CES500103

Optimize road design with Dynamo for Civil 3D and Generative Design

Paolo Emilio Serra
Autodesk, Principal Implementation Consultant

Learning Objectives

- Define a computational design approach for road design
- Leverage generative design to optimize the design and increase the insight of the design challenge
- Automate the creation of corridor models
- Assess the next steps for new use cases and implementations

Description

This class covers how to apply automation and generative design principles to optimize the design of a road network. The design starts from GIS inputs describing the site its features such as the boundary, existing buildings, and streams of water. Using InfraWorks and Civil 3D it is possible to create a model to start the definition of the design. Dynamo enables a computational design approach and define modeling strategies to enable optimization. With Generative Design it is possible to explore multiple scenarios and find the best solutions to the problem balancing multiple objectives. Finally with Dynamo for Civil 3D it is possible to create complex objects like alignments, profile, and corridors to curate the selected solutions and refine the modeling as needed.

Speaker

Paolo Emilio Serra is a construction engineer by trade, he worked as BIM Manager in an architectural firm for 5 years, now he is a Principal Implementation Consultant for Autodesk since 2014. With Autodesk he has been delivering Customer Success Services to engineering Companies, supporting BIM workflows and Digital Transformation in their business processes. Main focuses are on automation, generative design, integration between AEC and ENI industries. He is an architecture enthusiast, Revit user since 2006, API and Dynamo knowledge seeker. He wrote the CivilConnection Dynamo package that creates dynamic relationships between Civil 3D and Revit for Linear Structures BIM workflows. He is also the co-creator of the Civil 3D Toolkit package for Dynamo for Civil 3D. He owns the Punto Revit blog.

Problem Statement

The goal is to find methods to leverage the Generative Design approach to generate options to connect key-locations (defined by the designer) with a system of roads.
The road cross-section has a fixed width and fixed slopes for cut and fill.
The design considers existing structures such as buildings, water streams and, of course, the existing ground surface.

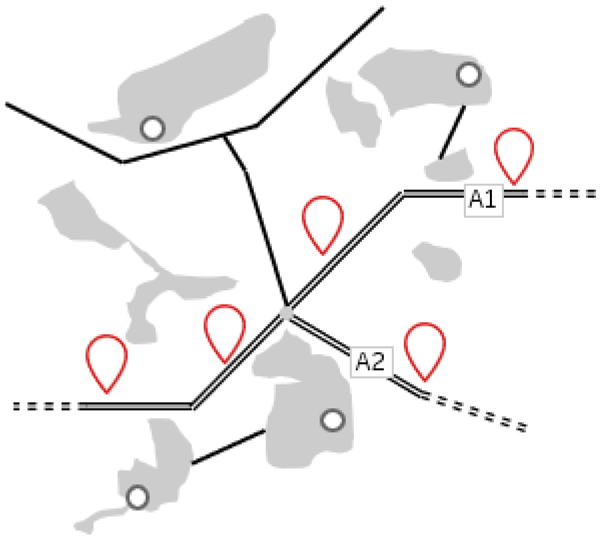


FIGURE 1: EXAMPLE OF A ROAD SYSTEM THAT CONNECTS KEY-LOCATIONS

The scope of this document is to focus more on the approach and show the new possible, there are working simplifications adopted to highlight critical steps in the approach. By no means this is the only way, or the “correct” way, to apply generative design for roads.

In this context a road in the system is the combination of three types of elements:

- A horizontal alignment, composed of straight lines and circular curves.
- A vertical profile, defined along the alignment and composed of straight segments; the combination of alignment and profile generates the road baseline.
- A set of cross-sections, each of which attaches to the baseline on the mid-point of the horizontal segment representing the road pavement.

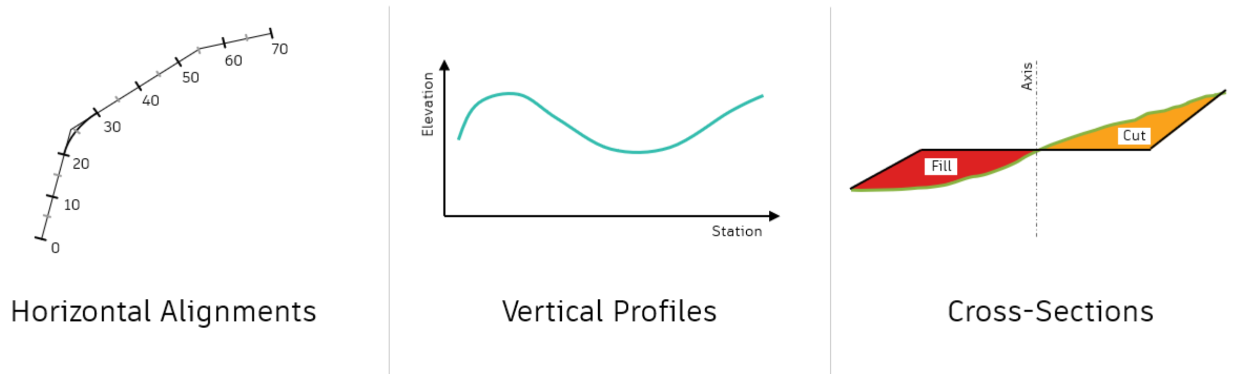


FIGURE 2: IN THIS CONTEXT A ROAD IS DEFINED BY THESE THREE ELEMENTS

Key Considerations

Measurable objectives allow to compare the performance of two different options.
Below a list of common objectives for road design optimization.

Minimize

- Nr. of Clashes
- Nr. of Water stream crossings (a proxy for costs of extra structures like bridges and culverts)
- Length
- Absolute Cut
- Absolute Fill
- Cut/Fill balance

Maximize

- Nr. of key-locations served
- Visibility

The result of the comparison of the objectives values is what allows to rank options and guide the evolution of the study towards optimum solutions.
They can even be in contrast with one another; quite often there is no simple analytical "shortcut" to find the optimum for this kind of problems or it might be expensive to develop an algorithmic solution that is also flexible.
These are the reasons why to solve these kinds of problems it is more convenient to adopt an evolutionary strategy and, luckily, the technology to solve a multi-objective optimization exists in Generative Design in Revit (GDiR).
This document illustrates how to leverage the technology in a road design context.

It might be surprising to the reader that in this approach there is tolerance for undesired options (e.g., with one or more clashes with buildings, or where no road serves one or more of the key-locations).

This is a critical difference from other approaches that leverage, at various degree, a form of automation or artificial intelligence.
Differently from Machine Learning applications, in Generative Design the goal is not to mimic what a designer does, but to explore a problem in search for optimal solutions.
These might be quite different from what a human would think of via leveraging their instincts or experience, and this is part of the value of an unbiased objective-driven approach.

The designer defines the strategy used in a Generative Design study, this allows suboptimal solutions because, over time, it will uncover the relationships among the design variables and performing solutions thanks to the evolutionary algorithm used by GDiR.

In this new paradigm, the designers' creativity is in how they frame a problem, defining which strategies to adopt and what objectives to consider to setup a Generative Design study. New skills are required to unlock the potential of this approach; this document is a humble attempt to accompany the reader along an innovation journey for the Civil Infrastructure Industry.

Get the Data in via InfraWorks and Civil 3D

It is beyond the scope of this document to give a detailed overview of these software for Civil Infrastructure design. The Generative Design capabilities for gradings in Civil 3D are not taking part to the workflow analyzed in this document.
Suffice to say that this software integration is key to acquire the necessary data to start the design instead. The proposed use case starts from InfraWorks connecting to the ArcGIS services and downloading the data of the context such as buildings, utilities, etc. and then it is transferred and refined in Civil 3D, integrating also other data source (e.g., SHP files) or directly via designer input.

Define the logic in Dynamo for Civil 3D

Using Dynamo for Civil 3D it is possible to read the information contained in a DWG document, derive inputs, and combine them into a custom logic to create, for example, AutoCAD or Civil 3D objects. In this context, Dynamo for Civil 3D allows to acquire the fixed inputs that will not change during a Generative Design study, for example the existing ground mesh, the coordinates of the key-locations, the boundary of the study area, etc.

In essence, Dynamo allows to extract the underlying geometry entities from the inputs of AutoCAD and Civil 3D/Map 3D objects. This is a critical step of the Generative Design study as, at the time of writing, it is not possible to leverage the nodes that operates on AutoCAD or Civil 3D objects directly (e.g., calculate the profile projecting a horizontal alignment on the existing ground).

This is the reason optimizing a Civil 3D Corridor directly it is not allowed, but it is still possible to optimize its main geometrical components (alignment, profile and cross sections definitions).

Civil 3D Toolkit

The Civil 3D toolkit is a free package for Dynamo for Civil 3D that contains extensions of the out-of-the-box nodes to cover more workflows in AutoCAD and Civil 3D using Dynamo. The user can install the package from the Dynamo Package Manager.
In this context, it facilitates the acquisition of information such as the geometries of Map 3D polygons and polylines or extracting a quick mesh representation of the existing ground. Unfortunately, at the time of writing, there is not much that we can do with a mesh via Dynamo nodes or via Dynamo packages. This is one of the reasons to develop an external Python module to deal with these kinds of geometry entities.
It also provides facilitations to directly create more complex objects such as alignments, profile, assemblies, subassemblies and combine them into a Civil 3D corridor with baseline, regions ad targets. I refer to this usage of the Civil 3D Toolkit as "automation" in the companion presentation. For more information about the Civil 3D Toolkit, refer to this couple of classes Computational Design for Civil Engineers, and Supercharge Your Dynamo Graph with Civil 3D Toolkit and the Dynamo Forum here.

Leverage Generative Design in Revit and use the results in Civil 3D

To use Generative Design in Revit (GDiR), we need to first create the logic in a Dynamo graph and then export it with its dependencies for Generative Design. If there are any problems, the tool will report the corrective actions to take. During the export it is possible to add a description and a thumbnail to clearly identify the study and provide high-level documentation.

From the Revit user interface, under Manage, it is possible to launch generative design and create a new study. A list of the available studies will appear and selecting one of them will open the setup window from which it is possible to launch the study. More information on how to setup a Generative Design study here.

When the study completes all the cycles or when the designer stops the calculations, the designer can select a solution from the GDiR interface and send it back to Dynamo. From here it is possible to leverage the same concept of serialization but in this case applying it to the output, or surrogate model, representing the Alignments and Profiles. These create the basic ingredients of Civil 3D objects that combined generate a Corridor.

Introduction to Dynamo and Computational Design

Dynamo is a platform that can define algorithms visually and the sequence of instructions are generated using block called Nodes that perform predefined tasks, joined together in sequence using Connectors. A Dynamo algorithm is called Graph to emphasize the visual programming approach which is one of the main differences between Dynamo and any other scripting language.

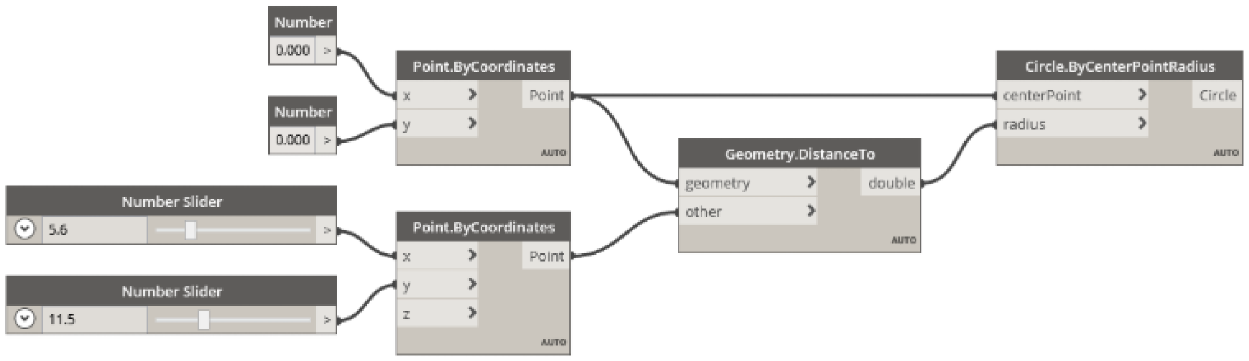


FIGURE 3: VISUAL PROGRAMMING IN DYNAMO

Dynamo is some sort of a digital Swiss-knife that can contain just the right blade for the job and even if it is very intuitive in the way it can be used, it is also highly effective. Another important characteristic of Dynamo is that, for whatever reason, a particular blade should be missing it supports several ways to either add one from a public repository of external packages or create more sophisticated functionalities using traditional scripting languages such as Python or C#.

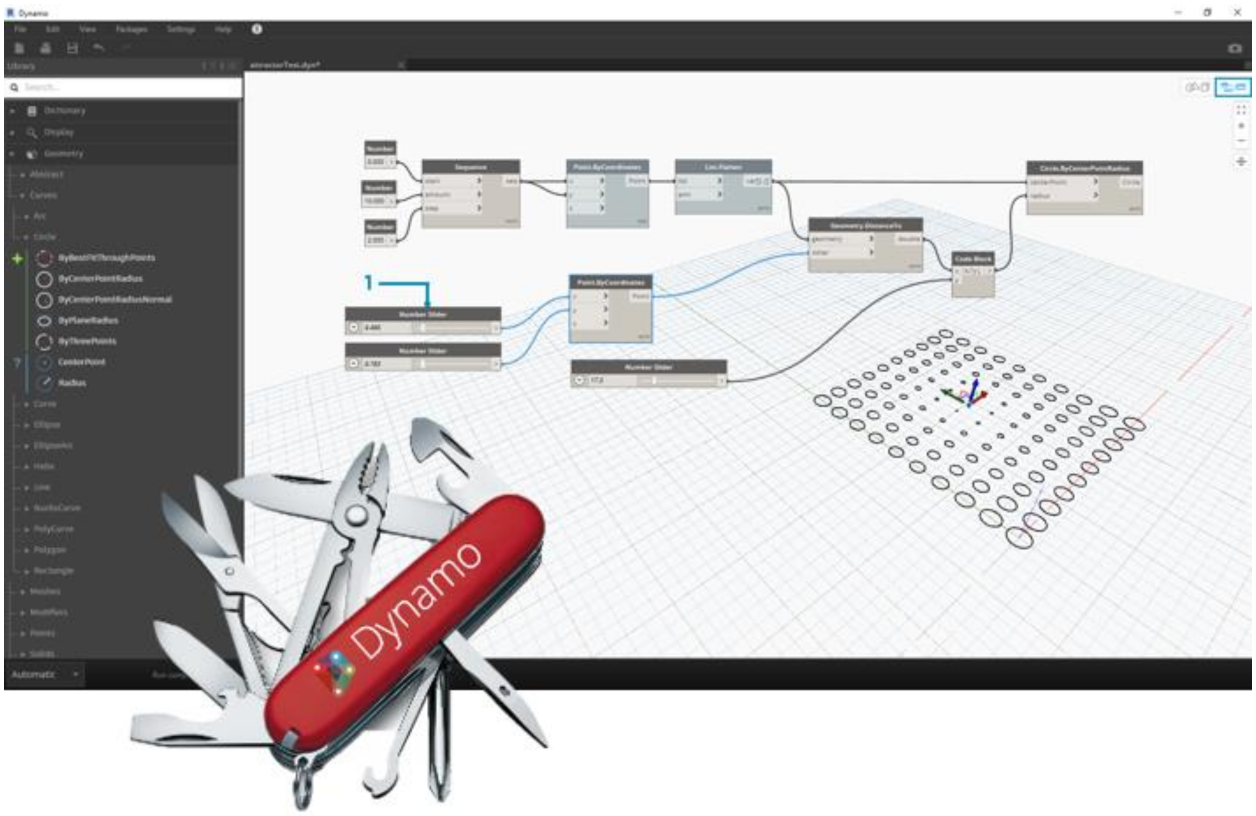


FIGURE 4:DYNAMO IS A FLEXIBLE PLATFORM FOR AUTOMATION AND VISUAL PROGRAMMING THAT SUPPORTS CUSTOMIZATION

Dynamo commoditizes programming and brings it closer to the designers, taking care of the sophisticated operations such as handling databases and transactions, serializing changes and updating the models rather than performing a “fire-and-forget” automation. Dynamo is in the sweet spot in between interactive tools such as Civil 3D or Revit and more traditional scripting languages to leverage product APIs.

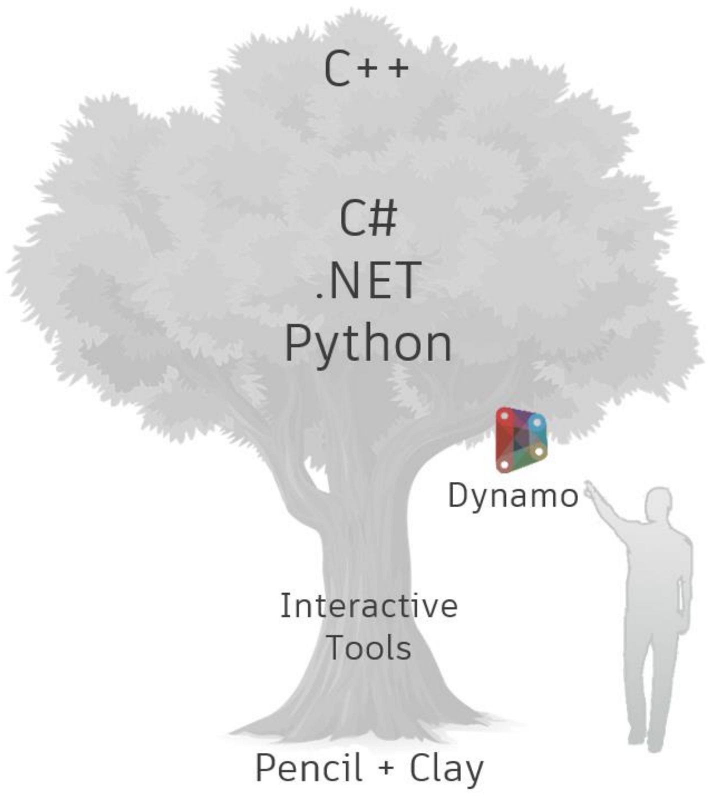
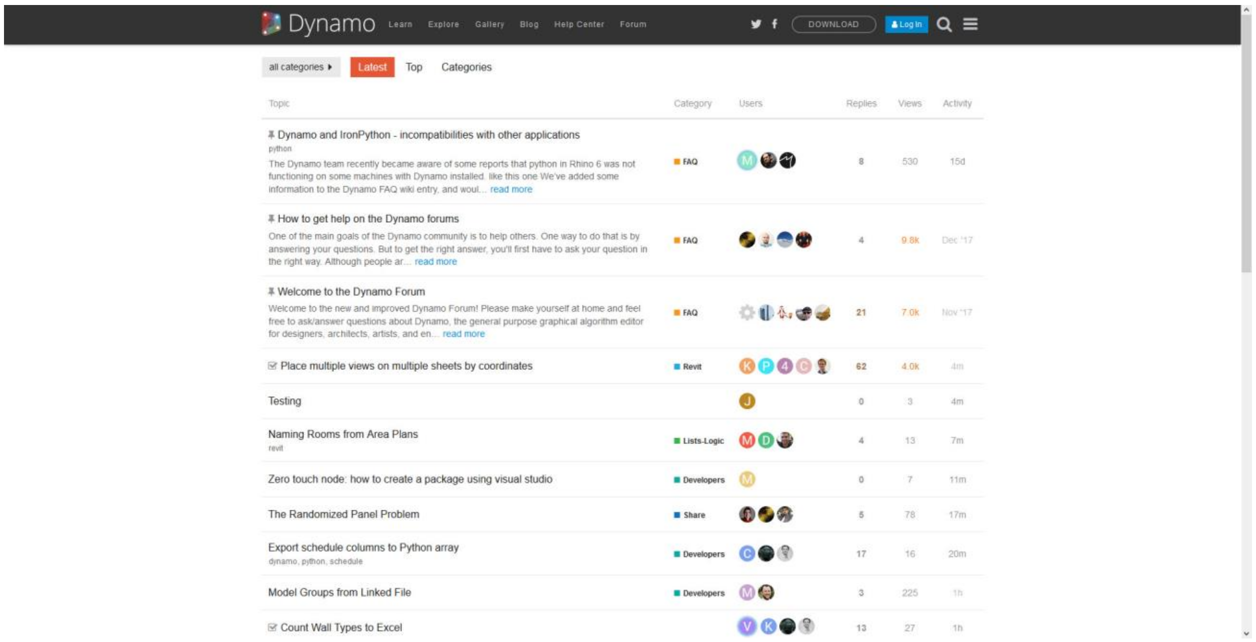


FIGURE 5: DYNAMO AS LOW-HANGING FRUIT OF THE TREE OF AUTOMATION TOOLS

Dynamo can be found as a standalone application called Sandbox but also integrated with a host application such as Revit or Civil 3D. For more information about Dynamo integrations and where to get it visit the link https://dynamobim.org/download/

The community of Dynamo users has grown with a fast and steady pace, mostly because of the ease of use of visual programming and because the community started to collect and share useful sets of nodes for free. These are bundled into packages to extend Dynamo capabilities and simplify the creation of more sophisticated workflows and boost adoption and productivity.

The community is also proactively engaging in knowledge sharing on the Dynamo forums: necessary for anyone and everyone who wants to connect, learn, find creative solutions to problems and contribute back.



Dynamo Learn Explore Gallery Blog Help Center Forum DOWNLOAD Log In

all categories Latest Top Categories

Topic	Category	Users	Replies	Views	Activity
Dynamo and IronPython - incompatibilities with other applications python The Dynamo team recently became aware of some reports that python in Rhino 6 was not functioning on some machines with Dynamo installed. like this one We've added some information to the Dynamo FAQ wiki entry, and woul… read more	FAQ	M	8	530	15d
How to get help on the Dynamo forums One of the main goals of the Dynamo community is to help others. One way to do that is by answering your questions. But to get the right answer, you'll first have to ask your question in the right way. Although people ar… read more	FAQ		4	9.8k	Dec '17
Welcome to the Dynamo Forum Welcome to the new and improved Dynamo Forum! Please make yourself at home and feel free to ask/answer questions about Dynamo, the general purpose graphical algorithm editor for designers, architects, artists, and en… read more	FAQ		21	7.0k	Nov '17
Place multiple views on multiple sheets by coordinates	Revit	K P 4 C	62	4.0k	4m
Testing		J	0	3	4m
Naming Rooms from Area Plans revit	Lists-Logic	M D	4	13	7m
Zero touch node: how to create a package using visual studio	Developers	M	0	7	11m
The Randomized Panel Problem	Share		5	78	17m
Export schedule columns to Python array dynamo, python, schedule	Developers	C	17	16	20m
Model Groups from Linked File	Developers	M	3	225	1h
Count Wall Types to Excel		V K	13	27	1h

FIGURE 6: THE DYNAMO FORUM ON THE OFFICIAL WEBSITE DYNAMOBIM.ORG

In this context, Dynamo defines the computational strategies to generate options in the study and the same graph is used to specify the variables and their ranges as well as the logic to measure the objectives.

Leverage Python to expand the possibilities

Dynamo supports the execution of Python scripts. Python is closer to a traditional scripting language, and it is close to reading and writing English, it has a quite simple syntax and yet is powerful. For more information on the language itself visit this link, and to know how to leverage Python in Dynamo follow this link.

At the time of writing, Dynamo supports two different versions of Python that are compatible with .NET (IronPython 2.7 and CPython 3.7), more information on the differences and the work in progress to improve the support can be found here.

During the development of the solution, it is convenient to write external Python modules for more complex functions that can be reused in other projects and load them inside Python nodes in Dynamo. GDiR leverages multiple processes that launch Dynamo Sandbox instances (e.g., a version of Dynamo that does not possess "host nodes" such as those specific to Civil 3D) and each of them supports the load of external modules. This can be used to leverage custom functionalities and existing libraries in the Generative Design study.

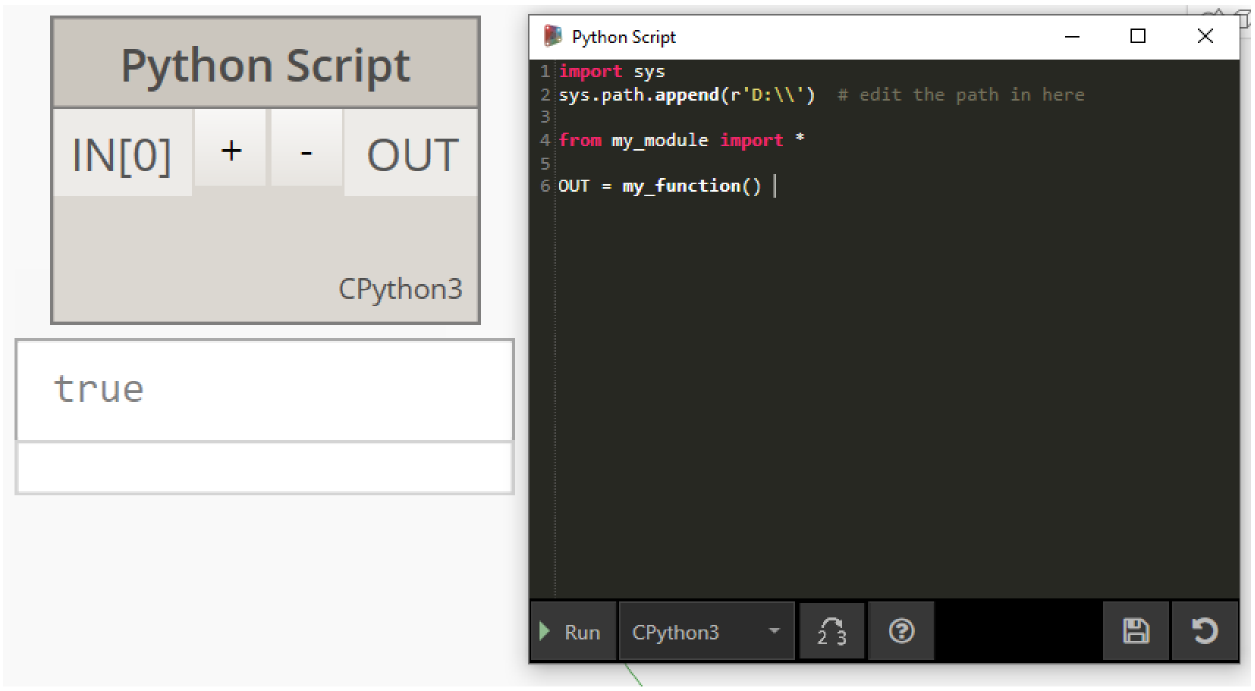
Using an external module allows to also leverage a Python Integrated Development Environment (e.g., PyCharm) and have more functionalities than those available in the Python nodes in Dynamo.

A simple application of this approach is adding a logging capability to a custom function to capture errors, exceptions, warnings or simply to follow the execution of the code for timing purposes.

Open a text editor and save the file as my_module.py in folder of your choosing.
In the file copy the following code:

```
import logging
import os
import Tempie

FORMAT = '[%(asctime)s] %(levelname)s: %(processName)s [%(process)d]:
%(module)s.%(funcName)s %(message)s'
logging.basicConfig(filename=os.path.join(tempfile.gettempdir(), 'MyLog.txt'),
level=logging.DEBUG, format=FORMAT, datefmt='%Y/%m/%d %H:%M:%S')


def my_function():  # this is the function to call from Dynamo
        logging.info('Start')

        # do something here

        logging.info('End')

        return True
```

In Dynamo open a new file and insert a Python Script node, right click and select Edit.
In the Python editor delete the content and past the following, then edit the path to the folder that contains my_module.py:



```python
import sys
sys.path.append(r'D:\\')  # edit the path in here

from my_module import *

OUT = my_function()
```

FIGURE 7: EXAMPLE OF HOW TO LOAD AN EXTERNAL PYTHON MODULE IN DYNAMO

The code in the external module writes a file called MyLog.txt in the temporary folder and when my_function() is executed appends two INFO entries capturing when the function starts and when it ends like the following:

```
[2021/09/25 02:16:42] INFO: MainProcess [34052]: my_module.my_function Start
[2021/09/25 02:16:42] INFO: MainProcess [34052]: my_module.my_function End
[2021/09/25 02:16:58] INFO: MainProcess [34052]: my_module.my_function Start
[2021/09/25 02:16:58] INFO: MainProcess [34052]: my_module.my_function End
[2021/09/25 02:17:59] INFO: MainProcess [34052]: my_module.my_function Start
[2021/09/25 02:17:59] INFO: MainProcess [34052]: my_module.my_function End
```

FIGURE 8:EXAMPLE OF CUSTOM LOGGING

In this example other than the time of entry there is the severity (ERROR, EXCEPTION, WARNING, INFO, DEBUG), the name of the process and its PID (in this case 34052), the name of the module containing the custom function, the function name, and the message.

In Generative Design there are multiple processes running in parallel, so it is possible to check if there are any issues in one of them and more importantly why, capturing the error messages.

In this context, I have been developing my own modules in Python to facilitate some computationally expensive operations via recursion, dealing with Civil 3D objects without leveraging the Dynamo for Civil 3D nodes (e.g., find the projection of an alignment on the mesh of the existing ground), implementing algorithms such as those to solve the Minimum Spanning Tree (e.g., used to calculate the alignments), read and write custom JSON schemas to store fixed inputs, etc.

Optimization Overview

The generative design workflow starts with gathering the data about the problem to solve. The result is like the problem DNA: it contains the essential traits used to define a surrogate model to explore the design space. In the generate phase, the "recipe" to create a potential solution to the problem is defined—in other words, an algorithm that transforms inputs and variables into a candidate solution. At the beginning, a few candidates are generated randomly.

Each candidate solution is then evaluated against a set of criteria or objectives so that can be scored and ranked against the others. The generative design enters in a cycle in which the genetic algorithm guides the exploration of the design space. At each cycle, it collects information about the performance of the candidates and over time it will refine the quality of the proposed solutions.

At the end of the study, it is possible to navigate the results to gather a better insight into the problem, uncovering the correlation between inputs and outputs to select the combinations that produced the more interesting results.
The surrogate model can then be used as a starting point to further the detailing of the selected solution.

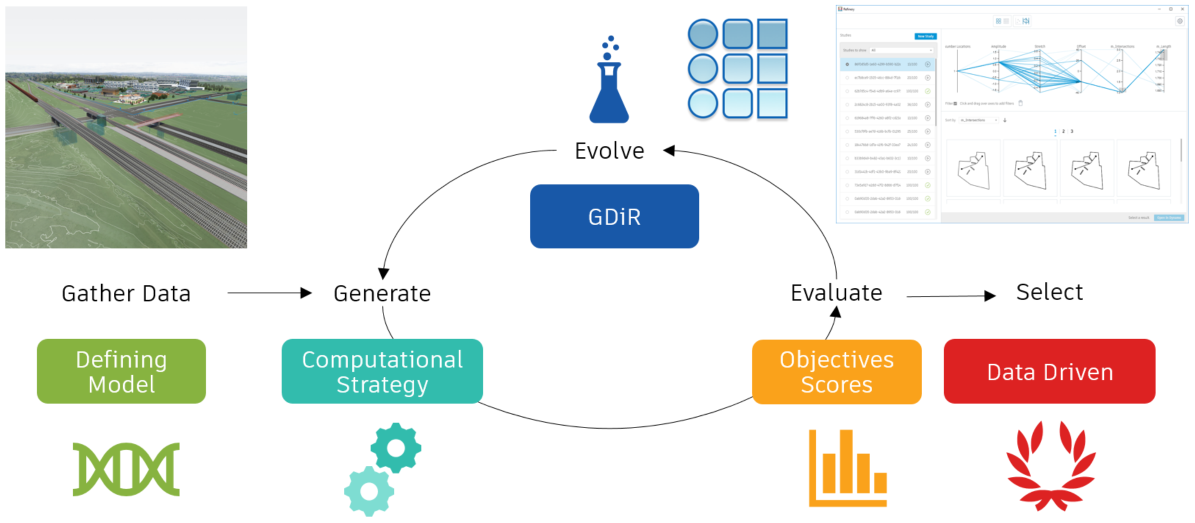


FIGURE 9: OVERVIEW OF THE GENERATIVE DESIGN APPROACH

Define the approach

To create abstract schemas of the processes, I recommend the Business Process Modeling Notation (BPMN 2.0) with the free online platform Cawemo.com.
As a convention, I adopted the following syntax to capture the key components of the process:

Inputs—Parameters that the designer can change but that are fixed for the duration of the study:

I: <name>=<value> (e.g., I: Width=30m)

Variables—What parameters are considered as part of the design space to explore with generative design, specifying the ranges and the step and how they are affecting the elements in the model:

V: <name> [<lower>, <upper>] | <step> (e.g., V: Rotation [0,359] | 1)

Generate—A high-level description of what constitutes a surrogate model, used to validate an option with the necessary steps to create a candidate solution.

Evaluate—A collection of objectives used to measure as proxies for the fitness/quality of the candidate options, specifying for each one if they should be minimized (minus sign) or maximized (plus sign):

e.g., Area [+], Cost [-]

Evolve – This is when Generative Design in Revit takes over and starts exploring the problem and generate options.

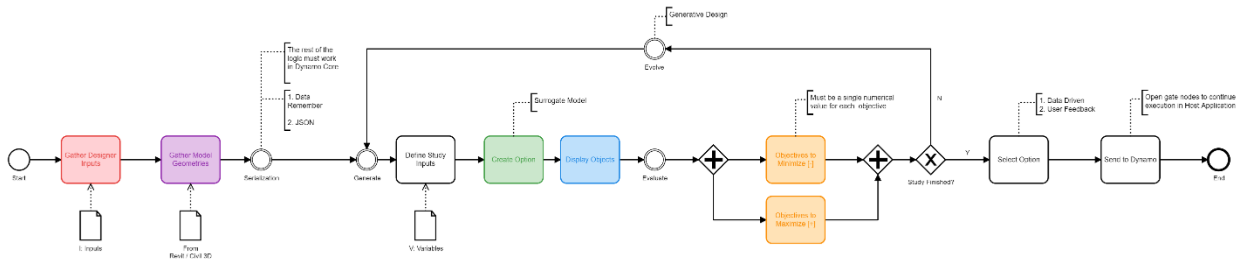


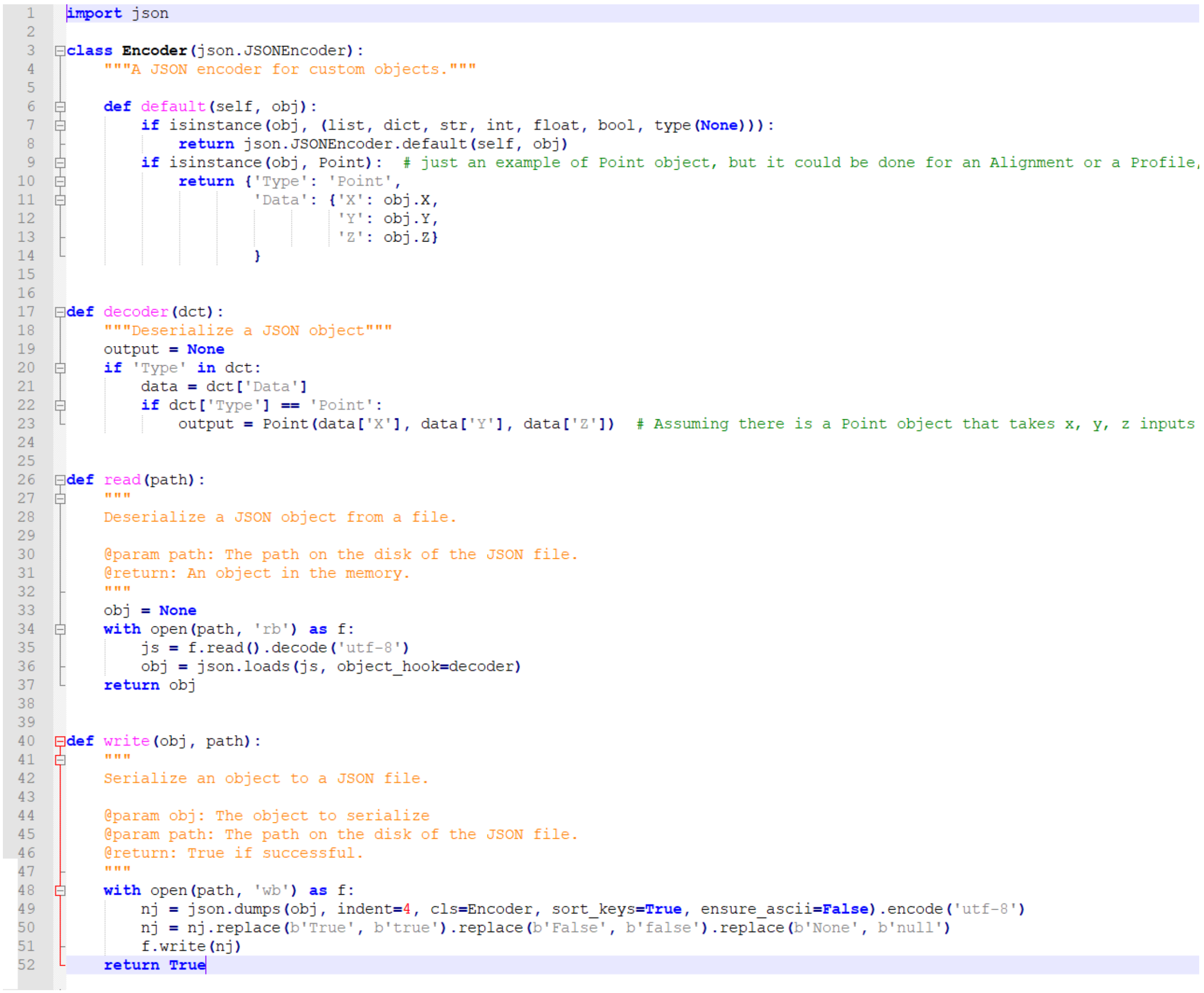
FIGURE 10: STRUCTURE OF A GENERATIVE DESIGN WORKFLOW

Serialize Data

GDiR offers the possibility to store information that is not changing during a study directly into the JSON structure of the study type. This is possible via the Data.Remember node and because the Dynamo file under the hood is based on a JSON schema, and a lot of work has been done to enable the serialization (in other words to convert to a stable representation) the geometry entities in Dynamo.
In this context, once the data from Civil 3D is embedded in the study type it can be transferred to another application that launches GDiR.
Similarly, data can also be brought back to Civil 3D following the reversed workflow, storing the results of the optimization in new Data.Remember nodes. A good planning of the Dynamo graph can help find where exactly put these nodes to run a successful study, and in the most recent versions of GDiR there have been improvements to highlight where these should go.

The same can be obtained leveraging Python built-in capabilities to read and write JSON files. The difference is that the data is not embedded in the Dynamo graph but into a separate JSON file decupling the logic from the data, which is always a good thing to do, as it makes it easier to manage different versions of the study. The Python approach allows also to develop custom Encoder objects and decoders functions for custom objects.

The code below defines the basic structure for a serialization module in Python.



```python
import json

class Encoder(json.JSONEncoder):
    """A JSON encoder for custom objects."""

    def default(self, obj):
        if isinstance(obj, (list, dict, str, int, float, bool, type(None))):
            return json.JSONEncoder.default(self, obj)
        if isinstance(obj, Point):  # just an example of Point object, but it could be done for an Alignment or a Profile,
            return {'Type': 'Point',
                    'Data': {'X': obj.X,
                             'Y': obj.Y,
                             'Z': obj.Z}
                    }


def decoder(dct):
    """Deserialize a JSON object"""
    output = None
    if 'Type' in dct:
        data = dct['Data']
        if dct['Type'] == 'Point':
            output = Point(data['X'], data['Y'], data['Z'])  # Assuming there is a Point object that takes x, y, z inputs


def read(path):
    """
    Deserialize a JSON object from a file.

    @param path: The path on the disk of the JSON file.
    @return: An object in the memory.
    """
    obj = None
    with open(path, 'rb') as f:
        js = f.read().decode('utf-8')
        obj = json.loads(js, object_hook=decoder)
    return obj


def write(obj, path):
    """
    Serialize an object to a JSON file.

    @param obj: The object to serialize
    @param path: The path on the disk of the JSON file.
    @return: True if successful.
    """
    with open(path, 'wb') as f:
        nj = json.dumps(obj, indent=4, cls=Encoder, sort_keys=True, ensure_ascii=False).encode('utf-8')
        nj = nj.replace(b'True', b'true').replace(b'False', b'false').replace(b'None', b'null')
        f.write(nj)
    return True
```

FIGURE 11: STRUCTURE OF A SERIALIZATION MODULE IN PYTHON

Design a Surrogate Model

The surrogate model is an important part of the generative design workflow. In fact, after a careful analysis you might even change your point of view regarding what kind of objects you need to deal with. This happens quite often in real-world applications of generative design. The surrogate model is an abstract representation of a real problem, a simplification if you will, and as such it helps focusing on the key features.

This allows you to turn down the “volume” of the other characteristics of the problem at hand, it helps in clarifying the relationships between the parts, and it highlights inner hierarchies that otherwise might not be immediately visible.
This is also a great support for visualization and effective communication of the results of a generative design study.

At the end of this process, the other characteristics of the problem are still there, and it is possible to take them into account using critical thinking as a separate exercise when it is time to select the solutions or integrate them one by one and refine the computational strategies to create more comprehensive studies.

In this context, the surrogate model of the road is the combination of alignment, profile and cross-sections.

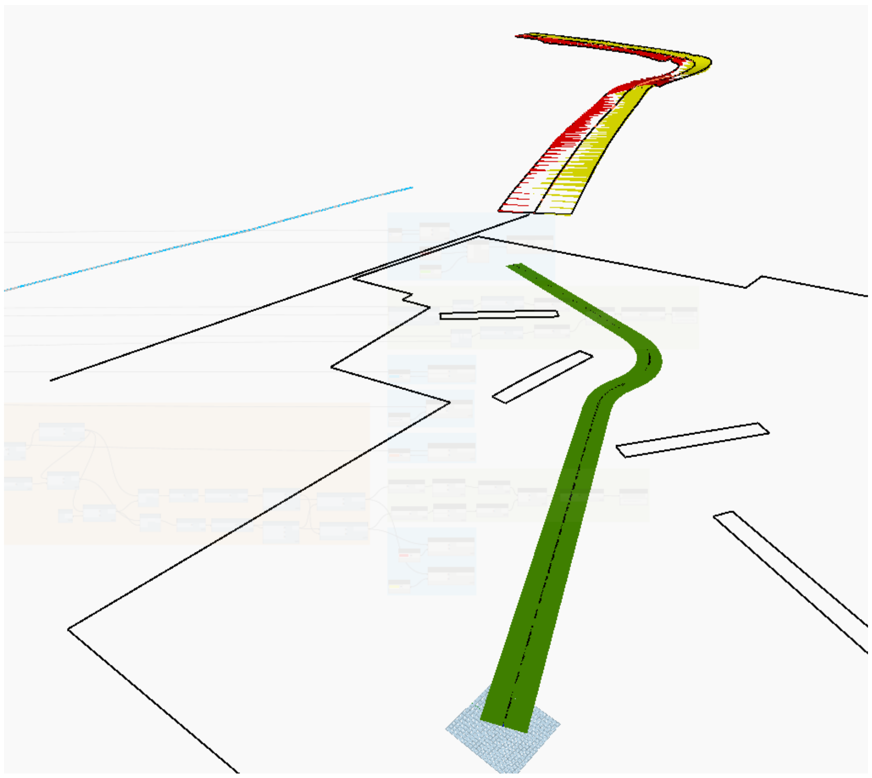


FIGURE 12: THE ROAD SURROGATE MODEL

Implement Computational Strategies

The first thing to realize is to define the design space for a given problem. This is a flexible system that is based on the principles of parametrization, in other words via the definition of the numbers of parameters necessary to describe a model.
It is crucial that the chosen parameters describe the functionality of the model as completely as possible while using the smallest number of parameters. Every design space and its

parametrization define a unique strategy which encodes the way in which the designer imagines solving a particular problem.

How to design a space? There are some factors to consider for an effective Generative Design study:

- **Bias**, refers to the condition of being too simple
- **Variance**, refers to the condition of being too flexible
- **Complexity**, refers to the potential of a design space to create unpredictable results beyond our own intuition
- **Continuity**, refers to the internal consistency and structure of the design space (consequently, random variables must be avoided)

Designing the space and refining the parameters involved takes time and it is an iterative approach by design. For more information I recommend reading Danil Nagy's articles on Generative Design.

The variables to be used can either be:

- Discrete, integer numbers
- Continuous, decimal numbers
- Permutations, combinations (these are not supported in Dynamo OOTB but can be implemented via Python)

Define Objectives and Metrics

These are the metrics by which the designer can drive the exploration of the design space with respect to a set of characteristics of the generated solutions. They can be only of two types: *minimize* or *maximize*.
As an implementation tip, GDiR defaults the objectives to minimize, and so it is possible to revert a maximize metric multiplying the result by -1, so that no matter how many objectives in a study type, they will be ready for execution without the risk of misplacing an assignment in the setup phase, only to discover after a few cycles that the calculations were looking in the "wrong direction".
When there is a constraint threshold involved, this can be represented with a binary clause (e.g., 0 if the condition is met, 1 otherwise – remember that the default behavior we are targeting is the minimization of the metric).
If there is a numeric target instead, it is possible to minimize the absolute value of the distance between the target and the actual metric.

Evolve the Design

This task is iterative in nature, not only in terms of evolutionary algorithm used by GDiR but also in terms of designer approach in refining the parameters of the computational strategies.
The goal of the design space exploration is to find the values of the variables that produce optimal solutions with respect to the objectives. For this reason, the search can produce different results in different executions.

The “landscape” of the design space can be complex and even difficult to grasp if there are more than three or four variables involved.

GDiR offers four different methods of exploration:

- **Cross Product**, this method allows to select which variables consider during the search and to sample a finite number of values for each variable (from two to ten); this allows to uncover high-level relationships between the variables and focus the attention on promising value ranges for the variables improving the chances to find optimal solutions in less cycles, every second saved counts. This is not really “learning or evolving” the design, but it is applying a brute-force approach to the search.
- **Randomize**, as the name suggests, this method samples the design space randomly and provides some quick and yet unstructured insight during the search. This method does not really “learn or evolve” the design but it can provide a cheap insight on the variable value ranges that produce the best options. It is recommended in the preliminary phases of the design to fine tune the designer input parameters and to keep the calculation for a single solution as quick as possible.
- **Optimize**, this method applies the Non-dominated Sorting Genetic Algorithm (NSGA2) which is a robust technique for solving multi-objective optimization problems. Differently from other algorithms, this technique allows to uncover the optimal solutions whilst maintaining diversity of the solutions, in other words it does not converge on a single result but does a better job at exploring the design space. The algorithm is capable of learning and evolving the design and it is computationally less expensive than a brute-force approach. On the other hand, it returns more of a “direction” rather than a “destination”, that is where the last method adds precision to the results found via Optimize.
- **Like This,** this method takes an initial configuration of the variables values and perturbs them in search for the true optimum. This further refines the search and the results obtained from previous iterations.

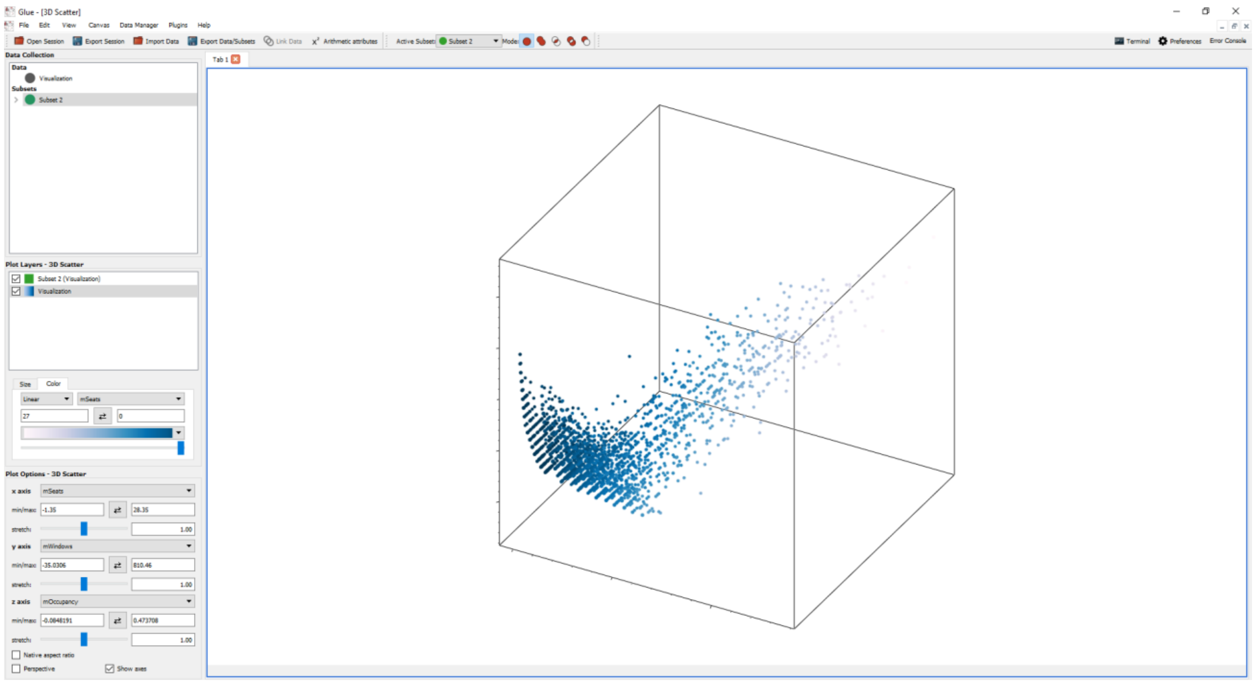
Select an Optimal Solution

GDiR offers different methods to visualize the inputs and the outputs, and more importantly allows to sort and filter the results based on specific performance thresholds. The goal is for the designer to make a choice based on numerical evidence (e.g., alignment length, cut/fill balance, etc.) to augment their experience and confidence when looking for the best solution.

It is worth noting that with the latest release of GDiR at the time of writing, the results of the computation can be exported to CSV files used in 3rd party platforms to further analyze data (e.g., GlueViz, Microsoft PowerBI and similar) to uncover relationships between inputs and outputs, get a better insight into the problem which is in a nutshell what the design optimization is trying to achieve.

Once a solution is selected and further refined in GDiR (e.g., refining the value range of the variables), it can be sent back to Dynamo and the surrogate model offers another visual support to inspect the characteristics of the solution.

If the designer is satisfied, the solution can be serialized to JSON and consumed in Dynamo for Civil 3D to continue with the next phases of the workflow.


Glue - [3D Scatter]
File Edit View Canvas Data Manager Plugins Help
Open Session Export Session Import Data Export Data/Subsets Link Data Arithmetic attributes Active Subset: Subset 2 Mode:
Terminal Preferences Error Console
Data Collection
Data
Visualization
Subsets
Subset 2
Tab 1
Plot Layers - 3D Scatter
Subset 2 (Visualization)
Visualization
Size Color

Linear, mSeats, 27 ⇄ 0

Plot Options - 3D Scatter

x axis: mSeats
min/max: -1.35 ⇄ 28.35
stretch: 1.00
y axis: mWindows
min/max: -35.0306 ⇄ 810.46
stretch: 1.00
z axis: mOccupancy
min/max: -0.0848191 ⇄ 0.473708
stretch: 1.00
☐ Native aspect ratio
☐ Perspective ☑ Show axes

FIGURE 13: LOADING GDIR RESULTS IN GLUEVIZ FOR VISUALIZATION PURPOSES

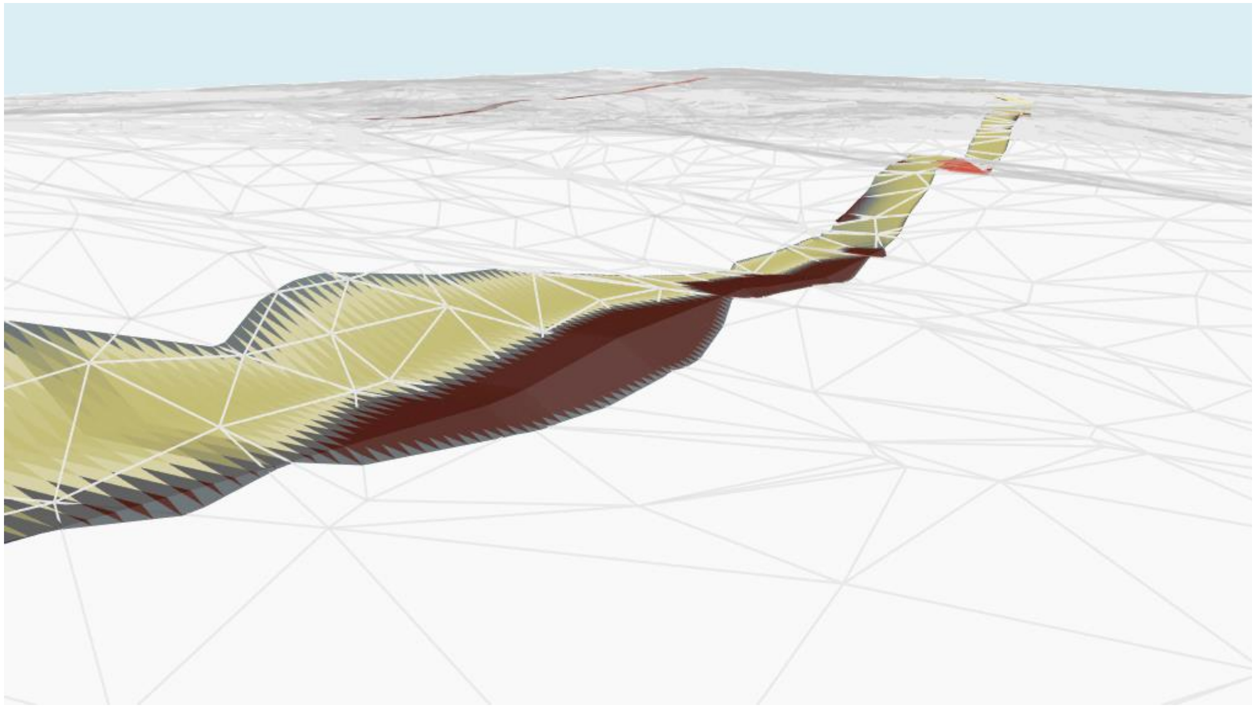
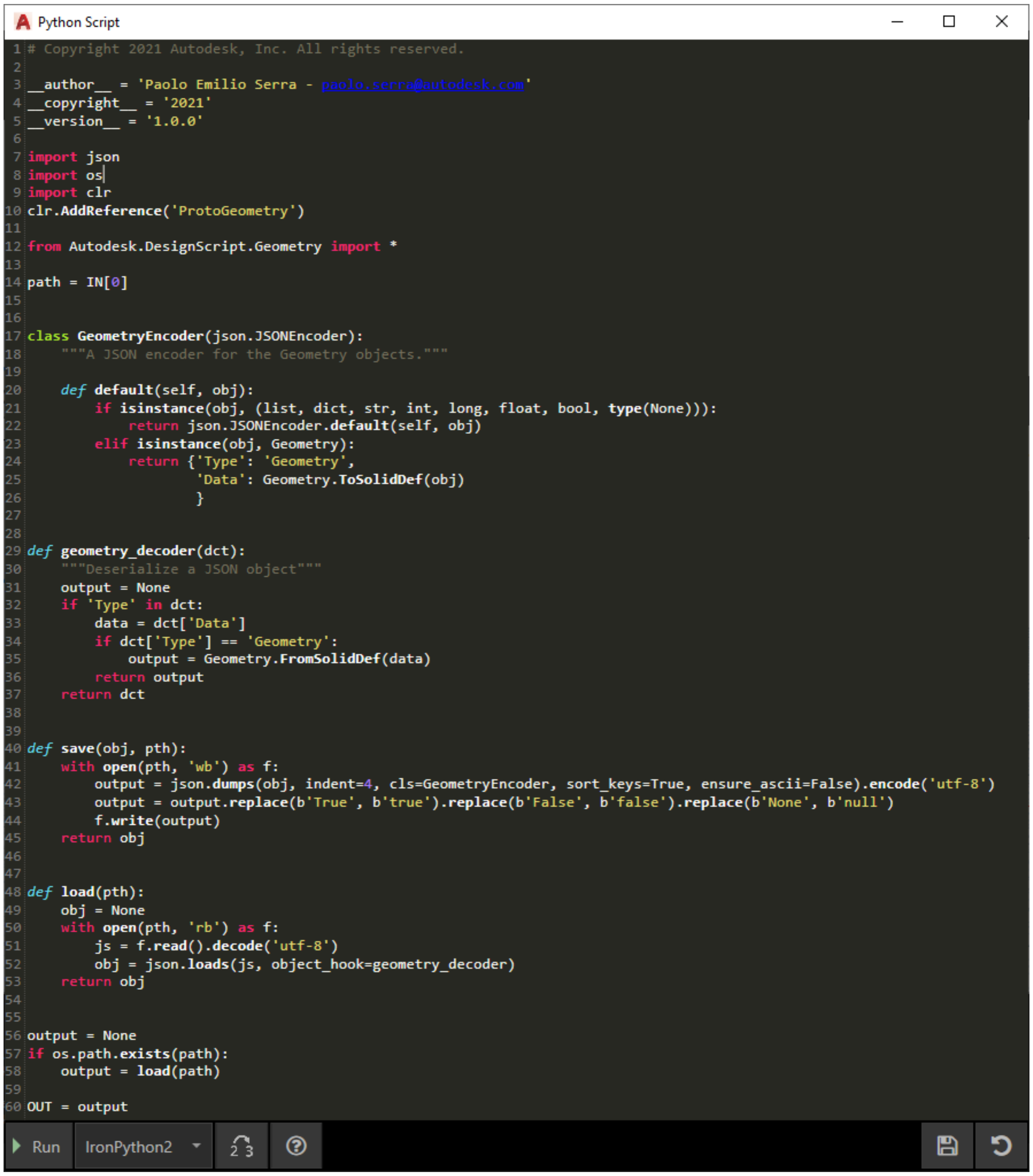


FIGURE 14: EXAMPLE OF ROAD SOLUTION

Automation

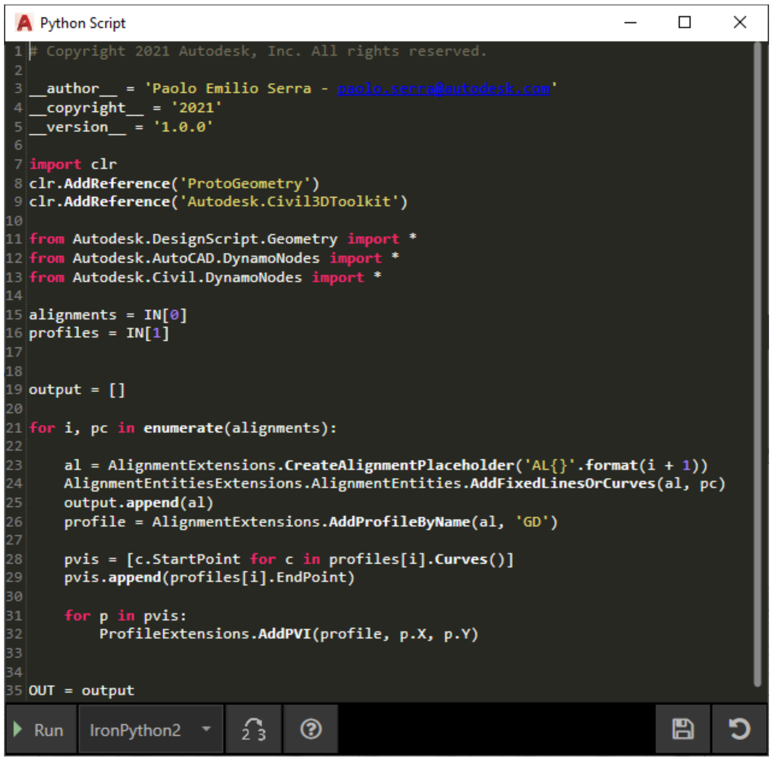
Once the solution has been selected and sent back to Dynamo it is possible to serialize the results from GDiR to JSON and read them in a dedicated graph in Dynamo for Civil 3D.
This information is than used to recreate the Dynamo geometry entities underlying the definition of more complex Civil 3D Objects.



```python
# Copyright 2021 Autodesk, Inc. All rights reserved.

__author__ = 'Paolo Emilio Serra - paolo.serra@autodesk.com'
__copyright__ = '2021'
__version__ = '1.0.0'

import json
import os
import clr
clr.AddReference('ProtoGeometry')

from Autodesk.DesignScript.Geometry import *

path = IN[0]


class GeometryEncoder(json.JSONEncoder):
    """A JSON encoder for the Geometry objects."""

    def default(self, obj):
        if isinstance(obj, (list, dict, str, int, long, float, bool, type(None))):
            return json.JSONEncoder.default(self, obj)
        elif isinstance(obj, Geometry):
            return {'Type': 'Geometry',
                    'Data': Geometry.ToSolidDef(obj)
                    }


def geometry_decoder(dct):
    """Deserialize a JSON object"""
    output = None
    if 'Type' in dct:
        data = dct['Data']
        if dct['Type'] == 'Geometry':
            output = Geometry.FromSolidDef(data)
        return output
    return dct


def save(obj, pth):
    with open(pth, 'wb') as f:
        output = json.dumps(obj, indent=4, cls=GeometryEncoder, sort_keys=True, ensure_ascii=False).encode('utf-8')
        output = output.replace(b'True', b'true').replace(b'False', b'false').replace(b'None', b'null')
        f.write(output)
    return obj


def load(pth):
    obj = None
    with open(pth, 'rb') as f:
        js = f.read().decode('utf-8')
        obj = json.loads(js, object_hook=geometry_decoder)
    return obj


output = None
if os.path.exists(path):
    output = load(path)

OUT = output
```

FIGURE 15: PYTHON SCRIPT TO LOAD A JSON FILE THAT CONTAINS DYNAMO GEOMETRIES

Using the Civil 3D Toolkit, it is possible to create Alignments and associated Profiles in Civil 3D. Assuming that they are both returned as Dynamo PolyCurve entities from the GDiR, it is possible to leverage a Python script node to generate these elements in Civil 3D.



```python
# Copyright 2021 Autodesk, Inc. All rights reserved.

__author__ = 'Paolo Emilio Serra - paolo.serra@autodesk.com'
__copyright__ = '2021'
__version__ = '1.0.0'

import clr
clr.AddReference('ProtoGeometry')
clr.AddReference('Autodesk.Civil3DToolkit')

from Autodesk.DesignScript.Geometry import *
from Autodesk.AutoCAD.DynamoNodes import *
from Autodesk.Civil.DynamoNodes import *

alignments = IN[0]
profiles = IN[1]


output = []

for i, pc in enumerate(alignments):

    al = AlignmentExtensions.CreateAlignmentPlaceholder('AL{}'.format(i + 1))
    AlignmentEntitiesExtensions.AlignmentEntities.AddFixedLinesOrCurves(al, pc)
    output.append(al)
    profile = AlignmentExtensions.AddProfileByName(al, 'GD')

    pvis = [c.StartPoint for c in profiles[i].Curves()]
    pvis.append(profiles[i].EndPoint)

    for p in pvis:
        ProfileExtensions.AddPVI(profile, p.X, p.Y)


OUT = output
```

FIGURE 16: LEVERAGE CIVIL 3D TOOLKIT NODES IN A PYTHON SCRIPT TO CREATE ALIGNMENTS AND PROFILES

It is also possible to create a Corridor placeholder with a single Baseline, it only needs the names of the Corridor, the Alignment and the Profile.

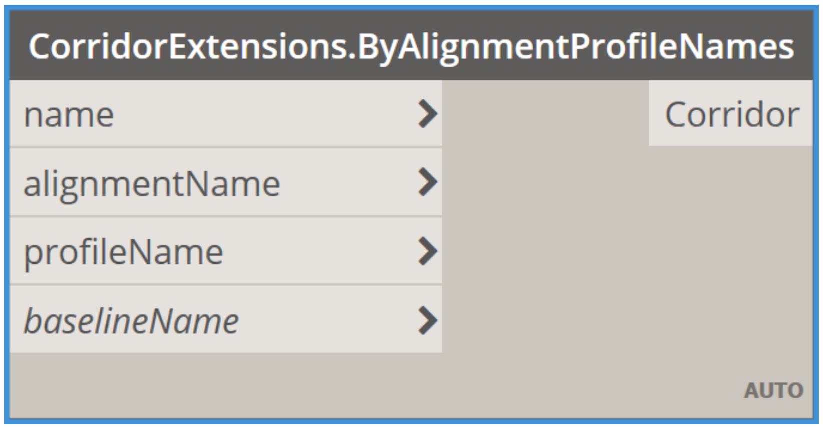


FIGURE 17: CREATE A CORRIDOR WITH THE CIVIL 3D TOOLKIT

The Assembly and Subassembly can be generated on the fly via Civil 3D Toolkit, but it is also possible to prepare the Civil 3D file manually in advance. Once the name of the Assembly is known it can be used to create the Baseline Region on the corridor and set the Targets, for example the Existing Ground.

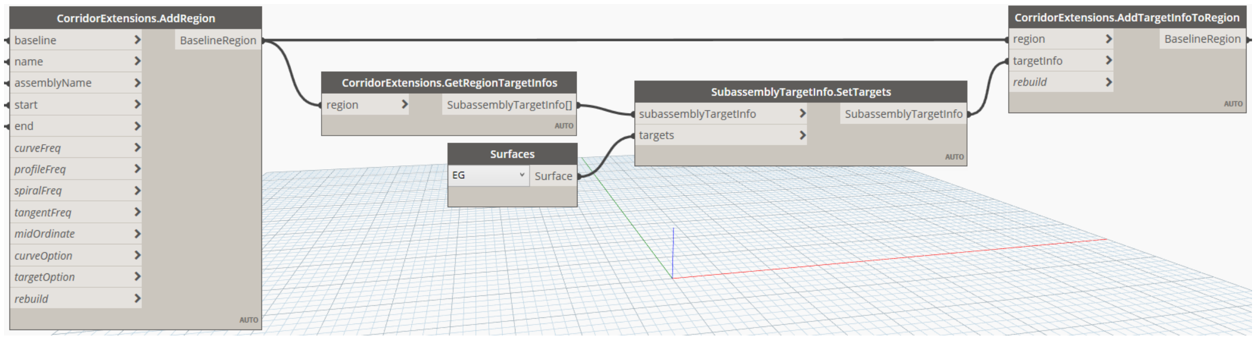


FIGURE 18: CREATE THE BASELINE REGION AND SET TARGETS

At the end of this process, it is possible to rebuild the Corridor, access to the definition of the Alignment and Profile via the Civil 3D interface and curate the model even further via traditional means in Civil 3D.

Conclusions

The performance of the exploration of the design space depends on the speed by which each solution is obtained. The optimization algorithm requires a substantial number of individuals in the population to effectively search for optimal solutions, this means that every second counts and optimizations of the calculations are necessary.
In the road design context, the performance is affected by the number of elements involved such as the number of triangles in the existing ground surface or the frequency of the cross sections along the road. Reducing the numbers of triangles using oriented bounding boxes along the Alignment entities and a sufficiently large “buffer” will help in reducing the calculation time and even avoid unexpected errors linked to memory limitations.

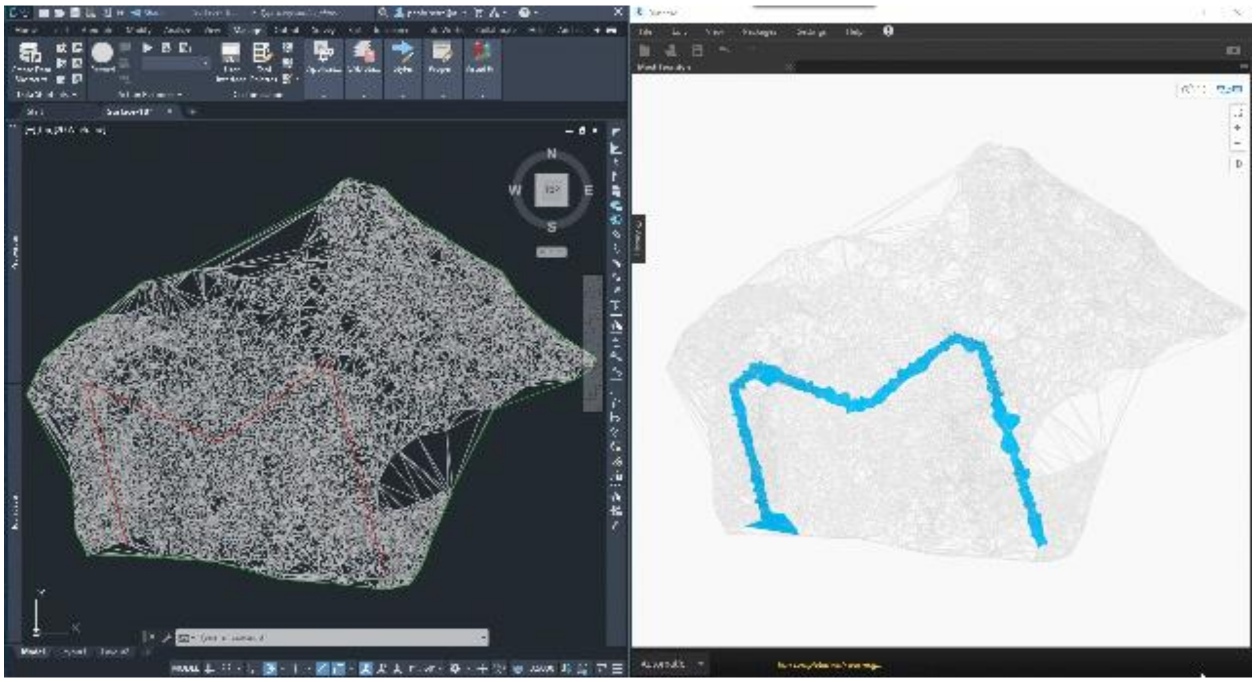


FIGURE 19: REDUCE THE NUMBER OF TRIANGLES TO IMPROVE THE PERFORMANCE

The frequency of the cross sections follows the logic you would apply for the parameters of a Corridor in Civil 3D, but the intersection performance is much slower in Dynamo and, depending on the length of the road, it could even take several minutes to compute, which is not ideal for a Generative Design study that might repeat thousands of times.
Another useful technique to keep in mind is the recursion, for example to join the triangles together to create a single PolySurface.

This approach can be extended to cover other kinds of linear structures such as tunnels, bridges and retaining walls, etc.

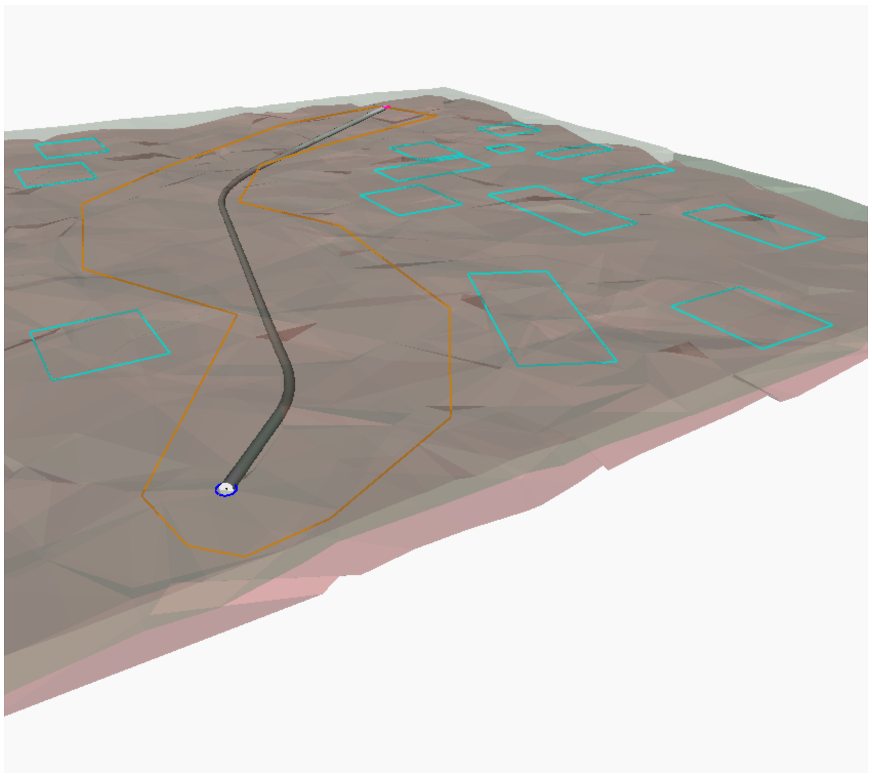


FIGURE 20: GENERATIVE DESIGN FOR A TUNNEL TO MINIMIZE VIBRATIONS AND THE USE OF EXPLOSIVE

There is no single way of adopting an optimization approach for linear structures design, nor a single computational strategy to cover all kinds of design. The designer controls how the problem is defined and what strategies are working better than others on a case-by-case basis. It will be increasingly important to acquire the necessary skills to be free to make these decisions and leverage the new possibilities offered by the technology.

References

Learning resources:

- Autodesk University
- https://primer.dynamobim.org/
- https://www.generativedesign.org/
- https://forum.dynamobim.com/
- https://dynamobim.org/blog/

Autodesk Classes used as references for this document:

- CES322249 Computational Design for Civil Engineers
- BES471869 Non-Geeks Guide to Optimizing Daily Workflows with Generative Design
- AS473693-L Generative Design at Hogwarts: Using Tech Instead of Magic
- CES473668 Supercharge Your Dynamo Graph with Civil 3D Toolkit

Appendix A – Graphs and Minimum Spanning Trees

A graph in mathematical terms is an abstraction used to represent "connectivity information" (see source).
A graph is made of vertices or nodes and edges that connect them. The edges can have a weight, or a number that represents the intensity of a connection, in geometrical applications the length of the edge is used as a weight.
Connected graphs can have loops or cycles (some vertices can be reached following more than one path). A tree is a graph with no cycles and a spanning tree of a graph is a connected graph that contains all the vertices of the original graph and only a subset of the edges so that there are no cycles. If the tree is composed by only by edges with minimum weight this is called the minimum spanning tree.

There are several algorithms to solve this problem and several implementations in Python that can be used in the road design context, in the companion presentation this is used to connect key-locations to determine the shortest path to connect them and generate pseudo-alignments. The algorithm per se does not provide a design space, so the introduction of extra vertices other than the key-locations provides the flexibility to define a design space for the study. Which extra vertices to add and how many depend on the computational strategy that the designer wants to adopt and there is no single answer to that.
In case it was already known that a loop or cycle between the vertices is needed, this approach will not work.
An alternative is the "Travelling salesperson problem" which explores the permutations of the connections between the vertices to return the shortest path that visits all the vertices exactly once and returns to the starting point.
The two approaches can be combined to obtain a more comprehensive design space to cater for more complex solutions.

Appendix B – Behavioral Systems and Cellular Automaton

Among the possible strategies to solve a generative design problem there is a branch called *behavioral systems* that is based on the concept of discrete agents that interact with their neighbors and the environment over a series of time-steps to generate solutions.
In this approach the parametrization of the design space resides in the rules that are chosen to model the behavior of the agents. The results of the interactions affect the internal state of the agents; over a series of time-steps the final state of the agents determines the design solution.

A cellular automaton is a discrete collection of cells, usually in a lattice-like structure (a row, a planar grid or a spatial grid). The cells evolve their state in parallel at discrete time steps following dynamic transition rules considering the states of the cells in their local neighborhood.
As the evaluation of the states can go on indefinitely it is recommended to introduce a termination criterion keeping track of overall metrics that do not change after several steps.

In the companion presentation, this approach is introduced as a method to calculate design vertical profiles for the road. The agents are the Points of Vertical Intersection (PVIs) that can be represented as cells in a row sorted by their station value. The states used are only two: active and inactive: when the PVI is inactive it will not be part of the PVIs of the design profile.

Other potential states for a PVI could increase or decrease their associated station or shift up and down their associated elevation. The rules by which these state changes are triggered is entirely up to the designer. Some criteria to consider are the slope of the resulting connecting profile segments, the overall visibility, the need for bridges or tunnels, etc.

These custom objects can be implemented as Python classes to keep track of their states and internal metrics and to assign custom behaviors and rules.