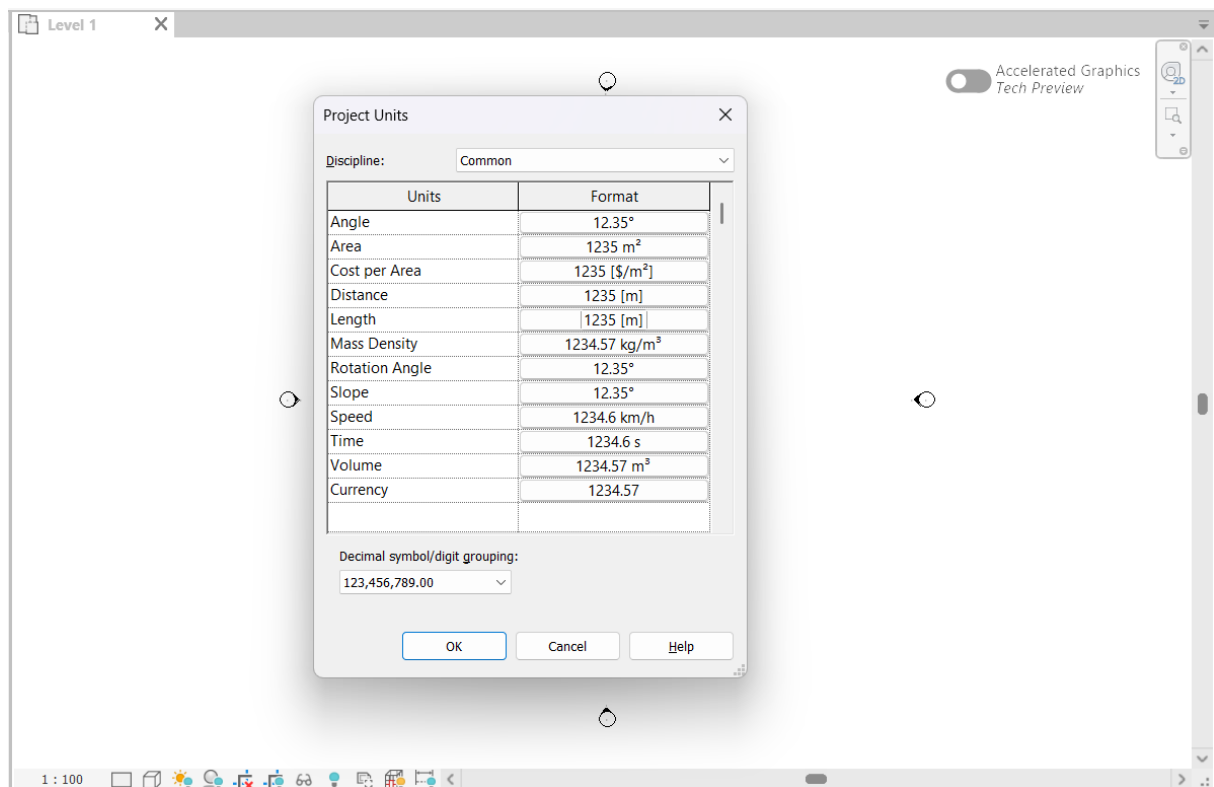


Figure 36. Setting project units in Revit

With the foundational metric environment securely established, the next critical operation is introducing the 3D structural and geological data into the project. At this stage, it is fundamentally important to use the "Link IFC" function rather than simply "Importing" or "Opening" the file. In BIM methodology, opening an IFC directly creates an entirely new, isolated Revit project, which defeats the purpose of combining multiple disciplines into a single shared space. Conversely, importing the file embeds the external geometry permanently into the Revit database; this not only inflates the file size to unmanageable levels but also severs the connection to the original source data. By choosing to link the IFC models, the software establishes a dynamic, live reference to the original files provided by the structural and geotechnical teams. This approach is the true backbone of the advanced BIM interoperability: if the Leapfrog soil model or the tunnel alignment is updated during the design iteration process, the linked files can simply be reloaded. This instantaneously updates the baseline model without losing any established coordinates or requiring manual rework. Ultimately, linking ensures that the digital information supply chain remains unbroken.

After linking the Tunnel model into Revit using the Link IFC command as shown in Figure 37 using the “IFC Origin to Internal Origin” positioning, the model's spatial position is calibrated.

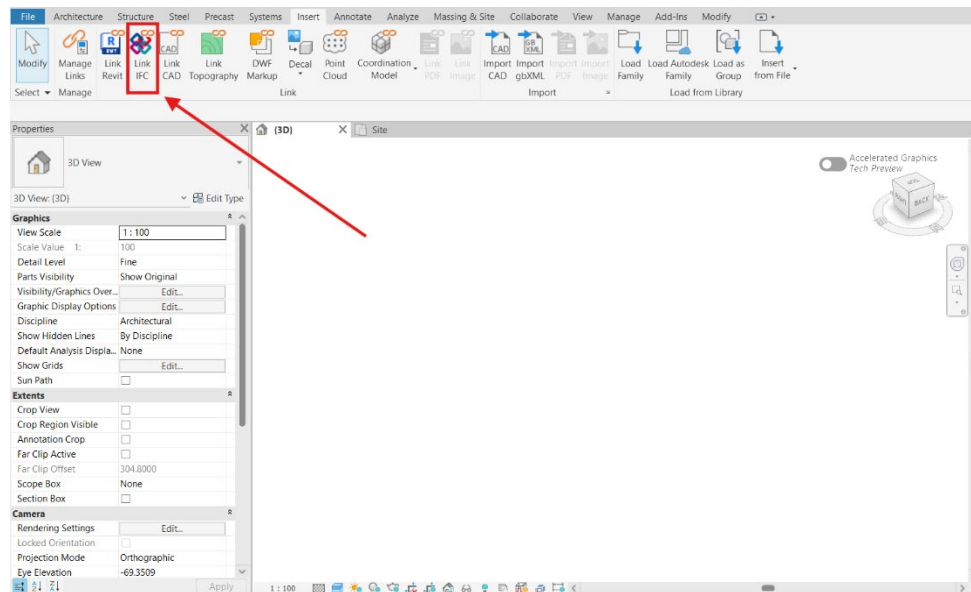


Figure 37. Utilizing the Link IFC tool.

Once the structural tunnel IFC model is successfully linked into the workspace, it initially sits at the software's internal local origin. However, to build a reliable BIM-to-FEM workflow, the digital model must be perfectly georeferenced to match its real-world physical location. To achieve this vital spatial calibration, the "Specify Coordinates at Point" tool is used within Revit. This specific function is essential because it allows the local project base point to be overridden and accurately aligned with a global coordinate system. This tool mathematically pins the geometry exactly where it belongs in the real world.

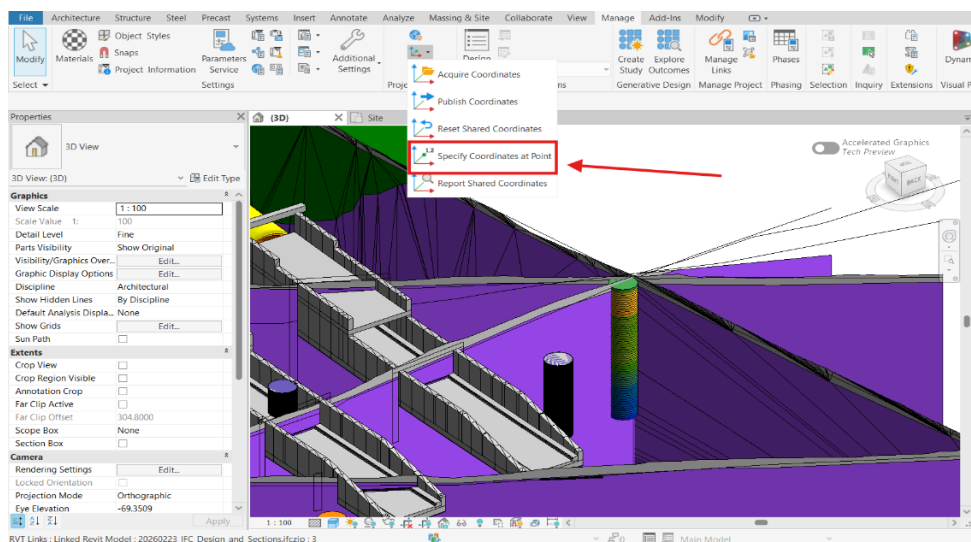


Figure 38. Georeferencing using “Specifying Coordinates at Point” function

As seen in Figure 39, establishing the correct position within the software requires a highly accurate benchmark to guarantee spatial consistency between the structural and geological data.

To achieve this, a specific reference point and its exact spatial coordinates (X, Y, Z) are extracted directly from the Leapfrog model. It was given specifically as fundamental input data and a primary reference for this phase of the project. By using this official data as the main anchor, it guarantees that the digital model aligns perfectly with the actual physical site. Relying on this verified geotechnical X,Y,Z input ensures that the entire georeferencing process is completely accurate, built on a solid foundation, and eliminates any need for spatial guesswork.

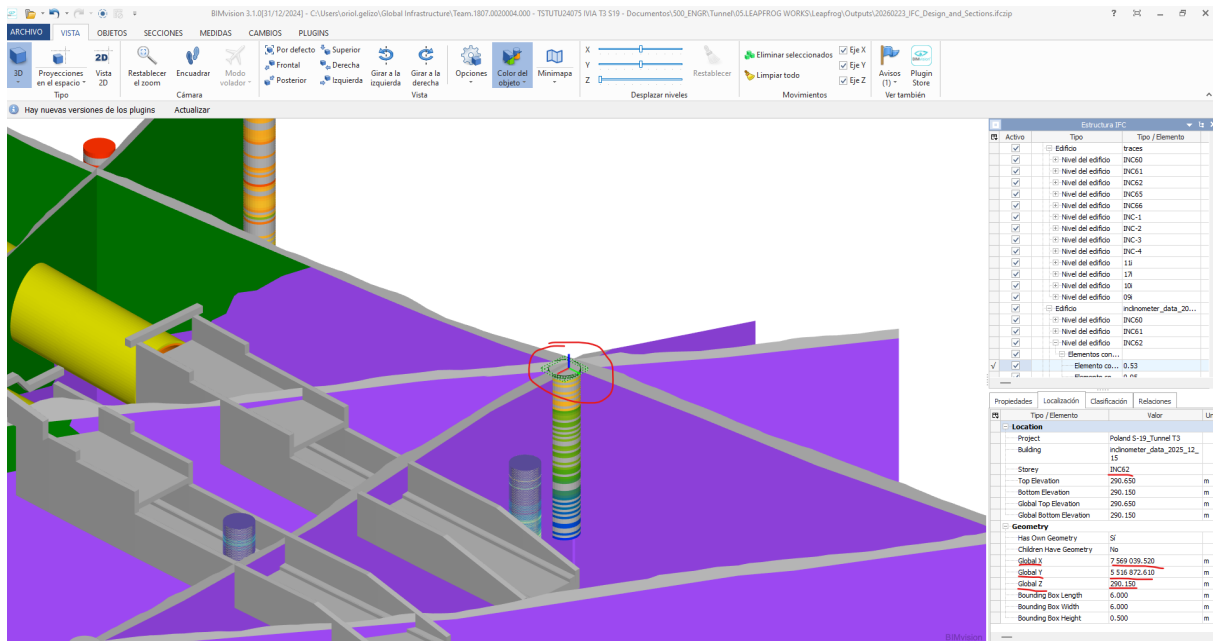


Figure 39. Identification of the localized reference point within the geological model

Once the correct coordinates are set up, the final step for the project baseline is to bring in the 3D geological model. To make sure the location is completely accurate, the soil IFC model is linked into Revit using the "IFC Origin to Project Base Point" option (Figure 40). This setting locks the soil data directly to the new coordinates so that it aligns perfectly with the tunnel structure. The result is the complete open-BIM model shown in Figure 41.

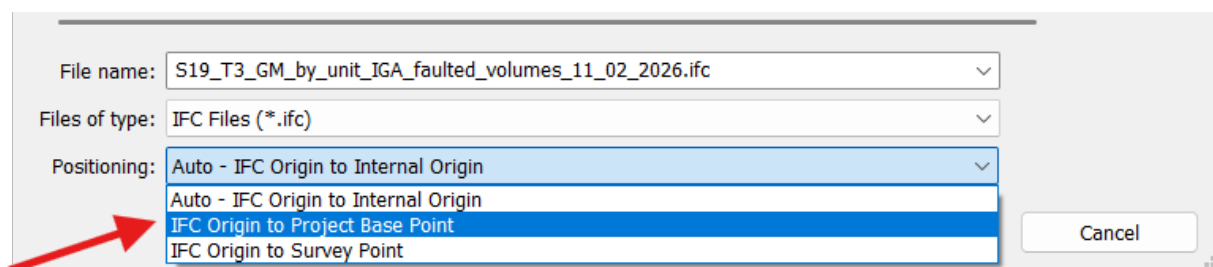


Figure 40. Selection of the "IFC Origin to Project Base Point" positioning

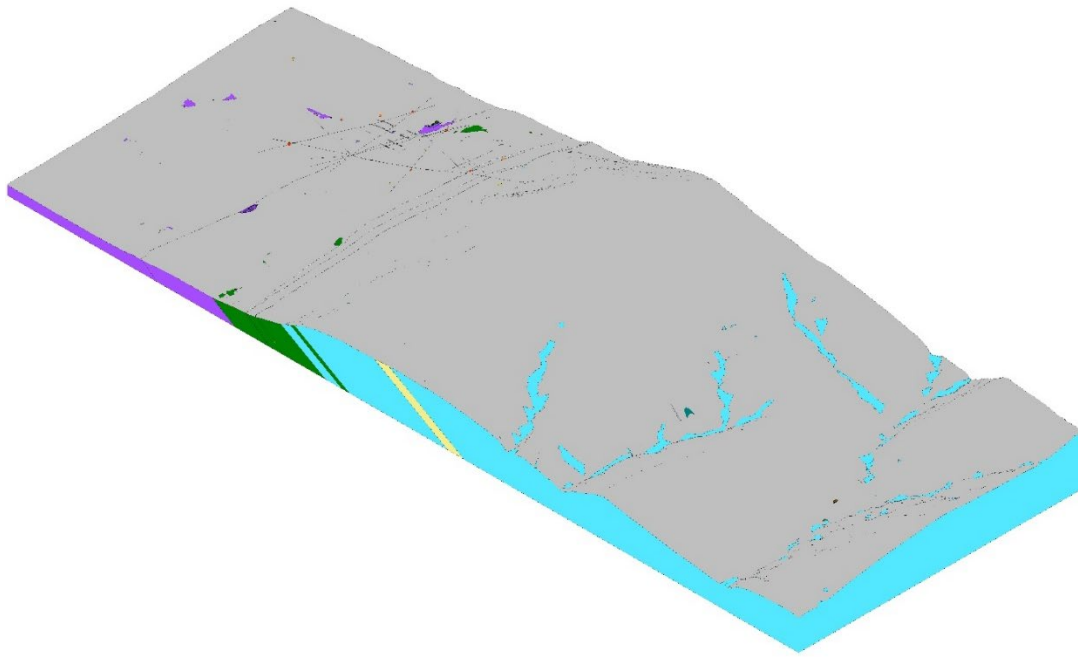


Figure 41. Tunnel + Soil (Open-BIM) Model.

To visualize better the positioning and combination of the Soil + Tunnel model, an underlay model of the soil is presented in Figure 42.

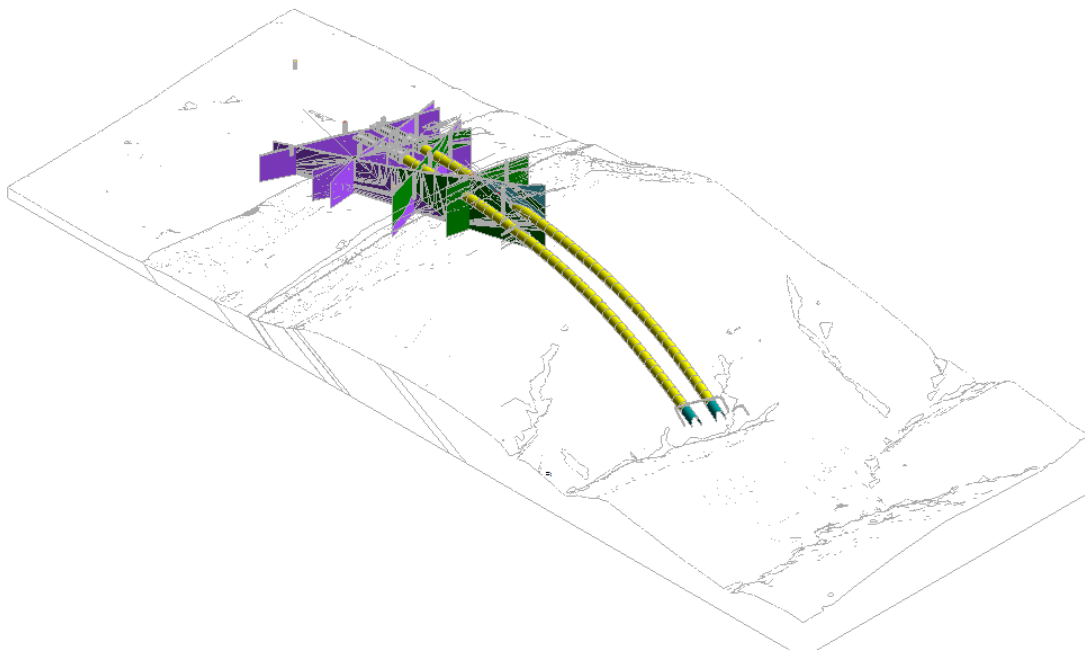


Figure 42. Underlay Soil+Tunnel Model

This complete open-BIM model is the starting point for the entire project. If the model is not placed in the exact right coordinates, all the steps that follow will give unreliable results. By combining the 3D soil layers with the tunnel structure, this phase connects geotechnical and structural data, which are usually kept separate in traditional workflows. Now, this combined Open-BIM provides a reliable and detailed model. It prepares us for Phase 2, where we will use visual programming language to pull out exact 2D cross-sections for our analysis.

In the next steps of the research, this unified open-BIM model will be our main source of information. It is more than just a 3D picture; it works as a central database. It holds both the exact shapes and the important details, like the color-coded soil layers and their material properties. In Phase 2, our automated system will read this model to automatically cut specific 2D sections along the tunnel path. Because we use this single, correctly located model as our "source of truth," we completely avoid the need to draw anything by hand. This direct process ensures that complex soil boundaries and tunnel sizes move smoothly into the numerical simulation software (RS2). It keeps the data connected and greatly lowers the chance of human mistakes when moving from the design phase to the analysis phase.

4.3 Phase 2: Algorithmic interoperability engine

This phase marks the critical transition where the static digital model begins to evolve into a functional BIM-to-FEM interoperability framework. As the core technical contribution of this research, this subchapter details the automated procedures used to extract, clean, and prepare 2D cross-sections of the integrated soil and tunnel model for numerical simulation in RS2.

A vital factor for successful automation within Building Information Modelling (BIM) and infrastructure environments is the application of Visual Programming Languages (VPL). By combining the visual node-based logic of Dynamo with the advanced scripting capabilities of Python, an "algorithmic interoperability engine" is created. This engine bypasses the limitations of traditional file exports, allowing for a direct, programmatic manipulation of the model's data.

To overcome the "information islands" commonly found in underground engineering, this phase is executed through two primary automated operations, which will be explained in detail in the subsequent sections:

1. **Automated 2D section generation (Autodesk Revit):**
Utilizing customized Dynamo scripts and Python to interact directly with the Revit API, this step systematically slices the unified 3D model to generate precise 2D cross-sections at specific intervals along the tunnel alignment.
2. **Geometry cleaning and standardization (Autodesk Civil 3D):**
Once extracted, the sections are exported as DXF files and processed through a

secondary programmatic cleaning phase in Civil 3D. Using Python and Dynamo, the geometric linework is automatically filtered, closed, and standardized to ensure it is structurally sound and ready for finite element analysis.

Replacing manual, repetitive tasks with this automated pipeline ensures high geometric precision and prevents the loss of vital data during software transfers. Beyond improving accuracy, the framework unlocks a highly adaptable design process. Should the project require changes to the tunnel trajectory or the underlying geological model, the algorithm can simply be executed again to produce fresh 2D analytical sections within minutes. This rapid regeneration effectively eliminates the severe time constraints typically associated with traditional drafting methods.

To ensure the technical reproducibility of this automated pipeline, all operations were conducted using the 2026 releases of Autodesk Revit and Autodesk Civil 3D. The automation scripts were developed within Dynamo Core 3.6, utilizing essential custom packages to expand the software's native capabilities. Specifically, packages such as “bimorphNodes” were employed for rapid geometry intersection, while Clockwork for Dynamo 3.x, Genius Loci, and Rhythm provided crucial list management and API interaction tools necessary to sustain the algorithmic engine.

4.3.1 Automated 2D section generation

The entire automated 2D section creation was done in the Dynamo environment. So, to proceed with Dynamo, first, setting up the environment was the crucial step. Dynamo is a visual programming language (VPL) that represents a significant shift in how engineers interact with Building Information Modelling (BIM) software. Instead of relying on traditional, manual drafting methods or writing complex lines of traditional computer code, VPL allows users to construct algorithmic workflows through an intuitive graphical interface. In Autodesk Revit, Dynamo serves as the primary visual programming environment. It operates by connecting functional blocks, known as “nodes,” with visual “wires” to establish a logical sequence of instructions. Each node performs a specific operation—ranging from basic mathematical calculations to complex geometry generation—allowing the user to manipulate Revit's internal database directly through its Application Programming Interface (API). This capability essentially transforms Revit from a static 3D design tool into a dynamic, programmable engine capable of handling highly repetitive and complex parametric tasks with absolute precision.

In this thesis, Dynamo works as the main link between the 3D design in Revit and the geotechnical analysis software. Building an automated workflow for underground tunnels requires more than just basic 3D shapes; it needs an automated and reliable way to transfer data. Since natural soil layers are highly irregular, they do not fit easily into the standard categories used by BIM software. Trying to model and transfer these details by hand is not only slow but also leaves a lot of room for human error.

The first practical step in the workflow is setting up the Dynamo interface. Since Dynamo is built directly into Autodesk Revit, it is very easy to open right from the manage tab, as shown in Figure 43, the tool is located under the "Manage" tab, inside the "Visual Programming" panel. Clicking the Dynamo icon opens the main workspace (Figure 44), providing the blank canvas needed to start creating the automated scripts for the project.

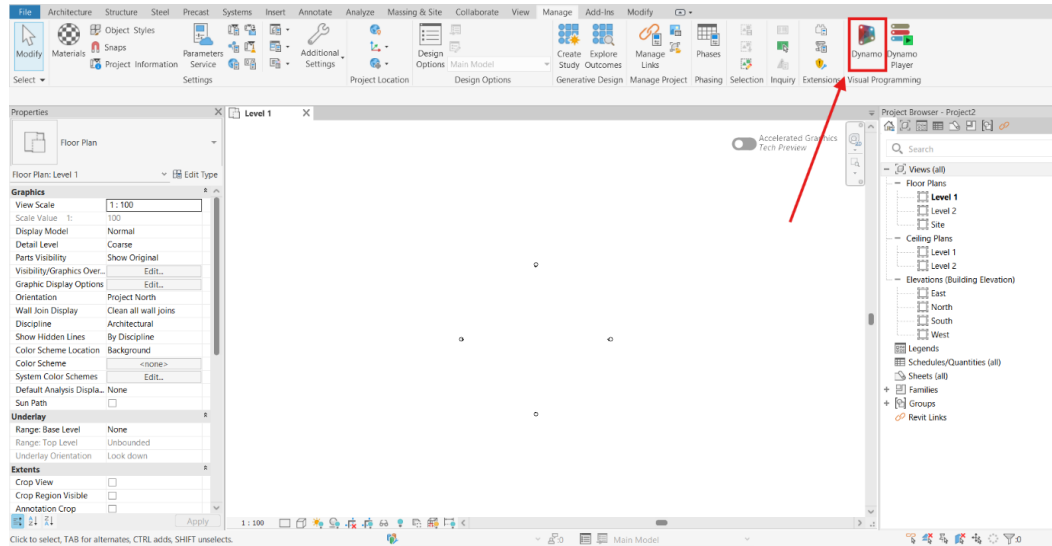


Figure 43. Dynamo tab in Revit

Once the Dynamo workspace is ready, the next step is to build the visual script using nodes and wires to generate the 2D cross-sections. To manipulate and process any 3D geometry, Dynamo first needs to read those specific elements into its active environment. Ideally, this would mean loading the entire Open-BIM model which combines both the structural tunnel and the geological soil IFC files straight into the Dynamo workspace.

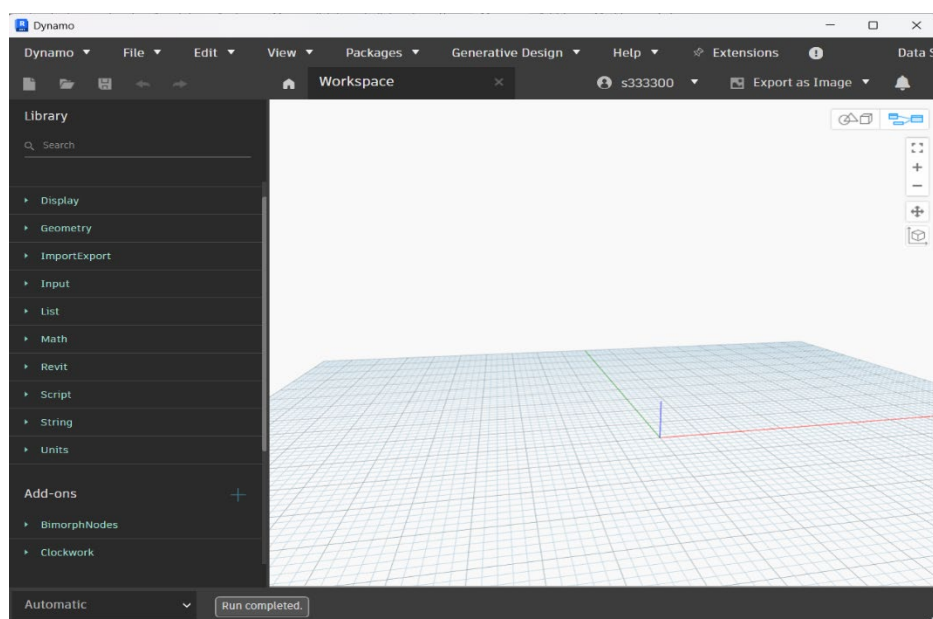


Figure 44. The default blank Dynamo workspace

However, geotechnical IFC models contain massive amounts of data. When imported into Revit, they are often processed as complex "DirectShapes." Attempting to force these heavy solid elements entirely into Dynamo usually overloads the system, resulting in incomplete geometry reading or causing the software to crash completely. Furthermore, Dynamo still struggles to seamlessly parse native IFC data structures. This software limitation is a practical example of the "information islands" problem discussed earlier in Chapter 2.

To overcome this computational barrier, a much more efficient approach was adopted. Instead of trying to process the entire solid model at once, the workflow isolates just the 3D alignment curve of the tunnel (Figure 45). By inserting this lightweight, simple line into Dynamo to serve as the geometric baseline, the script can establish the project's trajectory smoothly without overwhelming the system's memory.

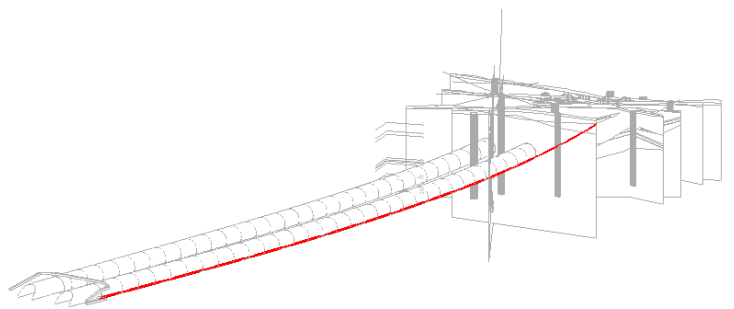


Figure 45. 3D Tunnel Curve of the Tunnel IFC model

4.3.2 Algorithmic discretization and mathematical setup

Once the Dynamo workspace is ready, the next practical step is to program the script to mathematically slice the tunnel. As discussed earlier, trying to load the massive, integrated IFC models directly into Dynamo simply does not work; the heavy "DirectShape" elements quickly overwhelm the system's memory. To solve this, the workflow relies entirely on the optimized 3D tunnel curve extracted earlier. As seen in Figure 46, this spatial curve is successfully loaded into Dynamo without any breaks or loss of geometric data.

It is critical to recognize that this extracted geometry is not a simple, straight line, but rather a complex, three-dimensional trajectory. Because real-world tunnels feature continuously varying horizontal curvatures and vertical slopes, the "forward" direction of the excavation is constantly shifting. If a simple straight line were used as a reference to cut the sections, the reference planes would not rotate with the tunnel's true geometry, resulting in diagonal, elongated slices. In 2D Finite Element Analysis, such as the simulations performed in RS2, these skewed slices would violate the fundamental assumption of plane strain, leading to

completely inaccurate calculations of soil stresses and tunnel deformations. By using this lightweight, perfectly accurate 3D curve as the central geometric spine, the algorithm guarantees a perfectly orthogonal (perpendicular) cut at every specific interval.

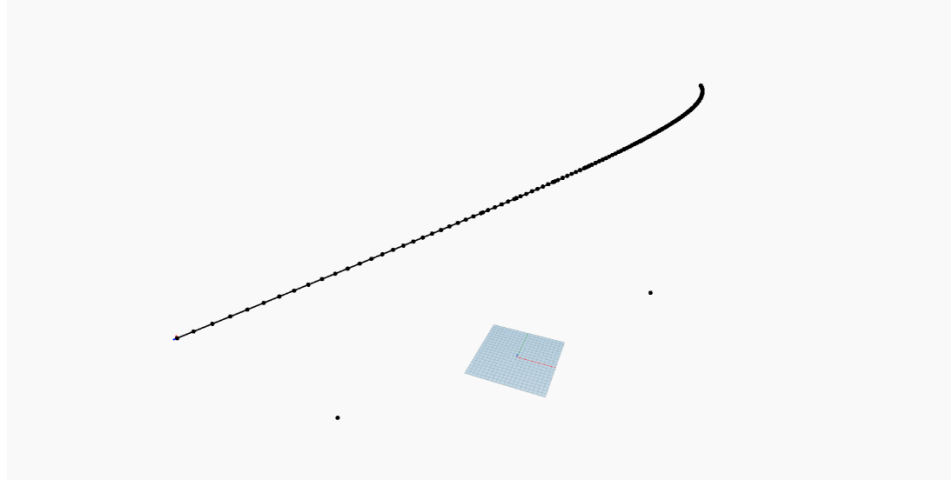


Figure 46. Tunnel curve correctly loaded in Dynamo

The core mathematical concept driving this part of the script is "discretization", the process of breaking down a continuous spatial line into a series of specific evaluation points. To perform an accurate 2D finite element analysis later in RS2, the tunnel cannot be treated as a single, continuous object; instead, it must be evaluated at discrete intervals.

In Dynamo, this logic is constructed using a sequence of mathematical nodes, as illustrated in Figure 47. The sequence begins with the Select Edge node to identify the extracted tunnel trajectory, immediately followed by the "Curve.Length" node to calculate its total absolute length. Next, an adjustable number slider is introduced to define the desired spacing for the

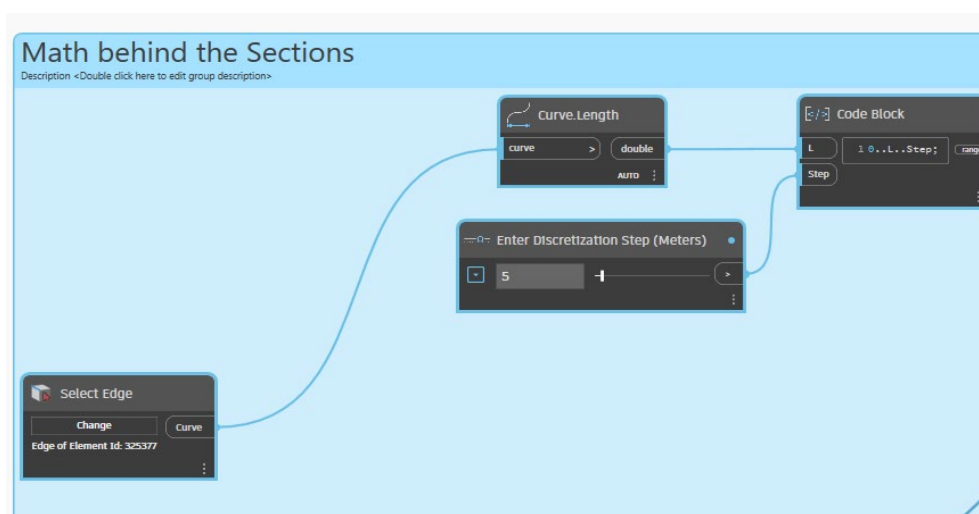


Figure 47. Dynamo node sequence for defining the spatial discretization intervals along the tunnel curve.

cross-sections. In this case study, it is set to a 5-meter interval. Notably, this slider is configured as an input parameter, meaning the spacing can be easily modified later directly through the Dynamo Player interface in Revit, without the user ever needing to reopen the visual script.

Finally, this user-defined step value and the total curve length are fed into a Code Block containing the formula $0 \dots L \dots \text{Step}$; This specific syntax automatically generates a mathematical array of exact chainage distances, dictating exactly where along the tunnel path each cross-section will be cut.

Once the distances are calculated, the script knows exactly how far apart the sections should be. However, there is a technical hurdle. Dynamo does not evaluate curves using physical measurements like meters. Instead, it uses a relative parameter system where 0 represents the start of the line and 1 represents the end. To solve this, the “Curve.ParameterAtSegmentLength” node is used. This node basically acts as a translator, converting the real-world meter distances into the relative parameters that Dynamo understands as seen in Figure 48.

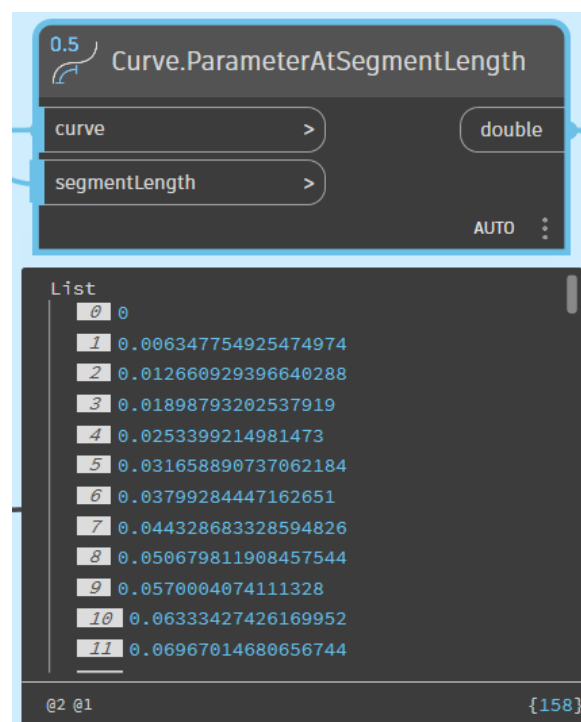


Figure 48. “Curves.ParameterAtSegmentLength” node converting real-world meter intervals into relative mathematical parameters

Next, these translated parameters are fed into two crucial geometry nodes to define exactly how the sections will be cut. The first is the “Curve.PointAtParameter” node. This tool uses the parameter to find the exact 3D coordinate (X, Y, Z) along the tunnel path, which will serve as the physical center point for the new cross-section. The second is the “Curve.TangentAtParameter” node, which calculates the exact direction the curve is pointing at that specific spot. As mentioned earlier, this tangent is essential because it provides the

"normal" direction. This guarantees that the 2D plane slices perfectly perpendicular to the tunnel, no matter how the path curves or slopes.

In addition to these geometric steps, a Views node is included to select a specific 3D working view in Revit, currently named "Raycast3D." This step simply tells the software where to generate the new elements.

As depicted in Figure 49, these data streams specifically the origin points obtained from the curve, their corresponding tangent vectors, and the designated 3D view are fed directly into the respective IN[0], IN[1], and IN[2] ports of a Python Script node. Integrating this node represents a fundamental transition from a visual programming interface to text-based coding. Ultimately, this Python component acts as the primary execution mechanism, leveraging the compiled inputs to mathematically instantiate the actual section views directly within the Revit model.

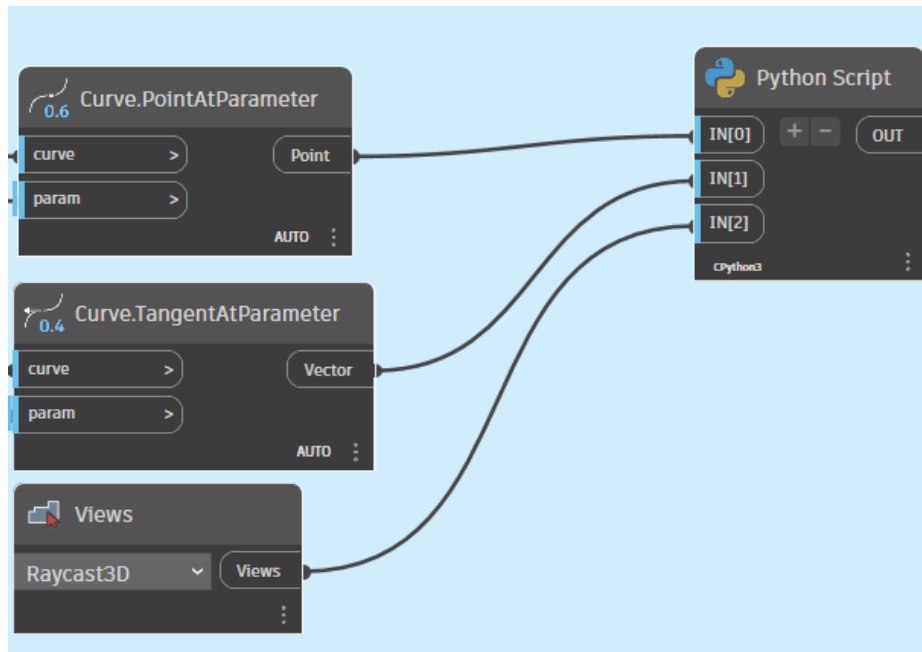


Figure 49. Dynamo inputs connected to Python

4.3.3 Algorithmic execution of the geological radar

The Python script embedded within the visual programming workflow serves as the core analytical engine of the methodology. It bridges the gap between Dynamo's visual geometry and the underlying Revit Application Programming Interface (API). By leveraging custom code, the script bypasses the limitations of standard nodes, allowing for advanced mathematical operations, spatial raycasting, and automated data extraction from linked building information models. The code is structured into several distinct logical sequences, each responsible for a

specific phase of the geological and spatial analysis. Each phase will be explained in detail in the following stages.

The initial section of the script establishes the necessary software environment by importing required system and software-specific libraries as seen in Figure 50. The Common Language Runtime is used to add references to the core Revit API, enabling direct interaction with the active Revit document's database. Additionally, document management services are imported to ensure the script can read the current model space. A critical component of this initialisation is the inclusion of Dynamo's geometry conversion extensions, which provide the necessary translation mechanisms to convert Dynamo-native geometric data, such as points and vectors, into Revit-native coordinates that the API can natively process.

```
1  import clr
2  import math
3
4  # Import RevitAPI
5  clr.AddReference('RevitAPI')
6  from Autodesk.Revit.DB import *
7
8  # Import DocumentManager
9  clr.AddReference('RevitServices')
10 import RevitServices
11 from RevitServices.Persistence import DocumentManager
12
13 # Import Dynamo Geometry Conversion
14 clr.AddReference("RevitNodes")
15 import Revit
16 clr.ImportExtensions(Revit.GeometryConversion)
17
18 doc = DocumentManager.Instance.CurrentDBDocument
19
```

Figure 50. Importing required Revit and Dynamo libraries.

After the initial setup is complete, the Python script takes the geometry data that was just prepared in Dynamo. It saves the exact points along the curve, the direction the tunnel is pointing (the tangents), and the chosen 3D view into its memory. To easily keep track of where everything is, the script gives a simple number from 0 to 1 to every point along the tunnel's path, where 0 is the beginning of the tunnel and 1 is the end.

At the same time, the script prepares a powerful tool inside Revit called the "Reference Intersector." You can think of this tool as a virtual laser scanner. It shoots invisible, straight lines (rays) outwards through the 3D space to see what objects they hit.

Because the tunnel structure and the surrounding soil layers are combined, this virtual laser is programmed to scan and detect these specific 3D shapes. When a ray hits a layer of soil or rock, the script uses a custom command to read that object's "IfcName" parameter. By doing this, the code in Figure 51, can automatically identify exactly which type of geological material is surrounding the tunnel at any specific location.

```

20 # --- INPUTS ---
21 points = IN[0]
22 tangents = IN[1]
23 view = UnwrapElement(IN[2])
24
25 num_points = len(points)
26 parameters = [float(i) / (num_points - 1) for i in range(num_points)] if num_points > 1 else [0.0]
27
28 category_filter = ElementCategoryFilter(BuiltInCategory.OST_GenericModel)
29 intersector = ReferenceIntersector(category_filter, FindReferenceTarget.Element, view)
30 intersector.FindReferencesInRevitLinks = True
31
32 def get_ifc_name(hit):
33     if hit == None:
34         return "Unknown"
35     ref = hit.GetReference()
36     link_doc = doc.GetElement(ref.ElementId).GetLinkDocument()
37     if link_doc:
38         hit_element = link_doc.GetElement(ref.LinkedElementId)
39         param = hit_element.LookupParameter("IfcName")
40         if param and param.HasValue:
41             return param.AsString()
42     return "Unknown"

```

Figure 51. Data collection and virtual raycasting

Before starting the spatial scans, the script sets up the physical settings for a virtual "geological radar." It defines a search area by sending out rays every 30 degrees up to a distance of 50 meters from the alignment curve, while also including a roof offset to account for the tunnel's inner size. Since the Revit API internally uses imperial units, the code converts these metric measurements into feet to ensure the spatial searches are accurate.

At the same time, the script calculates the true distance along the tunnel's path, known as the chainage (distance from the start of the tunnel). It does this by iterating through the list of starting points and measuring the distance between each one, creating a running total in meters.

```

46 # --- THE GEOLOGICAL RADAR SETTINGS ---
47 laser_distance_ft = 50.0 * 3.28084
48 number_of_lasers = 12
49 dist_to_roof_ft = 7.0 * 3.28084
50
51 depths = []
52 soil_profiles = []
53
54 # --- THE FIX: Calculate chainage natively in meters ---
55 distances_m = [0.0]
56 for i in range(1, len(points)):
57     # Removed the * 0.3048 because Dynamo points are already in metric!
58     dist = points[i].DistanceTo(points[i-1])
59     distances_m.append(distances_m[i-1] + dist)
60
61 # --- PASS 1: SCAN THE CORRIDOR ---
62 for pt, tangent in zip(points, tangents):
63     revit_pt = pt.ToXYZ()
64     forward = tangent.ToXYZ().Normalize()
65     world_up = XYZ.BasisZ
66
67     # 1. Depth Check (For Max/Min Cover)
68     hits_up = intersector.Find(revit_pt, world_up)
69     z_values = [h.GetReference().GlobalPoint.Z for h in hits_up]
70
71     if len(z_values) > 0:
72         depths.append((max(z_values) - revit_pt.Z) - dist_to_roof_ft)
73     else:
74         depths.append(-1.0)
75

```

Figure 52. Python script for the simulated geological radar part 1

Finding this exact distance is crucial for the final results (Figure 52), as it guarantees that each generated cross-section is labelled with its correct real-world position along the infrastructure.

After the initial setup, the script at Figure 53 starts a continuous 50-meter radar scan. A distance approximating 3D (three times the tunnel diameter). It first sets up a local coordinate system perpendicular to the tunnel's direction, allowing search rays to project radially outward. The algorithm then calculates the exact angle and direction for every scan.

```

76 | # 2. The 50m Continuous Radar
77 | if abs(forward.Z) > 0.99:
78 |     right = XYZ.BasisX
79 | else:
80 |     right = forward.CrossProduct(world_up).Normalize()
81 |
82 | local_up = right.CrossProduct(forward).Normalize()
83 |
84 | current_step_materials = set()
85 |
86 | for i in range(number_of_lasers):
87 |     angle = (2 * math.pi / number_of_lasers) * i
88 |     dir_vec = right.Multiply(math.cos(angle)).Add(local_up.Multiply(math.sin(angle)))
89 |
90 |     hits = intersector.Find(revit_pt, dir_vec)
91 |
92 |     for h in hits:
93 |         hit_pt = h.GetReference().GlobalPoint
94 |         dist = hit_pt.DistanceTo(revit_pt)
95 |
96 |         if dist <= laser_distance_ft:
97 |             mat_name = get_ifc_name(h)
98 |
99 |             if not any(ignored in mat_name for ignored in ignore_list) and mat_name != "Unknown":
100 |                 current_step_materials.add(mat_name)
101 |
102 | if len(current_step_materials) == 0:
103 |     current_step_materials.add("Uniform Geology")
104 |
105 | soil_profiles.append(tuple(sorted(list(current_step_materials))))
106 |

```

Figure 53. Python script for the simulated geological radar part 2

After the primary scan finishes and all topological data is saved, the script starts a second loop to narrow down the large dataset to just the most important cross-sections. The main goal is to find the exact points along the curve where section views are needed for structural or geological reasons.

The algorithm first isolates the locations of maximum and minimum overburden by parsing the array of calculated vertical depths. It extracts the curve parameters corresponding to these extremes and generates highly specific view names denoting the condition and the exact chainage. Following this, the script sequentially compares the compiled radial soil profiles along the entire alignment. By checking the detected material set at a given location against the set from the immediately preceding location, the algorithm can precisely pinpoint where a geological transition occurs within the fifty-meter influence zone. Each identified transition is recorded, mapped to its curve parameter, and assigned a standardized name reflecting its sequential section number and chainage. Ultimately, these filtered critical parameters and their dynamically view names are packaged and pushed out of the Python node, ready to drive the final view creation process. The script is shown in Figure 54.

```

108 # --- PASS 2: FILTER THE CRITICAL SECTIONS ---
109 critical_params = []
110 view_names = []
111
112 # 1. Extract Maximum and Minimum Cover Locations
113 valid_depths = [d for d in depths if d > 0]
114 if valid_depths:
115     max_d = max(valid_depths)
116     min_d = min(valid_depths)
117
118     max_idx = depths.index(max_d)
119     min_idx = depths.index(min_d)
120
121     critical_params.append(parameters[max_idx])
122     view_names.append("Geo - Maximum @ " + str(round(distances_m[max_idx], 1)) + "m")
123
124     critical_params.append(parameters[min_idx])
125     view_names.append("Geo - Minimum @ " + str(round(distances_m[min_idx], 1)) + "m")
126
127 # 2. Extract All Geological Changes within the 50m Bubble
128 counter = 1
129 for i in range(1, len(soil_profiles)):
130     prev_profile = soil_profiles[i-1]
131     curr_profile = soil_profiles[i]
132
133     if prev_profile != curr_profile:
134         name = "Geo - Section " + str(counter) + " @ " + str(round(distances_m[i], 1)) + "m"
135
136         if parameters[i] not in critical_params:
137             critical_params.append(parameters[i])
138             view_names.append(name)
139             counter += 1
140
141 # --- OUTPUT ---
142 OUT = [critical_params, view_names]

```

Figure 54. Final script phase identifying and labelling critical geological sections.

The Code Block after the python node acts as a bridge between the custom Python code and the rest of the visual workspace. It takes the bundled results and unpackages them into two clear categories: "Params" for the cross-section data and "Names" for the drawing titles. This simple step makes sure the system can handle the numerical variables and the text labels independently as the workflow finishes up. The whole math behind the 2D sections dynamo script is shown in Figure 55.

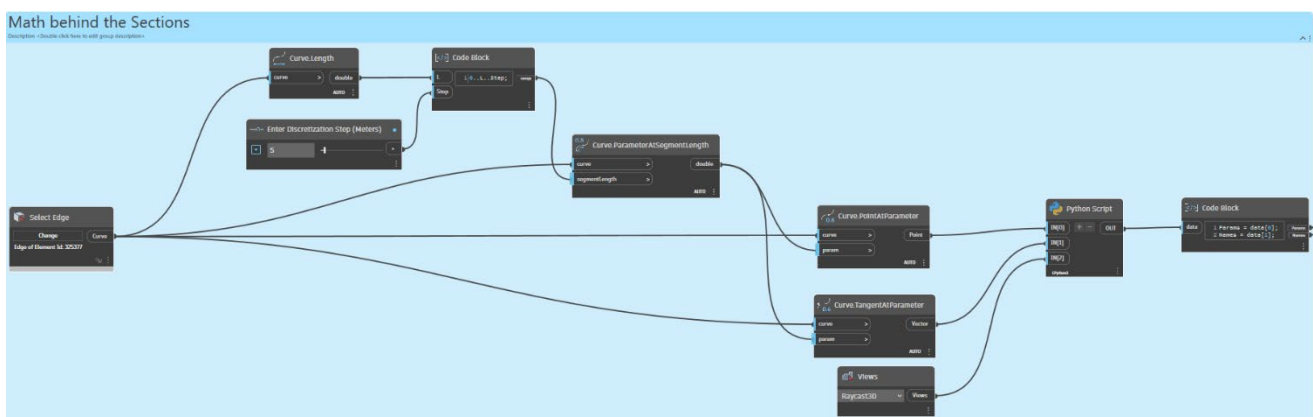


Figure 55. Math behind the 2D sections in Dynamo

4.3.4 Generation of orthogonal section views and boundary conditions

After calculating the math in the previous phase, the script moves on to the "2D section Creation" group, which you can see as a whole in Figure 56. While the first part figured out exactly where to make the cuts, this section actually builds the physical section views and their boundaries inside the Revit model.

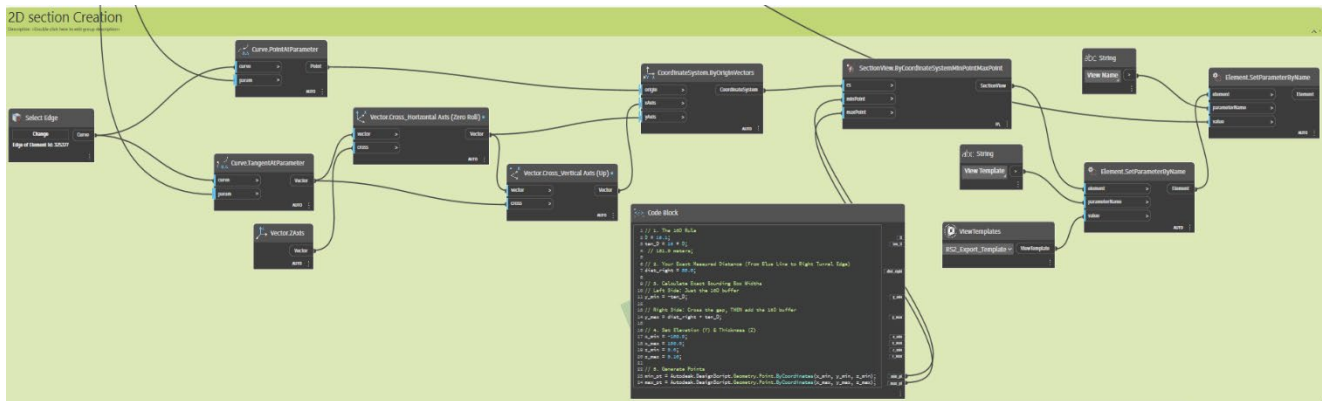


Figure 56. 2D Section Creation Group in Dynamo

The first step in generating the 2D sections is defining the exact location and orientation for each cut along the full tunnel alignment. This process takes place within the "2D section Creation" group in the visual workspace. The script first uses a "Select Edge" node to identify the main 3D curve of the tunnel model. To determine the specific locations for the sections, the script relies on the "Curve.PointAtParameter" and "Curve.TangentAtParameter" nodes. The parameter inputs for both nodes are fed directly from the Code Block that unpacks the numerical results from the custom Python script. By using these calculated parameters, the Point node

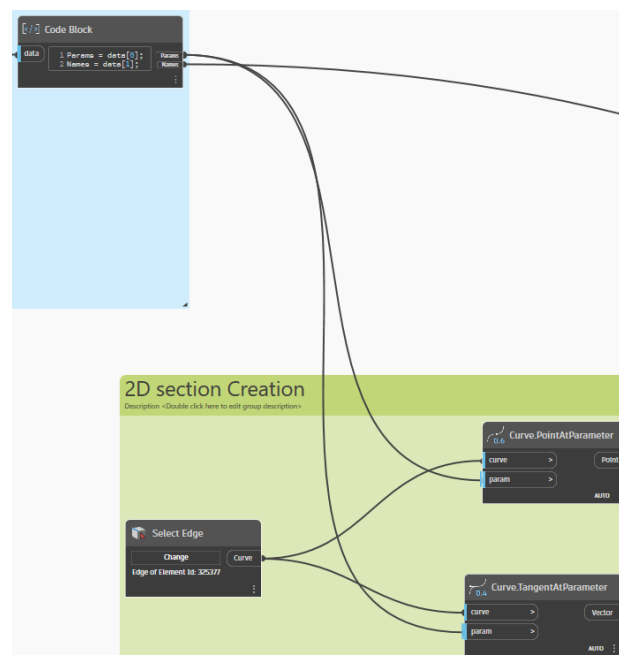


Figure 57. 2D Section Curve Parameter Node Inputs

establishes the exact 3D origin location for the slice, while the Tangent node calculates the forward directional vector, showing exactly where the tunnel is pointing at that specific interval. (Figure 57)

Once the starting point and forward direction are established, the script calculates how to orient the section plane, so it is perfectly perpendicular to the tunnel path. Because the tunnel constantly changes its slope and curves in 3D space, the script uses cross-product mathematics to find the correct angles. First, it takes the forward tangent vector and crosses it with the global Z-axis using the “Vector.ZAxis” node. This calculation is done inside the “Vector.Cross_Horizontal” Axis (Zero Roll) node to create a flat, sideways vector. After that, the script takes this new sideways vector and crosses it again with the original forward direction inside the “Vector.Cross_Vertical_Axis_(Up)” node. This final calculation finds the true vertical "up" direction that correctly matches the local slope of the tunnel for that specific cross-section.

Finally, all of these calculated geometric elements are brought together into the “CoordinateSystem.ByOriginVectors” node to establish the dynamic reference plane. This node requires three specific inputs to build the new local coordinate system, and the script wires them in directly from the previous steps. The *origin* input receives the exact 3D location generated by the “Curve.PointAtParameter” node. The *xAxis* input is fed the flat, sideways vector calculated by the first cross-product, which establishes the horizontal width of the plane. Lastly, the *yAxis* input receives the true vertical vector generated by the second cross-product. By combining this origin point with the custom X and Y axes, the node successfully generates a perfectly oriented local coordinate system that dynamically twists and slopes in complete alignment with the 3D tunnel geometry. The wiring and nodes for this step is seen in Figure 58.

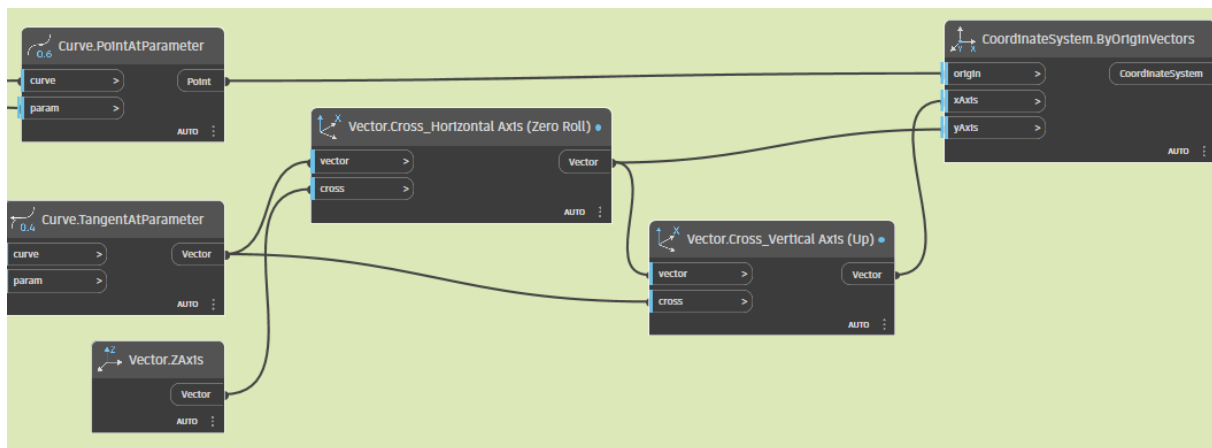


Figure 58. Integration of the calculated origin point and custom axes to generate the dynamic local coordinate system.

In geotechnical computer modelling, setting the correct physical boundaries for a simulation is a very important step. When we simulate a tunnel excavation using finite element software like RS2, the program calculates how the ground moves and how the stresses change. However, the computer model must have a limited size to work properly. If the artificial edges of this model are placed too close to the tunnel, they will act like rigid walls. This stops the soil from moving

naturally. This problem is called the "boundary effect." It makes the ground in the model seem much stiffer than it really is, which leads to incorrect calculations for ground settlement and the forces acting on the tunnel.

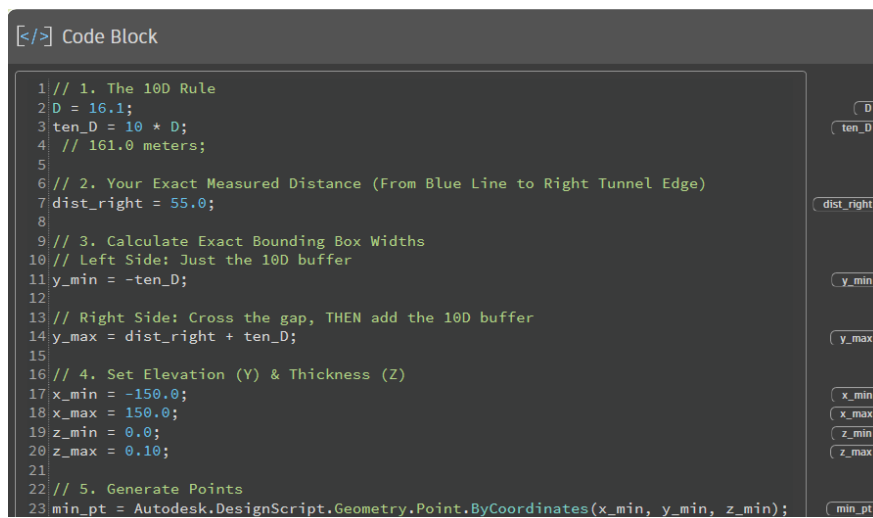
To avoid this problem, engineers widely use a standard guideline known as the 10D Rule. This rule states that the outer edges of the numerical model must be placed at a distance of at least ten times the tunnel diameter (10D) away from the center of the excavation. By leaving this large safety space, the main area where the soil deforms is kept completely away from the fixed edges of the model. Because the boundaries are far enough away, the stress calculations do not hit the edges and bounce back. This allows the software to accurately simulate an infinite, real-world ground condition.

The importance of the 10D rule and the dangers of the boundary effect have been confirmed by many recent geotechnical studies. For example, recent 3D numerical analyses of tunnel excavations emphasize that boundary distances of at least 8 to 10 times the tunnel diameter are necessary to eliminate artificial stiffness and correctly evaluate the structural behaviour of the tunnel (Li, et al., 2024).

After establishing the dynamic local coordinate system, the script uses a Code Block seen in Figure 59 to precisely calculate the outer limits of the numerical model. This is where the 10D Rule is mathematically applied to the workflow. First, the script defines the tunnel diameter as 16.1 meters and multiplies it by ten to create a strict 161.0-meter safety buffer.

The script then uses this buffer to calculate the horizontal width of the model box. For the left side boundary, it simply applies the 161.0-meter buffer directly. However, the calculation for the right side is different because of the adjacent second tunnel. To account for this, the script first adds the exact 55.0-meter gap between the two tunnels before applying the 161.0-meter safety buffer.

Next, the script sets the vertical limits of the model from -150.0 meters to +150.0 meters. These fixed elevations make sure that the model captures all the necessary underground soil layers. Finally, the script defines the thickness of the section view. It limits this thickness to a mere



```

[</>] Code Block
1 // 1. The 10D Rule
2 D = 16.1;
3 ten_D = 10 * D;
4 // 161.0 meters;
5
6 // 2. Your Exact Measured Distance (From Blue Line to Right Tunnel Edge)
7 dist_right = 55.0;
8
9 // 3. Calculate Exact Bounding Box Widths
10 // Left Side: Just the 10D buffer
11 y_min = -ten_D;
12
13 // Right Side: Cross the gap, THEN add the 10D buffer
14 y_max = dist_right + ten_D;
15
16 // 4. Set Elevation (Y) & Thickness (Z)
17 x_min = -150.0;
18 x_max = 150.0;
19 z_min = 0.0;
20 z_max = 0.10;
21
22 // 5. Generate Points
23 min_pt = Autodesk.DesignScript.Geometry.Point.ByCoordinates(x_min, y_min, z_min);
  
```

Figure 59. Code Block for creating the bounding box

0.10 meters, ensuring that the generated view remains a completely flat 2D slice without showing any extra 3D background objects.

In the final step of this sequence, all the calculated geometric data is brought together into the “SectionView.ByCoordinateSystemMinPointMaxPoint” node. This node acts as the primary generator for the automated views. As shown in the visual workspace Figure 60, it requires three specific inputs to function. First, the cs input receives the dynamic local coordinate system from the previous phase. This tells Revit exactly where to place the section plane and how to orient it perfectly perpendicular to the tunnel's 3D curve. Next, the minPoint and maxPoint inputs receive the exact diagonal corners of the bounding box that were just generated by the Code Block in Figure 59. By combining this perfectly aligned reference plane with the strict geotechnical boundaries of the 10D Rule, the node automatically draws the new 2D section views and places them directly inside the Revit project browser.

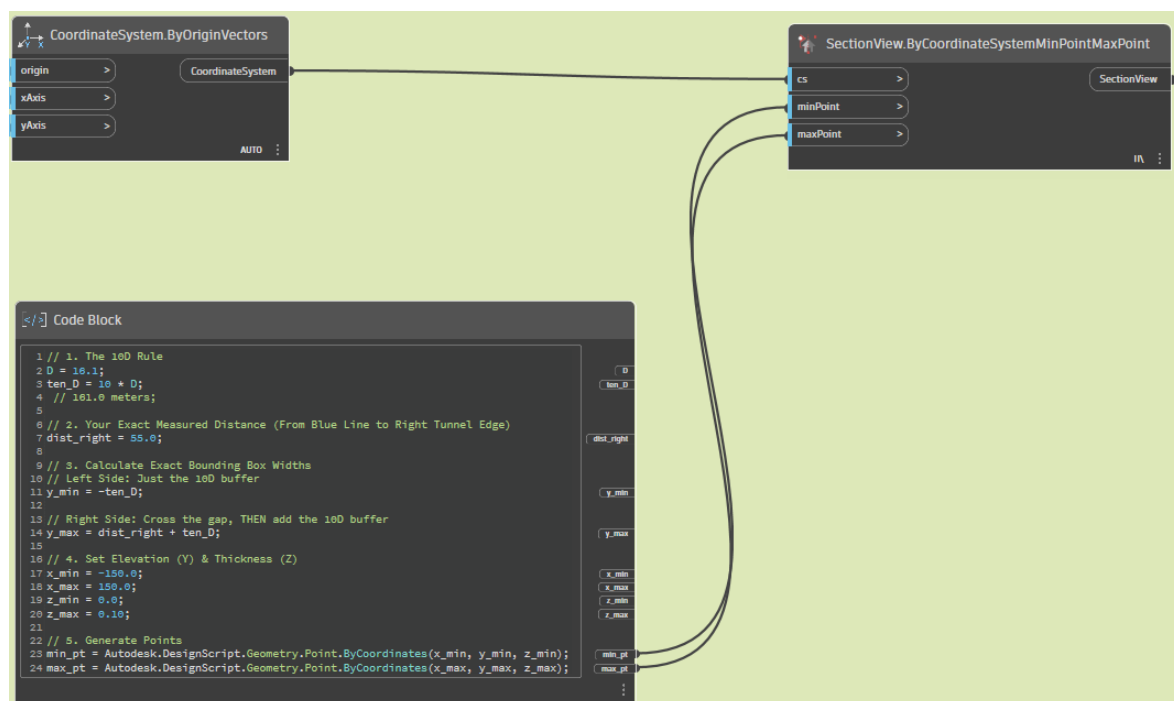


Figure 60. Wiring configuration for the final generation of the section views

Once the Dynamo script completes its execution, the automated results are immediately visible within the Revit environment. Instead of generating a disorganized list of generic sections, the script successfully organizes the project browser by assigning a specific, descriptive title to each new view. For example, a generated view is automatically given a structured name like "Geo - Section 7 @ 35.0m," which clearly indicates its exact position along the tunnel alignment. This automated naming convention saves significant manual labor and ensures the project remains perfectly organized for the numerical modelling phase.

Alongside the automated naming, the script also successfully applies the necessary visual settings to the view's properties. To achieve this, the custom *Rhythm* node package for Dynamo was used to specifically call and assign the pre-configured View Template. By applying the "RS2_Export_Template," Revit automatically adjusts the graphics of the newly created section.

It instantly hides unnecessary 3D elements, turns off background clutter, and standardizes the display of the soil layers and tunnel geometry. As a result of these automated property

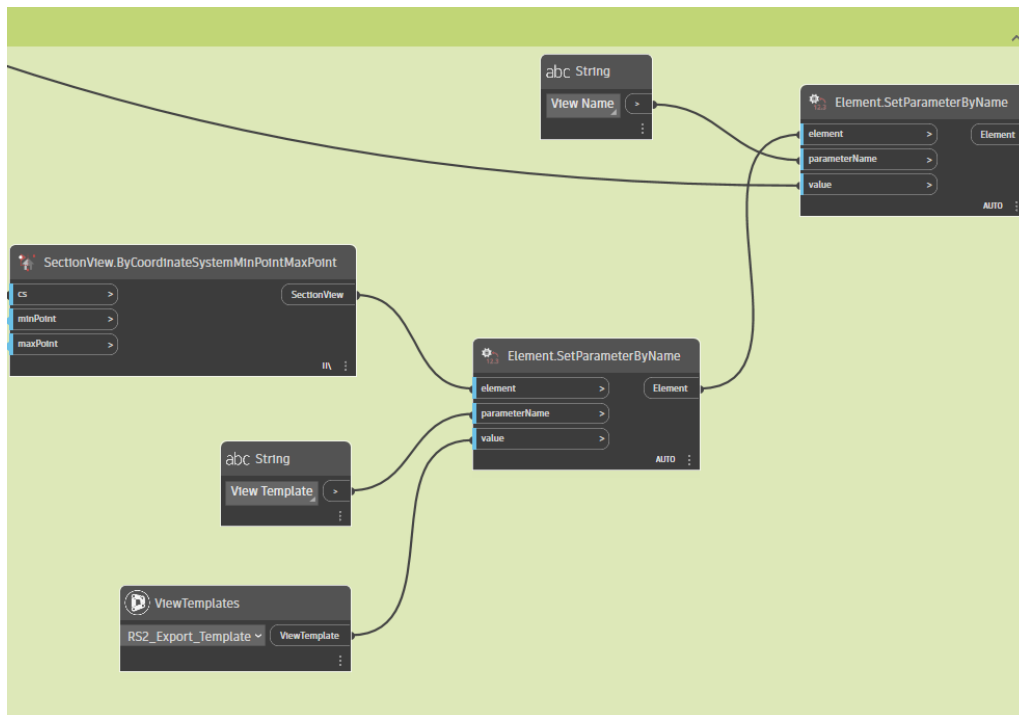


Figure 61. Automated application of the view name and view template properties

adjustments, the final cross-sections are completely clean, strictly two-dimensional, and fully optimized for immediate export into the numerical simulation software. The final part of the script for creating the View Templates and Setting Parameters is shown below in Figure 61.

To provide a clear perspective on the overall workflow, Figure 62 presents a comprehensive overview of the Dynamo script. It displays both the "Math behind the Sections" and the "2D section Creation" groups together, illustrating the complete structural logic required to generate the automated cross-sections. The results produced by this automated workflow, along with their subsequent application in the numerical modelling phase, will be presented in detail in the next chapter, Chapter 5. However, some examples of the created sections are shown below.

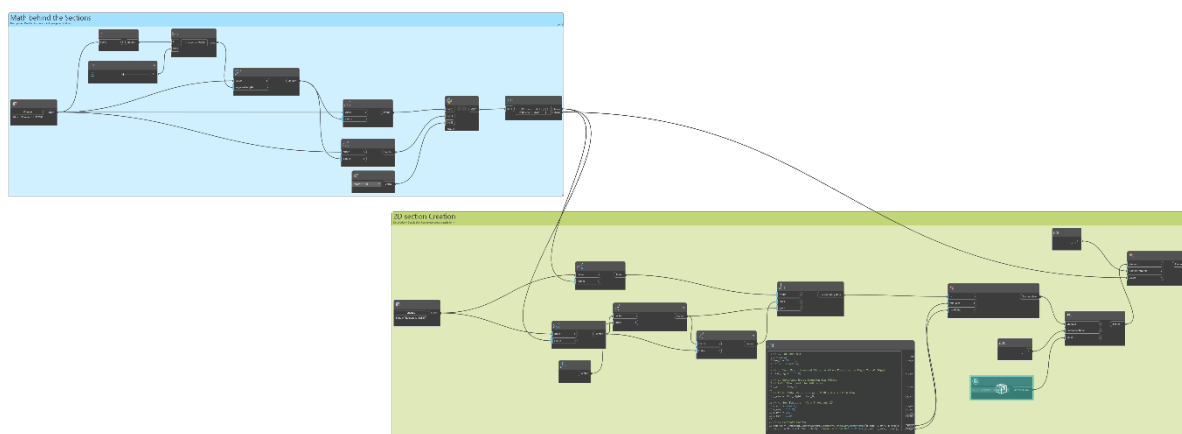


Figure 62. Complete structural layout of the Dynamo script for automated section generation

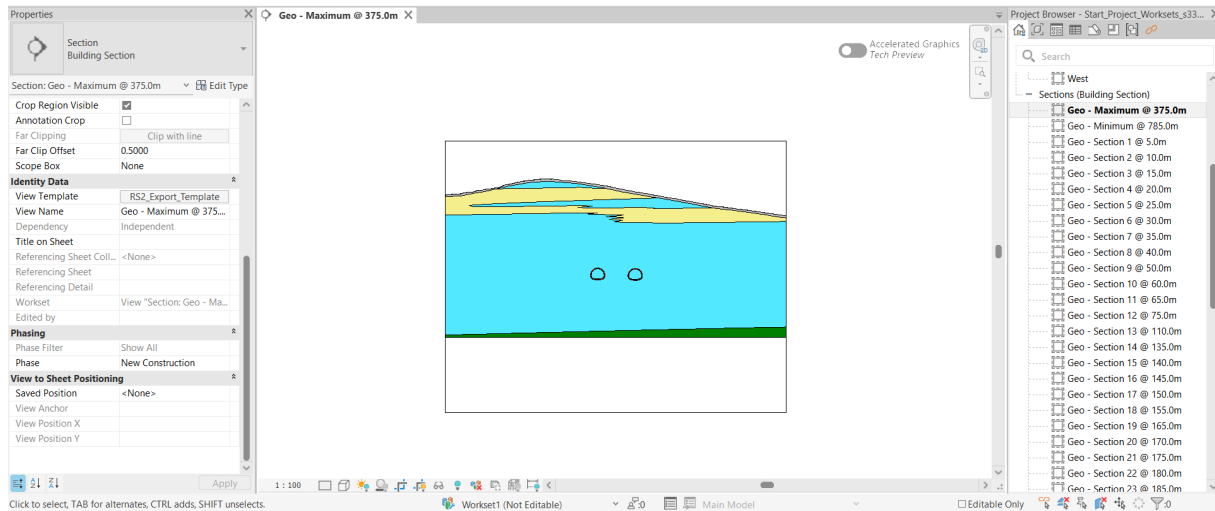


Figure 63. Automated 2D cross-section in Revit representing the maximum soil overhead at the 375.0-meter mark

As an example of the final output, Figure 63 illustrates the specific cross-section where the twin tunnels experience the maximum soil overhead. Located at the 375.0-meter mark along the alignment, this view clearly displays the thickest layer of stratigraphy above the excavation. Additionally, the project browser on the right side of the interface (Figure 64) demonstrates the script's success in batch-generating and sequentially organizing all the required sections along the entire tunnel path, fully preparing them for the DXF export process.

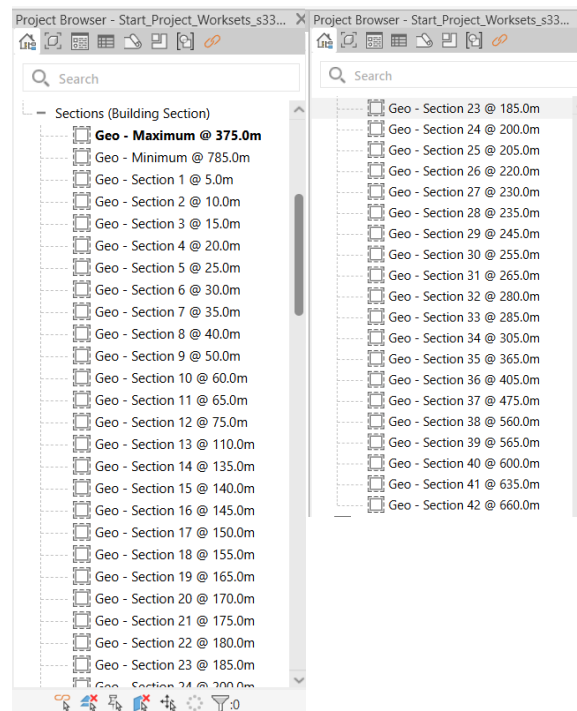


Figure 64. Sequential organization of the automated cross-sections along the tunnel alignment within the project browser.

To make the automated workflow more accessible and efficient, the script was designed to run directly through the Dynamo Player Located in the Manage tab of Revit shown in Figure 65, this built-in tool provides a simplified user interface that allows engineers to execute complex visual programming scripts without needing to open the backend Dynamo workspace.

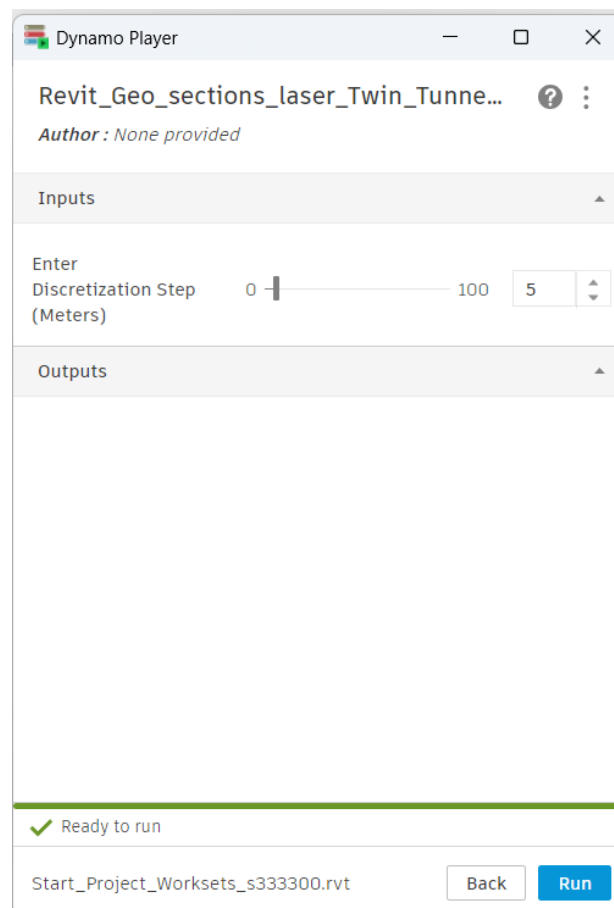


Figure 65. The Dynamo player interface displaying the adjustable discretization step

Within the Dynamo Player interface, a specific input parameter was established called the "Discretization Step." This control allows the user to define the baseline evaluation distance, in meters, along the tunnel alignment. For example, setting this slider to a value of 5 establishes a 5-meter interval for the script to analyze. However, it is important to note that the system does not simply generate a redundant section at every single 5-meter mark. Instead, this user-defined input feeds directly into the custom Python scripting discussed previously in this section.

4.3.5 DXF export and data transfer from Revit to Civil 3D

After successfully generating and organizing the 2D section views in Revit, the next critical step is transferring this geometric data into the RS2 numerical modelling environment. RS2 utilizes the Drawing Exchange Format (DXF) as its primary method for importing external geometry. While Revit possesses a native DXF export function, a direct data transfer from Revit to RS2 is highly problematic and in many cases non-functional due to strict geometric incompatibilities between building information modelling (BIM) software and finite element analysis (FEA) environments.

The core issue lies in how the two software packages interpret line data. For RS2 to successfully generate a mesh and calculate stresses, it requires all imported excavation boundaries and soil stratigraphy lines to be strictly formatted as continuous, unbroken polylines. However, Revit's native DXF exporter does not natively support this requirement. Instead, it breaks down complex section geometries into individual, disconnected line segments or groups them together as rigid block references. As highlighted in recent literature concerning BIM-to-FEM interoperability in mechanized tunnelling, direct geometric translation often results in data fragmentation, topology errors, and boundary disconnections that severely disrupt FEA meshing algorithms (Alsahly, Hegemann, König, & Meschke, 2020). Attempting to import these fragmented Revit blocks directly into RS2 leads to open boundaries, overlapping nodes, and complete topological failures during the meshing process.

This interoperability gap between BIM models and numerical simulation software is a widely recognized challenge in geotechnical engineering. Researchers have consistently noted that the automatic transition from parametric BIM geometries to the explicit nodes and elements required for finite element modelling is currently unreliable without manual intervention; therefore, an intermediate geometric processing stage is often required to ensure the model is fit for computational analysis (Fabozzi, Biancardo, Veropalumbo, & Bilotta, 2020).

To resolve this issue and bridge the gap between the software, AutoCAD Civil 3D is introduced as an intermediate processing platform in the workflow. The raw, fragmented DXF files generated by Revit are first transferred into Civil 3D. Within this environment, the fragmented lines and blocks undergo a dedicated geometric cleaning process where they are exploded, trimmed, and joined into continuous polylines. Once the linework is fully optimized and completely closed, the geometry is re-exported from Civil 3D as a clean DXF file. This intermediate step ensures the data is strictly formatted as polylines, making it perfectly tailored for a seamless import and successful meshing process within RS2.

One of the main challenges during the DXF export process is losing the BIM data. When Revit geometry is exported into a standard DXF format, all the material information and object properties are lost, leaving only simple 2D lines. To make sure that the different soil layers stay organized and are easy to classify when imported into Civil 3D (and later into RS2), a specific color strategy was applied directly in the Revit View Template.

To do this, rule-based filters were added to the RS2_Export_Template view template as shown in Figure 66. Since the geological model and the tunnel were originally brought into Revit through an IFC file, the 3D soil solids were read as "Generic Models" by the software.

Therefore, the filters were set up to identify each specific soil layer by using its unique “IfcGUID” parameter.

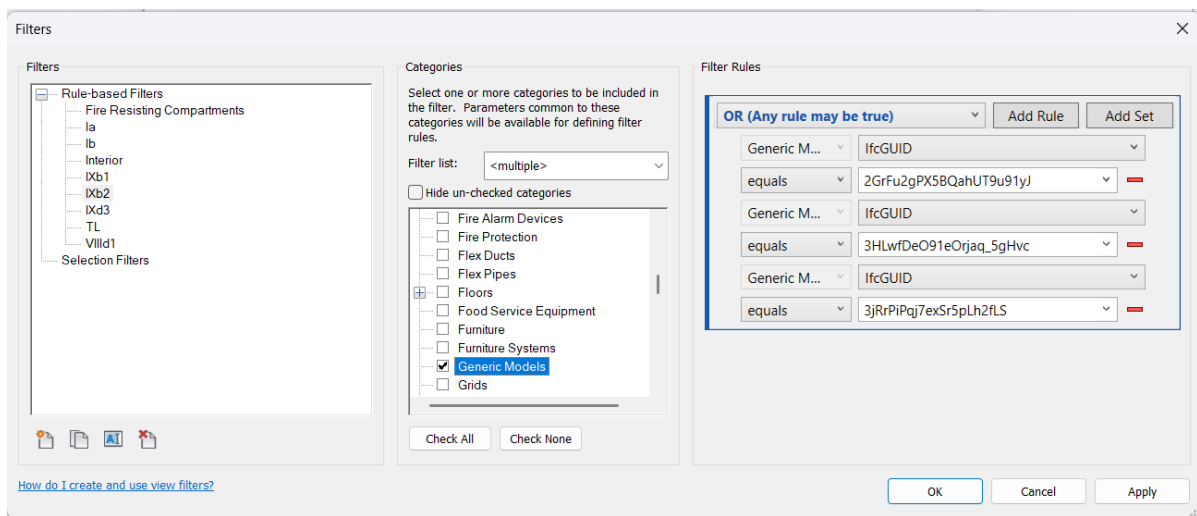


Figure 66. Configuration of the rule-based filters using the unique IfcGUID of the imported soil solids

By setting up these filters before exporting as seen in Figure 67, the 2D sections keep a clear visual organization. When the sections are finally exported, this color mapping helps to easily separate and classify the DXF lines into different layers inside Civil 3D, getting the soil geometry completely ready for the numerical modelling phase.

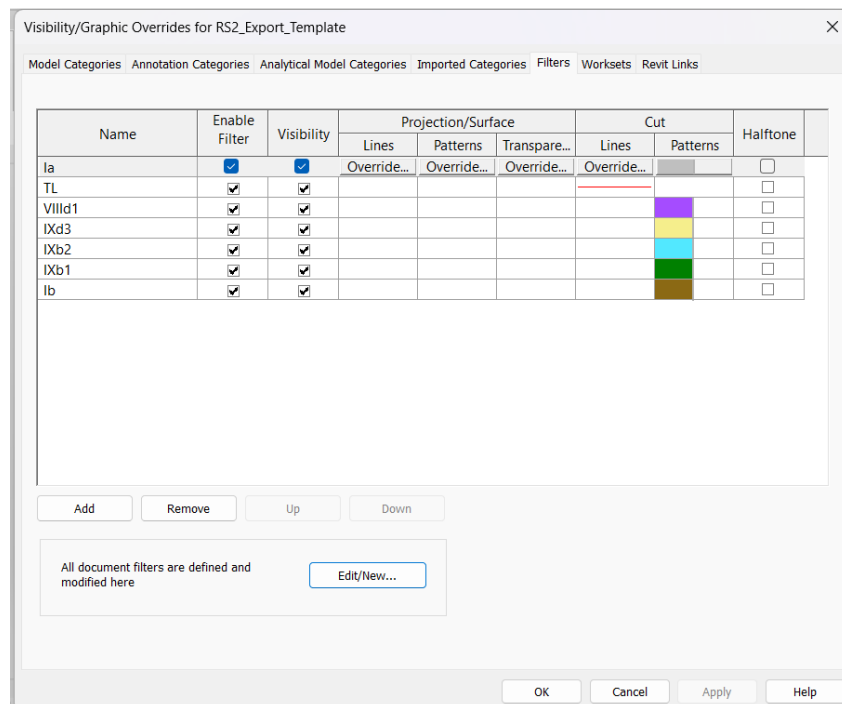


Figure 67. Application of the original geological colors to the soil layers in the View Template

To successfully transfer the generated 2D sections out of Revit, relying on the software's default export settings is insufficient. Default Revit export profiles are typically optimized for architectural drafting rather than geotechnical finite element analysis. Therefore, a dedicated, custom DXF export profile named RS2_Export was created to carefully control how the geometric and visual data translates into the CAD environment.

This custom setup was accessed through Revit's standard export options (File > Export > Options > Export Setups DWG/DXF) and configured across several critical parameters to ensure the data would be properly formatted for the subsequent Civil 3D cleanup phase as seen in Figure 68.

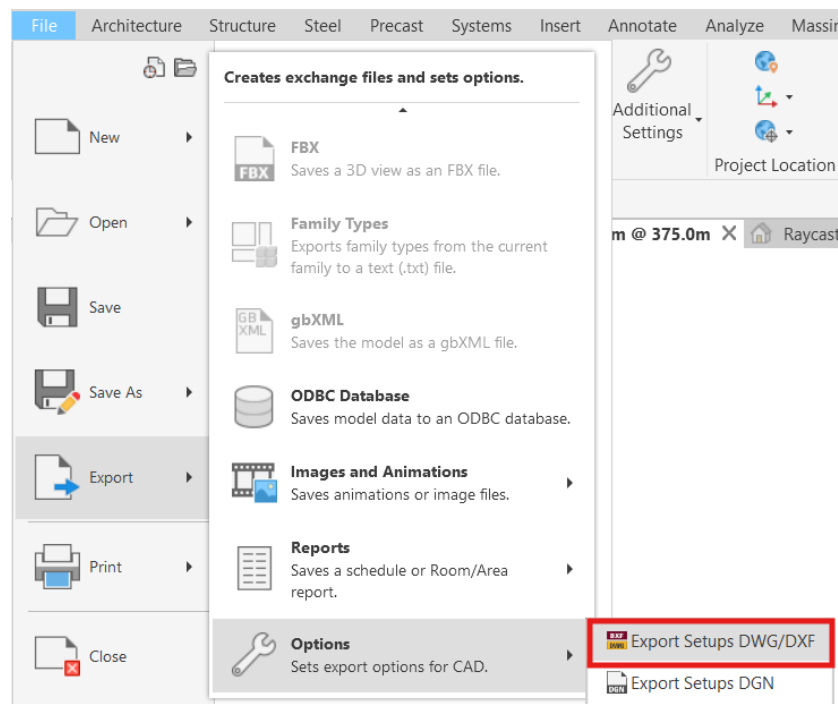


Figure 68. Export Setups DWG/DXF

The most vital modification was made within the Layers tab. As discussed in the previous section, the soil stratigraphy was isolated and color-coded using visual overrides in the View Template. If exported normally, Revit would group all these distinct soils into a single, unorganized "Generic Models" layer in the DXF file. To prevent this, the export layer options were explicitly set to *"Export all properties BYLAYER, and create new layers for overrides"* shown in Figure 69. This specific setting forces Revit to read the color overrides and generate a distinct, separate CAD layer for every single soil type, preserving the hard work done in the view template.

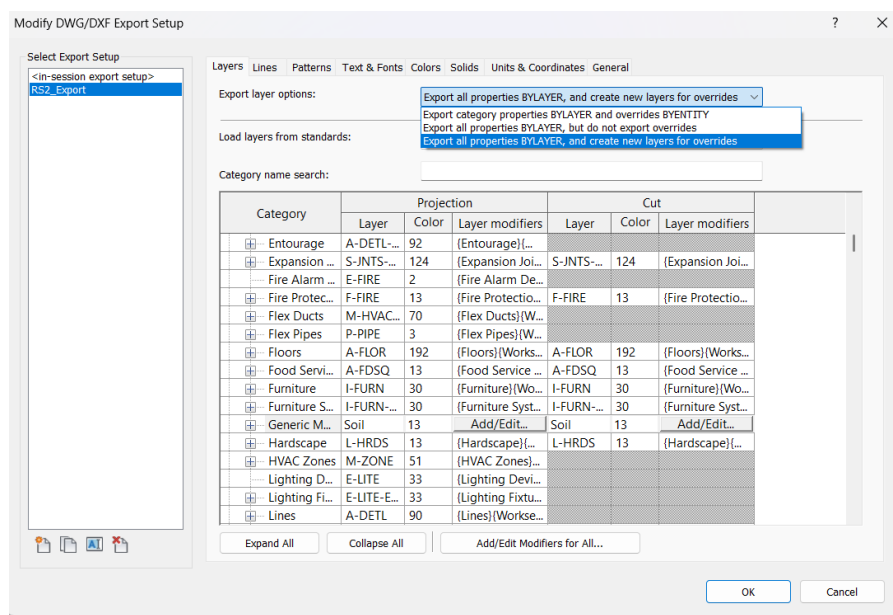


Figure 69. DXF export layers configuration

4.3.6 Automated cleaning in Civil 3D

Following the export process, the generated DXF files were opened in AutoCAD Civil 3D to verify the accuracy of the data transfer. As illustrated in Figure 70, the custom Revit export profile functioned exactly as intended. The visual inspection confirms that the overall geometry of the tunnel and the surrounding stratigraphy was imported correctly without any spatial distortion. Furthermore, the Layer Properties Manager within Civil 3D demonstrates that the view template filters were successfully translated; each distinct soil layer was automatically

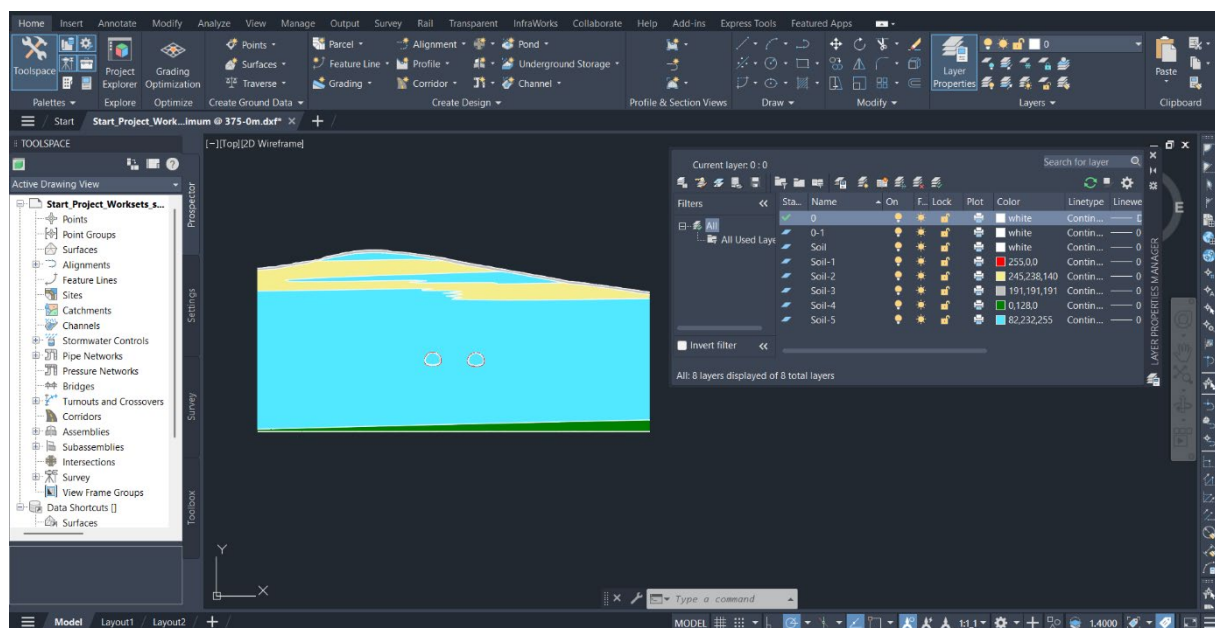


Figure 70. The exported DXF section successfully opened in Civil 3D, displaying the preserved layer organization and color coding

assigned to its own specific CAD layer (e.g., Soil-1, Soil-2), perfectly retaining the original RGB color coding.

While the visual representation of the exported DXF files appears correct in AutoCAD Civil 3D and the layers are neatly organized, the underlying geometry is simply not ready for finite element analysis (FEA). Numerical simulation software like Rocscience RS2 operates on strict geometric principles to build its computational models. To successfully generate a mesh and compute geotechnical stresses, RS2 requires all soil stratigraphy, material interfaces, and tunnel excavation boundaries to be defined by clean, continuous, and non-overlapping polylines. The software relies on these closed geometric loops to accurately identify distinct regions, assign specific material properties (like soil density or concrete strength), and simulate excavation stages. If the imported boundaries are open, fragmented, or self-intersecting, the software's meshing algorithm cannot accurately determine the closed domains, leading to topological failures.

Because the 3D tunnel solids in Revit were exported as polymeshes to preserve their true shape, the resulting raw DXF consists of thousands of fragmented, disconnected line segments, microscopic gaps, overlapping boundaries, and unjoined blocks. As recent studies on BIM-to-FEM interoperability have demonstrated, directly transferring native BIM geometry into structural or geotechnical analysis tools frequently results in severe topological inconsistencies and information loss (Colajanni, et al., Architectural and Structural Interoperability in the BIM DesignWorkflow, 2025). In fact, if an engineer attempted to import this raw, unedited DXF directly into RS2, the software would reject the data entirely. A highly fragmented file will often fail to open or import successfully, and it will certainly fail to reach the meshing stage due to boundary coordinate errors.

To resolve this critical interoperability gap, the raw CAD geometry must undergo a rigorous cleaning process. The fragmented elements must be exploded, flattened to a strict 2D plane (elevation $Z = 0$), trimmed of all overlapping segments, and precisely joined into continuous, closed polylines that RS2 can effortlessly interpret.

However, performing these repetitive and highly precise CAD commands manually for dozens of individual cross-sections along a tunnel alignment is incredibly inefficient and highly prone to human error. To achieve a reliable and seamless data transfer, a new visual programming script was developed using Dynamo inside Civil 3D. This dedicated script systematically targets the raw linework, executes all necessary geometric modifications, and fully automates the cleaning workflow. By bridging the gap between BIM geometry and FEA requirements, this automated approach ensures that every cross-section is flawlessly optimized and instantly ready for numerical simulation.

While Dynamo offers a variety of standard, out-of-the-box nodes for geometric manipulation, this specific automation was designed almost entirely using custom Python scripts. The decision to rely on Python rather than standard visual programming nodes was driven by the need for deep, system-level control over the CAD environment. Standard nodes can sometimes struggle or become overly complex when trying to filter, explode, and reconstruct thousands of raw, unjoined CAD lines. By utilizing Python, the script directly interacts with the AutoCAD and

Civil 3D Application Programming Interface (API). This programmatic approach allows us to dive directly into the internal layer database, precisely filter lines based on their assigned soil layers, and execute complex geometric commands—such as forcing lines to join into closed polylines and deleting overlapping segments—with much higher speed and reliability.

Similarly to Revit, Dynamo in Civil3D is opened from the Manage Tab under the Visual Programming section. To execute the custom automated cleaning operations, standard visual programming nodes were bypassed in favor of custom code. As illustrated in Figure 71, this process begins by searching for and placing a "Python Script" node directly onto the Dynamo canvas. By double-clicking this node, an embedded text editor opens, providing a blank interface where raw Python code can be written, edited, and executed directly within the visual programming workflow.

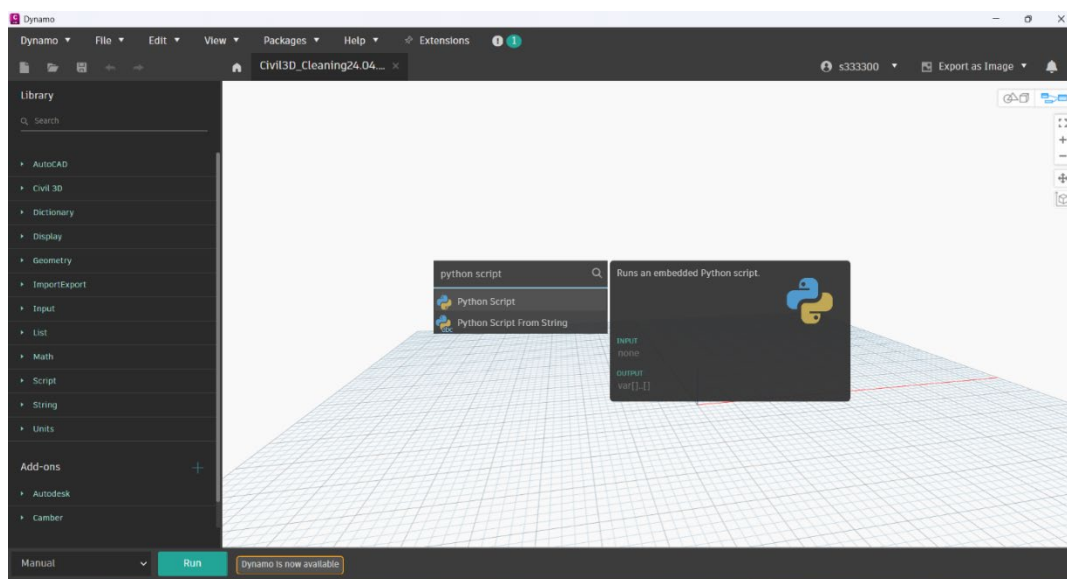


Figure 71. Opening a python script node in Dynamo

To bring everything together, the entire geometric cleaning process was consolidated into one clean visual script. As shown in Figure 72, instead of using a messy, tangled web of default Dynamo nodes which can easily break or become hard to read, this workflow relies on a straightforward sequence of Python scripts. This ensures the geometry is processed systematically and without errors every single time. The following subchapters will look under the hood of each of these four nodes, breaking down the specific math, AutoLISP routines, and geometric rules that drive them.

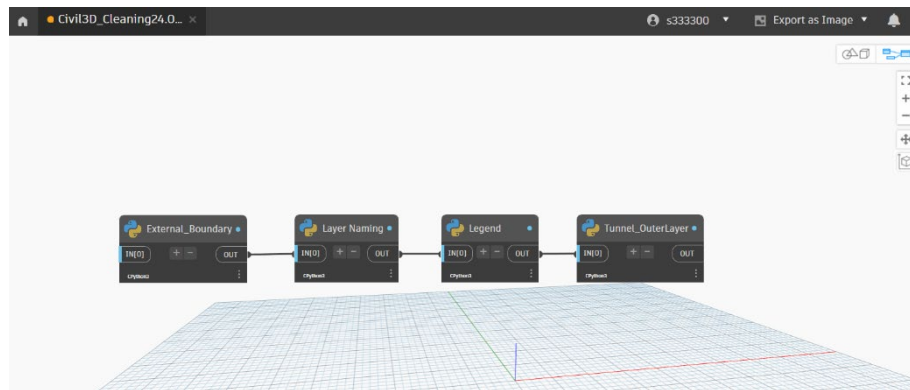


Figure 72. The complete Dynamo automation workflow in Civil3D

4.3.7 External boundary generation for RS2 import

Before the internal soil layers and tunnel geometries can be organized, the foundational limits of the numerical model must be established. In finite element analysis (FEA), software like RS2 requires a definitive "External Boundary." This boundary acts as the absolute mathematical limit of the physical domain, defining the outermost perimeter of the soil mass. It is crucial because this boundary is where the global boundary conditions such as zero-displacement restraints and far-field stresses are applied during the simulation.

The raw DXF exported from Revit contains the internal soil stratigraphy but lacks a single, unified, and closed polyline enclosing the entire geological model. Furthermore, the geometric data often contains overlapping lines where the tunnel cuts through the soil. To resolve these issues automatically, the first Python node in the Dynamo workflow operates as a programmatic wrapper, injecting a comprehensive AutoLISP routine directly into the Civil 3D command engine.

Instead of including the entire lengthy codebase, the primary automated operations executed by this script can be broken down into three critical phases:

1. Shrinkwrapping the model perimeter

To create the unified external boundary, the script first forces Civil 3D to "explode" the imported block references, exposing the raw hatch patterns that represent the soil. It then utilizes the `_LINEWORKSHRINKWRAP` command seen in Figure 73. This highly effective CAD operation detects the outermost limits of all selected elements and draws a perfectly closed, continuous polyline around them. The script isolates this new perimeter and assigns it to a newly created layer named "External."

```

25 ;; =====
26 ;; 2. EXTERNAL BOUNDARY (Shrinkwrap all hatches)
27 ;; =====
28 (if (not (tblsearch "LAYER" "External"))
29   (entmake (list '(0 . "LAYER") '(100 . "AcDbSymbolTableRecord") '(100 . "AcDbLayerTableRecord") '(2 . "External") '(70 . 0) '(62 . 6))))
30 )
31 (setvar "CLAYER" "External")
32
33 (if (setq ssHatch (ssget "_X" '((0 . "HATCH"))))
34   (progn
35     (command "._LINEWORKSHRINKWRAP" ssHatch "")
36     (while (> (getvar "CMDACTIVE") 0) (command ""))
37   )
38 )

```

Figure 73. LINEWORKSHRINKWRAP command in Python for Dynamo in Civil3D

2. Internal stratigraphy generation

(Visual vs. Analytical Data) Beyond establishing the external perimeter, the script is fundamentally responsible for reconstructing the internal soil stratigraphy. This step addresses a critical interoperability gap: the distinction between visual and analytical CAD data. When 2D sections are initially exported from Revit, the geological layers are often packaged as rigid CAD blocks containing only visual "hatches" (coloured fill patterns). RS2, however, cannot understand hatches; it strictly requires analytical geometric boundary lines (polylines) to generate a mesh.

To resolve this, the script executes a forced "double-explode" command (`._EXPLODE`), which shatters the imported block references and exposes the raw hatches. Once exposed, the script systematically loops through every hatch in the drawing using the `._HATCHGENERATEBOUNDARY` command visualized in Figure 74. This algorithm automatically traces the perimeter of each coloured fill, converting the visual pattern into a continuous joined polyline. Crucially, the script reads the source hatch's layer property and ensures the newly drawn boundary is placed on the exact same layer as presented in Figure 75.

```

64 ;; =====
65 ;; 3. GENERATE ALL INTERNAL BOUNDARIES
66 ;; =====
67 (if (setq ssHatch (ssget "_X" '((0 . "HATCH"))))
68   (progn
69     (setq i 0)
70     (while (< i (sslength ssHatch))
71       (setq ent (ssname ssHatch i))
72       (setq layer (cdr (assoc 8 (entget ent))))
73
74       (if (tblsearch "LAYER" layer) (setvar "CLAYER" layer))
75       (command "._HATCHGENERATEBOUNDARY" ent "")
76
77       (setq i (1+ i))
78     )
79   )
80   ;; Push colored hatches to the back
81   (command "._DRAWORDER" ssHatch "" "_B")
82 )

```

Figure 74. EXPLODE and HATCHGENERATEBOUNDARY Command in Python

3. Resolving topological overlaps (The "Auto-deleter")

One of the most severe interoperability issues occurs at the interface between the tunnel excavation and the surrounding soil. When Revit exports the 2D section, the soil layer

inherently contains a circular "hole" where the tunnel exists, while the tunnel element itself possesses its own identical outer boundary ring. If imported into RS2, these two perfectly overlapping circles cause fatal meshing errors because the software cannot distinguish between the soil boundary and the tunnel boundary.

To prevent this, the script executes a sophisticated spatial sorting algorithm. It calculates the exact bounding box and center point of the tunnel rings. Once identified, it retains the true "Tunnel" polyline but systematically deletes the redundant, overlapping soil hole sitting precisely beneath it.

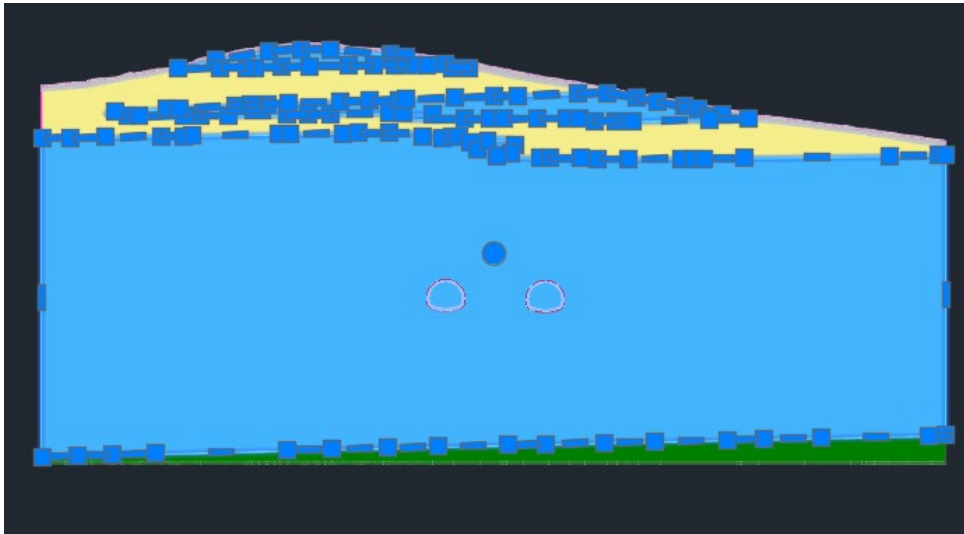


Figure 75. Analytical CAD polylines (highlighted) automatically generated from the visual soil hatch patterns

4. Polyline joining and final optimization

After the boundaries are generated and overlaps are deleted, the raw line segments must be merged. The script targets all relevant soil and tunnel layers and applies a global Polyline Edit (`_PEDIT`) command with a micro-tolerance of 0.001 meters, forcing all broken segments to fuse into closed loops. Finally, an `_OVERKILL` command is executed to wipe out any microscopic overlapping lines or microscopic fragments left behind by the Revit export. The script is seen in Figure 76.

Through this single Python node, the geometry is transformed from a fragmented architectural drawing into a topologically sound, mathematically closed framework (as illustrated in Figure 77). This unified perimeter fully prepares the model for the next stage of layer organization.

```

91  ;; =====
92  ;; 4. JOIN EVERYTHING FOR RS2
93  ;; =====
94  (setq targetLayers nil)
95  (setq layItem (tblnext "LAYER" T))
96  (while layItem
97    (setq layName (cdr (assoc 2 layItem)))
98    (if (wcmatch (strcase layName) "SOIL-*,TUNNEL*,EXTERNAL")
99      (setq targetLayers (cons layName targetLayers))
100    )
101    (setq layItem (tblnext "LAYER"))
102  )
103
104  (foreach layName targetLayers
105    (if (setq ss (ssget "_X" (list (cons 8 layName) '(0 . "LINE,LWPOLYLINE,ARC"))))
106      (progn
107        (command "._PEDIT" "_M" ss "" "_J" "0.001")
108        (while (> (getvar "CMDACTIVE") 0) (command ""))
109      )
110    )
111  )
112
113  ;; =====
114  ;; 5. FINAL OVERKILL
115  ;; =====
116  (if (setq ss (ssget "_X" '(0 . "LINE,LWPOLYLINE,ARC,CIRCLE"))))
117    (command "._OVERKILL" ss "" "Tolerance" "0.001" "Done")
118  )
119  (command)
120  (setvar "OSMODE" oldOsnap)
121  (setvar "CMDECHO" 1)
122  (princ)
123  )
124  ""

```

Figure 76. OVERKILL and PEDIT commands in Python for the external boundary

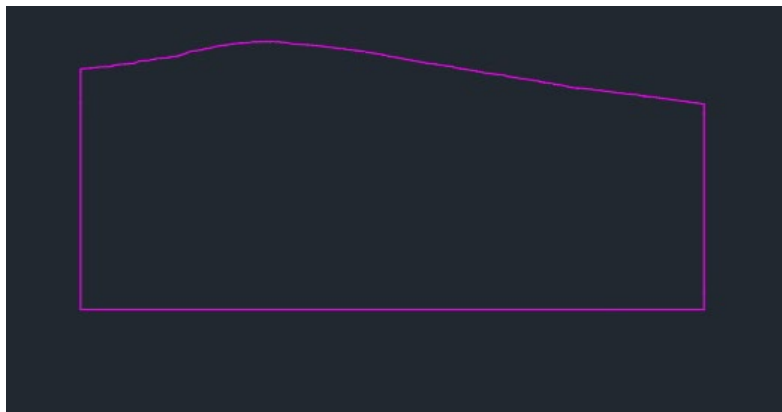


Figure 77. Isolated view of the generated external soil boundary polyline, with internal soil layers hidden for clarity

Through this Python node, the geometry is fully transformed from a fragmented architectural drawing into a topologically sound, mathematically closed framework. This unified geometry fully prepares the model for the next stage of layer organization.

4.3.8 Geological layer classification and naming standardization

With the external and internal boundaries fully optimized and converted into continuous polylines, the geometric topology of the numerical model is complete. However, some issues remain before the file can be seamlessly imported into RS2.

During the initial export from the BIM environment, the isolated soil layers were assigned generic, sequential CAD layer names (e.g., "Soil-1", "Soil-2", "Soil-3"). While this generic naming convention successfully kept the geometries separated during the export phase, it does not hold any geotechnical meaning. When setting up a finite element analysis, the software requires distinct material models to be assigned to each region. To streamline this assignment process and avoid manual renaming errors, the CAD layers must perfectly match the standardized geological classifications established in the project's geotechnical profile (e.g., "Ia", "IXd3", "IXb1").

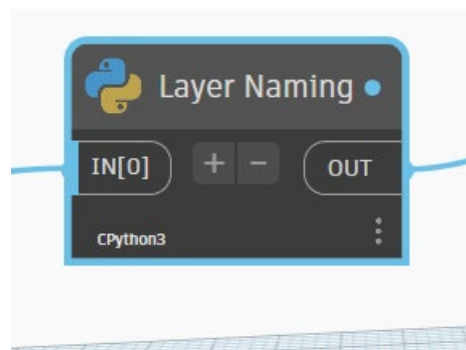


Figure 78. The second stage of the Dynamo workflow, utilizing the 'Layer Naming' Python node

To automate this data translation, the second Python node in the linear Dynamo sequence—titled "Layer Naming" illustrated in Figure 78 employs a highly effective reverse-engineering strategy. During the initial Revit View Template configuration (discussed previously in the previous sub-sections), specific geological colours were assigned to the generic soil models. While the DXF export stripped away the parametric BIM names, it successfully preserved these exact RGB (Red, Green, Blue) colour values. Therefore, the Python script utilizes the layer's RGB color as a unique "fingerprint" to accurately identify the soil types.

At the core of the script is a predefined dictionary array (`colour_map`). This dictionary explicitly links the known RGB tuples to their corresponding geotechnical target names as evident from . For example, the RGB value (245, 238, 140) is mapped directly to the stratigraphy layer "IXd3", and (255, 0, 0) is mapped to the "Tunnel" excavation boundary.

```

13 # Dictionary mapping RGB tuples to target layer names
14 color_map = {
15     (191, 191, 191): "Ia",
16     (139, 105, 20): "Ib",
17     (0, 128, 0): "IXb1",
18     (82, 232, 255): "IXb2",
19     (245, 238, 140): "IXd3",
20     (165, 76, 255): "VIIId1",
21     (255, 0, 0): "Tunnel"
22 }
23
24 adoc = acapp.DocumentManager.MdiActiveDocument
25 db = adoc.Database
26
27 renamed_layers = []

```

Figure 79. The Python dictionary array utilized to map extracted RGB color values to standardized geological layer names

By importing the native AutoCAD Database Services API (AcDbMgd), the script initiates a secure transaction within the Civil 3D layer table. It systematically iterates through every existing layer in the active file. During this loop, the script extracts the True Colour RGB properties of each layer and cross-references them against the dictionary. When an exact match is detected, the script overwrites the generic BIM name with the standardized geological name.

To prevent database corruption, the script includes built-in logic to bypass default CAD layers (such as "0") and actively checks the drawing database for existing naming conflicts before applying any modifications. Upon completion, the transaction is committed, and the node outputs a verification log directly to the Dynamo interface, confirming the successful translation of all layers (e.g., "Renamed 'Soil-1' to 'Tunnel'"). This ensures that when the file is ultimately imported into RS2, the material assignments can be mapped instantly without any manual CAD layer management as successfully demonstrated in Figure 80.

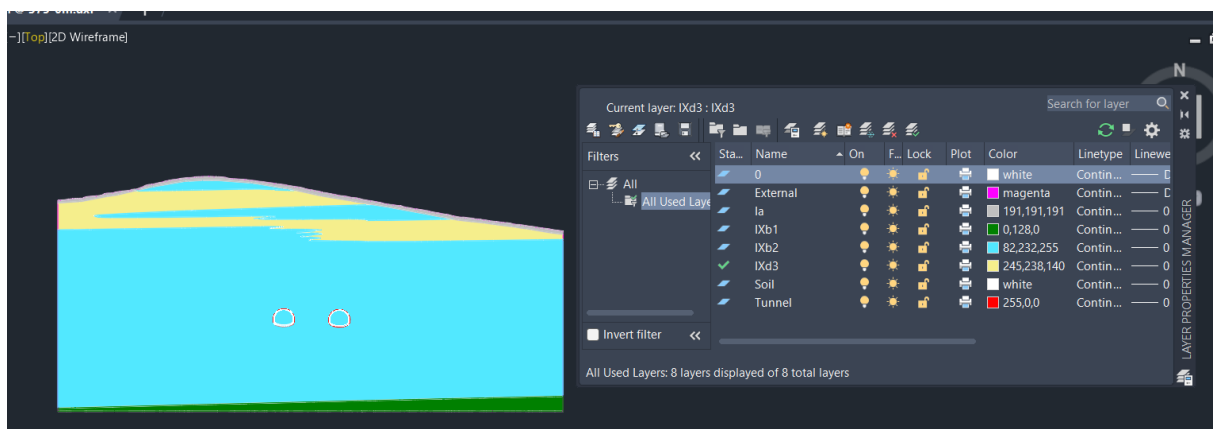


Figure 80. The updated Civil 3D Layer Properties Manager demonstrating the successfully renamed geological layers

4.3.9 Dynamic legend generation and visualization

While cleaning the geometry and fixing the layer names handles the software side of things (Sections 4.3.7 and 4.3.8), we also need to make sure the CAD files are readable for the people not knowing them. When dealing with a complex geological model that stretches across several kilometres, looking at dozens of individual cross-sections can get confusing very quickly if there isn't a clear visual reference.

To fix this, the third Python node in the Dynamo workflow, called "Legend," was designed to automatically draw a color-coded legend right inside the Civil 3D model space.

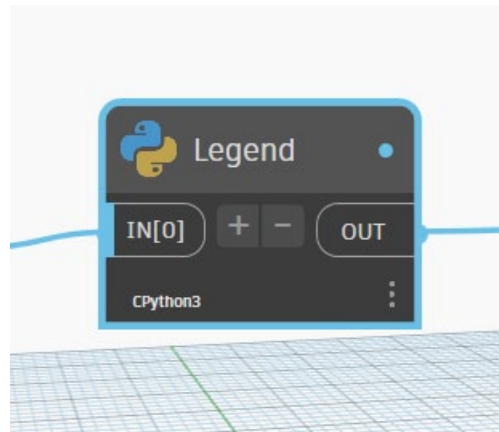


Figure 81. The third stage of the Dynamo workflow, using the 'Legend' Python node to dynamically draft a visual reference guide.

The problem with just inserting a standard, pre-drawn CAD block for a legend is that not every soil layer exists in every single cross-section. A static block might show soil types that aren't present at that specific tunnel location. To get around this, the script starts by scanning the active drawing. It checks the RGB colour of every layer currently being used and matches them against the master “color_map” dictionary we set up in the previous step.

By filtering these results, it builds a specific list of “active_legend_items”. This means if a soil layer like “Ib” doesn't appear at that exact chainage along the tunnel, the script simply leaves it out. The legend only shows what is there.

Next, the script must figure out exactly where to place the legend and how big to make it. Instead of dropping it at a random coordinate, it uses the native “Extmin” and “Extmax” database properties to calculate the total bounding box (the width and height) of the 2D cross-section.

Using the bounding box data in Figure 82, the script automatically places the title and color swatches just to the right of the model, leaving a neat 5% gap so it doesn't overlap with the drawing. It also scales the text height (txt_ht) to perfectly match 2.5% of the model's total vertical height. This ensures the legend always looks proportional, whether it is a shallow or deep section of the tunnel.

```

53 # Calculate Automatic Placement and Size based on Drawing Extents
54 db.UpdateExt(True)
55 ext_min = db.Extmin
56 ext_max = db.Extmax
57
58 width = ext_max.X - ext_min.X
59 height = ext_max.Y - ext_min.Y
60
61 # Fallback if extents are invalid
62 if width <= 0 or height <= 0 or width > 1e10:
63     ins_pt = Point3d(0, 0, 0)
64     txt_ht = 5.0
65 else:
66     # Place legend to the right of the model's top-right corner with a 5% gap
67     gap = width * 0.05
68     ins_pt = Point3d(ext_max.X + gap, ext_max.Y, 0)
69
70     # Scale text height proportionally to the model's total height (2.5%)
71     txt_ht = height * 0.025
72

```

Figure 82. Bounding box python script for the legend creation

Finally, the script loops through the active items, drawing thick CAD polylines to act as the color swatches and placing the standardized soil names right next to them. This completely removes the need for manual drafting, ensuring that every exported section gets a fast, accurate, and customized visual reference.

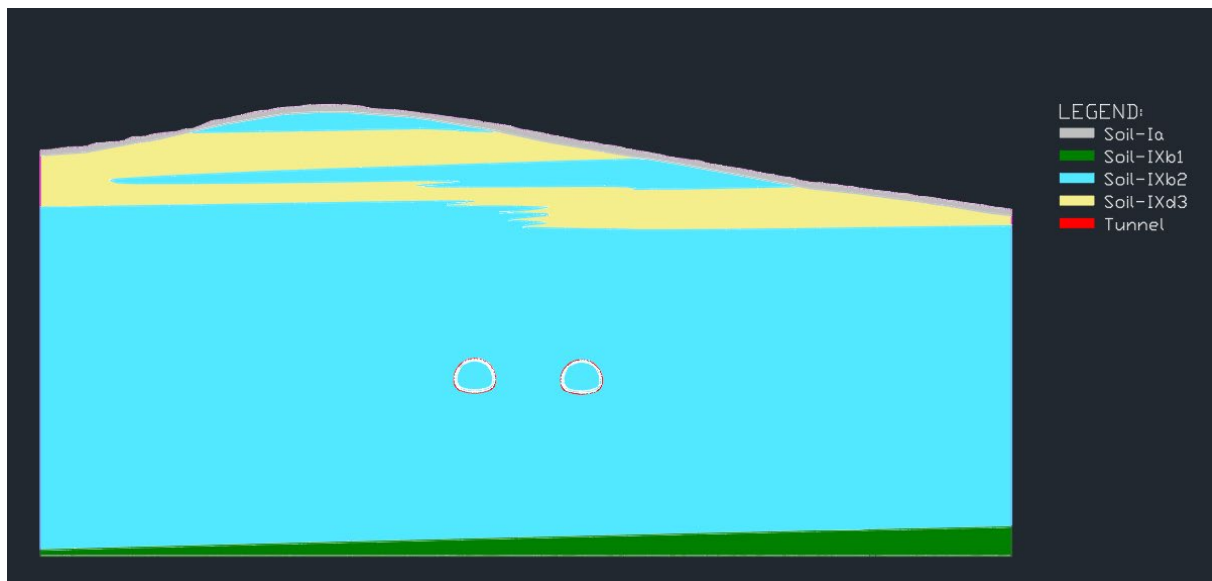


Figure 83. Final output of the 'Legend' script, showing the active soil types

Once the script finishes running, it instantly draws the complete, formatted legend just to the right of the model. As you can see in Figure 83, it lines up perfectly with the cross-section and only lists the soil layers that are present in this specific slice, complete with their exact RGB colour swatches.

4.3.10 Excavation boundary tunnel isolation

The final automated step in the Dynamo workflow addresses the geometric complexity of the tunnel structure itself. In geotechnical numerical modelling, the primary concern is the interaction between the surrounding soil mass and the initial excavation boundary. However, the BIM model exported from Revit typically contains detailed, multi-layered architectural and structural geometry, including inner concrete linings, shotcrete layers, and internal partitions.

When exported to DXF, these layers manifest as multiple concentric geometric rings. If imported directly into RS2, these internal rings complicate the meshing process and are entirely unnecessary for calculating the initial ground relaxation and stress distribution around the primary excavation. Therefore, the numerical model strictly requires only the single, outermost polyline representing the true excavation boundary.

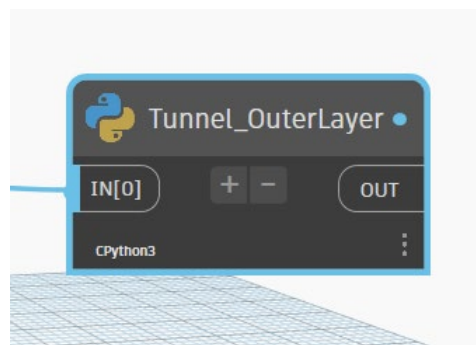


Figure 84. The final stage of the Dynamo workflow, utilizing the 'Tunnel_OuterLayer' Python node

To achieve this, the fourth Python node in the sequence was built with a targeted AutoLISP algorithm designed to mathematically extract only the outermost rings of the twin tunnels. The script starts by scanning the drawing database to locate the CAD layers holding the tunnel geometry, looking for the “Tunnel” layer name, we formalised in the previous steps or the specific red RGB colour code assigned to the infrastructure.

```

32      ;; 2. Select the tunnels and EXPLODE TWICE
33      (if (setq ssTunnels (ssget "_X" (list (cons 8 tunnelLayers))))
34          (progn
35              ;; QAFLAGS=1 allows exploding multiple objects at once
36              (setvar "QAFLAGS" 1)
37              (command "._EXPLODE" ssTunnels "")
38
39              ;; Grab whatever was just created and explode it one more time
40              (if (setq ssPrev (ssget "_P"))
41                  (command "._EXPLODE" ssPrev "")
42              )
43              (setvar "QAFLAGS" 0)
44          )
45      )
46
47      ;; 3. JOIN the raw lines and arcs back into closed rings
48      (if (setq ssFragments (ssget "_X" (list '(0 . "LINE,LWPOLYLINE,ARC") (cons 8 tunnelLayers))))
49          (progn
50              (command "._PEDIT" "_M" ssFragments "" "_J" "0.5")
51              (while (> (getvar "CMDACTIVE") 0) (command ""))
52          )
53      )

```

Figure 85. Double EXPLODE and JOIN Commands in python for "Tunnel Outer Layer"

Because BIM geometry is often highly grouped and block-referenced, the script forces a double-explode command on all elements residing on these layers. This breaks the architectural blocks down into their fundamental arcs and line segments. It then immediately follows up with a polyline edit command, forcing these raw CAD fragments to join back together into closed, continuous concentric rings.

The core challenge is then figuring out exactly which ring in the group represents the true outer excavation boundary. To solve this, the script in Figure 86, relies on a simple geometric rule: the outermost boundary will always have the longest perimeter. The script iterates through the isolated group and utilizes a custom function to precisely calculate the total length of each concentric ring. It identifies the polyline with the maximum length and protects it, and then systematically deletes all the shorter inner rings representing the structural linings.

```

87      ;; The function that keeps the longest ring and deletes the rest
88      (defun processGroup (ents / maxLen bestEnt l obj)
89        (setq maxLen -1.0 bestEnt nil)
90        (foreach e ents
91          (setq obj (vlax-ename->vla-object e))
92          (setq l (vlax-curve-getDistAtParam obj (vlax-curve-getEndParam obj)))
93          (if (> l maxLen) (setq maxLen l bestEnt e))
94        )
95        (if bestEnt
96          (foreach e ents
97            (if (not (equal e bestEnt)) (entdel e))
98          )
99        )
100      )
101
102      ;; Run the purge!
103      (if leftTunnels (processGroup leftTunnels))
104      (if rightTunnels (processGroup rightTunnels))
105    )

```

Figure 86. AutoLISP script utilizing perimeter length evaluation

The completed Dynamo automation sequence can be seen in Figure 87. The twin tunnels are now cleanly defined strictly by their outermost red perimeters, with all non-geotechnical elements successfully removed. With the external boundary closed, the soil layers standardized, the legend specific for each section created, and the tunnel excavation perfectly isolated, the entire Dynamo automation sequence is complete. The fragmented 2D DXF file exported from Revit has been fully transformed into a mathematically sound, optimized section, rendering it fully prepared for direct, error-free import into the Rocscience RS2 finite element software.

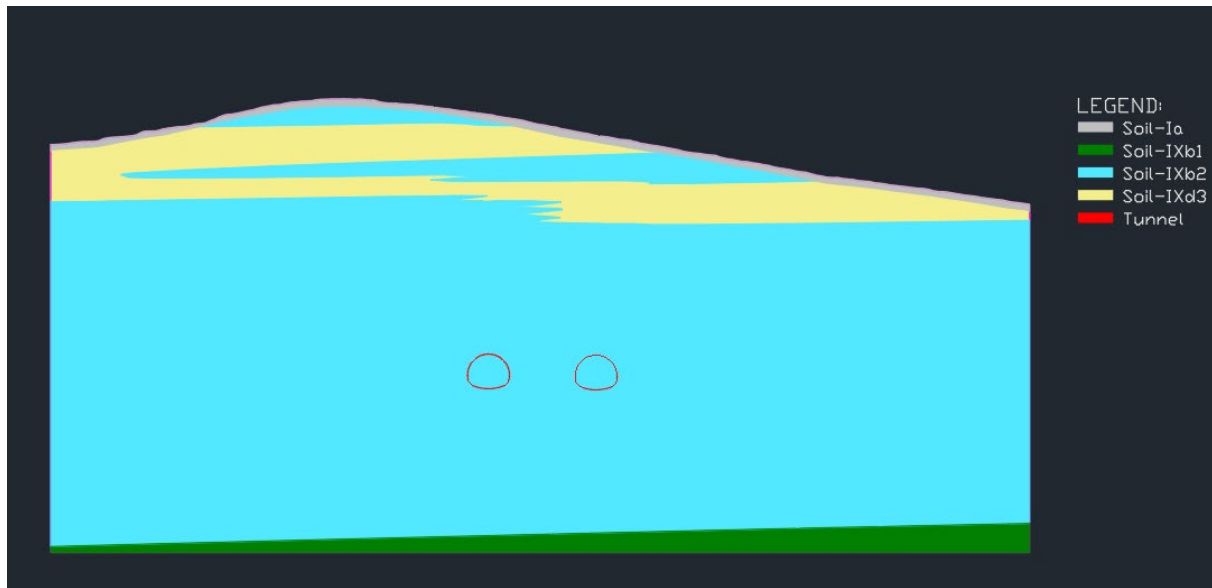


Figure 87. Final output of the Dynamo workflow, displaying the isolated outer tunnel boundaries, standardized geological layers, and dynamic legend ready for RS2 import.

4.3.11 Final export for RS2 interoperability (DXF 2013)

Now that the Dynamo script has done all the heavy lifting to clean and optimize the geometry, the very last step in Civil 3D is getting the file ready for our geotechnical software. To do this, the finalized cross-section is exported out as a standard DXF file.

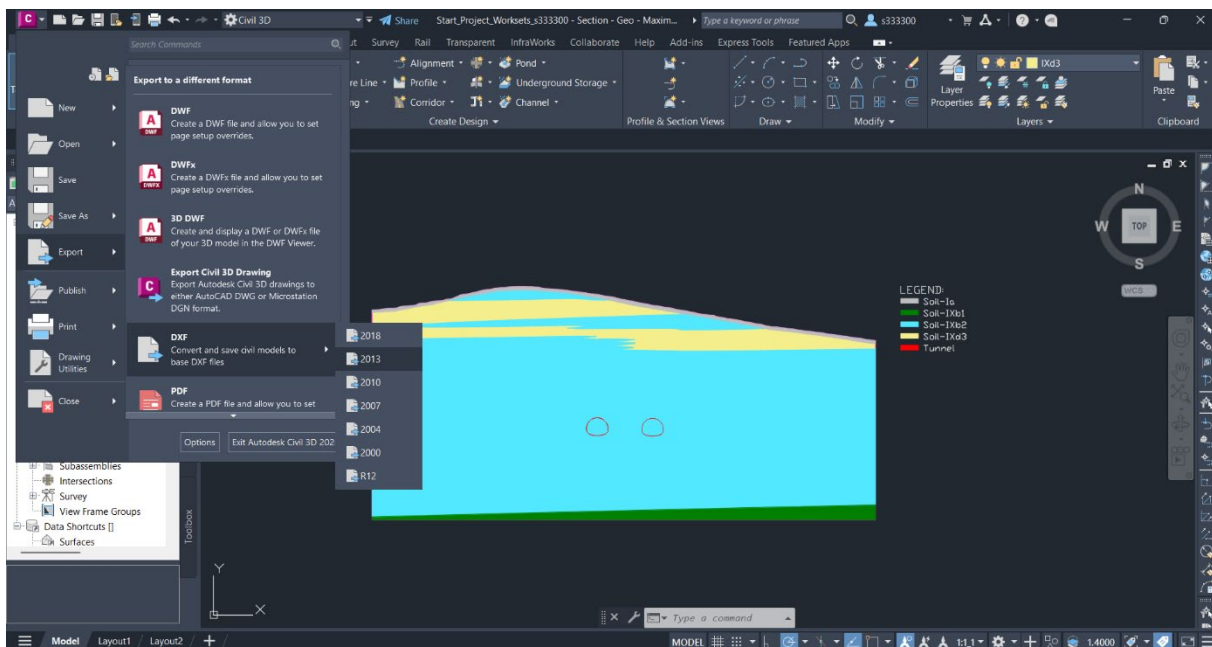


Figure 88. Exporting the optimized Civil 3D cross-section to 2013 DXF format

As you can see in Figure 88, even though Civil 3D offers newer export options, the file is specifically saved down to the older DXF 2013 format. This isn't random; it is a deliberate choice made for maximum reliability.

Following the final export settings illustrated previously in Figure 88, it is important to point out that the detailed methodology described above demonstrates the extraction and optimization process for just one single cross section. However, this workflow is completely scalable and remains the same for any other 2D sections being exported along the tunnel's alignment. In practice, the specific sections chosen for this process depend entirely on the requirements of the FEM analyst, who identifies the critical zones that need careful numerical simulation.

Furthermore, there is also the possibility of exporting all the required sections at once for transport into Civil 3D. While transferring the entire set of cross-sections simultaneously can speed up the workflow, it is a highly resource intensive task. Processing and flattening complex 3D geometries into multiple 2D planes at the same time requires a very strong computer. Specifically, a workstation equipped with substantial RAM and a powerful dedicated graphics card is necessary to handle the heavy computational load. Without sufficient hardware, the software is likely to freeze or crash, which can lead to corrupted files and missing geometric lines. Therefore, unless a high-performance machine is available, exporting the model section by section is generally the safest and most reliable approach to ensure no data is lost.

4.4 Phase 3: Analytical extraction & simulation

With the geometrical complexities resolved and the 2D cross sections successfully standardized and exported to the DXF 2013 format, the workflow now transitions from spatial coordination to rigorous geotechnical analysis. This third phase is a critical bridge in the framework, connecting the optimized digital model with the finite element simulations required to assess the structural and geological behaviour of the tunnel excavation.

The core software selected for this numerical analysis is Rocscience RS2, a comprehensive 2D finite element program highly suited for evaluating stress and deformation in underground engineering. Due to academic licensing restrictions, the RS2 software is accessed and operated remotely via the Virtual Lab portal provided by Politecnico di Torino. Utilizing this remote desktop environment ensures full access to the required computational tools while maintaining strict compliance with the institution's software distribution policies.

This phase details the methodology for bringing the geometries into the analytical environment. The following subsections will outline the step-by-step procedure for importing the finalized DXF files, defining the geotechnical boundaries, and establishing the parameters for the finite element simulation.

4.4.1 The finite element framework and Rocscience RS2

Rocscience RS2 is a powerful, two-dimensional finite element analysis (FEA) software heavily utilized in geotechnical and civil engineering to model soil and rock mechanics. Originally

known as Phase2, the program is primarily designed to evaluate the stress, deformation, and overall stability of underground excavations, surface slopes, and retaining structures.

At its core, RS2 operates by discretising complex geological profiles into a network of smaller, mathematically manageable geometric shapes, creating a finite element mesh. This allows the software to solve complex differential equations across the entire ground profile. By applying boundary conditions and material properties to this mesh, the software computes how the ground will react to various induced stresses. One of the fundamental techniques RS2 employs for stability analysis is the Shear Strength Reduction (SSR) method. During an SSR simulation, the software systematically and progressively reduces the shear strength parameters of the modelled soil or rock until the model reaches failure. This computational approach allows engineers to identify the exact failure mechanism and determine a highly accurate Factor of Safety (FoS) for the project.

In geotechnical engineering, accurately modelling the failure criteria of the surrounding ground is essential. RS2 seamlessly integrates advanced, non-linear material models into its calculations. For rock masses, the software allows the generalized Hoek-Brown failure criterion to be used directly to evaluate yielding and plastic deformation, ensuring a more realistic assessment of the rock's behaviour compared to oversimplified linear models (Kang, et al., 2022). This becomes particularly important when analysing highly heterogeneous subsurface conditions or complex stratigraphy.

To accurately simulate the geomechanical behaviour of the surrounding rock mass within RS2, the numerical model relies on the generalized Hoek-Brown failure criterion. This non-linear empirical model is highly regarded in underground engineering because it seamlessly bridges the gap between laboratory-tested intact rock properties and the highly variable, macro-scale conditions of a fractured rock mass. By quantitatively downgrading the intact rock strength based on the rock mass quality, it provides a realistic estimation of the overall ground behaviour.

The fundamental equation governing this failure criterion defines the relationship between the major and minor principal stresses σ_1 (Sigma 1) and σ_3 (Sigma 3) at the point of rock mass yielding. It is expressed mathematically as (Kang, et al., 2022):

$$\sigma_1 = \sigma_3 + \sigma_{ci} \left(m_b \frac{\sigma_3}{\sigma_{ci}} + s \right)^a$$

where:

- σ_{ci} represents the uniaxial compressive strength (UCS) of the intact rock material.
- m_b , s , and a are empirically derived material constants that account for the structural and surface conditions of the rock mass.

These rock mass constants are not arbitrary; they are determined by a constant m_i , the Geological Strength Index (GSI), and a disturbance factor (D). The factor D accounts for the degree of damage the rock mass has sustained, ranging from 0 for undisturbed in-situ conditions

to 1 for heavily disturbed ground due to poor excavation practices. The constants are derived using the following relationships:

$$m_b = m_i \exp\left(\frac{GSI - 100}{28 - 14D}\right)$$

$$s = \exp\left(\frac{GSI - 100}{9 - 3D}\right)$$

$$a = \frac{1}{2} + \frac{1}{6}(e^{-GSI/15} - e^{-20/3})$$

By integrating these specific formulas, RS2 dynamically adjusts the strength envelope of the computational mesh based on the assigned GSI, allowing for a highly realistic simulation of plastic deformation around the tunnel boundary.

Furthermore, analysing the deformation and convergence of the tunnel requires an accurate estimation of the rock mass stiffness. The finite element analysis utilizes the rock mass deformation modulus (E_m), which downgrades the intact rock deformation modulus (E_i) to account for the presence of natural joints and discontinuities. Following the established empirical methodology, E_m is calculated as:

$$E_m = E_i \left(0.02 + \frac{1 - D/2}{1 + e^{(60 + 15D - GSI)/11}} \right)$$

By embedding these constitutive equations directly into the RS2 solver, the simulation can rigorously assess both the ultimate structural capacity and the stress-strain distribution of the proposed tunnel support systems (Kang, et al., 2022).

When dealing with underground excavations, the ground rarely behaves uniformly as a single static block. RS2 excels in modelling staged construction, which is a vital feature for tunnelling projects. It simulates the gradual, step-by-step advancement of a tunnel face and models the subsequent redistribution of in-situ stresses within the surrounding rock mass. This staged approach is crucial because it helps engineers analyse dynamic soil-structure interactions over time. By observing how the rock mass responds at each stage of excavation, analysts can determine the optimal timing, placement, and capacity for support systems, such as rock bolts, steel ribs, or fibre-reinforced shotcrete. Integrating numerical simulations in RS2 with traditional empirical rock mass classifications provides a robust framework for optimising these tunnel support designs, ultimately ensuring structural stability without the need for overly conservative and costly material usage (Speakman, 2022).

4.4.2 Interoperability validation and automated model initialization

The physical transition of geometric data from the BIM environment to the numerical analyser occurs through the DXF import tab in Rocscience RS2. Once the software is launched and a new project is setup as illustrated in Figure 89, the user is presented with the primary modelling interface.

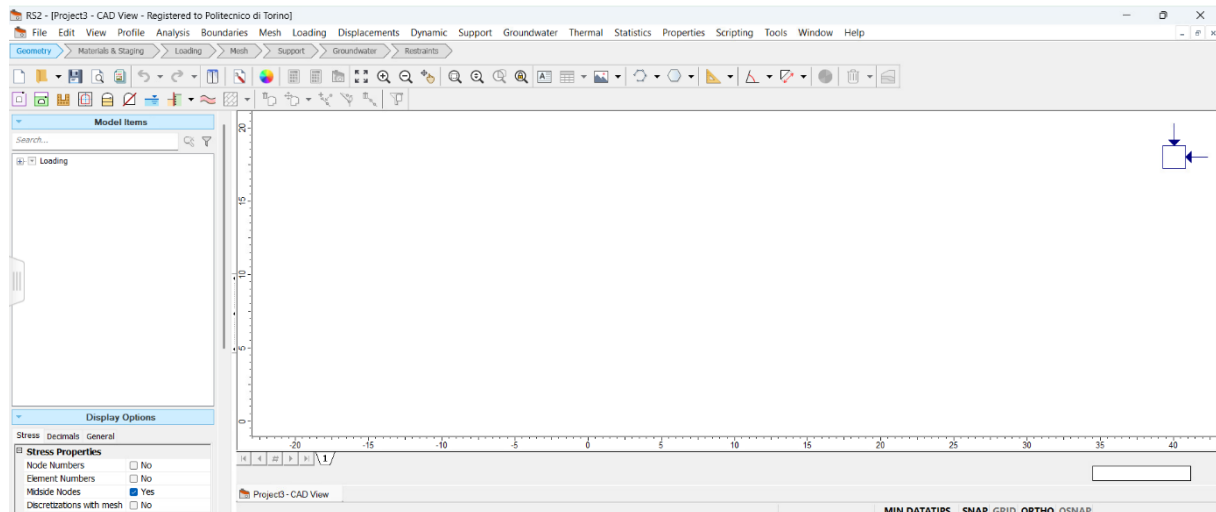


Figure 89. Rocscience RS2 user interface

Rocscience RS2 provides a versatile suite of import options designed to facilitate the integration of external data into the numerical modelling environment. The software supports the importation of proprietary file types, such as Slide for Rocscience itself and Examine2D, which allows for seamless interoperability within the broader Rocscience software ecosystem. Additionally, the interface includes options to import reference images and predefined material properties. However, when transferring complex spatial geometries from external architectural

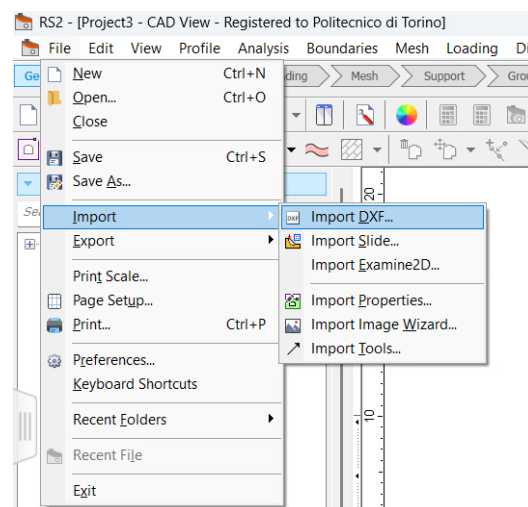


Figure 90. Rocscience RS2 import formats

or infrastructure models, the DXF (Drawing Exchange Format) serves as the primary and most robust import setup. Because DXF is a universally recognized CAD standard, it acts as the critical bridge connecting the optimized cross sections generated in Civil 3D directly to the finite element solver, ensuring that the structural framework is translated with absolute precision. A set of import options in RS2 are shown in Figure 90.

Bringing the 2D tunnel + Soil cross section into this environment is a highly sensitive process. During the import sequence, RS2 reads the incoming file and opens a critical mapping dialog. Because the geometric data was strictly organized into standardized layers during the Civil 3D phase, the analyst can easily map these CAD layers to specific RS2 entity types. For instance, the predefined layers representing the geological stratigraphy are assigned as "Material Boundaries," while the isolated tunnel perimeter is strictly categorized as an "Excavation Boundary." This layer-mapping process is the exact moment where dead geometric lines are translated into active, mathematically bounded geotechnical domains.

To fully validate the necessity of the intermediate processing steps outlined in Phase 2 (Civil3D and Dynamo), it is crucial to examine what would happen if this optimization had been skipped. If a raw 2D section were exported directly from the Revit model and imported straight into RS2, the numerical solver would be completely unable to process the data due to issues discussed in phase 2.

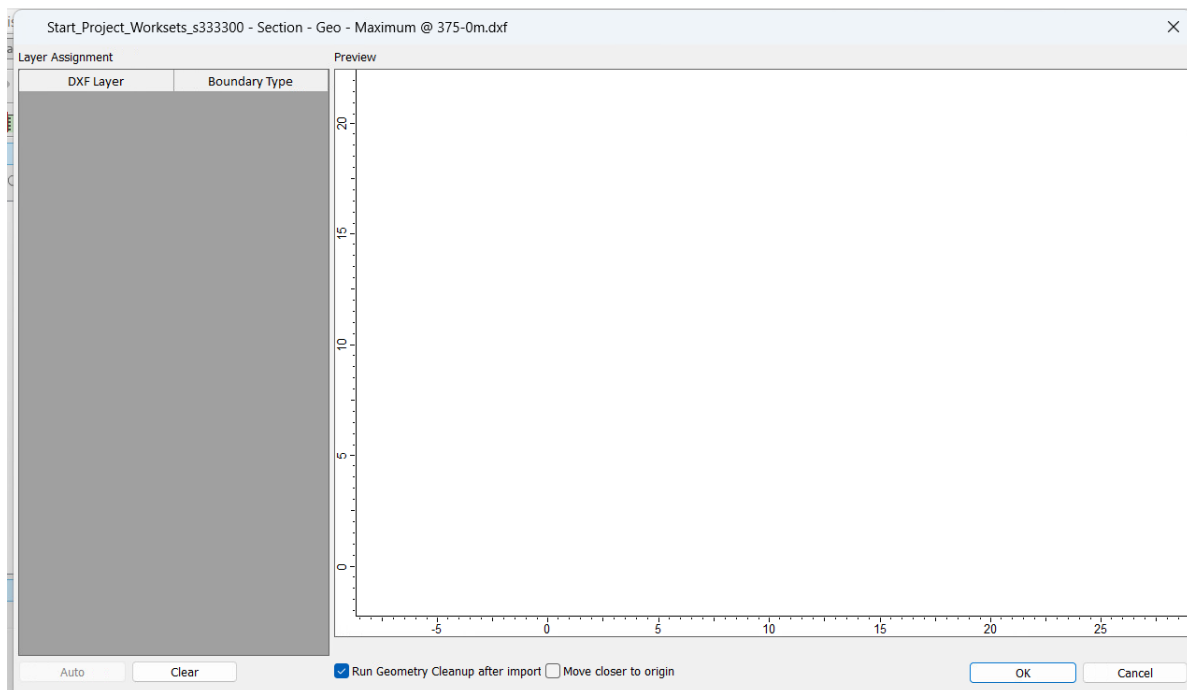


Figure 91. RS2 displaying raw 2D cross section import from Revit

As evident in Figure 91, importing raw BIM geometry results in complete breakdown. When RS2 attempts to discretize this uncleaned geometry, it is unable to process the blocks and can't transform the geometry. The software relies on perfectly closed polygons to define soil regions; if a boundary line has even a millimeter wide gap or overlap, the software cannot determine

where one material ends and another begins, completely halting the finite element analysis before it can even begin.

In contrast, the optimized DXF file generated through the Dynamo and Civil 3D framework loads flawlessly into the RS2 interface Figure 92. As visualized is the software's modelling pane, the complex stratigraphy and the tunnel cross-section are imported with perfect topological integrity.

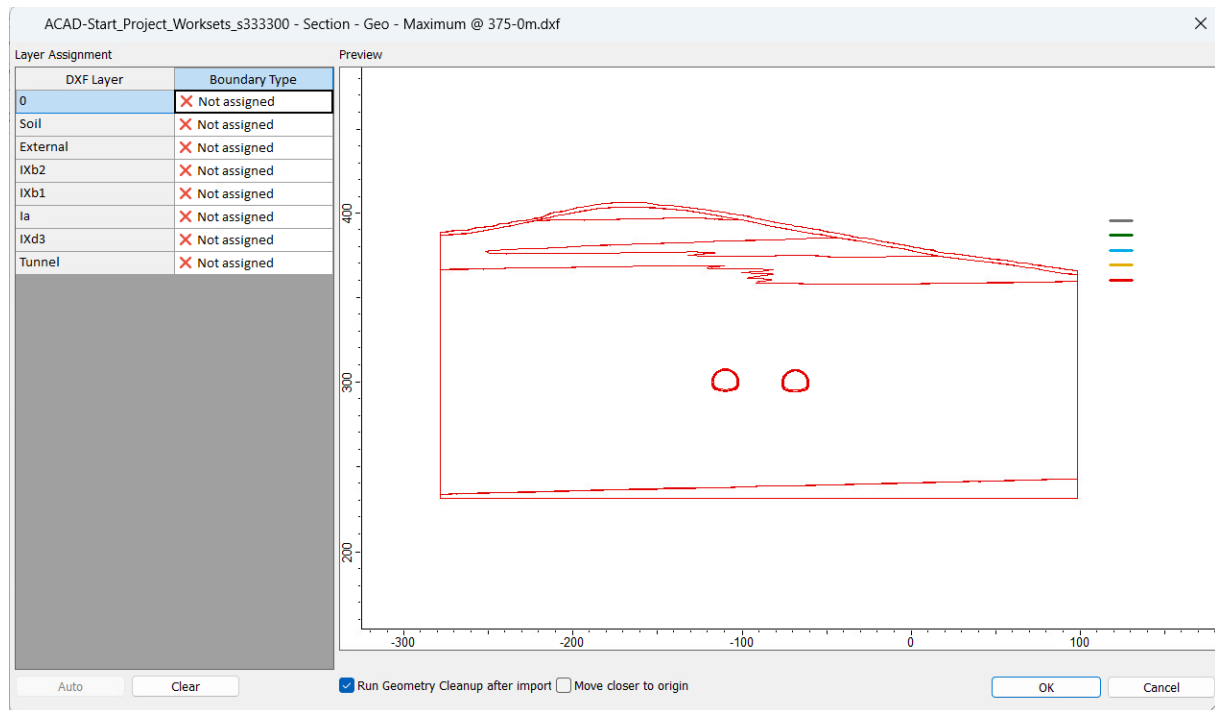


Figure 92. Flawless DXF geometry import inside RS2

There are no fragmented lines, disconnected vertices, or overlapping elements. Because the geometry is mathematically sound and strictly organized, the model is immediately ready for discretization. This successful import serves as the ultimate validation of the proposed interoperability framework, proving that the automated cleaning process prevents the fatal geometric errors that typically disrupt BIM-to-FEM workflows.

During the DXF importation process, the RS2 interface presents a dedicated layer assignment dialog that allows the user to map each individual CAD/DXF layer to a specific geotechnical boundary type. As illustrated in Figure 93, the software provides a comprehensive dropdown menu containing various boundary classifications essential for finite element modelling, such as 'External', 'Material', 'Excavation', and 'Stage'. Because the geometric data was strictly organized and color-coded into standardized layers during the prior Civil 3D phase, this mapping process is highly efficient and eliminates the need for manual geometry redrawing.

Following the project's structural requirements, the outermost perimeter layer is assigned as the 'External' boundary, establishing the absolute mathematical limits of the numerical model. The

distinct geological strata (e.g., IXb2, IXb1, Ia, and IXd3) are accurately classified as 'Material' boundaries, which will later allow the software to differentiate the mechanical properties of each soil zone. Simultaneously, the designated tunnel perimeter is explicitly defined as an 'Excavation' boundary. Furthermore, the interface allows extraneous or redundant CAD layers—such as the default '0' layer—to be marked as 'Not assigned', ensuring that only the optimized, relevant geometry is passed forward into the finite element mesh.

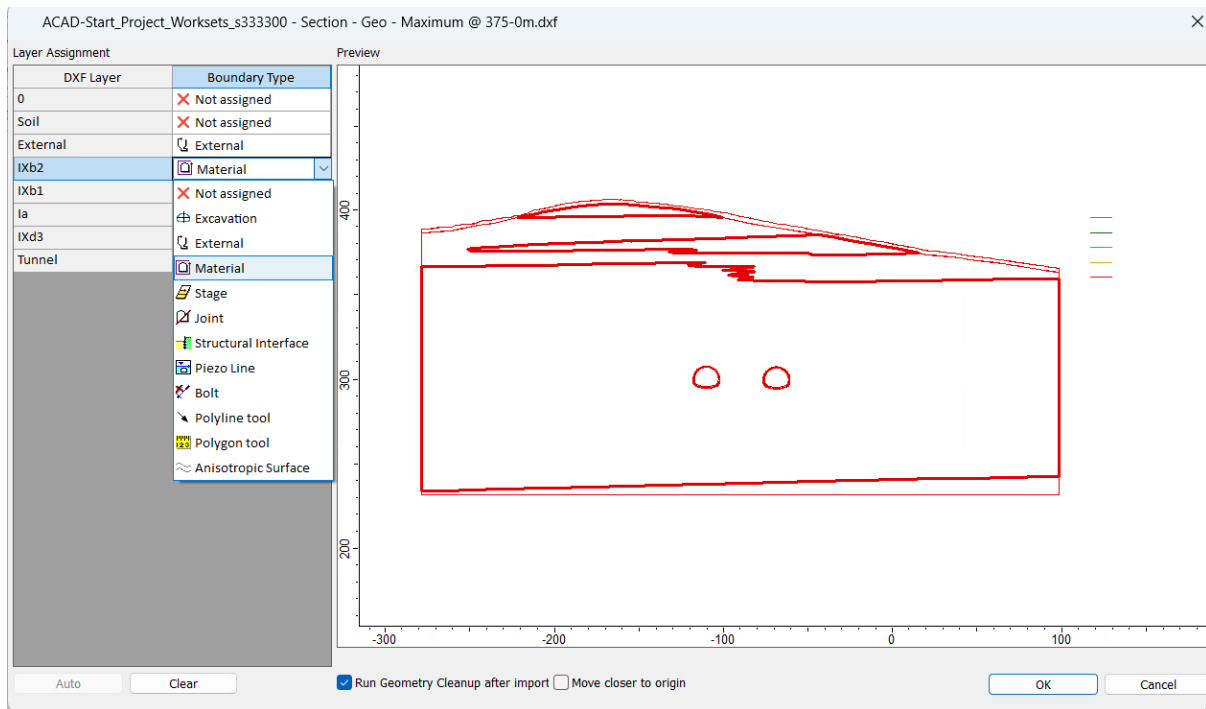


Figure 93. The RS2 DXF import dialog, demonstrating the direct assignment of optimized CAD layers to their corresponding geotechnical boundary types

Once the layer assignment is confirmed, the 2D cross-section is generated within the RS2 workspace (Figure 94). The software effectively translates the mapped CAD lines into functional geotechnical boundaries. The outer limits of the model are outlined in purple, while the internal geological strata are defined by green material boundaries. Additionally, the twin tunnels are correctly recognised as excavation boundaries. Seeing these continuous lines with clearly placed vertices confirms that the geometrical cleanup in Civil 3D was successful, leaving no topological errors or overlapping segments to disrupt the mesh. The model tree on the left side of the interface also dynamically updates to organise these imported elements.

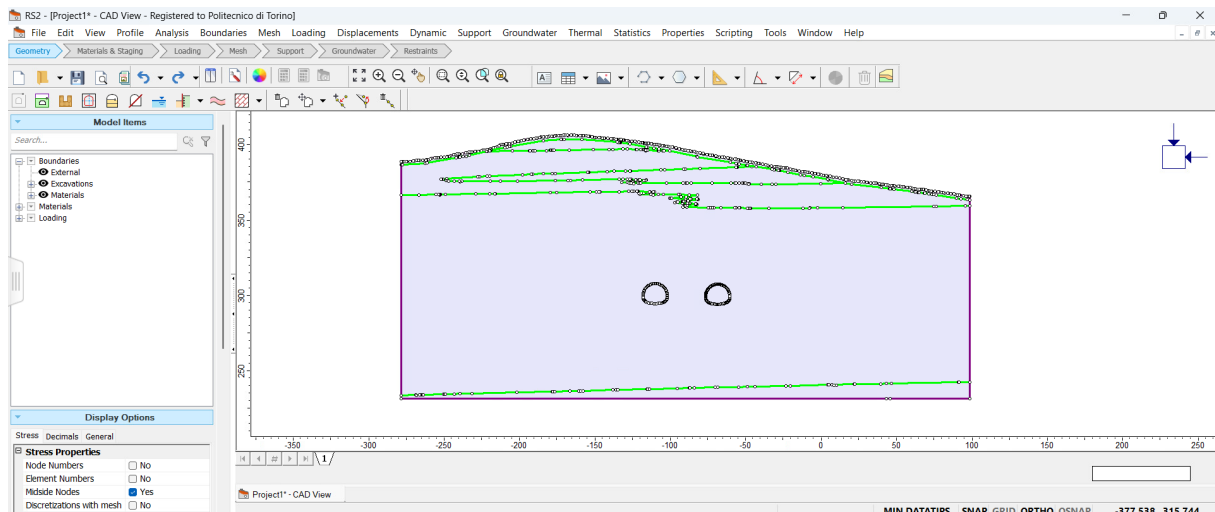


Figure 94. Initial DXF import in RS2, displaying the recognised material, external, and excavation boundaries

At this initial stage, the entire model is displayed in a single default material colour. This indicates that while the physical soil regions and spatial domains are perfectly established, their specific geomechanical properties have not yet been applied. The model essentially acts as a blank geometric canvas. With the geometry successfully imported and verified, the next phase of the workflow involves defining the structural parameters of each soil layer and assigning these material models to their corresponding regions.

While the DXF import process successfully establishes the physical boundaries of the model, manually inputting complex geomechanical parameters such as Hoek-Brown constants, unit weights, and stiffness properties for every individual soil layer across multiple cross-sections remains a highly repetitive and error-prone task. To fully close the interoperability loop and eliminate this manual data entry, a custom Python script was developed to interact directly with the RS2 Application Programming Interface (API) to automate the model initialization.

The automation sequence begins by establishing a secure communication link with the active RS2 modelling environment. By importing the `rs2.modeler` library, the Python script connects to the software's backend via a designated local server port. As seen in Figure 95 Port “60000” was chosen for connecting the server with the python scripter. Once this connection is verified, the algorithm automatically locates and opens the targeted finite element project file (.fez) to prepare it for data manipulation.

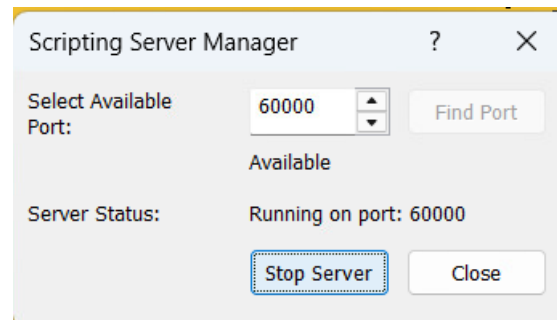


Figure 95. Scripting server port connector

4.4.3 Parameter injection and model initialization

To begin the automated configuration of the numerical model, the Python script must first establish a stable communication link with the active RS2 environment. As illustrated in Figure 96 the initial codebase, this connection is achieved by routing the programmatic commands through a designated local server port, for this example port 60000 was chosen. By utilizing a robust try-except block, the algorithm ensures that the target finite element project file (.fez) is securely located and opened within the workspace. This error-handling mechanism is critical; it provides immediate terminal feedback, allowing the script to safely halt execution rather than crash if the file path is invalid or the modeler fails to respond.

```

1  import pandas as pd
2  import math
3  from rs2.modeler.RS2Modeler import RS2Modeler
4  from rs2.modeler.properties.PropertyEnums import *
5
6  # =====
7  # FILE PATHS
8  # =====
9  fez_path = r"P:\Section-375m.fez"
10 rs2_csv_path = r"P:\RS2_Automated_Workflow\RS2.csv"
11
12 # =====
13 # CONNECT TO RS2 MODELER
14 # =====
15 port_num = 60000
16 modeler = RS2Modeler(port=port_num)
17
18 try:
19     model = modeler.openFile(fez_path)
20     print(f"Successfully opened: {fez_path}")
21 except Exception as e:
22     print(f"Failed to open model. Error: {e}")

```

Figure 96. Python script illustrating the connection to RS2 via a server port

Once the .fez file is successfully loaded, the workflow transitions to its most data-intensive phase: the automated definition of the geomechanical stratigraphy. In traditional modelling practices, assigning complex material properties across numerous soil layers is a highly repetitive manual task, making it incredibly susceptible to human error. To mitigate this, the proposed workflow completely externalizes the data management.

Once the Python script establishes a connection to the RS2 environment, the next step involves filling the empty numerical model with the correct geomechanical data. Instead of manually entering these values into the software, a process that is time-consuming and highly susceptible to errors, the algorithm relies on an external CSV database to manage the information.

The raw data extracted from the CSV includes intact rock properties such as the compressive strength (UCS), the intact Young's modulus (E_i), the intact rock parameter (m_i), and the Geological Strength Index (GSI). However, to accurately evaluate the tunnel's stability, RS2 requires the strength and stiffness parameters of the fractured rock mass rather than the intact rock. To manage this conversion, the Python script is programmed to automatically compute the Generalized Hoek-Brown constants (m_b , s , and a). By embedding these established empirical formulas directly into the code, the algorithm dynamically scales down the intact rock stiffness based on the site-specific GSI and disturbance factor. Figure 97 illustrates the script for extracting the raw data from the CSV database.

```

24 # =====
25 # PART 1: NUMERICAL INJECTION (RS2.csv)
26 # =====
27 print("\n--- Starting Material Property Injection ---")
28 df = pd.read_csv(rs2_csv_path)
29 materials = model.getAllMaterialProperties()
30
31 def get_val(row, col_name):
32     if col_name not in row: return None
33     val = row[col_name]
34     if pd.isna(val) or str(val).strip() == '': return None
35     return float(val)
36
37 for i, (index, row) in enumerate(df.iterrows()):
38     layer_name = str(row['LayerName']).strip()
39
40     if i < len(materials):
41         mat = materials[i]
42         mat.setMaterialName(layer_name)
43
44         gamma = get_val(row, 'Unitweight')
45         nu = get_val(row, 'PoissonsRatio')
46         e_i = get_val(row, 'YoungsModulus') or 0.0
47         ucs = get_val(row, 'UCS') or 0.0
48         gsi = get_val(row, 'GSI') or 0.0
49         mi = get_val(row, 'mi') or 0.001
50         d_factor = get_val(row, 'D') or 0.0

```

Figure 97. Python script for extracting raw geomechanical parameters from an external CSV database

The decision to embed the empirical Generalized Hoek-Brown formulas directly into the Python script was driven by a specific technical limitation of the RS2 Application Programming Interface (API). Although the software features a convenient, built-in GSI Calculator within its graphical user interface, the Python Scripser is currently unable to access or trigger this internal tool programmatically. Because the API cannot reach through the required parameters, they had to be derived externally. By implementing the established empirical equations directly within the codebase, the script successfully bypasses this software constraint, ensuring the automated workflow remains entirely unbroken and functional without requiring manual intervention. Figure 98 shows the code and formulas used for the parameters, all acquired from chapter 4.4.1.

```
--
52     e_rm = e_i
53     if e_i > 0 and gsi > 0:
54         exponent = (60.0 + 15.0 * d_factor - gsi) / 11.0
55         e_rm = e_i * (0.02 + ((1.0 - d_factor / 2.0) / (1.0 + math.exp(exponent))))
56
57     mb, s, a = 0.0, 0.0, 0.5
58     if gsi > 0:
59         mb = mi * math.exp((gsi - 100.0) / (28.0 - 14.0 * d_factor))
60         s = math.exp((gsi - 100.0) / (9.0 - 3.0 * d_factor))
61         a = 0.5 + (1.0 / 6.0) * (math.exp(-gsi / 15.0) - math.exp(-20.0 / 3.0))
62
```

Figure 98. Generalized Hoek-Brown parameters script

With the raw geotechnical data extracted and the rock mass parameters mathematically derived, the workflow advances to the critical stage of material initialization. This phase involves the direct, programmatic injection of these finalized values into their specific, dedicated property fields within the finite element model, effectively bridging the gap between the dataset and the computational solver.

```
63     if gamma is not None and gamma > 0:
64         mat.InitialConditions.setUnitWeight(gamma / 1000.0)
65     if nu is not None and nu > 0:
66         mat.Stiffness.Isotropic.setPoissonsRatio(nu)
67     if e_rm > 0:
68         mat.Stiffness.Isotropic.setYoungsModulus(e_rm)
69
70     mat.Strength.setFailureCriterion(StrengthCriteriaTypes.GENERALIZED_HOEK_BROWN)
71     try:
72         ghb = mat.Strength.GeneralizedHoekBrown
73     except AttributeError:
74         ghb = mat.Strength.GeneralizedHoekBrownStrength
75
76     ghb.setCompressiveStrength(ucs)
77     ghb.setMbParameter(mb)
78     ghb.setSPParameter(s)
79     ghb.setAPParameter(a)
80
81     print("Part 1 Complete: Mathematical parameters injected.")
```

Figure 99. Python script demonstrating the insertion of Generalized Hoek-Brown parameters directly into their fields

As illustrated in the codebase (Figure 99), the Python algorithm utilizes the software's application programming interface to systematically target the designated material slots within the backend. Rather than dumping the data indiscriminately, the script precisely maps the calculated variables to their exact, required locations.

4.4.4 Programmatic formulation of structural support systems

After initializing the soil layers, the next step in the automation process is to define the structural support elements. To simulate the excavation accurately, the model needs to include the tunnel support system. This specific case study uses a standard two-pass lining method: a temporary shotcrete layer and a final cast-in-place concrete lining. Instead of setting these components up manually in RS2, the Python script directly inputs their geometric and elastic properties.

```

108 # PART 2: INDIVIDUAL LINER DEFINITIONS
109 # =====
110 print("\n--- Configuring Individual Liners ---")
111 liners = model.getAllLinerProperties()
112
113 if len(liners) < 2:
114     print("\n[ERROR] Missing Liner slots. Please add them in the RS2 GUI.")
115 else:
116     # Safely define Liner 1
117     try:
118         liner1 = liners[0]
119         liner1.setLinerName("Shotcrete")
120         set_liner_properties(liner1, 0.05, 30000, 0.2)
121         print(" Shotcrete configured.")
122     except Exception as e:
123         print(f" Failed to configure Shotcrete. Error: {e}")
124
125     # Safely define Liner 2
126     try:
127         liner2 = liners[1]
128         liner2.setLinerName("Final Concrete")
129         set_liner_properties(liner2, 0.3, 30000, 0.2)
130         print(" Final Concrete configured.")
131     except Exception as e:
132         print(f" Failed to configure Final Concrete. Error: {e}")
133 print("Part 2 Complete.")

```

Figure 100. Python script showing the automated setup of the tunnel liners

Figure 100 shows the script used for this task. First, the algorithm checks the RS2 model to retrieve the available liner slots. It includes a basic check to ensure that at least two slots exist, printing a warning in the terminal if they are missing. Once the setup is confirmed, the script targets the first structural slot and renames it to "Shotcrete". It then uses a helper function to assign a thickness of 0.05 m, an elastic modulus of 30,000 MPa, and a Poisson's ratio of 0.2.

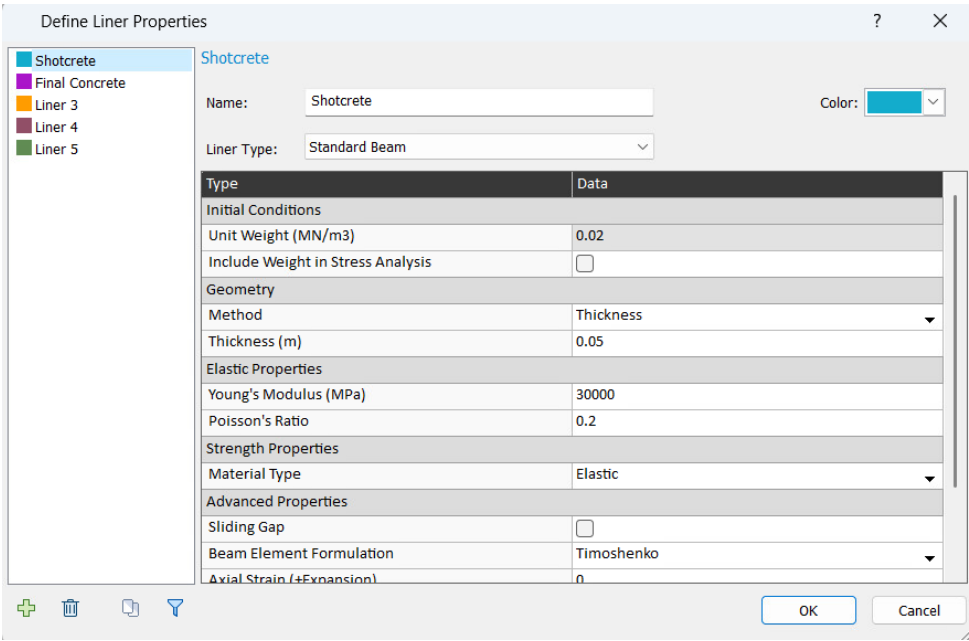


Figure 101. RS2 property dialog for the primary Shotcrete liner support. RS2 property dialog for the primary Shotcrete liner support

Figure 101 demonstrates how these values are applied in the RS2 interface under the define liner properties. This setup allows the finite element model to simulate the initial support provided by the shotcrete during ground relaxation.

Next, the script defines the permanent lining. It selects the second available liner object, changes its name to "Final Concrete", and applies the parameters required for the long-term support. While the elastic properties remain the same ($E = 30,000$ MPa, $\nu = 0.2$), the structural thickness is set to 0.3 m.

After defining the individual properties for the shotcrete and the cast-in-place concrete, the automation process moves to combining these elements into a unified support system. In modern tunnel construction, structural supports are rarely installed simultaneously. Typically, a primary flexible lining is applied immediately after excavation to stabilize the rock face, followed later by a secondary, permanent lining. To simulate this phased construction sequence accurately in RS2, the script must configure a composite liner.

```

135 # =====
136 # PART 3: COMPOSITE LINER CONFIGURATION
137 # =====
138 print("\n--- Configuring Composite Liner System ---")
139 try:
140     if len(liners) >= 2:
141         composite = model.getAllCompositeLinerProperties()[0]
142
143         # 1. Name the system
144         composite.setCompositeName("Tunnel Support System")
145
146         # 2. Set to 2 layers
147         composite.setNumberOfLayers(2)
148
149         # 3. Assign the specific Liners we just created by their exact new names
150         composite.setCompositeLinerPropertyByName(1, "Shotcrete")
151         composite.setCompositeLinerPropertyByName(2, "Final Concrete")
152
153         # 4. Set the 2-stage installation delay for Layer 2 (Final Concrete)
154         composite.setInstallDelay(2, 2)
155
156         print("Part 3 Complete: Composite Liner configured with 2-stage delay.")
157     else:
158         print("Skipped Composite Liner configuration due to missing Liner slots.")
159 except Exception as e:
160     print(f"Failed to configure Composite Liner. Error: {e}")

```

Figure 102. Python script demonstrating the automated setup of a two-layer composite liner

Figure 102 illustrates the Python code responsible for setting up this interaction. First, the script checks if the required liner slots are available to avoid execution errors. It then retrieves the first available composite liner property from the model database and renames it to "Tunnel Support System" to maintain clear organization within the project. Following this, the algorithm configures the system to consist of two distinct structural layers.

Instead of defining new material properties from scratch, the script directly links the individual liners that were generated and parameterized in the previous step. It assigns "Shotcrete" as the first layer, representing the immediate support applied directly against the excavated boundary. Then, it assigns "Final Concrete" as the second layer.

To accurately reflect the physical construction timeline, the permanent concrete cannot be activated at the exact same time as the shotcrete. To handle this, the script applies a targeted installation delay to the second layer. By setting this parameter, the final concrete is programmed to activate exactly two construction stages after the primary shotcrete is installed. This specific delay ensures that the numerical model accounts for the real-world time gap between the initial stabilization of the tunnel and the casting of the permanent structure, allowing the surrounding ground to undergo realistic stress relaxation. By executing this configuration through the Python API, the entire multi-stage support system is fully integrated into the model without requiring any manual setup in the graphical interface.

The successful execution of this automated configuration is directly reflected in the RS2 graphical user interface, as presented in Figure 103. The Define Composite Properties dialog confirms that the Python script accurately displays the two distinct structural layers exactly as

programmed. The primary Shotcrete layer is positioned as the first liner at stage 5, meanwhile, the secondary Final Concrete layer is correctly assigned its specific installation parameter, set to activate at stage 7, exactly two stages after the initial support is placed. This result fully validates the scripted approach, verifying that the numerical model properly incorporates the staggered construction sequence and is completely prepared for the staged excavation analysis without requiring any manual data entry.

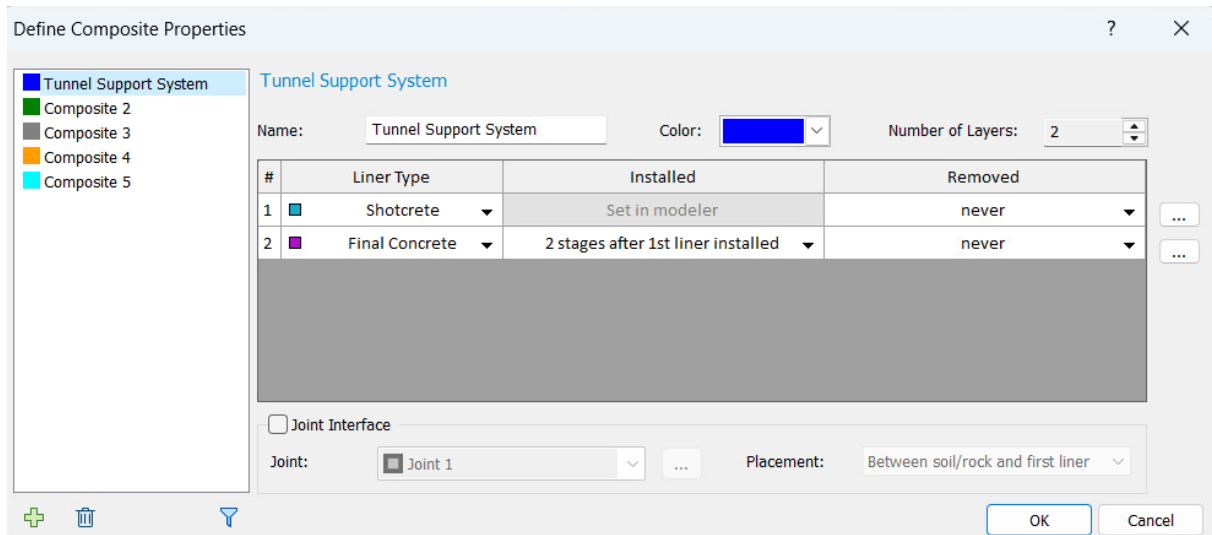


Figure 103. The RS2 Define Composite Properties dialog

4.5 Summary of the interoperability framework

While the Python scripts developed in this chapter significantly reduce the need for manual data entry, it is important to acknowledge that a completely automated, end-to-end finite element analysis is not currently possible. This limitation primarily stems from the existing constraints of the RS2 Application Programming Interface (API). Although the API effectively handles the creation of materials, the injection of geomechanical parameters, and the definition of structural liners, it currently lacks the capacity to execute complex topological commands. For example, the actual generation of the finite element mesh cannot be reliably automated through the script. The software still requires an engineer to manually review the geometric boundaries and trigger the discretization process through the graphical interface. Furthermore, assigning specific construction stages to the excavated voids and applying the exact boundary conditions to the outer perimeter still rely on manual selection. Because these critical analytical steps demand

human oversight, the framework is best understood as a highly optimized, semi-automated pipeline rather than a fully autonomous system.

Despite these software limitations at the final computation stage, the methodology presented in this chapter successfully establishes a functional, unidirectional workflow from Building Information Modelling to finite element analysis. The process began by combining the 3D geological data and the structural tunnel alignment into a single Open-BIM baseline within Autodesk Revit. Using a custom Dynamo and Python script, the workflow acted as a spatial radar, intelligently slicing the 3D model to extract 2D cross-sections specifically at critical chainages where the ground profile varied.

Because raw BIM exports are inherently too fragmented for numerical solvers to process, the workflow introduced AutoCAD Civil 3D as a necessary intermediate step. Within this environment, a secondary automation script cleaned the fragmented linework, closed the geometric boundaries into continuous polylines, and standardized the CAD layer names by cross-referencing their RGB color codes. Finally, the optimized DXF geometry was imported into RS2, where the final Python script automatically injected the complex rock mass parameters and structured the composite liner systems, entirely bypassing the need for manual data input.

In summary, this chapter demonstrates a practical method for bridging the "information islands" that traditionally separate structural design from geotechnical analysis. By replacing manual drafting and repetitive data transcription with targeted visual programming and API scripting, the proposed workflow effectively prevents the geometric errors that usually disrupt BIM-to-FEM data transfers. This automated preparation provides a reliable and structurally sound foundation for the numerical simulations that will be evaluated in the following chapter.

Chapter 5: Workflow validation and numerical results

The previous chapters detailed the technical steps needed to connect the BIM environment with finite element analysis software. Now, it is time to put this automated framework to the test. In a study focused on developing a new computational workflow, the main goal of the results section is not to provide a final geotechnical design for the tunnel. Instead, it is to prove that the data pipeline is reliable, functional, and produces models that are ready for immediate analysis.

To show that the Dynamo script is flexible and not just tailored to one perfect location, the first part of this chapter presents several different 2D cross-sections extracted from the Revit model. Generating these sections at various chainages along the alignment demonstrates how well the algorithm adapts to changes in the tunnel's curvature and the shifting geological layers.

After confirming that the geometry extraction works consistently across different sections, the final step is to validate the workflow inside the RS2 software. Crucially, detailing the programmatic injection of material parameters and structural liners serves as direct proof that the model initialization process within RS2 can be successfully automated via Python scripting. By generating a computational mesh and running a staged excavation simulation on this script-prepared setup, this chapter proves that the imported structural data can genuinely support complex geotechnical calculations. Ultimately, these practical tests confirm that the proposed framework successfully prevents the geometric errors typically caused by software interoperability, while simultaneously saving significant manual drafting and data entry time.

5.1 Automated section extraction across varying chainages

To prove that the Dynamo workflow is genuinely robust, it must be tested across multiple locations along the tunnel alignment. Relying on a single, ideal cross-section is not enough to validate the methodology. Therefore, the automated script was run to generate several slices at different chainages. This step confirms that the algorithm can consistently handle the real, shifting conditions of the 3D model.

As the tunnel path progresses, the depth of the soil cover changes, the geological layers shift, and the tunnel itself curves and slopes. Despite these constant spatial variations, the script successfully calculated the correct perpendicular angles for every single cut. It also automatically applied the 10D boundary rule to each new section, ensuring that every generated view was perfectly sized and geometrically prepared for the geotechnical export.

The following figures display a selection of these automated cross-sections directly within the Revit environment. As seen in the images, the script dynamically adjusts to the specific geology, tunnel depth, and structural orientation at each chainage, proving the scalability and reliability of the extraction process.

To demonstrate the script's capability in identifying critical geological profiles, Figure 104 presents the cross-section located at the 235.0-meter chainage, this section reveals a different topography compared to the rest of the alignment. Furthermore, Figure 105 illustrates a complex stratigraphic profile where multiple distinct soil layers (IXb1, IXb2, VIIId1, and Ia) are present within a single cross-section.

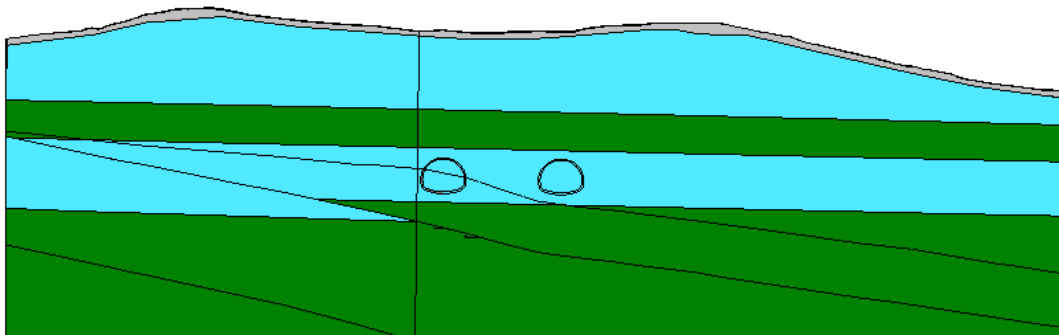


Figure 104. Automated 2D cross section at the 235.0-meter chainage

Beyond identifying critical extremes like minimum and maximum cover, the true efficiency of the workflow is demonstrated by how the script decides when to generate the next view.



Figure 105. Automated 2D cross section at 60-meter chainage

A standard, basic algorithm would typically slice the model at rigid intervals for example, every five meters along the entire length of the tunnel. While this approach guarantees nothing is missed, it generates a massive amount of redundant data when the ground profile is completely uniform, which wastes both time and space.

Instead, the Dynamo script is programmed with conditional logic. It continuously evaluates the alignment and only triggers a new cross-section cut at a five-meter mark if it detects a meaningful spatial variation. This variation could be a transition into a different geological stratum, or a significant change in the tunnel's vertical slope. In these highly variable zones, frequent cross-sections are necessary to ensure the geotechnical model accurately reflects the shifting ground.

However, if the script checks the next five-meter segment and determines that both the soil layers and the tunnel orientation remain unchanged, it skips that chainage. By intelligently bypassing these identical, redundant cuts, the algorithm prevents data overload.

A clear, practical example of this intelligent extraction logic can be seen when analysing the tunnel alignment between the 600.0-meter and 635.0-meter chainages.

To determine whether a new section is necessary, the algorithm is programmed to monitor a specific geotechnical zone around the excavation. In this workflow, that critical boundary is defined as a radius of three times the tunnel diameter (3D). The script actively scans this 3D range to detect any intersections with new geological strata.

Figure 106 displays the cross-section generated at the 600.0-meter mark. Here, the visualised 3D influence zones (represented manually by the circular boundaries around the twin tunnels) capture a specific arrangement of soil layers. After extracting this view, the algorithm continued to evaluate the subsequent five-meter segments along the 3D model. Because the position of the soil layers within that critical 3D radius remained uniform, the script deliberately skipped the standard intervals at 605m, 610m, and so on. Extracting sections at these points would have simply produced duplicate data.

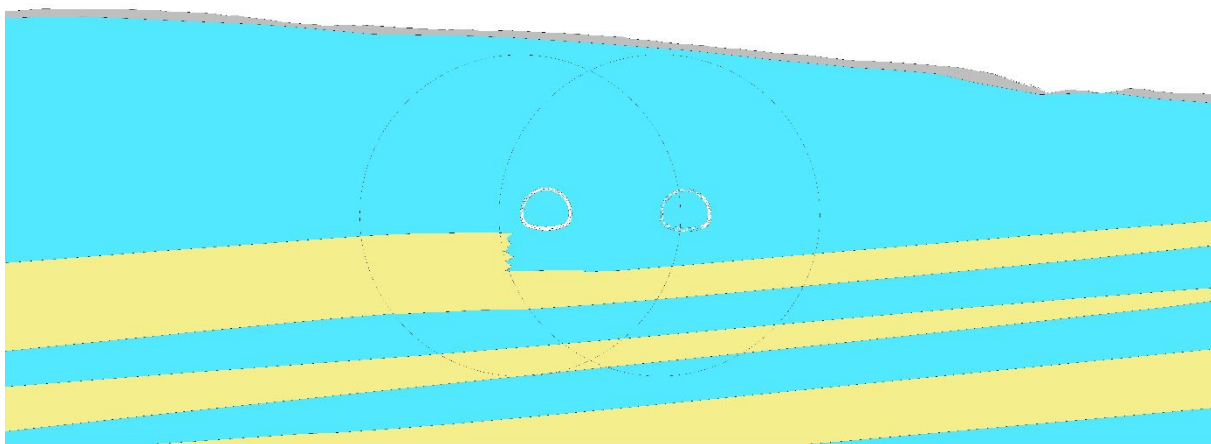


Figure 106. Automated cross-section at 600.0m chainage

The algorithm bypassed 30 meters of uniform ground until it reached the 635.0-meter chainage, shown in Figure 107. At this exact location, the script detected that the geological stratum had shifted, intersecting new soil layers. Recognising this as a meaningful spatial variation that would impact the finite element analysis, the script immediately triggered the creation of a new cross-section.

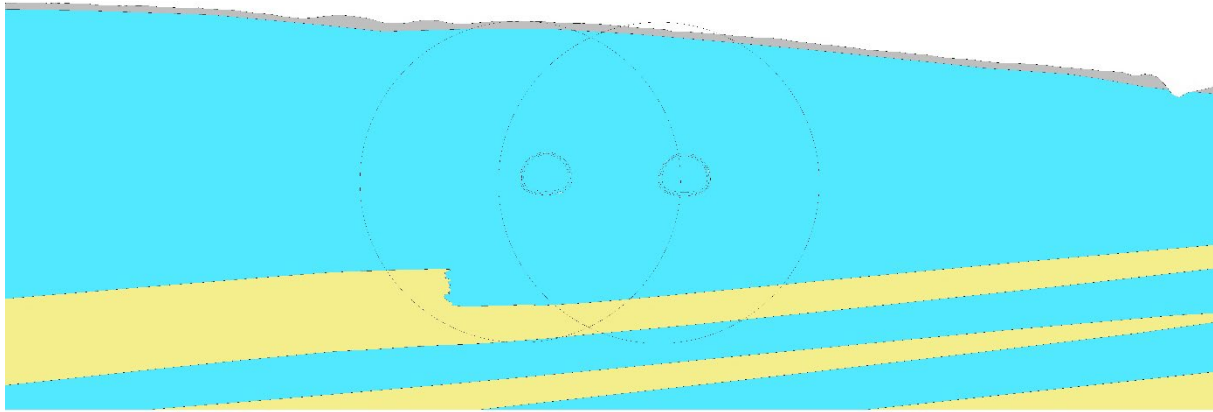


Figure 107. Automated cross-section at 635.0m chainage

Ultimately, this conditional extraction strategy ensures that the automated BIM-to-FEM framework remains efficient without minimizing geotechnical accuracy. With the geometry successfully extracted and optimized, this 635.0-meter cross-section was selected to undergo a full finite element simulation to validate the final stage of the interoperability workflow.

5.2 Finite element simulation and mesh generation in RS2

To validate the proposed framework, this section separates the automated RS2 initialization from the complete finite element analysis (FEA). It first demonstrates the practical execution of the automation scripts, detailing the geometric import and programmatic material injection using the cross-section extracted at the 375.0-meter chainage. Following this proof of automation, a comprehensive FEA of the 635.0-meter cross-section is presented in the next section to evaluate the computational mesh, the implementation of the load relaxation method, and the sequential construction stages.

5.2.1 Geometry importation and material assignment

The first step in the numerical modelling phase involves bringing the optimized DXF file into the RS2 workspace. To demonstrate the automated initialization process discussed in Chapter 4, the cross-section located at the 375.0-meter chainage was selected. As established earlier during the geometric extraction phase, this specific location represents the zone of maximum soil overburden.

While the DXF import successfully establishes the physical boundaries of the section, the model initially acts as a blank geometric canvas, devoid of any geomechanical properties or structural

support definitions. Manually inputting these complex parameters across multiple soil layers and composite liners is inefficient and highly prone to errors. To resolve this, the Python automation scripts were executed directly on this imported geometry.

The script rapidly iterated through the active CAD layers, systematically mapping the extracted unit weights, stiffness properties, and Generalized Hoek-Brown constants from the external CSV database to their respective boundary domains. The immediate result of this automation is visually confirmed within the RS2 graphical user interface.

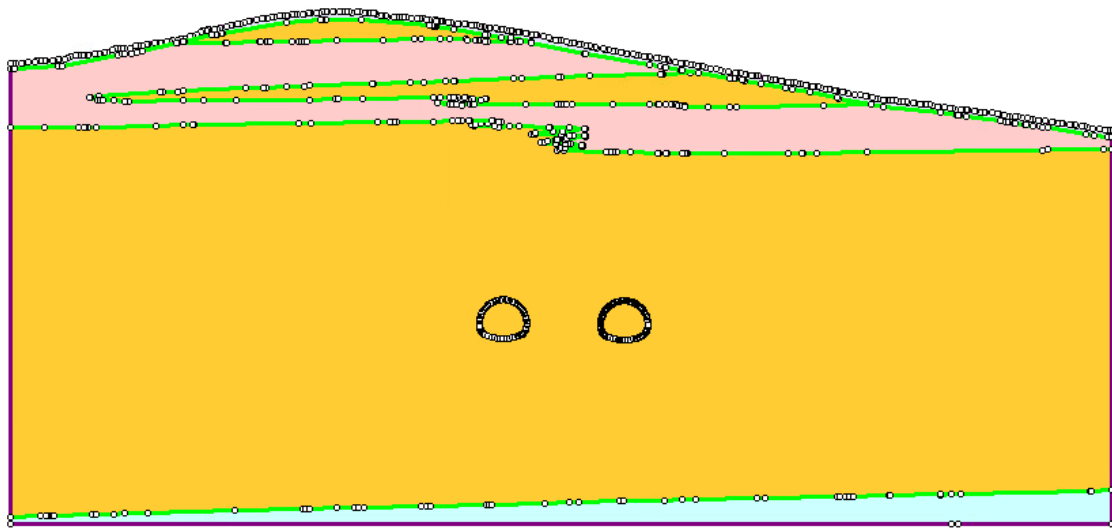


Figure 108. RS2 interface displaying the automated initialization of the 375.0-meter cross-section

As illustrated in Figure 108, the 375.0-meter cross-section is no longer a collection of empty boundaries. Without any manual intervention, the script accurately recognized the distinct

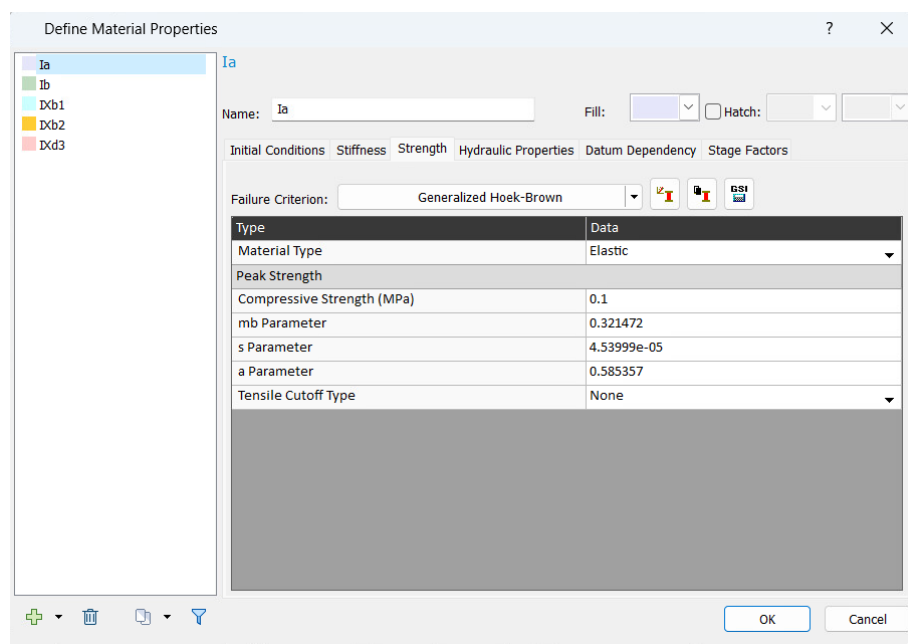


Figure 109. Strength material properties tab

geological strata and injected the precise numerical parameters required for the solver, which is reflected by the populated, color-coded material regions.

By successfully establishing the physical and structural parameters programmatically, the model was completely prepared for the computational meshing and staging phases. As seen in Figure 109, with the automation framework effectively validated for material property assignment. Furthermore, Figure 110 confirms the successful programmatic configuration of the composite liners. With this automation framework fully validated, the study now transitions to the core numerical simulation. To evaluate the ground response and structural stability, the subsequent sections will detail the mesh generation and stress analysis performed on the 635.0-meter cross-section.

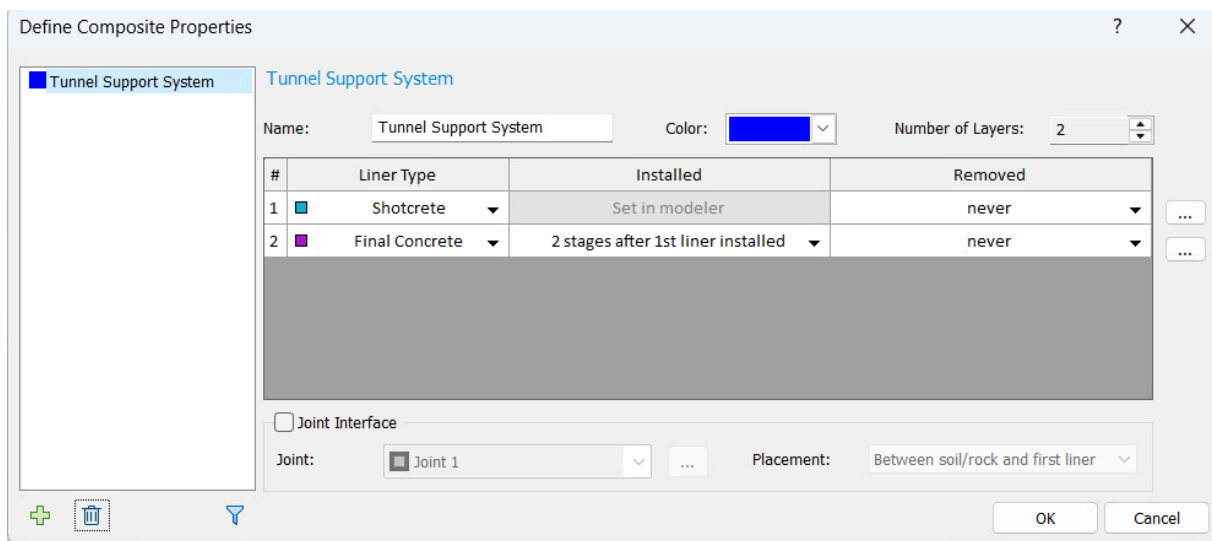


Figure 110. Composite liner properties panel in RS2 configured automatically

5.3 Mesh generation and staged excavation analysis of the 635.0-meter chainage

As seen in Figure 111, the 635.0-meter chainage cross-section was imported into the RS2 workspace following its optimization in Civil 3D. The geological boundaries are defined and, crucially, completely free of the intersecting lines and open polygons that typically induce meshing failures during numerical simulations. While the material parameters and structural supports are injected via the automation scripts seen in the previous sectors, the actual generation of the finite element mesh must currently be finalized manually due to existing software API limitations. Before the imported geometry can be analysed, the specific geomechanical properties of each soil stratum must be accurately defined within the RS2 database. As shown in Figure 112, the material profiles are configured based on the verified geotechnical data provided. This process involves establishing both the initial state and the ultimate strength characteristics of the ground.

For the initial conditions, parameters such as the unit weight of the soil, set to 0.026 MN/m³ for the Soil-IXb2 layer are entered to allow the software to correctly calculate the in-situ geostatic stresses prior to excavation.

To simulate how the surrounding rock mass behaves and yields during excavation, the model relies on the Generalized Hoek-Brown failure criterion. As established in Chapter 4, this non-linear empirical model is a standard approach in rock mechanics. The RS2 interface allows for the direct input of the material's uniaxial compressive strength along with its specific rock mass constants. By setting the material type to 'Plastic', the software computes residual strength parameters to account for any post-yield deformation in the ground illustrated in Figure 113.

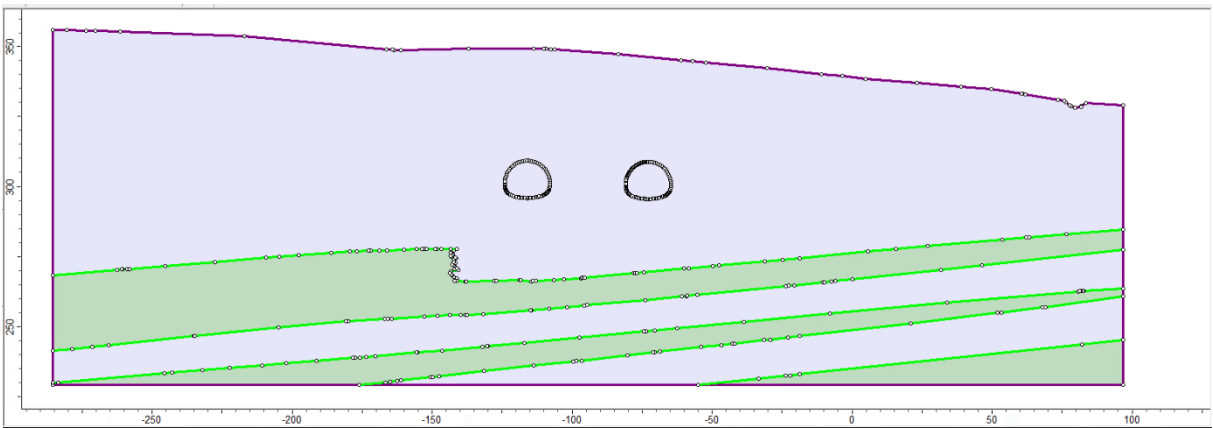


Figure 111. The 635m chainage cross-section imported into RS2 after being processed in Civil 3D

Once these profiles are defined in the material library, they are assigned to their respective regions in the 2D model. This step transforms the empty polygons imported from the BIM

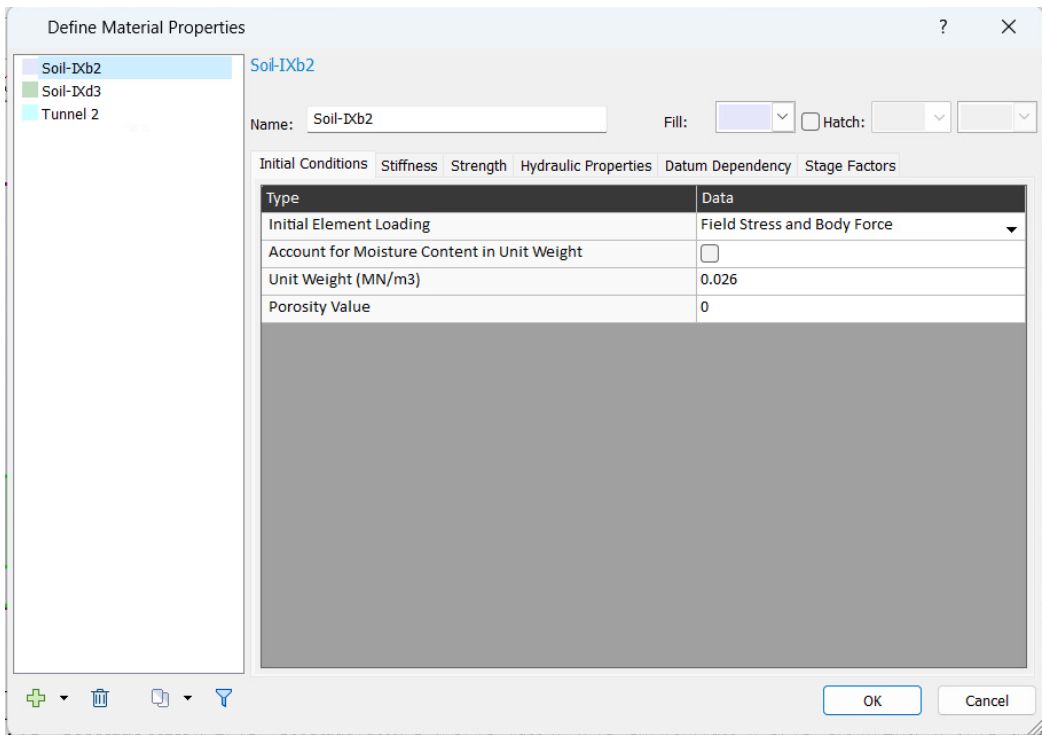


Figure 112. The RS2 material properties dialog

environment into active computational domains, ensuring the numerical model accurately reflects the physical mechanics of the real-world geology.

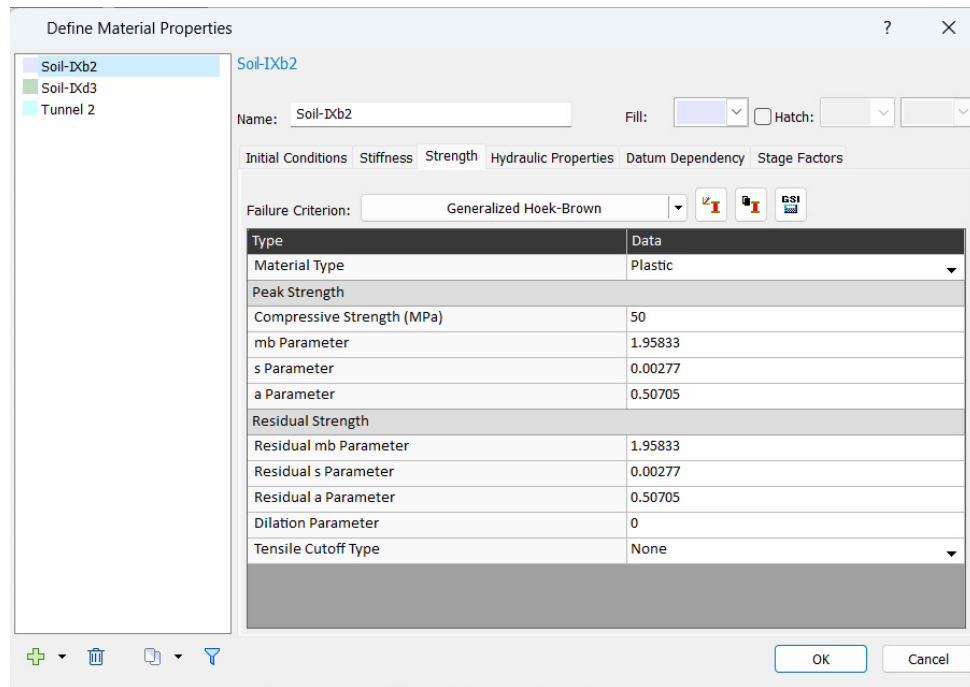


Figure 113. The RS2 Material Properties dialog, illustrating the strength parameters based on the Generalized Hoek-Brown failure criterion

5.3.1 Project settings and stress analysis

Once the materials are assigned, the next step is to configure the global computational parameters in the RS2 Project Settings. Because the model relies on the non-linear Generalized Hoek-Brown criterion to simulate rock mass behaviour, the solver is set to a 'Comprehensive' convergence type using effective stress analysis Figure 114. This ensures the software can accurately calculate how the ground yields and redistributes stress around the excavation.

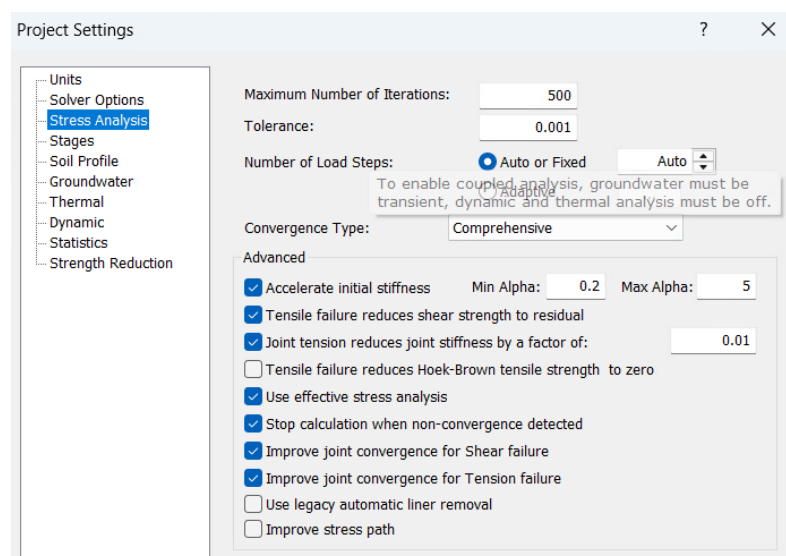


Figure 114. Project Settings dialog for Stress Analysis

However, tunnelling is essentially a three-dimensional process. As a tunnel advances, the surrounding ground relaxes, and structural supports are installed in a specific sequence. To accurately replicate this complex 3D progression within a 2D plane strain model, the analysis must be divided into sequential construction stages.

As outlined in the Project Settings dialog (Figure 115), the simulation is broken down into 14 distinct phases: seven for the first tunnel (T1), followed by an identical sequence for the second (T2), which will be explained in the next paragraph.

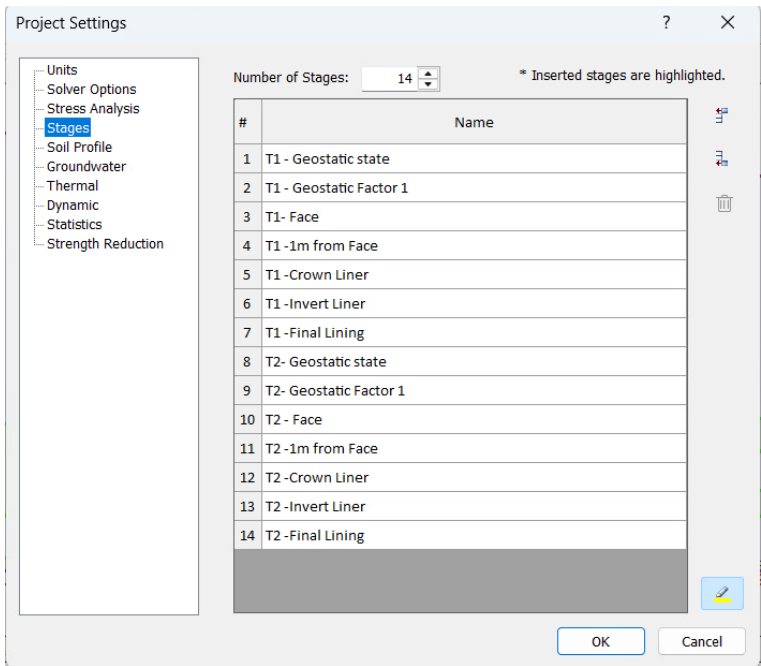


Figure 115. The 14 construction stages listed in RS2

To provide a comprehensive overview of the calculation process, the entire staging strategy is summarized in Table 3. This table details the specific excavation actions, support system activations, and the internal load relaxation factors applied at each step to simulate 3D convergence. It is important to note that upon the installation of the primary crown liner, the artificial internal pressure is entirely removed (reduced to 0%). This crucial step ensures the realistic transfer of the rock mass loads directly onto the physical structural elements.

Stage	Phase Name	Excavation Status	Support Activation	Load Split / Internal Pressure
1	T1 - Geostatic state	Initial ground conditions	None	Initial State
2	T1 - Geostatic Factor 1	Tunnel 1 core excavated	None	100% (Factor 1.0)
3	T1 - Face	Tunnel 1 excavated	None	70% (Factor 0.70)
4	T1 - 1m from Face	Tunnel 1 excavated	None	59% (Factor 0.59)
5	T1 - Crown Liner	Tunnel 1 excavated	Primary shotcrete (Crown)	0% (Pressure removed)
6	T1 - Invert Liner	Tunnel 1 excavated	Primary shotcrete (Invert)	0% (Pressure removed)
7	T1 - Final Lining	Tunnel 1 excavated	Final concrete lining (Full ring)	0% (Permanent support)
8	T2 - Geostatic state	T1 supported, T2 initial	None	Initial State
9	T2 - Geostatic Factor 1	Tunnel 2 core excavated	None	100% (Factor 1.0)
10	T2 - Face	Tunnel 2 excavated	None	70% (Factor 0.70)
11	T2 - 1m from Face	Tunnel 2 excavated	None	59% (Factor 0.59)
12	T2 - Crown Liner	Tunnel 2 excavated	Primary shotcrete (Crown)	0% (Pressure removed)
13	T2 - Invert Liner	Tunnel 2 excavated	Primary shotcrete (Invert)	0% (Pressure removed)
14	T2 - Final Lining	Both tunnels excavated	Final concrete lining (Full ring)	0% (Permanent support)

Table 3. Summary of the 14-stage finite element calculation sequence

The numerical analysis begins by establishing the in-situ stress field and undisturbed ground conditions (Stage 1). Once this mathematical baseline is set, the excavation of the first tunnel begins. In Stage 2, the physical soil elements within the tunnel perimeter are excavated. However, to temporarily support the resulting void, an internal boundary traction that perfectly opposes the initial in-situ stresses is applied (Factor 1.0). Once this mathematical baseline is set, the physical excavation of the first tunnel commences. Rather than removing the soil core and leaving the cavity entirely unsupported, the initialized internal pressure is sequentially reduced. This pressure drops to 70% at the excavation face (Stage 3), and further to 59% at one meter behind the face (Stage 4), allowing the ground to undergo controlled, realistic relaxation.

Following this managed convergence, the physical support system is introduced to stabilize the opening. The primary shotcrete liner is applied exclusively along the top heading to secure the excavation crown (Stage 5). Concurrently, the artificial internal pressure is completely removed, dropping to 0% which effectively transfers the rock mass loads entirely onto the newly activated structural elements. The simulation then progresses to excavate and support the invert (Stage 6), closing the structural ring. Finally, to reflect the physical delay in the real-world construction cycle, the permanent cast-in-place concrete lining is activated across the entire tunnel perimeter (Stage 7).

Once the first tunnel is fully supported and stabilized, this exact seven-stage cycle of computational initialization, load relaxation, crown support, ring closure, and final lining is repeated for the second tunnel (Stages 8 through 14). Executing this staging sequentially, is a fundamental requirement for the accuracy of the numerical model. This step-by-step progression allows the finite element mesh to realistically capture the asymmetric stress redistributions and complex interaction effects that occur when driving a new tunnel through a rock mass already disturbed by the primary excavation.

5.4 Boundary conditions, discretization, and numerical computation

Before the model can be meshed and solved, it must be spatially constrained. If gravity is applied without properly securing the boundaries, the software cannot achieve geostatic equilibrium, as the entire model would essentially fall infinitely downward in the 2D workspace. To accurately simulate a semi-infinite geological space within a confined computational box, standard displacement boundary conditions are applied to the external limits.

Using RS2's Restrain function in the Displacements tab; specific restrictions are placed on the model's perimeter. As shown in Figure 116, the vertical lateral edges are assigned a 'Restrain X' condition, represented by roller symbols. This allows the ground along these edges to settle vertically under its own weight but strictly prevents any horizontal movement. Meanwhile, the bottom boundary is assigned a 'Restrain Y' condition (pin symbols). This fully fixes the base. The top surface of the model is intentionally left completely free, allowing the ground to naturally deform and accurately capture any surface settlements caused by the tunnel excavation below. The corner edges of the section are set to Restrain X,Y since they should not move in both directions.

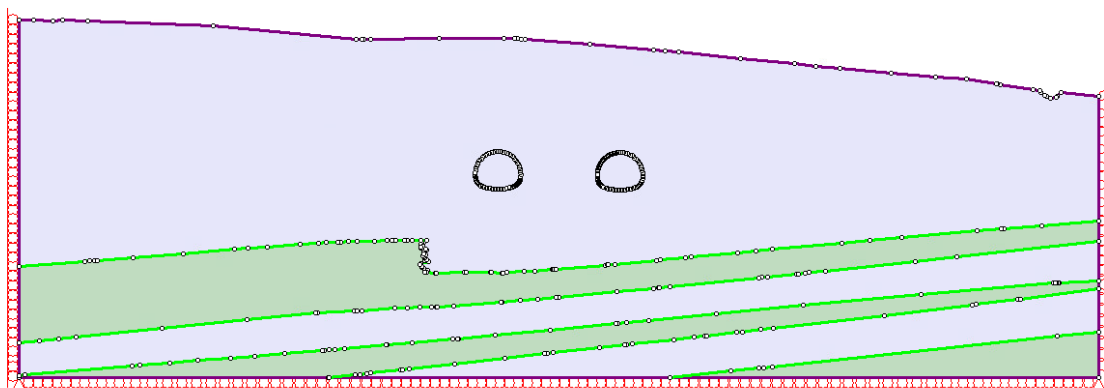


Figure 116. The imported cross-section with standard displacement boundary conditions applied

With the physical parameters and boundary conditions established, the geometric model is discretized into a finite element mesh. This step represents the ultimate validation of the proposed BIM-to-FEM workflow. Traditionally, directly importing architectural CAD geometry into numerical software results in severe meshing failures due to unclosed polygons, microscopic gaps, or intersecting lines. However, because the extracted Revit linework was programmatically cleaned and flattened in Civil 3D, RS2 seamlessly generated the mesh without a single topological error.

To generate the finite element mesh, the discretization parameters were configured within the RS2 Mesh Setup dialog (Figure 117). To balance computational efficiency with analytical accuracy, a 'Graded' mesh type was selected, utilizing higher-order 6-noded triangular elements. Because the most critical stress concentrations and displacement gradients occur directly at the excavation face, the mesh requires significant refinement in these localized areas.

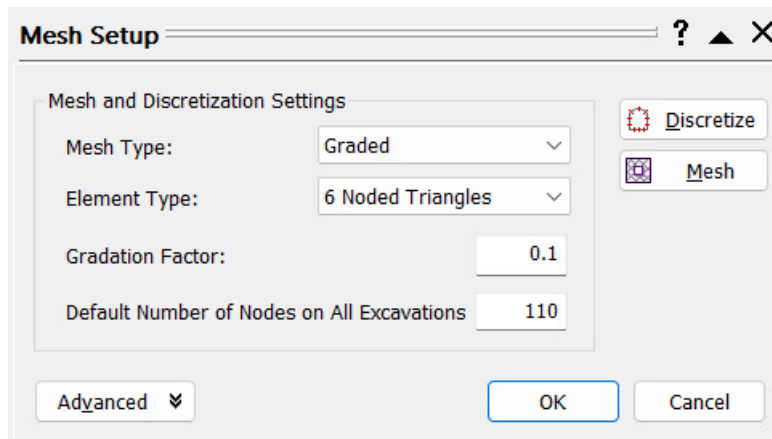


Figure 117. The RS2 mesh setup dialog

As illustrated in Figure 118, the 2D section is divided into triangular meshes. The mesh density has been refined around the perimeters of the tunnels. This refinement is crucial for the analysis; it ensures the software can accurately capture the intense stress concentrations and high displacement gradients that occur directly at the excavation face, while utilizing a slightly coarser mesh near the outer boundaries to optimize overall computational time.

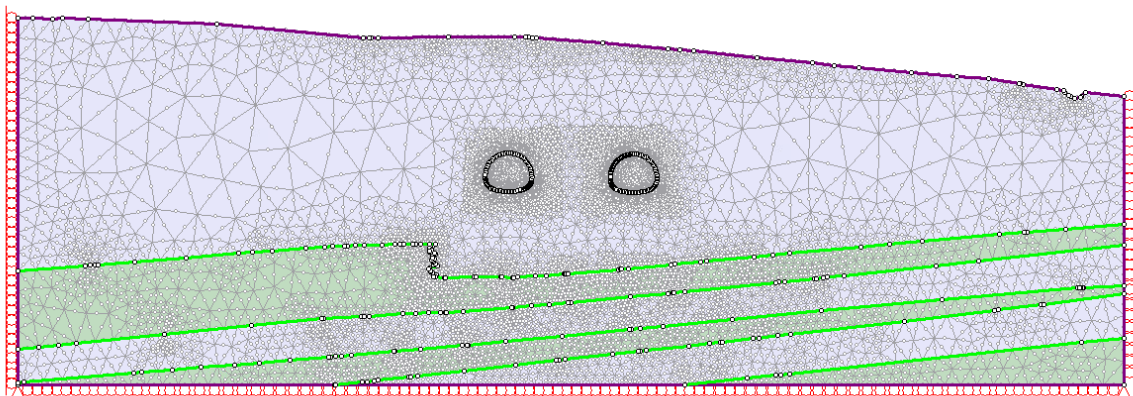


Figure 118. The generated finite element mesh

Once the mesh is finalized and verified, the computational engine is executed. The software solves the governing equations across all 14 sequential construction stages defined previously.

5.4.1 Implementation of the load relaxation method

With the boundary conditions set and the mesh generated, the numerical computation begins by simulating the sequential excavation. As outlined previously, this process relies on the load relaxation (or λ) method to safely transition the model from its initial undisturbed state to a supported excavation, preventing immediate numerical collapse.

In Stage 2 (T1 - Geostatic Factor 1), the physical soil elements within the Tunnel 1 perimeter are excavated. To temporarily support the resulting void, an internal boundary traction that perfectly oppose the initial in-situ stresses is applied in RS2. At this stage, the load split factor is exactly 1.0 (100%), meaning the internal tractions are fully supporting the excavation, and zero ground relaxation is permitted (Figure 119).

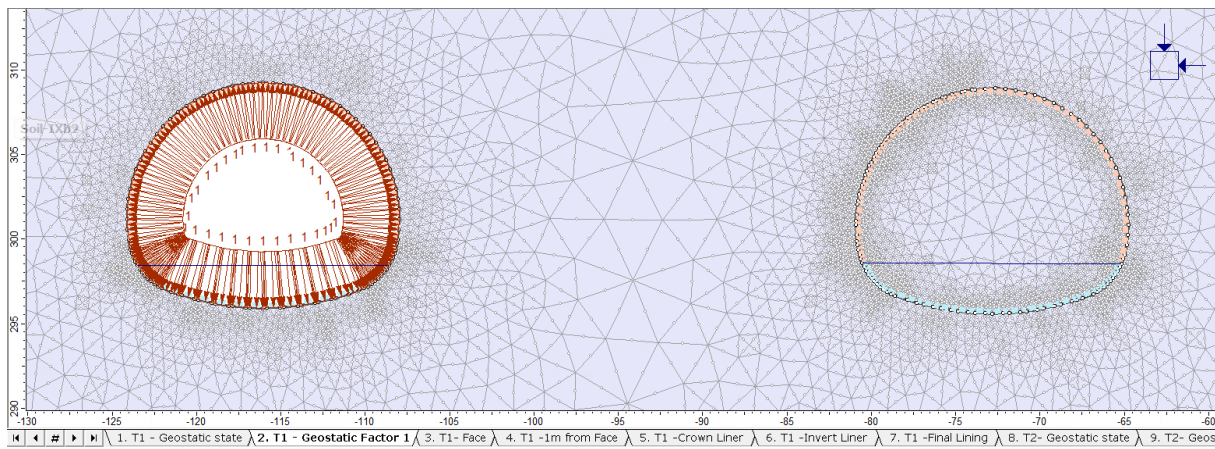


Figure 119. Stage 2, showing the geostatic factor 1 with 100% internal pressure

To accurately simulate the three-dimensional advancement of a tunnel within a 2D plane strain model, the exact degree of ground relaxation must be mathematically determined. To ensure the finite element analysis reflects true convergence-confinement principles, the RocSupport software was utilized to calculate the precise relaxation factors for the RS2 staging.

The initial step in this analytical process involved defining the specific tunnel and rock mass parameters. As shown in Figure 120, the equivalent tunnel radius was established at 8.07 m, subject to an initial in-situ stress calculated as 1.21 MPa. The surrounding geological medium was characterized using the Generalized Hoek-Brown failure criterion. Based on the project data, an intact uniaxial compressive strength of 40 MPa, a Geological Strength Index (GSI) of 47, and an intact rock parameter (m_i) of 13 were inputted to generate the rock mass stiffness and peak strength profiles.

Figure 120. Tunnel and rock mass parameters defined in RocSupport

Once the ground conditions were established, the next step was to define the temporary support system. The primary support was modelled as a 50 mm thick shotcrete liner with a uniaxial compressive strength of 35 MPa and a Young's modulus of 30,000 MPa (Figure 121). To accurately simulate the real-world construction sequence, the installation point for this liner was set exactly 1 meter behind the advancing tunnel face.

Figure 121. Support parameters in RocSupport, detailing the shotcrete properties at the distance of 1m from the face.

With these parameters defined, RocSupport generates the Ground Reaction Curve (GRC). As shown in Figure 122, this curve illustrates how the required support pressure naturally drops as the tunnel walls begin to converge. Plotted alongside it is the Support Capacity Curve for the shotcrete. The exact point where these two curves intersect pinpoints the final equilibrium state of the excavation, resulting in a strong Factor of Safety of 11.24.

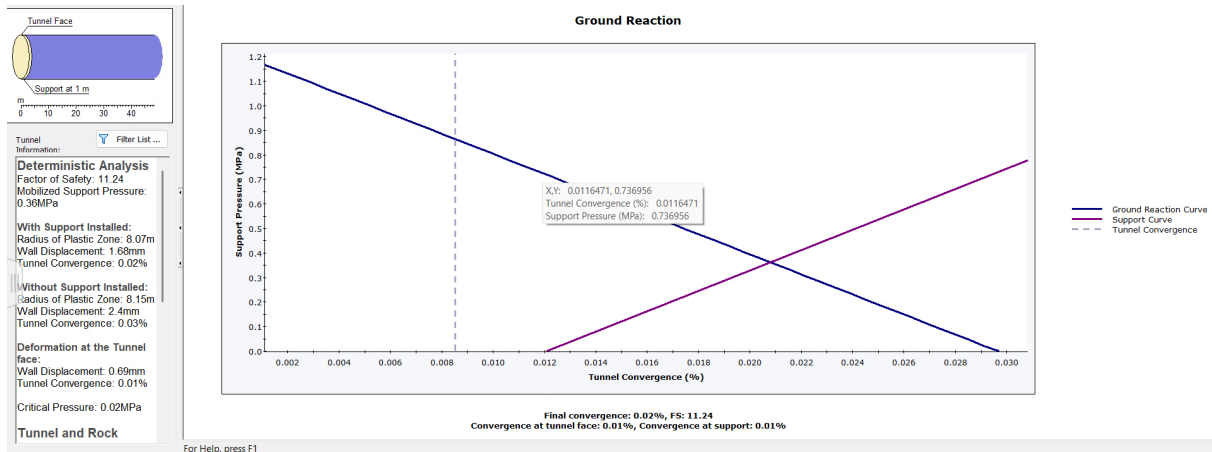


Figure 122. The ground reaction curve generated by RocSupport

While the GRC confirms the overall stability of the design, the most critical output for the RS2 simulation is the longitudinal displacement profile (LDP). The load split factors of 0.70 and 0.59 used in the finite element stages are derived by linking the GRC with the LDP. To find the factor for the tunnel face (Stage 3), RocSupport uses the LDP to determine the exact ground convergence right as the face is reached. By mapping this convergence onto the GRC, we find the remaining internal earth pressure. Dividing this required face pressure by the initial 1.21 MPa in-situ stress yields 0.70. This means that 70% of the original earth pressure is still actively supported by the unexcavated core.

The exact same logic dictates the factor for Stage 4, where the shotcrete is installed 1 meter behind the advancing face. According to the LDP, the ground converges further over this 1-meter span. Mapping this increased displacement back onto the GRC shows a natural drop in the required support pressure. Normalizing this updated value against the 1.21 MPa baseline results in the 0.59 factor. By directly linking the 3D analytical data from RocSupport to the 2D numerical stages in RS2, it ensures the simulation physically replicates the real-world 3D relaxation of the ground.

5.4.2 Sequential support installation and composite liners

Before the structural supports can be applied within the finite element model, the geometry itself must be prepared to reflect the physical construction method. In conventional tunnelling, large cross-sections are rarely excavated all at once. Instead, a top heading and invert sequence is typically employed. To accommodate this in RS2, a horizontal construction line was

explicitly drawn across the lower section of the tunnel profile. This line acts as a geometric divider, splitting the continuous excavation boundary into two distinct operational segments: the upper crown and the lower invert. This simple division is crucial because it allows the software to apply structural liners independently to the top and bottom halves at different times. The fictional splitting line was drawn using the Line command in the Tools tab, drawing a straight line from where the crown and invert are separated.

With the perimeter correctly divided, the specific structural materials must be established. Two distinct liner types were defined for this project. The primary temporary support is the 50 mm shotcrete layer discussed previously. The permanent support is defined as a cast-in-place final concrete lining. As detailed in Figure 123, this final lining is modelled using standard beam elements with a designated thickness of 0.3 meters and an elastic Young's modulus of 35,000 MPa.

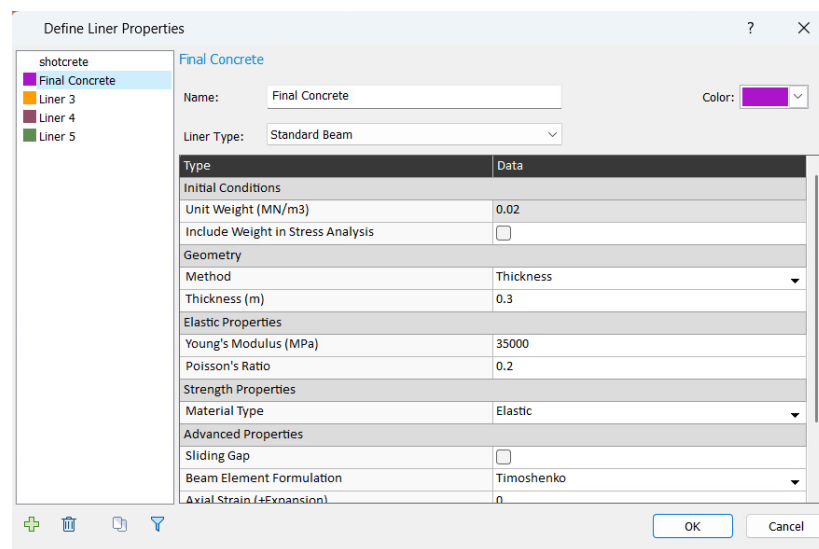


Figure 123. The Define Liner Properties dialog

To replicate the real-world construction timeline, these two materials were combined using the software's Composite Properties function (Figure 124). Instead of manually deleting temporary supports to replace them with permanent ones, the composite framework allows multiple structural layers to co-exist on the same boundary. The system is configured with the shotcrete as the primary layer and the final concrete as the secondary layer, with the latter programmed to activate automatically two stages after the initial installation. This setup realistically captures the physical delay between spraying the temporary stabilization layer and casting the permanent concrete structure.

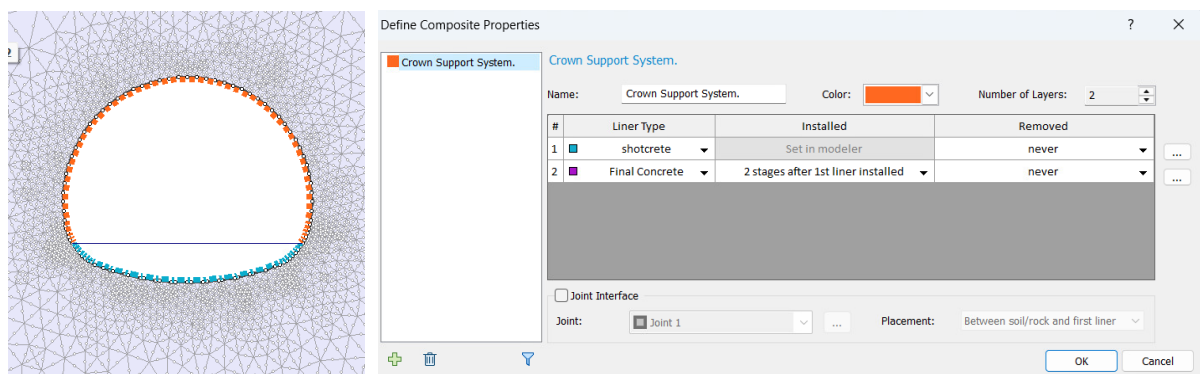


Figure 124. The "Define Composite Properties" dialog along with the crown and invert liner shown

With the composite system fully configured, the numerical staging proceeds to the structural support phases. In Stage 5 (T1 - Crown Liner), the internal load relaxation factor drops to 0.59. At this point of managed ground convergence, the shotcrete layer of the composite support is activated exclusively along the upper crown boundary to secure the tunnel roof (Figure 125).

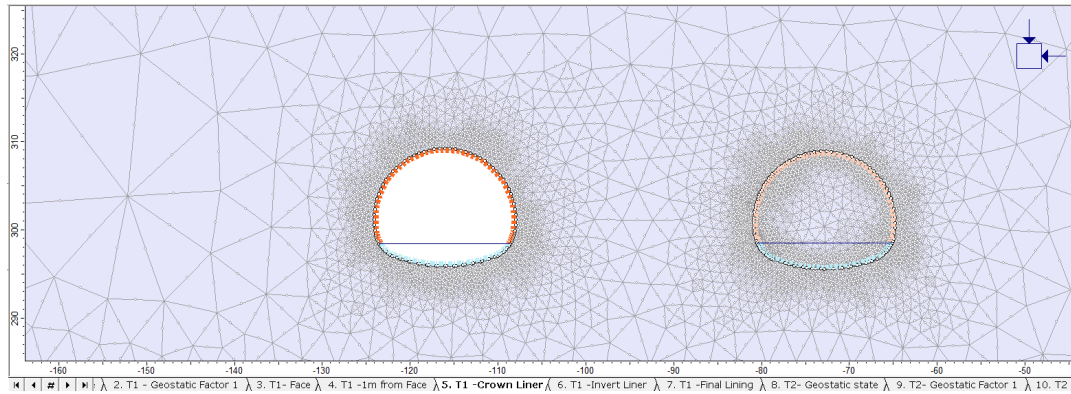


Figure 125. Stage 5 (Crown Liner), showing the activation of the support elements along the top heading

As the simulation advances to Stage 6 (T1 - Invert Liner), the primary shotcrete support is extended along the bottom boundary of the profile. Because the full-face core has already been removed in the earlier stages, this step focuses entirely on completing the liner application along the invert. Closing the structural ring in this manner is essential to stabilize the tunnel floor and prevent floor heave.

Finally, in Stage 7 (T1 - Final Lining), the model reaches the specific delay interval designated in the composite settings. The 0.3-meter final concrete lining automatically activates across the entire tunnel perimeter, establishing the permanent, long-term structural condition of the first tunnel.

Once Tunnel 1 is fully completed and stable, the same cycle of load relaxation, crown support, ring closure, and final lining is repeated for Tunnel 2 from Stage 8 to 14. Simulating the tunnels sequentially rather than at the same time is vital for the accuracy of the model. This setup allows the finite element mesh to register the asymmetric stress shifts and interaction effects that occur when driving a second tunnel through ground already disturbed by the first.

With all 14 construction stages fully configured and the composite support system assigned across both tunnels, the numerical setup is complete. The model is now ready for computation. As shown in Figure 126, the analysis is executed, prompting the software to solve the governing finite element equations sequentially across every stage. This computational phase generates the complete set of stress, strain, and displacement data required to evaluate the final twin-tunnel interaction.

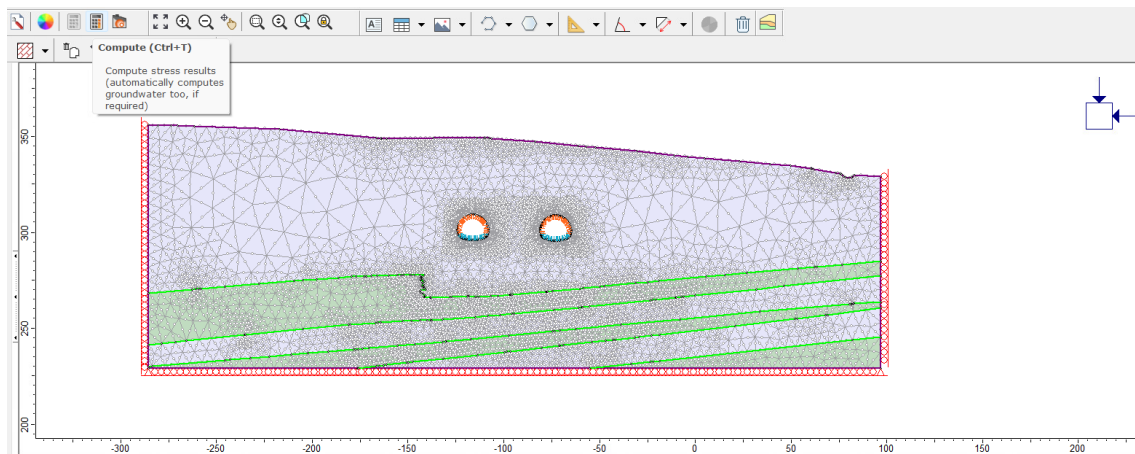


Figure 126. Computing (Run) the numerical analysis

5.5 Analysis of computational results

With the finite element computation complete, the focus shifts to interpreting the structural and geotechnical behaviour of the rock mass. The RS2 software generates a vast amount of data across all 14 stages, allowing for a detailed tracking of stresses, displacements, and yielding zones as the twin tunnels advance.

5.5.1 Verification of the initial geostatic state

Before evaluating the complex interactions caused by the excavation, it is standard practice to verify the initial undisturbed ground conditions established in Stage 1 (T1 - Geostatic state). This phase acts as the mathematical baseline for all subsequent calculations; any errors here would generate through the entire simulation.

A review of the effective stress contours confirms that the initial stress field is correctly generated. Both the major principal stress (σ_1) and minor principal stress (σ_3) exhibit a classic, gravity driven distribution. The contour bands are perfectly horizontal, indicating that the stresses increase linearly with depth from the ground surface down to the model's base boundary. As illustrated in Figure 127, the major effective stress reaches a maximum of approximately 1.85 MPa at the deepest elevation. Correspondingly, the minor effective stress follows the same uniform gradient, peaking at roughly 0.90 MPa, which reflects the horizontal at-rest earth pressure (K_0) of the specific geological layers (Figure 128).

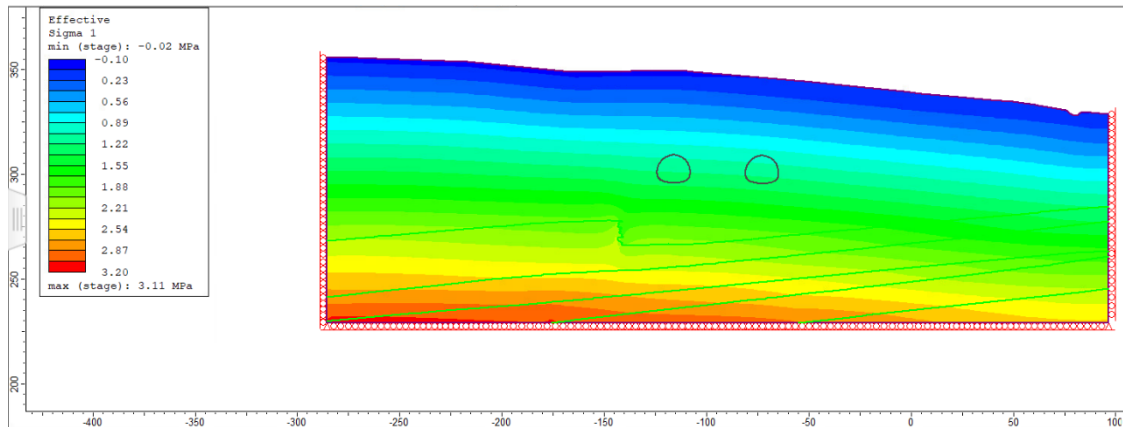


Figure 127. Stage 1 Major effective stress (σ_1) contour plot

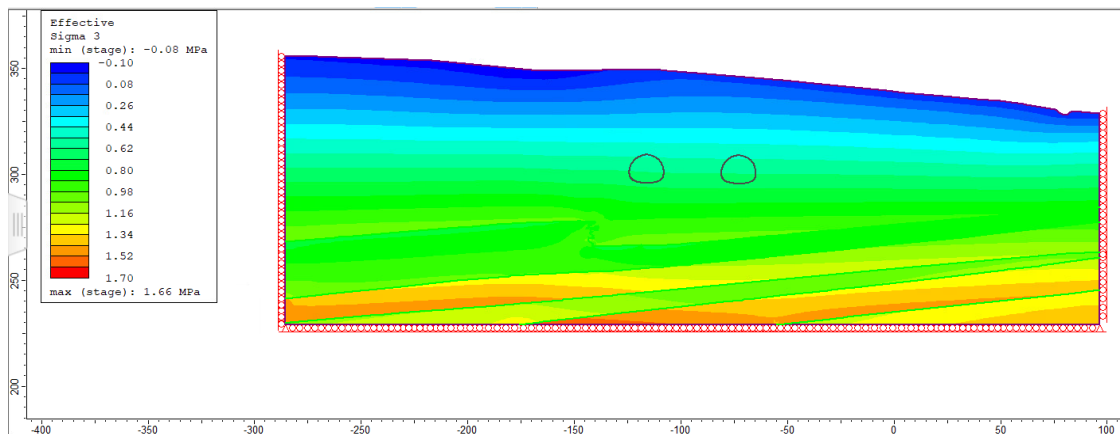


Figure 128. Stage 1 Minor effective stress (σ_3) contour plot

5.5.2 Evolution of ground displacements and surface settlement

Excavating twin underground openings alters the initial stress state of the rock mass, triggering ground movements that propagate upward toward the surface. Monitoring these deformations is a critical aspect of the design, especially in twin tunnelling configurations where the construction of one bore can directly influence the stability and settlement profile of the adjacent one. To analyse this behaviour, total displacement contours and surface settlement trends were evaluated at two major project milestones: the completion of the first tunnel (Stage 7) and the final state where both tunnels are fully supported (Stage 14).

At Stage 7, Tunnel 1 (T1) is fully excavated and stabilized by its primary shotcrete and secondary concrete lining, while the ground around the future Tunnel 2 (T2) remains completely undisturbed. The numerical results in Figure 129 show that ground deformation is highly localized around the perimeter of the active excavation. The maximum total displacement with the highest concentrations occurring at the tunnel crown and along the sidewalls. This convergence reflects the immediate elastic and plastic response of the rock mass during the load relaxation phases before the composite lining system fully engages to halt further movement.

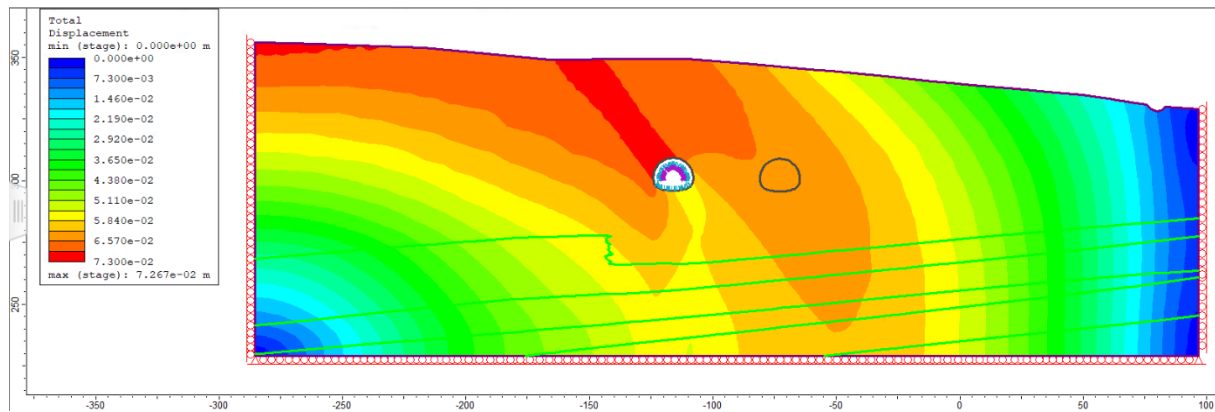


Figure 129. Total displacement contour map after the completion and final lining of Tunnel 1 (Stage 7).

When the simulation transitions to the excavation of Tunnel 2 (Stages 8 to 14), the displacement field undergoes a significant transformation. Tunnel 2 is driven through a rock mass that has already been stress-relieved and partially disturbed by the excavation of the first bore. Consequently, the ground response around the second tunnel is slightly more pronounced. By Stage 14, the maximum total displacement in the model increases to approximately 14 mm, concentrated predominantly around the crown and upper shoulders of Tunnel 2 shown in Figure 130.

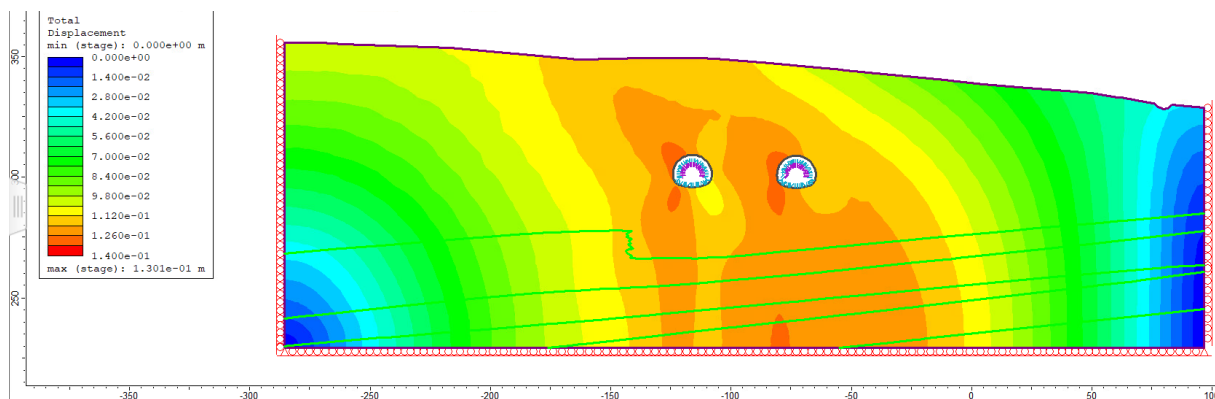


Figure 130. Final total displacement contour map of the twin-tunnel system at the completion of construction (Stage 14).

A key point to note in the Stage 14 results is how the central rock pillar between the two cavities behaves. Looking at the contour map, the light blue low-displacement zones from both tunnels

spread outward and meet in the middle. This confirms that the pillar is absorbing a combined load, feeling the structural weight and stress from both excavations at the same time.

Crucially, however, the core of this pillar shows no signs of significant movement or distress; its deformation values remain low and perfectly stable. This indicates that the rock mass in the center is holding up well under the double-sided pressure, proving that the chosen clear distance between the twin alignments is completely adequate to prevent any localized ground failure or collapse between the bores.

5.5.3 Analysis of rock mass yielding and plastic zone development

Aside from measuring ground movement, it is essential to look at where the rock mass yields under the new stress conditions. In RS2, yielded elements mark the specific zones where induced stresses have pushed past the shear or tensile strength of the rock, calculated here using the Generalized Hoek-Brown failure criterion. Mapping these points helps determine whether the localized failure is safely managed by the support system or if there is a genuine risk of deep, progressive ground collapse.

Looking at the results for Stage 7 in Figure 131, with Tunnel 1 fully excavated and lined, the distribution of these yielded elements is quite limited. The model output shows that plastic yielding is tightly confined to a narrow ring right around the excavation boundary of the first bore. These failure points appear as a close-knit perimeter around the tunnel profile, spanning the crown, sidewalls, and invert. Because the primary shotcrete and final concrete lining are already active, they successfully stop these failure zones from spreading deeper into the rock strata.

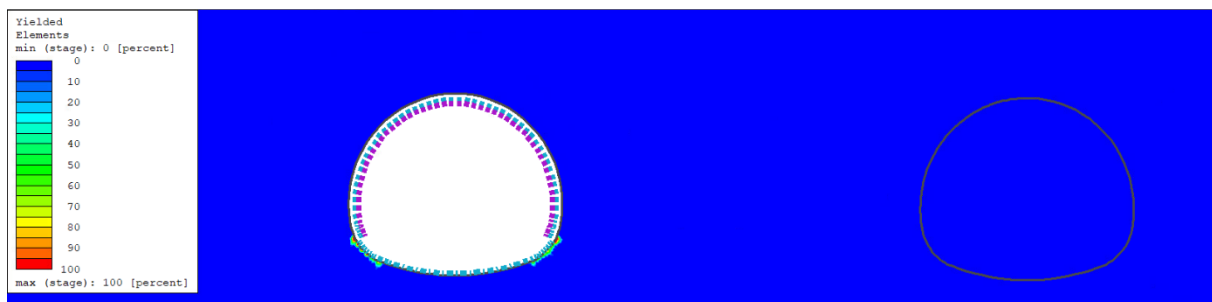


Figure 131. Distribution of yielded elements around Tunnel 1 at the completion of Stage 7

Crucially, the area designated for Tunnel 2 remains completely clear of any yielded elements, showing that the excavation of the first tunnel has not negatively impacted the undisturbed ground next to it. This confirms that the chosen load relaxation factor allowed the rock mass around Tunnel 1 to deform just enough to relieve the intense initial pressures without losing its self-supporting capacity or causing wide-scale damage.

Moving to Stage 14, digging Tunnel 2 creates a similar yielding pattern around the second opening, mimicking what occurred in the first bore. Since the two tunnels are close together, the main area of concern is the behaviour of the rock pillar separating them. The computational results in Figure 132 show that while some failure elements develop along the inner sidewalls facing the pillar, they do not cross or break through its centre.

The core of the pillar remains entirely elastic and unyielded. This proves that the overlapping stresses from both excavations do not overstress the intermediate rock mass to the point of structural failure. Ultimately, this result validates both the clear spacing chosen for the twin alignment and the long-term effectiveness of the composite lining design.

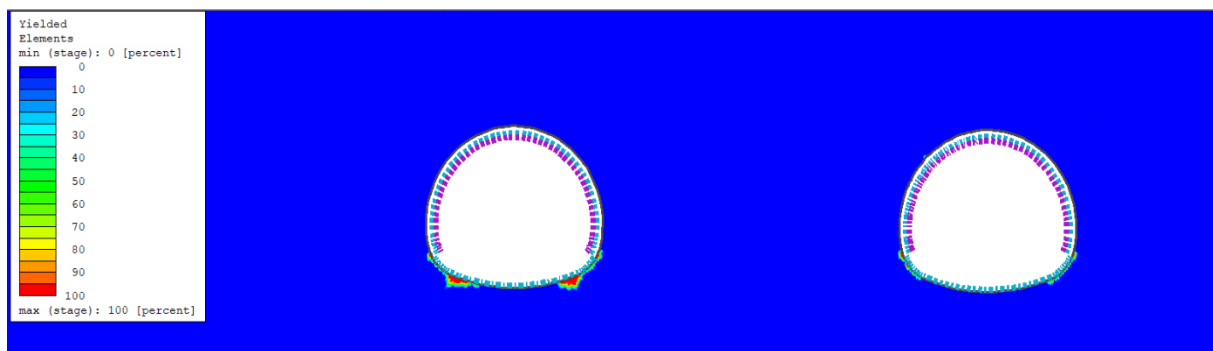


Figure 132. Final distribution of yielded elements across the twin-tunnel system at Stage 14

Chapter 6: Conclusion and future development

This research has successfully demonstrated the development and application of an automated pipeline that bridges the fields of Building Information Modelling (BIM) and geotechnical finite element analysis. At the centre of this study was the challenge of overcoming "information islands", a barrier in underground engineering where critical structural and geological data remain isolated within incompatible software environments, leading to costly manual remodelling, coordination errors, and project delays. By establishing an automated workflow connecting Autodesk Revit and Rocscience RS2 via Dynamo and Python, this thesis has provided a practical, scripted methodology to streamline design iterations during the planning and design phases of complex tunnel projects.

The primary research objective focused on using visual programming languages to automate the extraction of geometric and parametric data from the central BIM model. This was achieved through a customized Dynamo script that successfully extracted the three-dimensional tunnel alignment curve from linked IFC structural models, serving as the central spatial spine for the entire pipeline. Because real-world tunnel alignments frequently feature complex horizontal curves and vertical gradients, a major challenge lay in ensuring that any extracted 2D analytical section was oriented perfectly perpendicular to the local direction of excavation. Traditional CAD-based methods often rely on simplified, straight line section cuts, which can introduce severe geometric skewing and violate the fundamental plane strain assumptions required for 2D numerical simulations.

To solve this spatial problem, the script was programmed to dynamically compute a local coordinate system at any given parameter along the alignment curve. By executing vector cross-product calculations using the local forward tangent vector and the global vertical axis inside custom Python nodes, the script successfully established a perfectly orthogonal coordinate system at each desired chainage. This geometric orientation was then mapped directly to the Revit section generator, establishing the spatial boundaries of each analytical section in strict compliance with the ten-times-diameter boundary rule. In finite element analysis, the artificial boundaries of a model can introduce unwanted stiffness if placed too close to the zone of excavation, skewing settlement and stress computations. By incorporating this rule directly into the Dynamo boundary-box code block, the script automatically sized each section to extend ten

times the tunnel diameter from the center of excavation, guaranteeing that far-field stress boundaries were placed sufficiently far away to eliminate boundary-induced errors.

The second research objective addressed the critical challenge of interoperability and data mapping between BIM authoring platforms and numerical analysis engines. Although standard neutral exchange file formats like IFC are designed to serve as universal bridges, this research confirmed that moving raw, native BIM models directly into geotechnical analysis tools is highly unstable. Geotechnical IFC files imported into Revit are often processed as complex, solid elements or polymeshes. When these shapes are natively exported to 2D exchange formats, the resulting geometries consist of thousands of fragmented, disconnected line segments, overlapping boundaries, and unjoined blocks. Because finite element meshes require mathematically closed, non-overlapping polylines to accurately define material domains, directly importing raw BIM geometry into RS2 causes fatal meshing failures. To resolve this bottleneck without resorting to tedious manual drafting, AutoCAD Civil 3D was introduced as a scripted intermediate processing platform. Within this environment, a dedicated Dynamo automation script was deployed to clean and standardize the raw CAD geometry. This script utilized custom Python nodes to access the AutoCAD Database Services API, executing a series of systematic CAD commands to explode fragmented blocks, generate analytical polylines from visual soil hatches, and automatically define the model's external perimeter. Additionally, the script resolved the issue of semantic data loss during standard file transfers. While standard exchange formats strip away parametric BIM attributes, they successfully preserve the visual RGB color codes assigned to the soil strata within the Revit View Template. By using these color codes as visual fingerprints, the Python script successfully queried each layer's RGB values and renamed the generic CAD layers to match the standardized geomechanically classifications of the geotechnical profile, such as IXd3 or Ia. This automation completed the data chain, providing a topologically sound file that could be directly imported and meshed in RS2 without manual geometry reconstruction.

The third research objective sought to validate the proposed framework by applying it to a real-world infrastructure case study provided by TYLin, based on the mined tunnel project. The tunnel alignment is characterized by highly irregular, alternating layers of sandstones and shales, together with overlying colluvial landslide deposits. To validate the pipeline, the automated Dynamo script was executed across varying chainages, demonstrating its high robustness and adaptability to changing topographies. To optimize performance, the script utilized conditional logic, continuously monitoring a zone of influence of three times the tunnel diameter around the alignment. By checking for geological transitions along this boundary, the algorithm bypassed uniform ground zones and only generated new section views when meaningful spatial variations were detected, preventing data overload. The cross-section at the 635.0-meter chainage was selected for a complete finite element analysis inside RS2, where geomechanical parameters for each layer were assigned in the material database using the Generalized Hoek-Brown failure criterion. By using the load relaxation method calculated in RocSupport, the 14-stage plane strain model successfully simulated the 3D stress-relaxation and deformation response of the ground during sequential excavation. Ultimately, the successful execution of this simulation serves as the final validation of the developed framework, demonstrating that replacing slow, manual, and error-prone geometric recreation

with an automated, forward-feeding pipeline successfully bypasses traditional interoperability barriers, providing a scalable and highly efficient workflow for modern underground design.

Moving forward, several key research directions have been identified to expand the current unidirectional design pipeline and progress toward a mature, dynamic Digital Twin. A primary area of future development involves the integration of active structural health monitoring directly into a digital twin workflow. Rather than relying on static structural models that remain unadjusted during construction, the framework should integrate continuous physical datasets. This can be achieved by embedding distributed fiber optic sensors along the concrete linings to monitor changes in strain, structural temperature, and local micro-deformations. Concurrently, surface displacement data from radar interferometry can be captured to track geotechnical and slope stability. By utilizing Python scripts to perform automated back-analyses on this live data, any detected deviation from expected structural behaviours can automatically trigger updates to the soil and lining parameters within the finite element model, establishing a real-time predictive safety system.

Another highly promising future development is the integration of geothermal activation, turning conventional transport infrastructure into clean, low-enthalpy energy systems. By thermally activating concrete tunnel linings through embedded absorber pipes, shallow geothermal energy can be extracted to heat and cool nearby buildings, reducing carbon emissions in urban zones. To support this transition, the developed BIM-to-FEM pipeline should be expanded to accommodate coupled thermo-hydro-mechanical numerical modeling. Programmatically transferring pipe layouts, fluid flow rates, and thermal properties from Revit to advanced numerical solvers would allow engineers to evaluate structural thermal yields and assess the structural integrity of concrete linings under cyclic thermal loads.

Furthermore, future research should incorporate time-dependent geomechanical behaviours, such as soil swelling and consolidation, which pose severe long-term stability challenges in over consolidated clay shales. The rapid stress unloading caused by tunnel excavation can induce significant swelling pressures over time, leading to delayed structural damage. Future scripting routines could incorporate empirical swelling laws directly into the material database of the digital model, allowing the system to run transient hydraulic-mechanical simulations to predict long-term lining pressures and support convergence, facilitating the design of optimized, yield-control supports rather than overly conservative structures.

Additionally, the framework can be expanded to incorporate stochastic modelling techniques to capture highly heterogeneous geological conditions, such as block-in-matrix rocks. These complex formations consist of strong rock blocks randomly distributed within a weak soil matrix, making traditional deterministic modelling unreliable. By writing custom Python scripts to generate randomized block arrangements within the BIM environment based on borehole statistics, engineers could automate probabilistic finite element analyses, providing a powerful tool to evaluate complex structural configurations, central rock pillars, and twin-tunnel interactions under variable geomechanical conditions.

Finally, the integration of deep-learning computer vision models directly into the Digital Twin database represents a highly practical avenue for automated tunnel inspection and structural

lifecycle management. By converting high-definition mobile laser scans of the concrete lining into two-dimensional images, segmentation models can automatically identify, locate, and measure structural defects like cracks and water leakage. These defect geometries can then be programmatically mapped back to their corresponding elements in the Revit model, enabling the finite element solver to automatically degrade local concrete material stiffness inside the mesh to simulate progressive structural decay and support proactive maintenance planning.

References

- Erhardter, G., Weil, J., Bacher, L., Heil, F., & Kompolschek, P. (2023). Building information modelling based ground modelling for tunnel projects – Tunnel Angath/Austria. *Tunnelling and Underground Space Technology*.
- Al Furjani, A., Younsi, Z., Abdulalli, A., Elsaeh4, M., Almahdi5,, A., Jouili, K., & Lashihar, S. (2020). Enabling the City Information Modeling CIM for Urban Planning with OpenStreetMap OSM. *The Fourth International Conference for Geospatial Technologies – Libya GeoTec 4*. Tripoli, Libya.
- Alaei, A. (2023). *Towards BIM implementation for Geotechnical projects*. Jönköping University, School of Engineering.
- Alsahly, A., Hegemann, F., König, M., & Meschke, G. (2020). Integrated BIM-to-FEM approach in mechanised tunnelling. *Geomechanics and Tunnelling*.
- Babanagar, N., Sheil, B., Nini’c, J., Zhang, Q., & Hardy, S. (2024). Digital twins for urban underground space. *Tunnelling and Underground Space Technology incorporating Trenchless Technology Research*.
- Bestyński, Z., Pacanowski, G., & Sieinski, E. (2017). Geophysical investigation and geotechnical classifications for stability assessment of Carpathian flysch slopes. *Przegląd Geologiczny*.
- Colajanni, P., Inzerillo, L., Pisciotta, A., Acuto, F., Mantalovas, K., & Di Mino, G. (2025). Architectural and Structural Interoperability in the BIM Design Workflow. *Buildings*.
- Colajanni, P., Inzerillo, L., Pisciotta, A., Acuto, F., Mantalovas, K., & Di Mino, G. (2025). Architectural and Structural Interoperability in the BIM DesignWorkflow. *buildings*.

- DartisTech. (2026). *Best Geotechnical Engineering Software Tools for 2026*. Retrieved from DartisTech: <https://dartistech.com/best-geotechnical-engineering-software-tools-for-2025/>
- Fabozzi, S., Biancardo, S., Veropalumbo, R., & Bilotta, E. (2020). I-BIM based approach for geotechnical and numerical modelling of a conventional tunnel excavation. *Tunnelling and Underground Space Technology incorporating Trenchless Technology Research*.
- Fahmy, A. (2025). *Deciphering Python Node in Dynamo for Revit v1.00*.
- Fonsati, A., Cosentini, R., Tundo, C., & Osello, A. (2023). From Geotechnical Data to GeoBIM Models: Testing Strategies. *Buildings*.
- HS2. (2020). *Automation of soil-structure interaction finite element modelling using Python coding and the project BIM model*. Major Projects Association.
- Huang, H., Ruan, B., Wu, X., & Qin, Y. (2024). Parameterized modeling and safety simulation of shield tunnel based on BIM-FEM automation framework. *Automation in Construction*.
- Kang, K., Huang, S., Liu, W., Cheng, H., Fomenko, I., & Zhou, Y. (2022). Sandstone Slope Stability Analysis Under Wetting-Drying Cycles Based on Generalized Hoek-Brown Failure Criterion. *Frontiers in Earth Science*.
- Kullvén, F., & Nyberg, K. (2014). *Possibilities with BIM in relation to cost estimation and scheduling*. Chalmers University of Technology.
- Lai, D. a. (2023). Interface impact assessment using BIM and Leapfrog on Sydney Metro West. In *Expanding Underground-Knowledge and Passion to Make a Positive Impact on the World* (pp. 2766--2774). CRC Press.
- LeeLee, M., Linn Lee, Y., Liang Goh, S., Hoon Koo, C., Hung Lau, S., & YahChong, S. (2021). Case Studies and Challenges of Implementing Geotechnical Building Information Modelling in Malaysia. *Infrastructures*.
- Li, F., Si, P., He, Y., Wang, H., Zhang, Z., & Zhao, S. (2024). Numerical Analysis of the Single-Directionally Misaligned Segment Behavior of Hydraulic TBM Tunnel. *Buildings*.
- Malata, T. (2008). Development of Polish Flysch Carpathians revealed in outcrops and landscape. *Przegląd Geologiczny*.

- Marmo, R., Polverino, F., Nicoletta, M., & Tibaut, A. (2020). Building performance and maintenance information model based on IFC schema. *Automation in Construction*.
- Menaka, K., & Sivakumar Babu, G. (2025). Building information modelling in geotechnical engineering. In A. Juneja, A. Joseph, & D. Murty, *GeoVadis: The Future of Geotechnical Engineering (Volume 2)* (pp. 229-234). London: CRC Press.
- Menaka, K., & Sivakumar Babu, G. (2025). Building Information Modelling in Geotechnical Engineering: A Case Study of Deep Excavation in South India. *Indian Geotechnical Journal*.
- Nguyen, T., & Adhikari, S. (2023). The Role of BIM in Integrating Digital Twin in Building Construction: A Literature Review. *Sustainability*.
- Nugraha, B., Iarasati, P., & Dũng, D. (2025). Assessing The Interoperability And Semantic Readiness Of BIM And IFC Data For AI Integration In The Architecture, Engineering, And Construction Industry: A Systematic Review. *International Journal of Intelligent Data and Machine Learning*, 14-24.
- Rajchel, J., & Myszkowska, J. (1998). Exotic clasts of organodetrritic algal limestones from lithosomes of the Babica Clay, Skole Unit (Outer Flysch Carpathians, Poland). *Annales Societatis Geologorum Poloniae*.
- Shi, C., Jin, Y., Lu, H., & Shi, J. (2023). A BIM-based framework for automatic numerical modelling and geotechnical analysis of a large-scale deep excavation for transportation infrastructures. *Intelligent Transportation Infrastructure*.
- Singh, M., Fuenmayor, E., Hinchy, E., Qiao, Y., Murray, M., & Devine, D. (2021). Digital Twin: Origin to Future. *Applied System Innovation*, 36.
- Speakman, H. (2022). A mixed-method approach for major excavation ground support evaluation. *Caving 2022: Proceedings of the Fifth International Conference on Block and Sublevel Caving* (pp. 279-288). Perth: Australian Centre for Geomechanics.
- Szwarkowski, D., & Pilecka, E. (2017). BIM TECHNOLOGY IN GEOTECHNICAL ENGINEERING IN TERMS OF IMPACT HIGH BUILDING “MOGILSKA TOWER” IN CRACOW OF EXISTING BUILDING DEVELOPMENT. *Technical Sciences*.
- Talebi, A., Potenza, F., & Gattulli, V. (2023). Interoperability between BIM and FEM for vibration-based model updating of a pedestrian bridge. *Structures*, 1092-1107.

- Wang, Y., Rezaei, A., & Hicks, S. (2025). Digital twin applications in geotechnical engineering: a systematic review. *Machine Learning and Data Science in Geotechnics*, 78 - 92.
- Wu, J., Chen, J., Chen, G., Wu, Z., Zhong, Y., Chen, B., . . . Huang, J. (2021). Development of Data Integration and Sharing for Geotechnical Engineering Information Modeling Based on IFC. *Hindawi*.
- Zobl, F., CHMELINA, K., FABER, R., KOOIJMAN, J., MARSCHALLINGER, R., & STOTER, J. (2011). Multidimensional aspects of GeoBIM data: new standards needed. *Mathematical Geosciences at the Crossroads of Theory and Practice*. Salzburg: University of Salzburg.

Appendix A

Interoperability framework source codes

A.1 Contents

The present Appendix collects the complete, intact source codes developed for the automated computational workflow described in Chapter 4. While the main text of the thesis highlights specific logic blocks to explain the methodology, this section provides the full scripts to ensure transparency and allow for the complete reproducibility of the data pipeline.

The codes are organized sequentially, reflecting the three primary stages of the interoperability framework: 3D geometric extraction, 2D boundary optimization, and finite element model initialization. In **Error! Reference source not found.**, a complete list of the developed automation scripts is reported.

Script Name	Software Environment	Primary Function
Extraction Script	Autodesk Revit (Dynamo/Python)	Automated slicing of 3D BIM models and 2D cross-section generation at critical chainages.
Optimization Script	AutoCAD Civil 3D (Dynamo/Python)	Cleaning fragmented CAD linework and generating mathematically closed boundary polylines.
Automated RS2 Setup Script	Rocscience RS2 (Python API)	Programmatic injection of Generalized Hoek-Brown parameters and composite liner configurations.

Table A.1 Summary of automation scripts developed for the workflow

A.2 Revit and Dynamo extraction script

This section contains the Dynamo nodes and Python code executed within the Dynamo visual programming environment in Revit to establish the initial Open-BIM baseline and extract the geometric cross-sections.

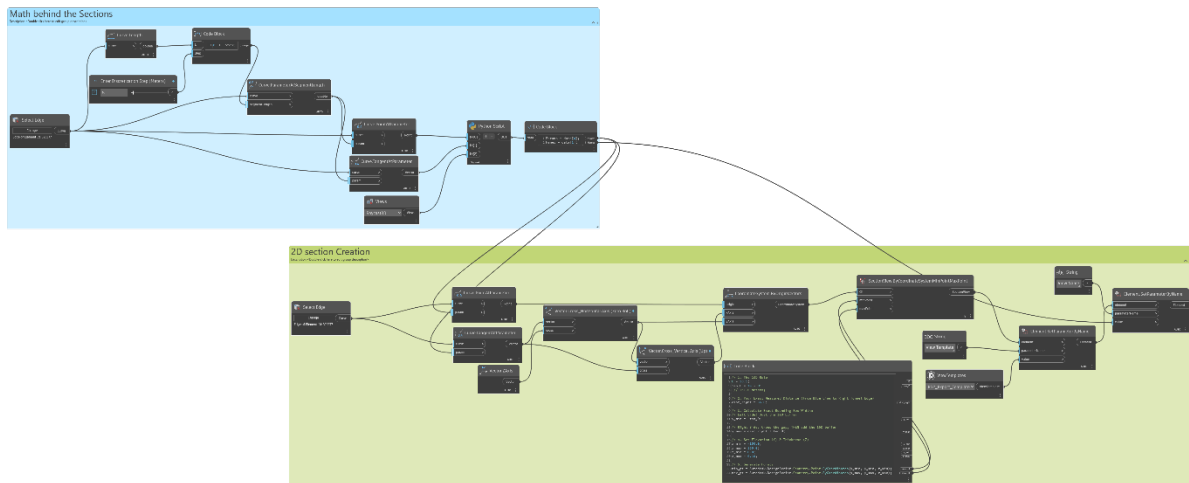


Figure A.1 The complete Dynamo visual programming graph utilized for automated geometric extraction and cross-section generation within the Revit environment.

Figure A.2. Python source code executed within the Revit Dynamo environment

```

1 import clr
2 import math
3
4 # Import RevitAPI
5 clr.AddReference('RevitAPI')
6 from Autodesk.Revit.DB import *
7
8 # Import DocumentManager
9 clr.AddReference('RevitServices')
10 import RevitServices
11 from RevitServices.Persistence import DocumentManager
12
13 # Import Dynamo Geometry Conversion
14 clr.AddReference("RevitNodes")
15 import Revit
16 clr.ImportExtensions(Revit.GeometryConversion)
17
18 doc = DocumentManager.Instance.CurrentDBDocument
19
20 # --- INPUTS ---
21 points = IN[0]
22 tangents = IN[1]
23 view = UnwrapElement(IN[2])
24
25 num_points = len(points)
26 parameters = [float(i) / (num_points - 1) for i in range(num_points)] if num_points > 1 else [0.0]
27
28 category_filter = ElementCategoryFilter(BuiltInCategory.OST_GenericModel)
29 intersector = ReferenceIntersector(category_filter, FindReferenceTarget.Element, view)
30 intersector.FindReferencesInRevitLinks = True
31
32 def get_ifc_name(hit):
33     if hit == None:
34         return "Unknown"
35     ref = hit.GetReference()
36     link_doc = doc.GetElement(ref.ElementId).GetLinkDocument()
37     if link_doc:
38         hit_element = link_doc.GetElement(ref.LinkedElementId)
39         param = hit_element.LookupParameter("IfcName")
40         if param and param.HasValue:
41             return param.AsString()
42     return "Unknown"
43
44 ignore_list = ["OTD ", "OTD LocalPreConfinement PermanentWall", "OTD LiningsParts"]
45
46 # --- THE GEOLOGICAL RADAR SETTINGS ---
47 laser_distance_ft = 50.0 * 3.28084
48 number_of_lasers = 12
49 dist_to_roof_ft = 7.0 * 3.28084
50
51 depths = []
52 soil_profiles = []
53
54 # --- THE FIX: Calculate chainage natively in meters ---
55 distances_m = [0.0]
56 for i in range(1, len(points)):
57     # Removed the * 0.3048 because Dynamo points are already in metric!
58     dist = points[i].DistanceTo(points[i-1])
59     distances_m.append(distances_m[i-1] + dist)
60
61 # --- PASS 1: SCAN THE CORRIDOR ---
62 for pt, tangent in zip(points, tangents):
63     revit_pt = pt.ToXYZ()
64     forward = tangent.ToXYZ().Normalize()
65     world_up = XYZ.BasisZ
66
67     # 1. Depth Check (For Max/Min Cover)
68     hits_up = intersector.Find(revit_pt, world_up)
69     z_values = [h.GetReference().GlobalPoint.Z for h in hits_up]
70
71     if len(z_values) > 0:
72         depths.append((max(z_values) - revit_pt.Z) - dist_to_roof_ft)
73     else:
74         depths.append(-1.0)
75
76     # 2. The 50m Continuous Radar
77     if abs(forward.Z) > 0.99:
78         right = XYZ.BasisX
79     else:
80         right = forward.CrossProduct(world_up).Normalize()
81
82     local_up = right.CrossProduct(forward).Normalize()
83
84     current_step_materials = set()
85
86     for i in range(number_of_lasers):
87         angle = (2 * math.pi / number_of_lasers) * i
88         dir_vec = right.Multiply(math.cos(angle)).Add(local_up.Multiply(math.sin(angle)))

```

```

90     hits = intersector.Find(revit_pt, dir_vec)
91
92     for h in hits:
93         hit_pt = h.GetReference().GlobalPoint
94         dist = hit_pt.DistanceTo(revit_pt)
95
96         if dist <= laser_distance_ft:
97             mat_name = get_ifc_name(h)
98
99             if not any(ignored in mat_name for ignored in ignore_list) and mat_name != "Unknown":
100                 current_step_materials.add(mat_name)
101
102     if len(current_step_materials) == 0:
103         current_step_materials.add("Uniform Geology")
104
105     soil_profiles.append(tuple(sorted(list(current_step_materials))))
106
107
108     # --- PASS 2: FILTER THE CRITICAL SECTIONS ---
109     critical_params = []
110     view_names = []
111
112     # 1. Extract Maximum and Minimum Cover Locations
113     valid_depths = [d for d in depths if d > 0]
114     if valid_depths:
115         max_d = max(valid_depths)
116         min_d = min(valid_depths)
117
118         max_idx = depths.index(max_d)
119         min_idx = depths.index(min_d)
120
121         critical_params.append(parameters[max_idx])
122         view_names.append("Geo - Maximum @ " + str(round(distances_m[max_idx], 1)) + "m")
123
124         critical_params.append(parameters[min_idx])
125         view_names.append("Geo - Minimum @ " + str(round(distances_m[min_idx], 1)) + "m")
126
127     # 2. Extract All Geological Changes within the 50m Bubble
128     counter = 1
129     for i in range(1, len(soil_profiles)):
130         prev_profile = soil_profiles[i-1]
131         curr_profile = soil_profiles[i]
132
133         if prev_profile != curr_profile:
134             name = "Geo - Section " + str(counter) + " @ " + str(round(distances_m[i], 1)) + "m"
135
136             if parameters[i] not in critical_params:
137                 critical_params.append(parameters[i])
138                 view_names.append(name)
139             counter += 1
140
141     # --- OUTPUT ---
142     OUT = [critical_params, view_names]

```

A.3 Civil 3D geometric optimization script

This section details the second phase of the interoperability framework, executed within the AutoCAD Civil 3D environment.

Figure A.3. External boundary script in Dynamo

```

1 import clr
2
3 clr.AddReference('acmgd')
4 clr.AddReference('acdbmgd')
5 from Autodesk.AutoCAD.ApplicationServices import Application
6
7 doc = Application.DocumentManager.MdiActiveDocument
8
9 master_lisp = r"""(progn
10 (vl-load-com)
11 (command)
12
13 (setq oldOsnap (getvar "OSMODE"))
14 (setvar "OSMODE" 0)
15 (setvar "CMDECHO" 0)
16 (setvar "QAFLAGS" 1)
17
18 ;; =====
19 ;; 1. DOUBLE EXPLODE (Exposes all hatches)
20 ;; =====
21 (if (setq ss (ssget "_X" '((0 . "INSERT")))) (command "._EXPLODE" ss ""))
22 (if (setq ss (ssget "_X" '((0 . "INSERT")))) (command "._EXPLODE" ss ""))
23 (setvar "QAFLAGS" 0)
24
25 ;; =====
26 ;; 2. EXTERNAL BOUNDARY (Shrinkwrap all hatches)
27 ;; =====
28 (if (not (tblsearch "LAYER" "External"))
29 (entmake (list '(0 . "LAYER") '(100 . "AcDbSymbolTableRecord") '(100 . "AcDbLayerTableRecord") '(2 . "External") '(70 . 0) '(62 . 6))))
30 )
31 (setvar "CLAYER" "External")
32
33 (if (setq ssHatch (ssget "_X" '((0 . "HATCH"))))
34 (progn
35 (command "._LINEWORKSHRINKWRAP" ssHatch "")
36 (while (> (getvar "CMDACTIVE") 0) (command ""))
37 )
38 )
39
40 ;; Delete any smaller duplicate polygons created by fragmented soil
41 (if (setq ssExt (ssget "_X" '((8 . "External") (0 . "LWPOLYLINE,POLYLINE,LINE,ARC,CIRCLE"))))
42 (progn
43 (setq i 0 maxLen -1.0 mainEnt nil)
44 (while (< i (sslength ssExt))
45 (setq ent (ssname ssExt i))
46 (setq obj (vlax-ename->vla-object ent))
47 (setq currlen (vlax-curve-getDistAtParam obj (vlax-curve-getEndParam obj)))
48 (if (and currlen (> currlen maxLen))
49 (progn (setq maxLen currlen mainEnt ent))
50 )
51 (setq i (1+ i))
52 )
53 (setq i 0)
54 (while (< i (sslength ssExt))
55 (setq ent (ssname ssExt i))
56 (if (not (equal ent mainEnt))
57 (entdel ent)
58 )
59 (setq i (1+ i))
60 )
61 )
62 )
63
64 ;; =====
65 ;; 3. GENERATE ALL INTERNAL BOUNDARIES
66 ;; =====
67 (if (setq ssHatch (ssget "_X" '((0 . "HATCH"))))
68 (progn
69 (setq i 0)
70 (while (< i (sslength ssHatch))
71 (setq ent (ssname ssHatch i))
72 (setq layer (cdr (assoc 8 (entget ent))))
73
74 (if (tblsearch "LAYER" layer) (setvar "CLAYER" layer))
75 (command "._HATCHGENERATEBOUNDARY" ent "")
76
77 (setq i (1+ i))
78 )
79 ;; Push colored hatches to the back
80 (command "._DRAWORDER" ssHatch "" "_B")
81 )
82 )
83
84 ;; =====
85 ;; 3.5 THE "AUTO-DELETER" (Mimics your manual fix!)
86 ;; =====
87 (setq trueOuterRings nil)
88
89 ;; Step A: Find the true tunnels on Soil-1, Tunnel, or Soil-TL
90 (if (setq ssT1 (ssget "_X" '((8 . "Soil-1,Tunnel*,Soil-TL") (0 . "LWPOLYLINE,POLYLINE,CIRCLE,ARC"))))
91 (progn
92 (setq clusters nil)
93 (setq i 0)
94 ;; Group the lines by location so the two tunnels don't confuse each other
95 (while (< i (sslength ssT1))
96 (setq e (ssname ssT1 i))

```

```

97 (vla-getboundingbox (vla-ename->vla-object e) 'minPt 'maxPt)
98 (setq p1 (vlax-safearray->list minPt) p2 (vlax-safearray->list maxPt))
99 (setq cx (+ (car p1) (/ (- (car p2) (car p1)) 2.0)))
100 (setq cy (+ (cadr p1) (/ (- (cadr p2) (cadr p1)) 2.0)))
101
102 (setq added nil)
103 (setq newClusters nil)
104 (foreach c clusters
105   (setq ccx (car (car c)) ccy (cadr (car c)))
106   (if (< (distance (list cx cy) (list ccx ccy)) 5.0)
107     (progn
108       (setq added T)
109       (setq newClusters (cons (list (car c) (cons e (cadr c))) newClusters))
110     )
111     (setq newClusters (cons c newClusters))
112   )
113 )
114 (if (not added)
115   (setq newClusters (cons (list (list cx cy) (list e)) newClusters))
116 )
117 (setq clusters newClusters)
118 (setq i (1+ i))
119 )
120
121 ;; Step B: For each tunnel, keep the LONGEST line (the outer ring) and delete the inner linings
122 (foreach c clusters
123   (setq ents (cadr c))
124   (setq maxlen -1.0 bestEnt nil)
125   (foreach e ents
126     (setq obj (vla-ename->vla-object e))
127     (setq l (vlax-curve-getDistAtParam obj (vlax-curve-getEndParam obj)))
128     (if (> l maxlen) (setq maxlen l bestEnt e))
129   )
130   (if bestEnt
131     (progn
132       (setq trueOuterRings (cons bestEnt trueOuterRings))
133       (foreach e ents
134         (if (not (equal e bestEnt)) (entdel e))
135       )
136     )
137   )
138 )
139 )
140
141 ;; Step C: VAPORIZE THE LAYERS ABOVE IT! (Deletes overlapping Soil-4 / Soil holes)
142 (if trueOuterRings
143   (progn
144     (setq ssAll (ssget "_X" '((0 . "LWPOLYLINE,POLYLINE,CIRCLE,ARC"))))
145     (if ssAll
146       (progn
147         (setq i 0)
148         (while (< i (sslength ssAll))
149           (setq e (ssname ssAll i))
150           (setq isTrueRing nil)
151           (foreach tr trueOuterRings (if (equal e tr) (setq isTrueRing T)))
152         )
153         ;; If this line isn't our protected Tunnel Ring, check if it's sitting on top of one
154         (if (not isTrueRing)
155           (progn
156             (vla-getboundingbox (vla-ename->vla-object e) 'minP 'maxP)
157             (setq p1 (vlax-safearray->list minP) p2 (vlax-safearray->list maxP))
158             (setq ecx (+ (car p1) (/ (- (car p2) (car p1)) 2.0)))
159             (setq ecy (+ (cadr p1) (/ (- (cadr p2) (cadr p1)) 2.0)))
160             (setq ew (- (car p2) (car p1)))
161             (setq eh (- (cadr p2) (cadr p1)))
162
163             (setq shouldDelete nil)
164             (foreach tr trueOuterRings
165               (vla-getboundingbox (vla-ename->vla-object tr) 'tminP 'tmaxP)
166               (setq tp1 (vlax-safearray->list tminP) tp2 (vlax-safearray->list tmaxP))
167               (setq tcx (+ (car tp1) (/ (- (car tp2) (car tp1)) 2.0)))
168               (setq tcy (+ (cadr tp1) (/ (- (cadr tp2) (cadr tp1)) 2.0)))
169               (setq tw (- (car tp2) (car tp1)))
170               (setq th (- (cadr tp2) (cadr tp1)))
171
172               ;; If it matches the exact location and size of our tunnel... Delete it!
173               (if (and (< (distance (list ecx ecy) (list tcx tcy)) 2.0)
174                 (< (abs (- ew tw)) 2.0)
175                 (< (abs (- eh th)) 2.0))
176                 (setq shouldDelete T)
177             )
178             (if shouldDelete (entdel e))
179           )
180         )
181       )
182     )
183     (setq i (1+ i))
184   )
185 )
186 )
187 )
188 )
189 )

```

```

190
191 ;; =====
192 ;; 4. JOIN EVERYTHING FOR RS2
193 ;; =====
194 (setq targetLayers nil)
195 (setq layItem (tblnext "LAYER" T))
196 (while layItem
197   (setq layName (cdr (assoc 2 layItem)))
198   (if (wcmatch (strcase layName) "SOIL-*,TUNNEL*,EXTERNAL")
199     (setq targetLayers (cons layName targetLayers))
200   )
201   (setq layItem (tblnext "LAYER"))
202 )
203
204 (foreach layName targetLayers
205   (if (setq ss (ssget "_X" (list (cons 8 layName) '(0 . "LINE,LWPOLYLINE,ARC"))))
206     (progn
207       (command "._PEDIT" "_M" ss "" "_J" "0.001")
208       (while (> (getvar "CMDACTIVE") 0) (command ""))
209     )
210   )
211 )
212
213 ;; =====
214 ;; 5. FINAL OVERKILL
215 ;; =====
216 (if (setq ss (ssget "_X" '((0 . "LINE,LWPOLYLINE,ARC,CIRCLE"))))
217   (command "._OVERKILL" ss "" "Tolerance" "0.001" "Done")
218 )
219 (command)
220 (setvar "OSMODE" oldOsnap)
221 (setvar "CMDECHO" 1)
222 (princ)
223 )
224 ""
225
226 doc.SendStringToExecute(master_lisp + "\n", True, False, False)
227
228 OUT = "Execution complete. The Auto-Deleter has successfully vaporized the overlapping soil holes!"

```

Figure A.4. Layer naming in Civil 3D

```

1  import clr
2
3  # Add Assemblies for AutoCAD and Civil3D
4  clr.AddReference('AcMgd')
5  clr.AddReference('AcCoreMgd')
6  clr.AddReference('AcDbMgd')
7
8  # Fixed import syntax for CPython3 compatibility
9  from Autodesk.AutoCAD.ApplicationServices import Application as acapp
10 from Autodesk.AutoCAD.DatabaseServices import *
11 from Autodesk.AutoCAD.Colors import *
12
13 # Dictionary mapping RGB tuples to target layer names
14 color_map = {
15     (191, 191, 191): "Ia",
16     (139, 105, 20): "Ib",
17     (0, 128, 0): "IXb1",
18     (82, 232, 255): "IXb2",
19     (245, 238, 140): "IXd3",
20     (165, 76, 255): "VIIIId1",
21     (255, 0, 0): "Tunnel"
22 }
23
24 adoc = acapp.DocumentManager.MdiActiveDocument
25 db = adoc.Database
26
27 renamed_layers = []
28
29 with adoc.LockDocument():
30     with adoc.Database.TransactionManager.StartTransaction() as t:
31         # Get the layer table
32         lt = t.GetObject(db.LayerTableId, OpenMode.ForRead)
33
34         # Get a list of existing layer names to avoid conflict errors
35         existing_names = []
36         for ltr_id in lt:
37             ltr = t.GetObject(ltr_id, OpenMode.ForRead)
38             existing_names.append(ltr.Name.lower())
39
40         # Iterate through layers and rename based on color
41         for ltr_id in lt:
42             ltr = t.GetObject(ltr_id, OpenMode.ForWrite)
43
44             # Protect default layers
45             if ltr.Name == "0" or ltr.Name.lower() == "defpoints":
46                 continue
47
48             # Extract RGB values from the layer
49             color = ltr.Color
50             rgb = (color.Red, color.Green, color.Blue)
51
52             # Check if the color matches our dictionary
53             if rgb in color_map:
54                 target_name = color_map[rgb]
55
56                 # Rename if the name isn't already correct and doesn't already exist in the drawing
57                 if ltr.Name != target_name and target_name.lower() not in existing_names:
58                     old_name = ltr.Name
59                     ltr.Name = target_name
60                     existing_names.append(target_name.lower()) # Update list to prevent duplicates
61                     renamed_layers.append("Renamed '{}' to '{}'".format(old_name, target_name))
62
63         t.Commit()
64
65 # Output the results so you can see what changed
66 if renamed_layers:
67     OUT = renamed_layers
68 else:
69     OUT = "No layers needed renaming or matched the exact RGB values."

```

Figure A.5. Legend in Civil3D

```

1  import clr
2
3  # Add Assemblies for AutoCAD and Civil3D
4  clr.AddReference('AcMgd')
5  clr.AddReference('AcCoreMgd')
6  clr.AddReference('AcDbMgd')
7
8  # CPython3 syntax for application services
9  from Autodesk.AutoCAD.ApplicationServices import Application as acapp
10 from Autodesk.AutoCAD.DatabaseServices import *
11 from Autodesk.AutoCAD.Geometry import *
12 from Autodesk.AutoCAD.Colors import *
13
14 adoc = acapp.DocumentManager.MdiActiveDocument
15 db = adoc.Database
16
17 # Master dictionary of RGB colors and corresponding base names
18 color_map = {
19     (191, 191, 191): "Ia",
20     (139, 105, 20): "Ib",
21     (0, 128, 0): "IXb1",
22     (82, 232, 255): "IXb2",
23     (245, 238, 140): "IXd3",
24     (165, 76, 255): "VIIId1",
25     (255, 0, 0): "Tunnel"
26 }
27
28 with adoc.LockDocument():
29     with adoc.Database.TransactionManager.StartTransaction() as t:
30
31         # --- NEW LOGIC: Scan the drawing for available layers ---
32         lt = t.GetObject(db.LayerTableId, OpenMode.ForRead)
33         found_colors = []
34
35         # Check every layer in the drawing
36         for ltr_id in lt:
37             ltr = t.GetObject(ltr_id, OpenMode.ForRead)
38             color = ltr.Color
39             rgb = (color.Red, color.Green, color.Blue)
40             # If the layer's color matches our master list, mark it as found
41             if rgb in color_map:
42                 found_colors.append(rgb)
43
44         # Filter the master list to ONLY include the colors we found, keeping the original order
45         active_legend_items = {rgb: name for rgb, name in color_map.items() if rgb in found_colors}
46
47         # If no matching layers are found, stop the script safely
48         if not active_legend_items:
49             OUT = "No matching layers found in the drawing to build a legend."
50         else:
51             btr = t.GetObject(db.CurrentSpaceId, OpenMode.ForWrite)
52
53             # Calculate Automatic Placement and Size based on Drawing Extents
54             db.UpdateExt(True)
55             ext_min = db.Extmin
56             ext_max = db.Extmax
57
58             width = ext_max.X - ext_min.X
59             height = ext_max.Y - ext_min.Y
60
61             # Fallback if extents are invalid
62             if width <= 0 or height <= 0 or width > 1e10:
63                 ins_pt = Point3d(0, 0, 0)
64                 txt_ht = 5.0
65             else:
66                 # Place legend to the right of the model's top-right corner with a 5% gap
67                 gap = width * 0.05
68                 ins_pt = Point3d(ext_max.X + gap, ext_max.Y, 0)
69
70                 # Scale text height proportionally to the model's total height (2.5%)
71                 txt_ht = height * 0.025
72
73             # Draw "LEGEND" Title
74             title = MText()
75             title.Contents = "LEGEND:"
76             title.Location = ins_pt
77             title.TextHeight = txt_ht * 1.2 # Make title slightly larger than normal text
78             btr.AppendEntity(title)
79             t.AddNewlyCreatedDBObject(title, True)
80
81             # Calculate spacing and layout metrics based on the auto-generated text height
82             current_y = ins_pt.Y - (txt_ht * 2.5)
83             swatch_len = txt_ht * 3.0
84             gap_txt = txt_ht * 1.0
85
86             # Loop through ONLY the active items to draw swatches and text
87             for rgb, name in active_legend_items.items():
88
89                 # --- Draw Color Swatch (Thick Polyline) ---
90                 pline = Polyline()
91                 pline.AddVertexAt(0, Point2d(ins_pt.X, current_y), 0, 0, 0)
92                 pline.AddVertexAt(1, Point2d(ins_pt.X + swatch_len, current_y), 0, 0, 0)
93                 pline.ConstantWidth = txt_ht * 0.8
94
95                 # Apply TrueColor to match exactly
96                 pline.Color = Color.FromRgb(rgb[0], rgb[1], rgb[2])

```

```

97      btr.AppendEntity(pline)
98      t.AddNewlyCreatedDBObject(pline, True)
99
100      # --- Draw Text ---
101      text = MText()
102
103      # Add "Soil-" prefix to everything except "Tunnel"
104      display_name = "Soil-" + name if name != "Tunnel" else name
105      text.Contents = display_name
106
107      # Position text to the right of the swatch, aligned vertically
108      text.Location = Point3d(ins_pt.X + swatch_len + gap_txt, current_y + (txt_ht * 0.4), 0)
109      text.TextHeight = txt_ht
110
111      btr.AppendEntity(text)
112      t.AddNewlyCreatedDBObject(text, True)
113
114      # Move down for the next row
115      current_y -= (txt_ht * 2.0)
116
117      t.Commit()
118
119      # Output the results
120      OUT = "Legend dynamically drawn with {} items at X: {:.2f}, Y: {:.2f}".format(len(active_legend_items), ins_pt.X, ins_pt.Y)
121

```

Figure A.6. Tunnel outer layer

```

1  import clr
2
3  clr.AddReference('acmgd')
4  clr.AddReference('acdbmgd')
5  from Autodesk.AutoCAD.ApplicationServices import Application
6
7  doc = Application.DocumentManager.MdiActiveDocument
8
9  isolate_and_explode_lisp = r"""(progn
10    (vl-load-com)
11    (setvar "CMDECHO" 0)
12
13    ;; 1. Find Red layers or "Tunnel" layers
14    (setq tunnelLayers "")
15    (setq layItem (tblnext "LAYER" T))
16    (while layItem
17      (setq layName (cdr (assoc 2 layItem)))
18      (setq layObj (entget (tblobjname "LAYER" layName)))
19      (setq layColor (cdr (assoc 62 layObj)))
20      (setq trueColor (cdr (assoc 420 layObj)))
21
22      (if (or (= layColor 1) (equal trueColor 16711680) (wcmatch (strcase layName) "*TUNNEL*"))
23        (if (= tunnelLayers "")
24          (setq tunnelLayers layName)
25          (setq tunnelLayers (strcat tunnelLayers "," layName))
26        )
27      )
28      (setq layItem (tblnext "LAYER"))
29    )
30    (if (= tunnelLayers "") (setq tunnelLayers "Tunnel,Soil-1"))
31
32    ;; 2. Select the tunnels and EXPLODE TWICE
33    (if (setq ssTunnels (ssget "_X" (list (cons 8 tunnelLayers))))
34      (progn
35        ;; QAFLAGS=1 allows exploding multiple objects at once
36        (setvar "QAFLAGS" 1)
37        (command "._EXPLODE" ssTunnels "")
38
39        ;; Grab whatever was just created and explode it one more time
40        (if (setq ssPrev (ssget "_P"))
41          (command "._EXPLODE" ssPrev "")
42        )
43        (setvar "QAFLAGS" 0)
44      )
45    )
46
47    ;; 3. JOIN the raw lines and arcs back into closed rings
48    (if (setq ssFragments (ssget "X" (list '(0 . "LINE,LWPOLYLINE,ARC") (cons 8 tunnelLayers))))

```

```

49 (progn
50   (command "_PEDIT" "M" ssFragments "" "J" "0.5")
51   (while (> (getvar "CMDDACTIVE") 0) (command "")))
52 )
53 )
54
55 ;; 4. Split Left/Right, Measure, and Delete Inner Rings
56 (if (setq ssJoined (ssget "_X" (list '(0 . "LWPOLYLINE,POLYLINE,CIRCLE") (cons 8 tunnelLayers))))
57   (progn
58     ;; Calculate the center line between the two tunnels
59     (setq i 0 avgX 0.0 count 0)
60     (while (< i (sslength ssJoined))
61       (setq e (ssname ssJoined i))
62       (vla-getboundingbox (vla-ename->vla-object e) 'minPt 'maxPt)
63       (setq cx (+ (car (vlax-safearray->list minPt)) (/ (- (car (vlax-safearray->list maxPt)) (car (vlax-safearray->list minPt))) 2.0)))
64       (setq avgX (+ avgX cx))
65       (setq count (1+ count))
66       (setq i (1+ i))
67     )
68
69     (if (> count 0)
70       (progn
71         (setq avgX (/ avgX count))
72
73         ;; Sort into Left and Right lists
74         (setq leftTunnels nil rightTunnels nil)
75         (setq i 0)
76         (while (< i (sslength ssJoined))
77           (setq e (ssname ssJoined i))
78           (vla-getboundingbox (vla-ename->vla-object e) 'minPt 'maxPt)
79           (setq cx (+ (car (vlax-safearray->list minPt)) (/ (- (car (vlax-safearray->list maxPt)) (car (vlax-safearray->list minPt))) 2.0)))
80           (if (< cx avgX)
81             (setq leftTunnels (cons e leftTunnels))
82             (setq rightTunnels (cons e rightTunnels))
83           )
84           (setq i (1+ i))
85         )
86
87         ;; The function that keeps the longest ring and deletes the rest
88         (defun processGroup (ents / maxLen bestEnt l obj)
89           (setq maxLen -1.0 bestEnt nil)
90           (foreach e ents
91             (setq obj (vla-ename->vla-object e))
92             (setq l (vla-curve-getDistAtParam obj (vla-curve-getEndParam obj)))
93             (if (> l maxLen) (setq maxLen l bestEnt e))
94           )
95           (if bestEnt
96             (foreach e ents
97               (if (not (equal e bestEnt)) (entdel e))
98             )
99           )
100         )
101
102         ;; Run the purge!
103         (if leftTunnels (processGroup leftTunnels))
104         (if rightTunnels (processGroup rightTunnels))
105       )
106     )
107   )
108   (princ "\nOuter Tunnel successfully isolated.")
109   (princ)
110 )
111 ""
112
113 doc.SendStringToExecute(isolate_and_explode_lisp + "\n", True, False, False)
114
115 OUT = "Final tunnel isolation complete: Exploded 2x, Joined, and Inner Rings Vaporized."
116

```

A.4 Automated finite element configuration script

This section details the final phase of the interoperability framework, executed within the Rocscience RS2 finite element Python Scripter. This initialization process was fully automated using the RS2 Python API. The source code presented below programmatically executes the complete automation sequence, mapping the required geomechanical parameters.

Figure A.7. The RS2 Python API automation script

```

1  import pandas as pd
2  import math
3  from rs2.modeler.RS2Modeler import RS2Modeler
4  from rs2.modeler.properties.PropertyEnums import *
5
6  # =====
7  # FILE PATHS
8  # =====
9  fez_path = r"P:\Section-375m.fez"
10 rs2_csv_path = r"P:\RS2_Automated_Workflow\RS2.csv"
11
12 # =====
13 # CONNECT TO RS2 MODELER
14 # =====
15 port_num = 60000
16 modeler = RS2Modeler(port=port_num)
17
18 try:
19     model = modeler.openFile(fez_path)
20     print(f"Successfully opened: {fez_path}")
21 except Exception as e:
22     print(f"Failed to open model. Error: {e}")
23
24 # =====
25 # PART 1: NUMERICAL INJECTION (RS2.csv)
26 # =====
27 print("\n--- Starting Material Property Injection ---")
28 df = pd.read_csv(rs2_csv_path)
29 materials = model.getAllMaterialProperties()
30
31 def get_val(row, col_name):
32     if col_name not in row: return None
33     val = row[col_name]
34     if pd.isna(val) or str(val).strip() == '': return None
35     return float(val)
36
37 for i, (index, row) in enumerate(df.iterrows()):
38     layer_name = str(row['LayerName']).strip()
39
40     if i < len(materials):
41         mat = materials[i]
42         mat.setMaterialName(layer_name)
43
44         gamma = get_val(row, 'UnitWeight')
45         nu = get_val(row, 'PoissonsRatio')
46         e_i = get_val(row, 'YoungsModulus') or 0.0
47         ucs = get_val(row, 'UCS') or 0.0
48         gsi = get_val(row, 'GSI') or 0.0
49         mi = get_val(row, 'mi') or 0.001
50         d_factor = get_val(row, 'D') or 0.0
51
52         e_rm = e_i
53         if e_i > 0 and gsi > 0:
54             exponent = (60.0 + 15.0 * d_factor - gsi) / 11.0
55             e_rm = e_i * (0.02 + ((1.0 - d_factor / 2.0) / (1.0 + math.exp(exponent))))

```

```

56
57     mb, s, a = 0.0, 0.0, 0.5
58     if gsi > 0:
59         mb = mi * math.exp((gsi - 100.0) / (28.0 - 14.0 * d_factor))
60         s = math.exp((gsi - 100.0) / (9.0 - 3.0 * d_factor))
61         a = 0.5 + (1.0 / 6.0) * (math.exp(-gsi / 15.0) - math.exp(-20.0 / 3.0))
62
63     if gamma is not None and gamma > 0:
64         mat.InitialConditions.setUnitWeight(gamma / 1000.0)
65     if nu is not None and nu > 0:
66         mat.Stiffness.Isotropic.setPoissonsRatio(nu)
67     if e_rm > 0:
68         mat.Stiffness.Isotropic.setYoungsModulus(e_rm)
69
70     mat.Strength.setFailureCriterion(StrengthCriteriaTypes.GENERALIZED_HOEK_BROWN)
71     try:
72         ghb = mat.Strength.GeneralizedHoekBrown
73     except AttributeError:
74         ghb = mat.Strength.GeneralizedHoekBrownStrength
75
76     ghb.setCompressiveStrength(ucs)
77     ghb.setMbParameter(mb)
78     ghb.setSPParameter(s)
79     ghb.setAPParameter(a)
80
81     print("Part 1 Complete: Mathematical parameters injected.")
82
83     # =====
84     # HELPER FUNCTION FOR LINER PROPERTIES
85     # =====
86     def set_liner_properties(liner_obj, thickness, youngs, poisson):
87         """Dynamically applies properties based on the RS2 API version."""
88         if hasattr(liner_obj, 'Standard'):
89             liner_obj.Standard.setThickness(thickness)
90             liner_obj.Standard.setYoungsModulus(youngs)
91             liner_obj.Standard.setPoissonsRatio(poisson)
92         elif hasattr(liner_obj, 'StandardBeam'):
93             liner_obj.StandardBeam.setThickness(thickness)
94             liner_obj.StandardBeam.setYoungsModulus(youngs)
95             liner_obj.StandardBeam.setPoissonsRatio(poisson)
96         elif hasattr(liner_obj, 'Beam'):
97             liner_obj.Beam.setThickness(thickness)
98             liner_obj.Beam.setYoungsModulus(youngs)
99             liner_obj.Beam.setPoissonsRatio(poisson)
100        elif hasattr(liner_obj, 'setThickness'):
101            liner_obj.setThickness(thickness)
102            liner_obj.setYoungsModulus(youngs)
103            liner_obj.setPoissonsRatio(poisson)
104        else:
105            print(f" [Warning] Could not automatically apply structural properties to '{liner_obj.getLinerName()}' due to API version differences.")
106
107        # =====
108        # PART 2: INDIVIDUAL LINER DEFINITIONS
109        # =====
110        print("\n--- Configuring Individual Liners ---")
111        liners = model.getAllLinerProperties()
112
113        if len(liners) < 2:
114            print("\n[ERROR] Missing Liner slots. Please add them in the RS2 GUI.")
115        else:
116            # Safely define Liner 1
117            try:
118                liner1 = liners[0]
119                liner1.setLinerName("Shotcrete")
120                set_liner_properties(liner1, 0.05, 30000, 0.2)
121                print(" Shotcrete configured.")
122            except Exception as e:
123                print(f" Failed to configure Shotcrete. Error: {e}")
124
125            # Safely define Liner 2
126            try:
127                liner2 = liners[1]
128                liner2.setLinerName("Final Concrete")
129                set_liner_properties(liner2, 0.3, 30000, 0.2)
130                print(" Final Concrete configured.")
131            except Exception as e:
132                print(f" Failed to configure Final Concrete. Error: {e}")
133
134        print("Part 2 Complete.")

```

```

136 # =====
137 # PART 3: COMPOSITE LINER CONFIGURATION
138 # =====
139 print("\n--- Configuring Composite Liner System ---")
140 try:
141     if len(liners) >= 2:
142         composite = model.getAllCompositeLinerProperties()[0]
143
144         # 1. Name the system
145         composite.setCompositeName("Tunnel Support System")
146
147         # 2. Set to 2 layers
148         composite.setNumberOfLayers(2)
149
150         # 3. Assign the specific Liners we just created by their exact new names
151         composite.setCompositeLinerPropertyByName(1, "Shotcrete")
152         composite.setCompositeLinerPropertyByName(2, "Final Concrete")
153
154         # 4. Set the 2-stage installation delay for Layer 2 (Final Concrete)
155         composite.setInstallDelay(2, 2)
156
157         print("Part 3 Complete: Composite Liner configured with 2-stage delay.")
158     else:
159         print("Skipped Composite Liner configuration due to missing Liner slots.")
160 except Exception as e:
161     print(f"Failed to configure Composite Liner. Error: {e}")
162 # =====
163 # FINAL SAVE
164 # =====
165 # model.save()
166 print("\nWorkflow Complete! Model properties are configured and meshed, but NOT saved to disk.")

```