AUTODESK UNIVERSITY

AU 2018; NEIL KATZ, FRANCIS SEBASTIAN
SKIDMORE, OWINGS & MERRILL LLP

PROGRAMMING INTELLIGENT WAYFINDING AND EGRESS PLANNING / AS227086

WEDNESDAY, 14 NOVEMBER 2018, 09:15-10:15

NEIL KATZ, FRANCIS SEBASTIAN
SKIDMORE, OWINGS & MERRILL

SOM

Programming Intelligent Wayfinding and Egress Planning

This class demonstrates collaborative workflows for using intelligent model data to perform wayfinding analysis and egress compliance checks. The first part of this two-part demonstration walks through a popular ‘Shortest Walk’ algorithm to evaluate egress conditions of a single floor plate using the Grasshopper Visual Programming interface. The second part demonstrates the process of transferring room and door data from Autodesk Revit to Grasshopper via the internet and back again.

Neil Katz; Skidmore, Owings & Merrill LLP; Architect
Francis Sebastian; Skidmore, Owings & Merrill LLP; BIM Specialist

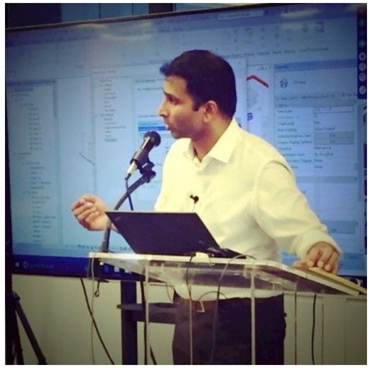
Duration: 60 minutes
Type: Instructional Demo
Topics: Architecture Services Building Information Modeling (BIM)
Class Focus: Exploring industry practice and workflows
Learning Objectives:
. Experiment with the potential of using a spatial element data from a Revit project as a planning tool
. Learn how to develop prototypical workflows based on the data set provided in this class to use room and door Revit elements for code compliance checking
. Learn how to capitalize on existing Grasshopper libraries to perform spatial analysis
. Learn how to use a limited set of Revit API and Dynamo methods to retrieve and inject data to and from the Revit model
Level of Expertise: Intermediate
Audience: Architect, Interior Designer

WHO WE ARE

Neil Katz, Skidmore, Owings & Merrill
Computational Design Specialist | Associate Director

Using a "computational design" approach to design, implements and develops methodologies to create geometry (simple and "complex") and to analyze and design in response to many project objectives, including environmental and sustainability goals.

"Computational design", aspects of which include algorithmic and parametric design as well as BIM, is as much a way of thinking about design as using and developing tools for design.



Francis Sebastian, Skidmore, Owings & Merrill
BIM Specialist

Develops BIM workflows for project teams. Builds software applications to bridge gaps in existing technologies. Enjoys working with Neil.

SOM

Skidmore, Owings & Merrill
Skidmore, Owings & Merrill LLP (SOM) is one of the largest and most influential architecture, interior design, engineering, and urban planning firms in the world. Founded in 1936, we have completed more than 10,000 projects in over 50 countries. We are renowned for our iconic buildings and our commitment to design excellence, innovation, and sustainability.
Our New York office has an applied research and development group focusing on digital design, in five areas: computational design (geometry, scripting), high performance design (sustainability, analysis), building information modeling (tools, templates, workflows), visualization (tools, templates, workflows), and realization (fabrication, materials). We collaborate closely with similar groups in other offices.

the ‘right’ tool for the job is one that the designer is most fluid with

WHY ARE WE DOING THIS?

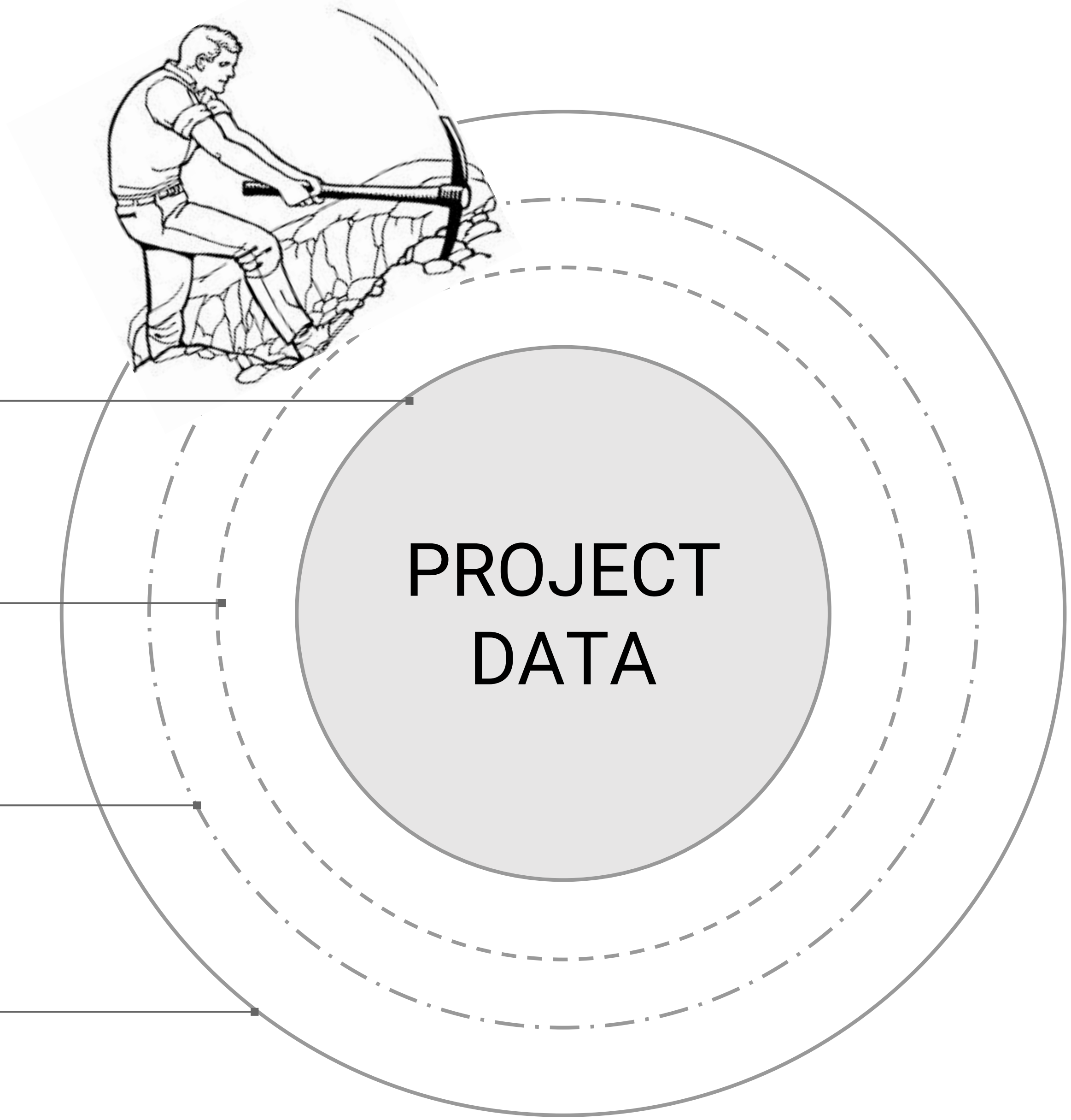
Obstacles to ‘fluidity’

MULTIPLE DATA FORMATS

COMPLEXITY

TRAINING

ACCESS



Programming Intelligent Wayfinding and Egress Planning

INTEROPERABILITY
. accessing the data in Revit:
    . rooms
    . doors
    . “obstacles” (furniture, columns, etc.)
. saving the data in an accessible format and location (JSON file)

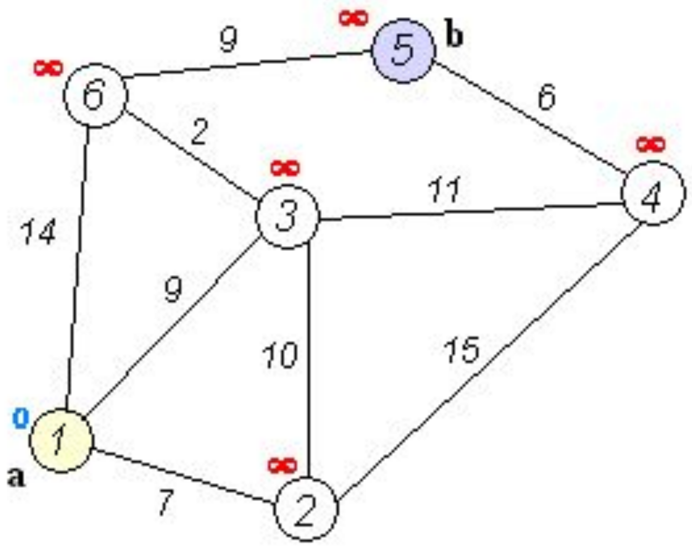
ANALYSIS
. pulling the data into Grasshopper
    . parsing the JSON file to identify rooms, doors, and obstacles
. associate doors and obstacles with rooms
. identify corridors and exit doors
. find “shortest paths”

INTEROPERABILITY (again)
. using a JSON file, make the shortest path diagram available to the Revit model

INTEROPERABILITY

GEOMETRIC SPATIAL ANALYSIS

Dijkstra’s “shortest path” algorithm:



Dijkstra's algorithm to find the shortest path between *a* and *b*. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.[1][2][3]

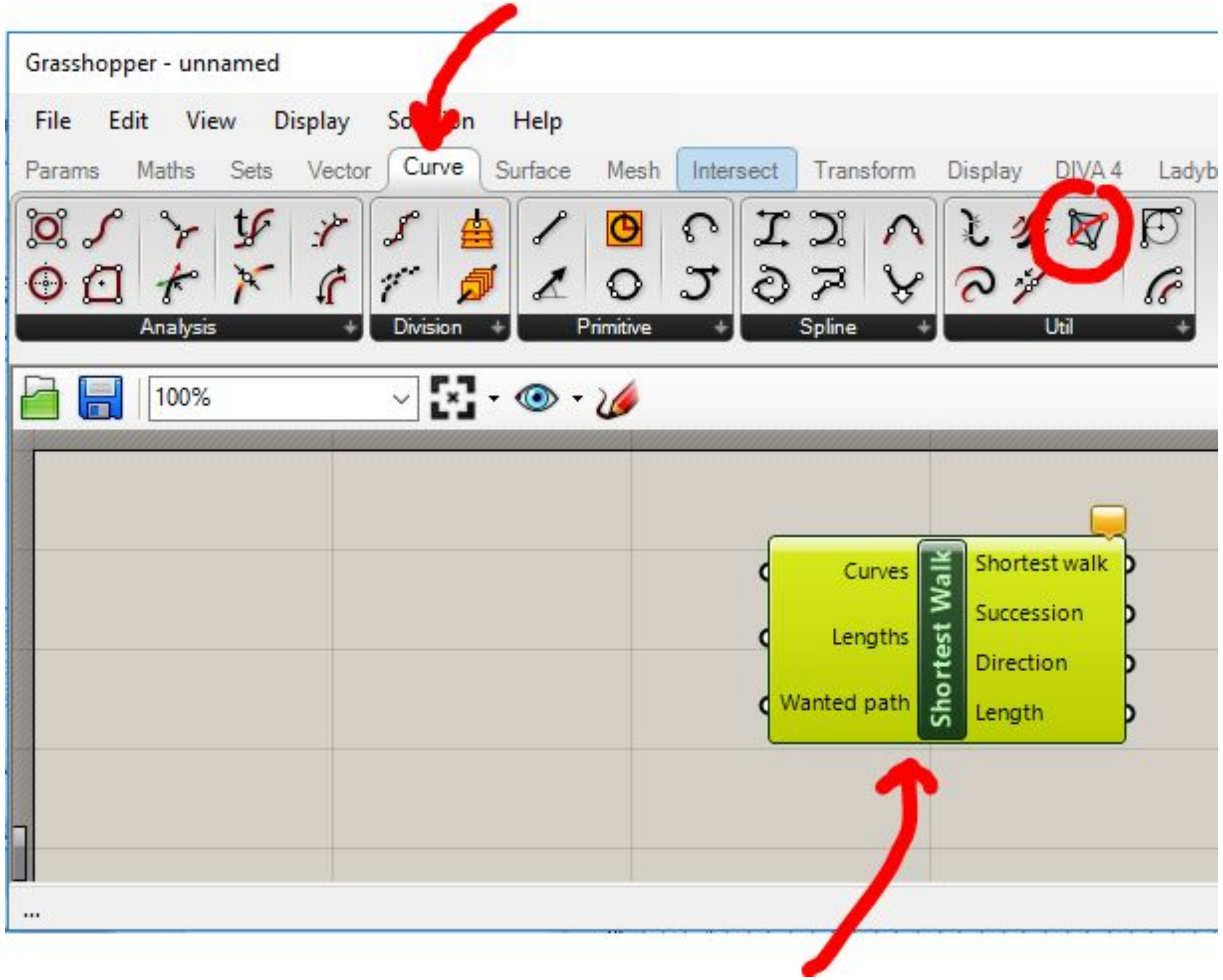
The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,[3] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other.[4]:196–206 It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's.

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Grasshopper has a “Shortest Path” component, which uses Dijkstra’s algorithm.

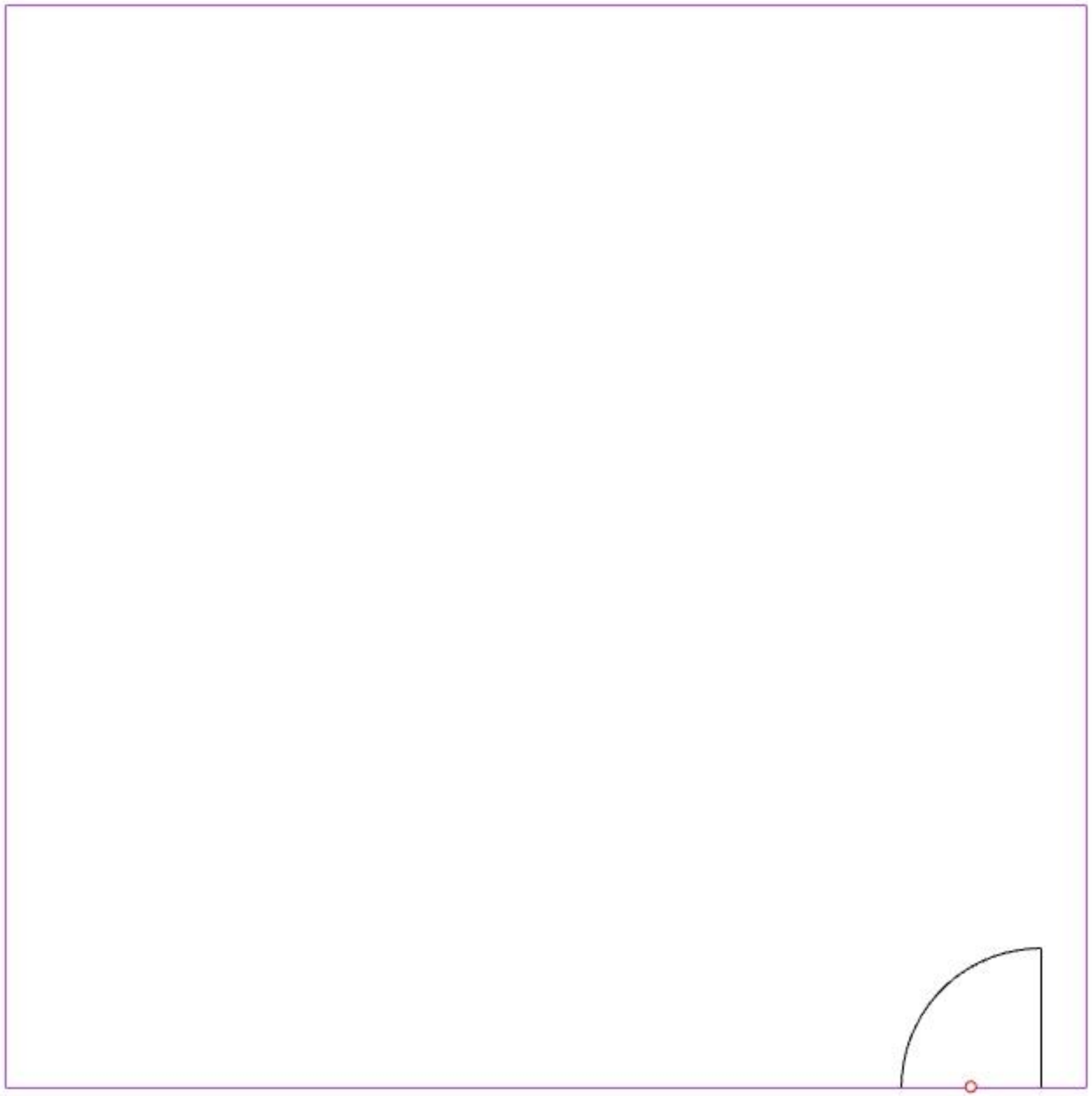
If we input a collection of lines (“Curves”) from with the Shortest Path tool will select, and a line indicated the desired path, from start point to end point (“Wanted path”), the component will compute the shortest path.

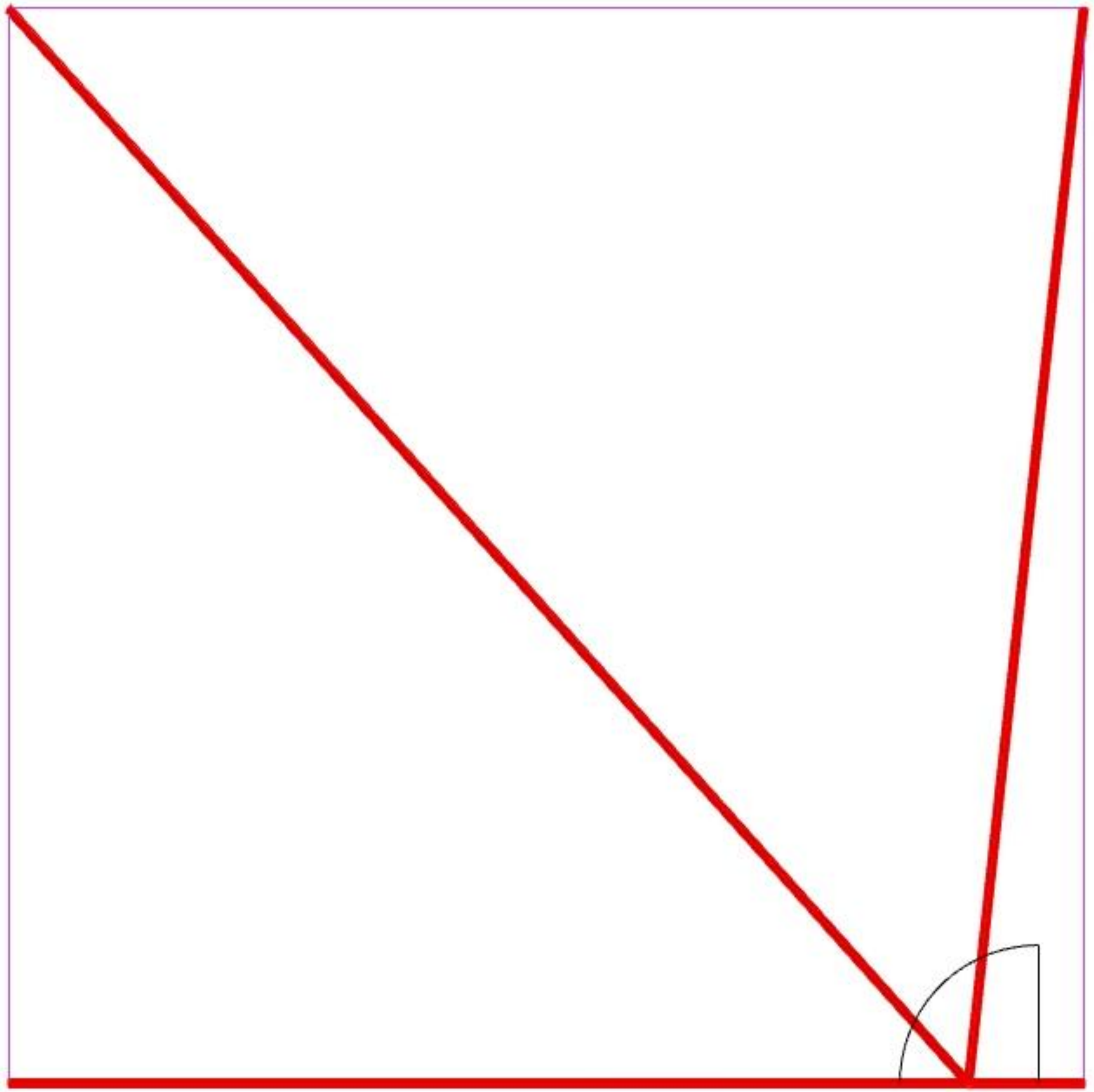


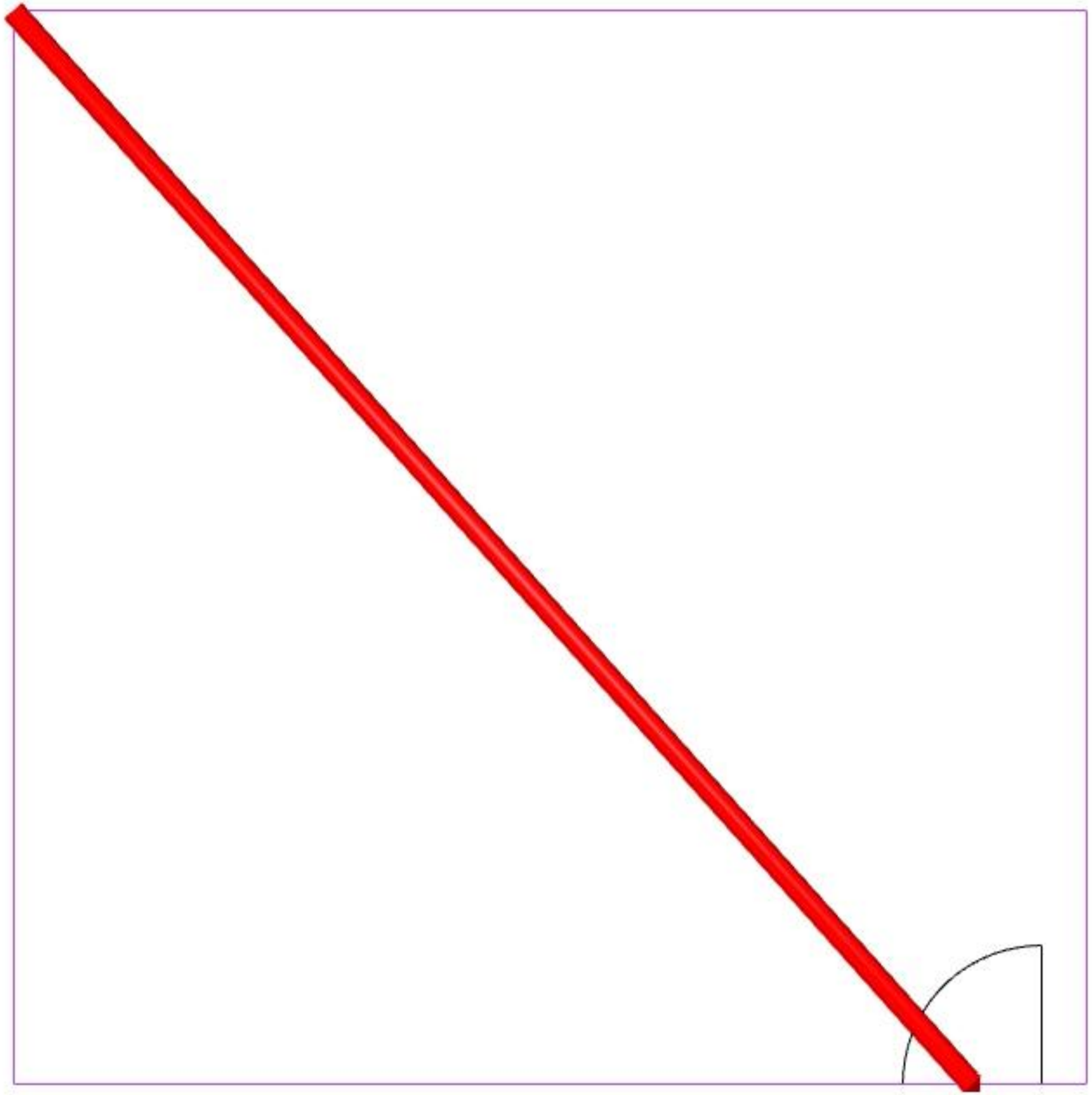
```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:             // Initialization
        dist[v] ← INFINITY                  // Unknown distance from source to v
        prev[v] ← UNDEFINED                 // Previous node in optimal path from source
        add v to Q                          // All nodes initially in Q (unvisited nodes)

    dist[source] ← 0                        // Distance from source to source

    while Q is not empty:
        u ← vertex in Q with min dist[u]    // Node with the least distance
                                            // will be selected first
        remove u from Q

        for each neighbor v of u:           // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:               // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

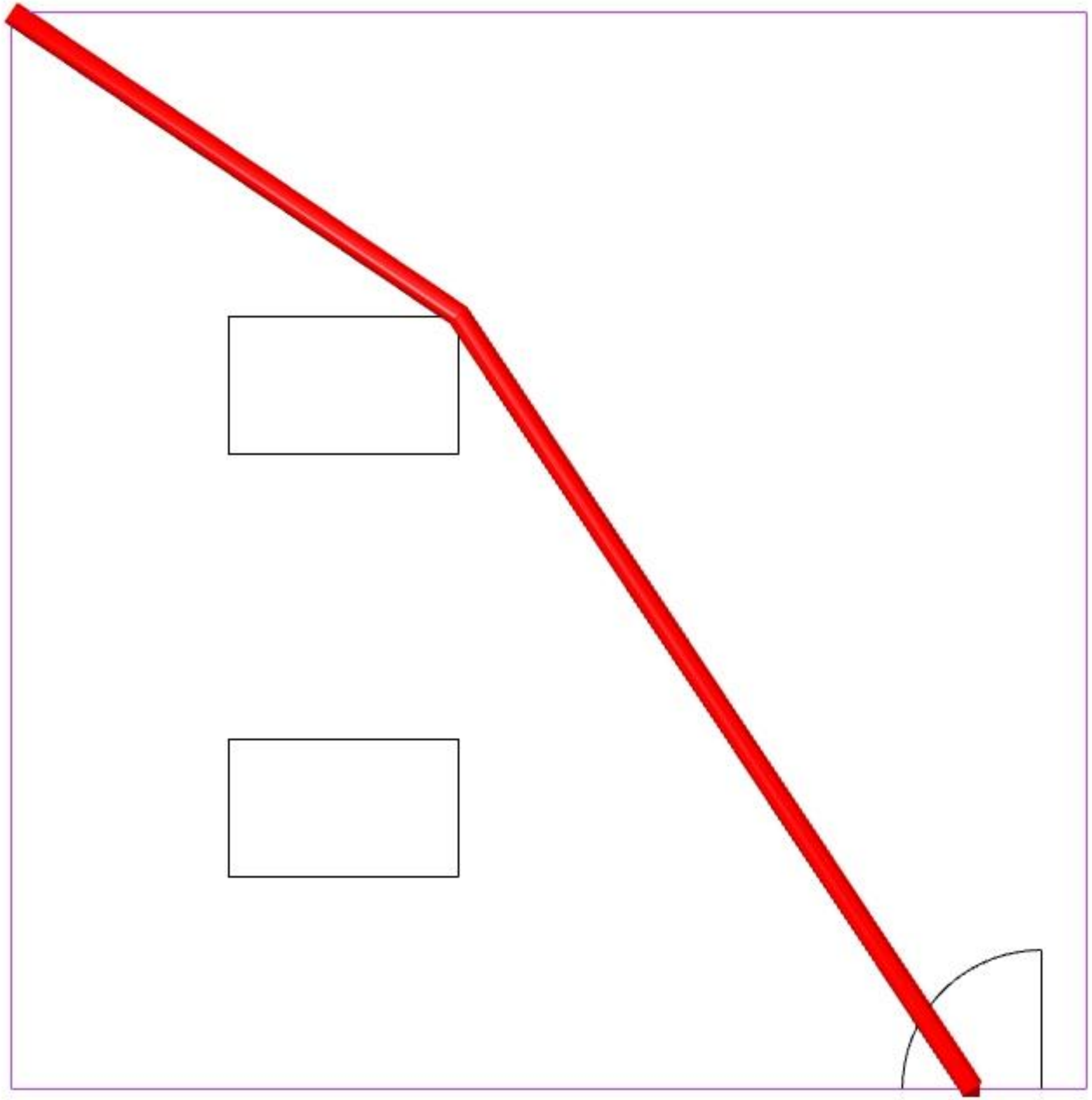
A demo of Dijkstra's algorithm based on Euclidean distance. Red lines are the shortest path covering, i.e., connecting *u* and prev[*u*]. Blue lines indicate where relaxing happens, i.e., connecting *v* with a node *u* in *Q*, which gives a shorter path from the source to *v*.

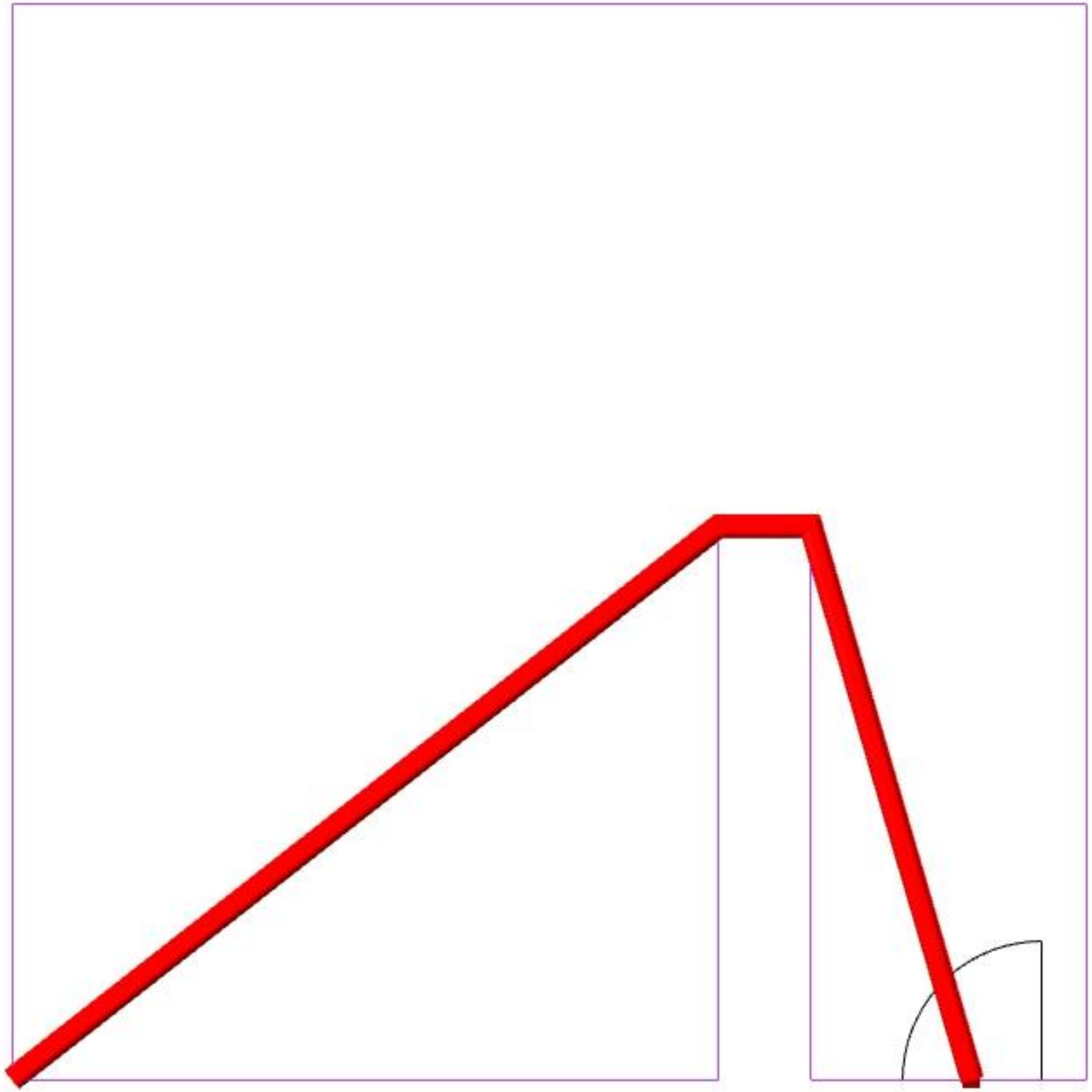
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

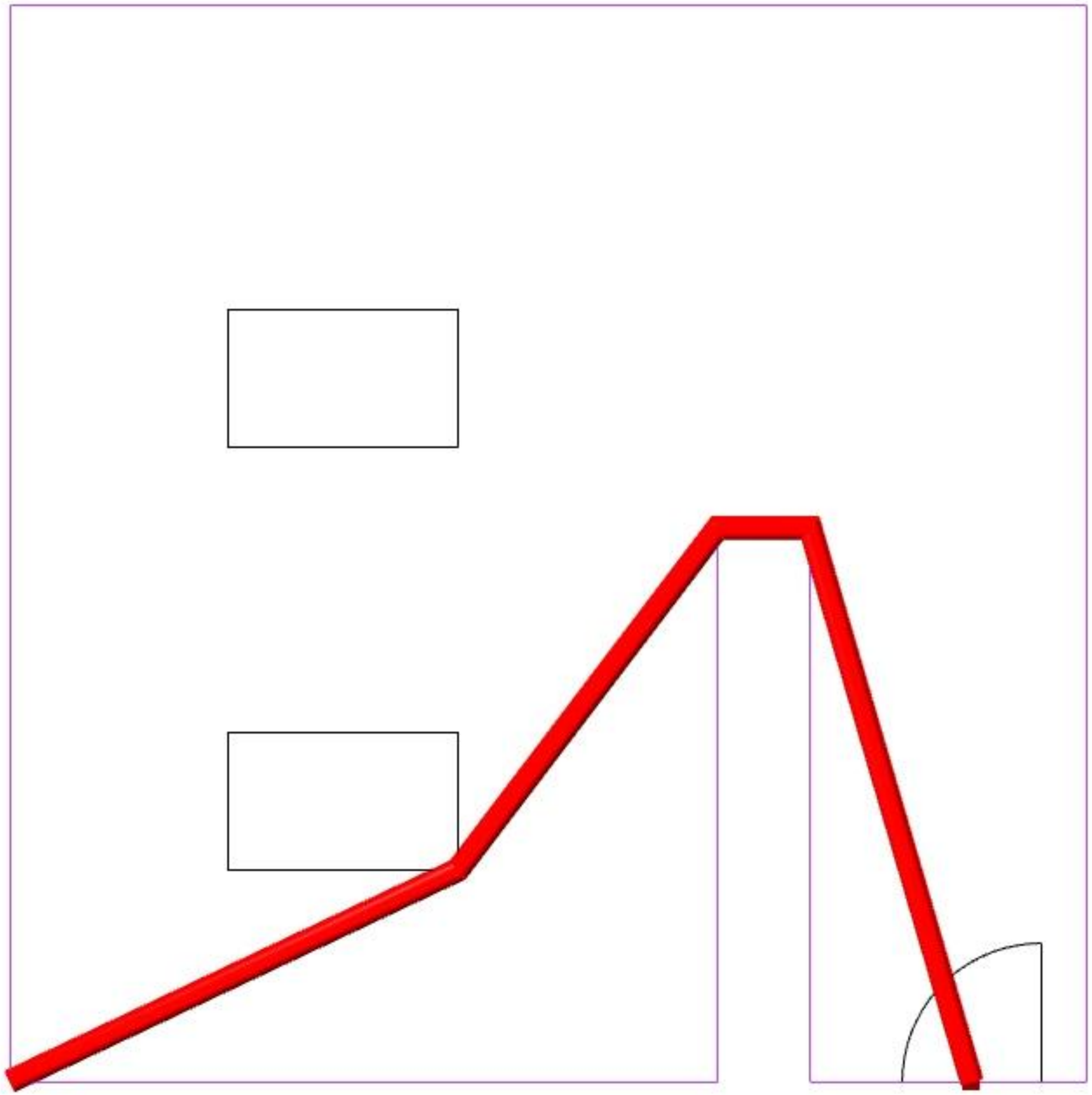


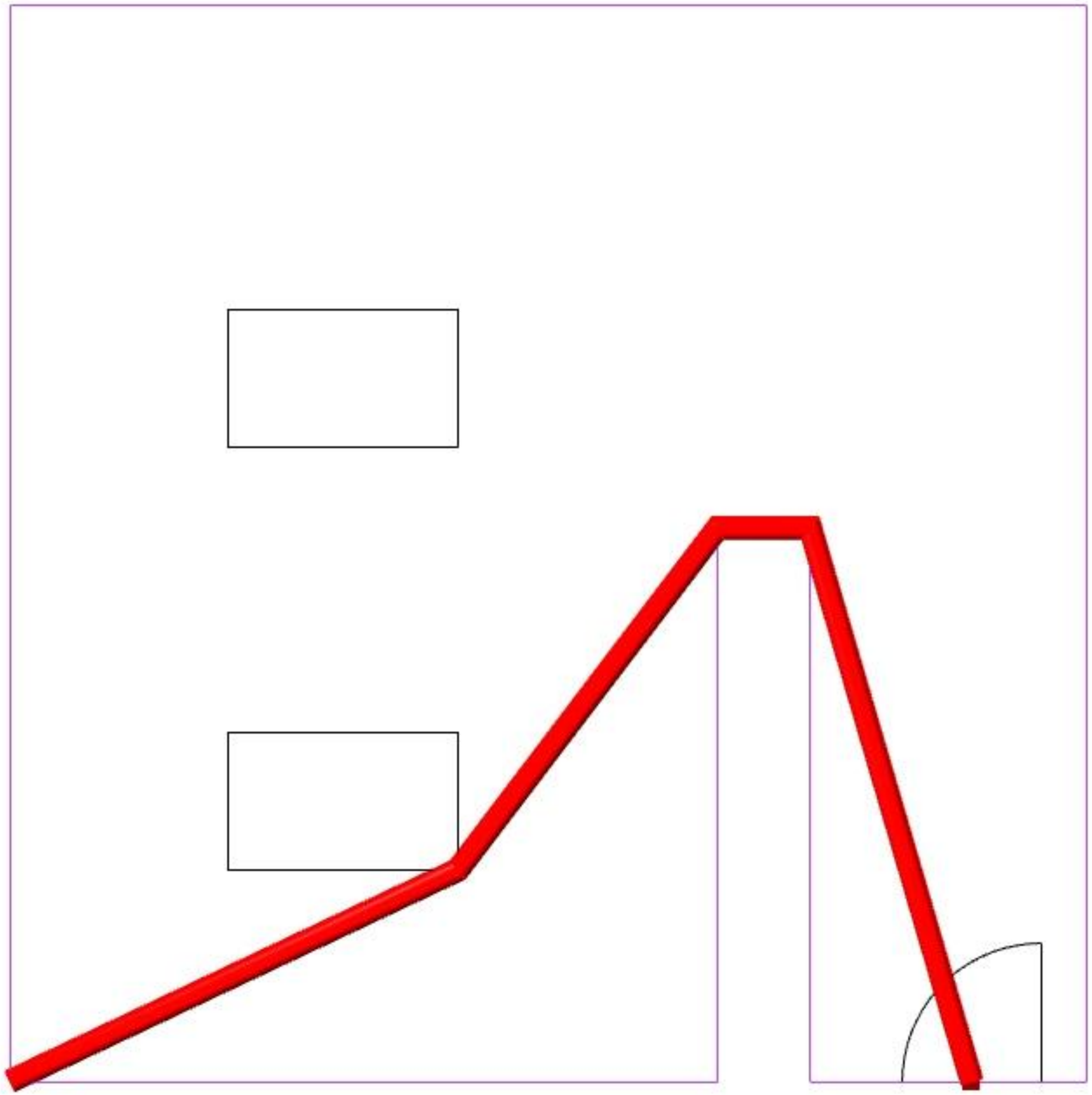




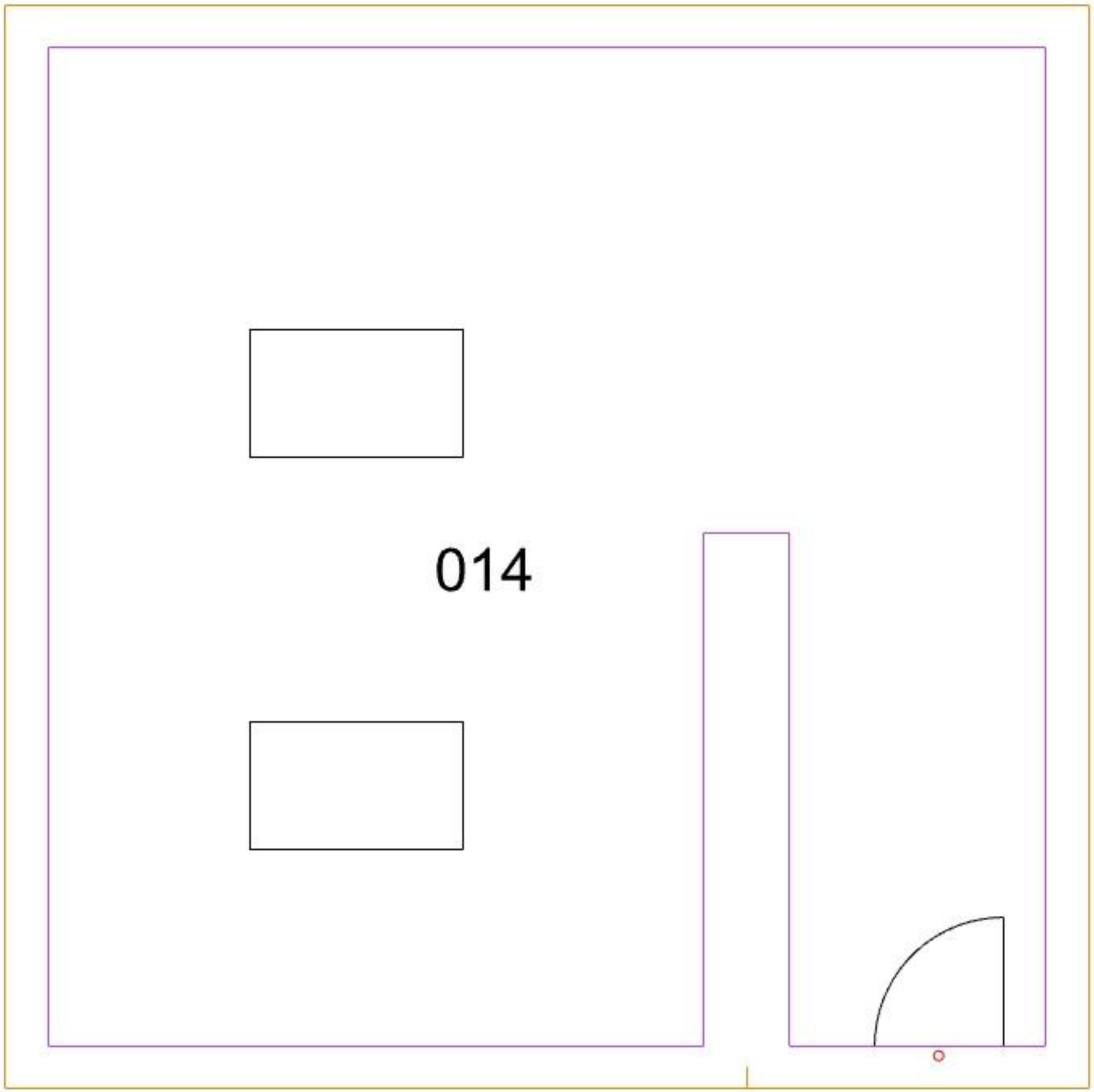






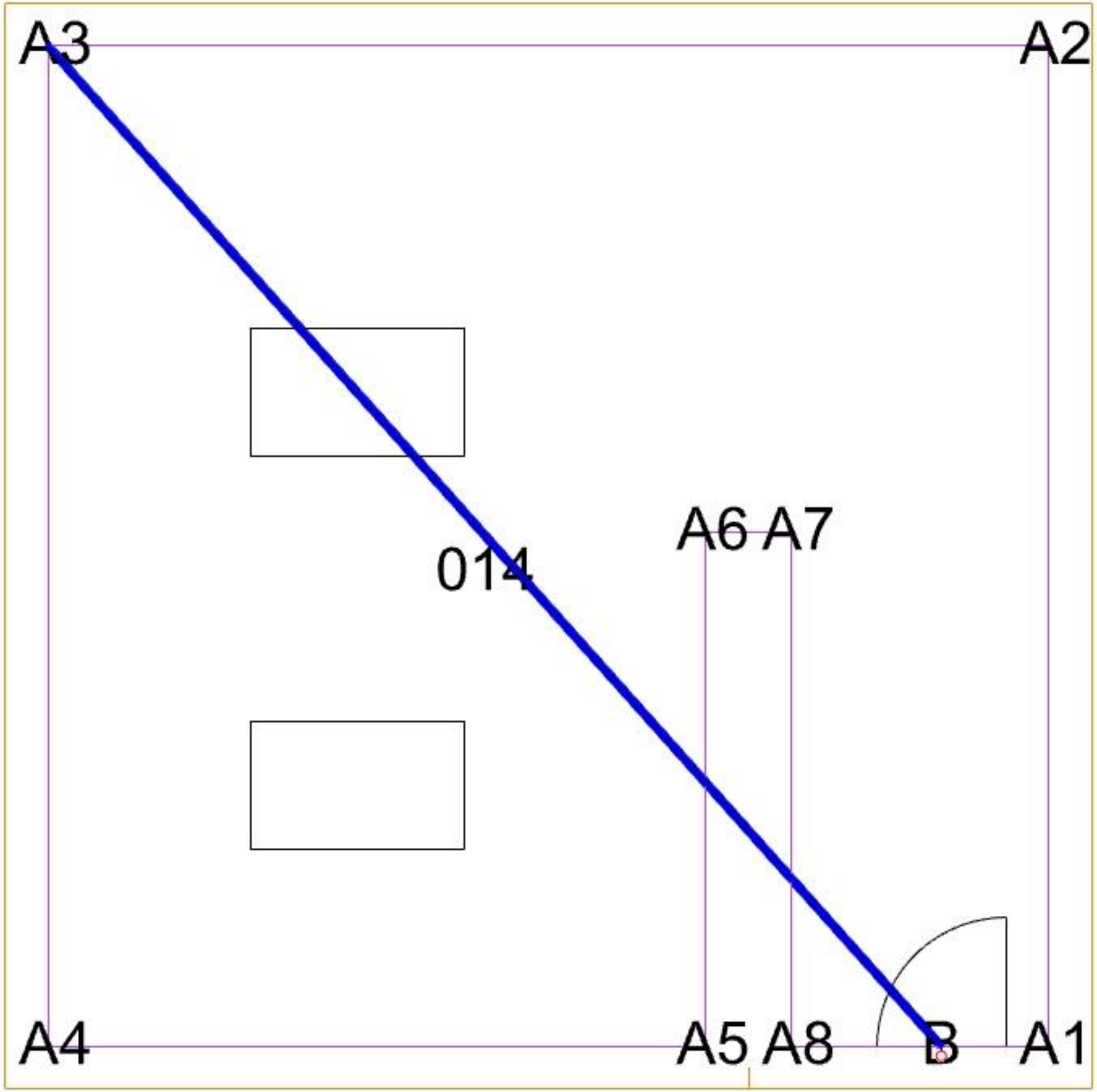


steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):



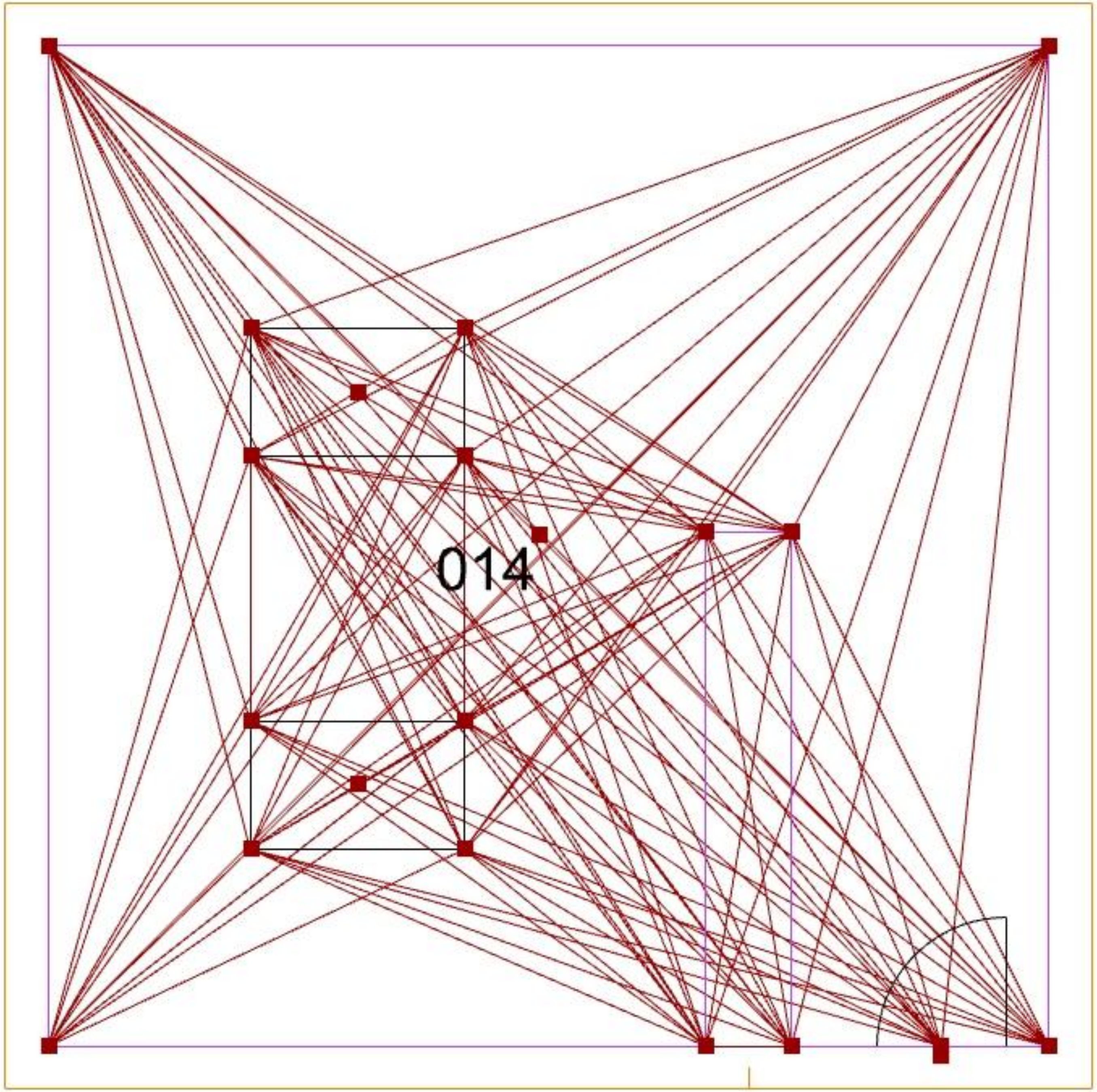
steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):

1. From a list of rooms to process, identify the room bounding polylines which contain those roomnames.
2. Offset each room outline outward. Find the door-points which are included inside each of these offset polylines. (The door-points will typically be slightly outside of the room polyline.)
3. Find any “obstacles” (furniture, columns, etc.) contained within the room polyline.



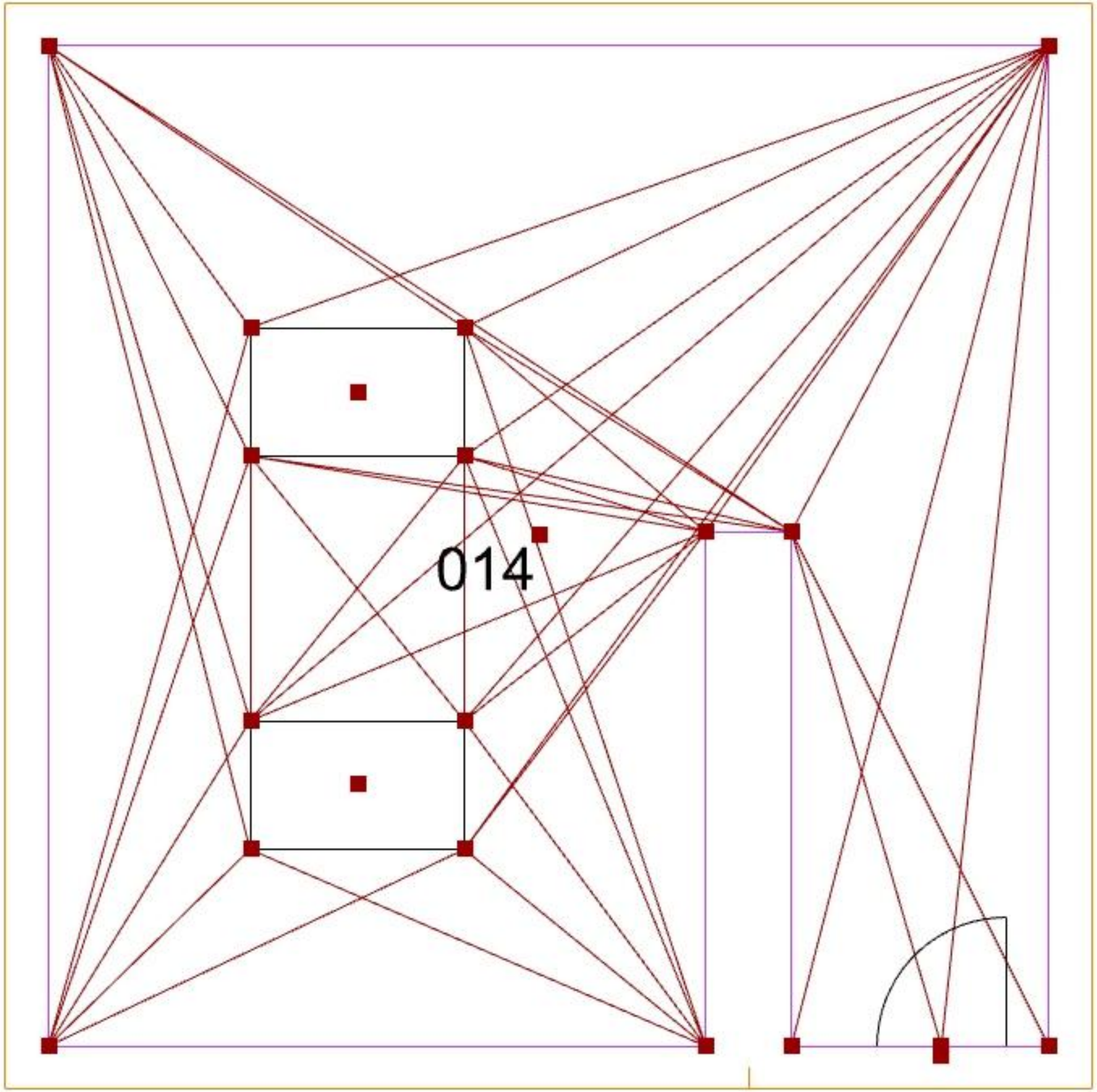
steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):

1. From a list of rooms to process, identify the room bounding polylines which contain those roomnames.
2. Offset each room outline outward. Find the door-points which are included inside each of these offset polylines. (The door-points will typically be slightly outside of the room polyline.)
3. Find any “obstacles” (furniture, columns, etc.) contained within the room polyline.
4. To start the shortest-path algorithm, create lines from each room vertex to the door point (these identify the initial “start” and “end” points for the algorithm).



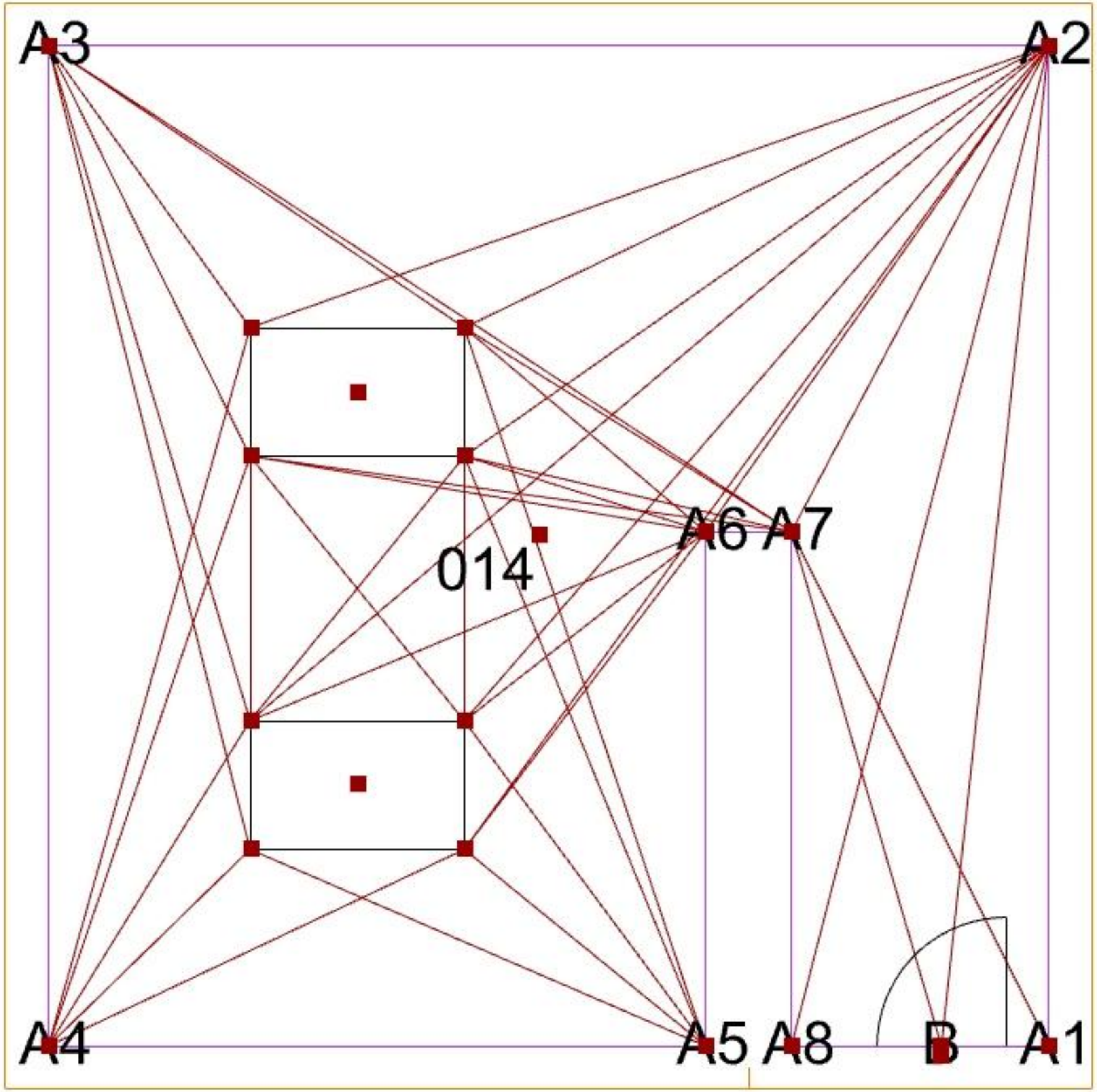
steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):

1. From a list of rooms to process, identify the room bounding polylines which contain those roomnames.
2. Offset each room outline outward. Find the door-points which are included inside each of these offset polylines. (The door-points will typically be slightly outside of the room polyline.)
3. Find any “obstacles” (furniture, columns, etc.) contained within the room polyline.
4. To start the shortest-path algorithm, create lines from each room vertex to the door point (these identify the initial “start” and “end” points for the algorithm).
5. Create lines from each vertex to every other vertex, including obstacle vertices and door points.



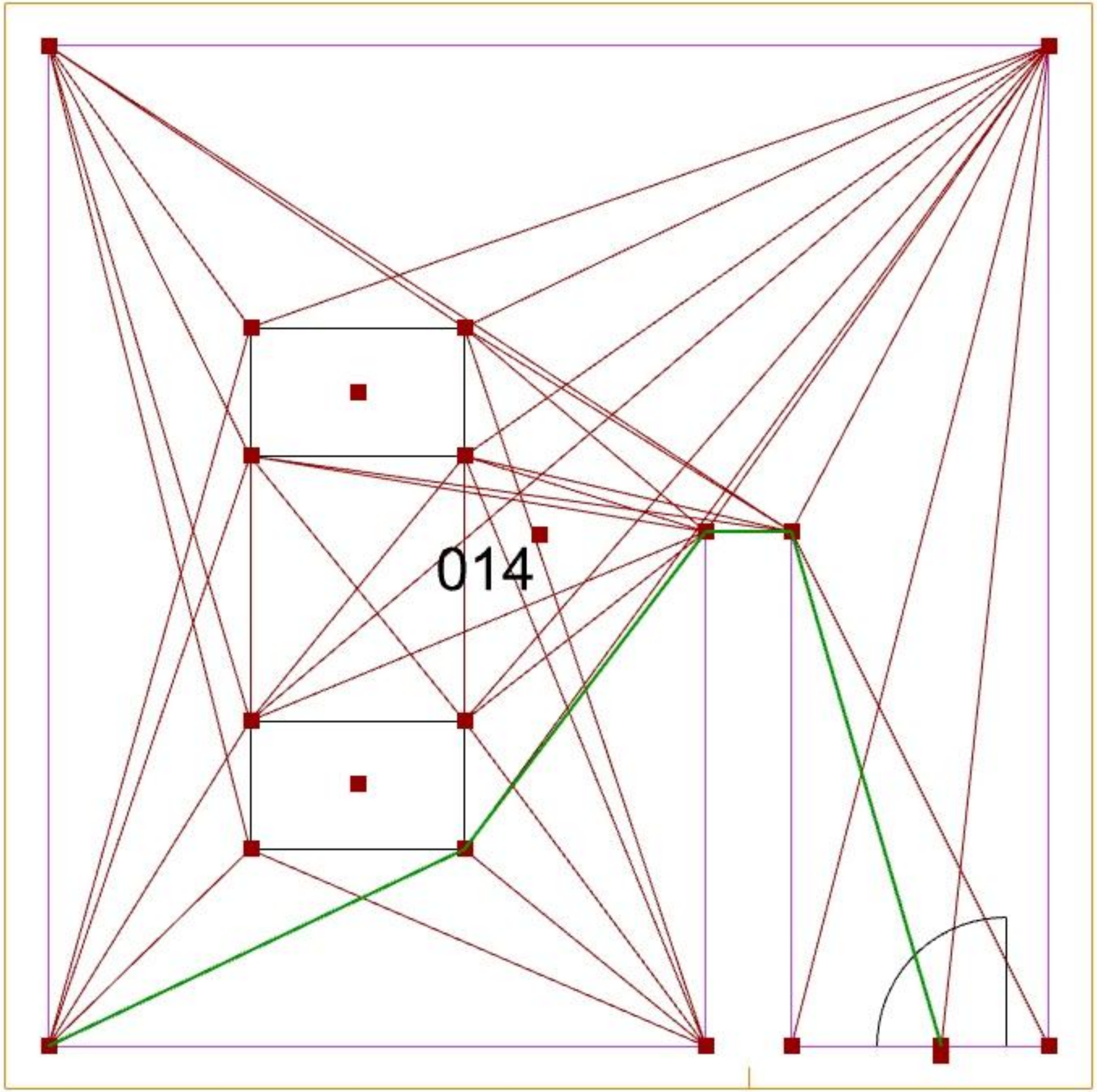
steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):

1. From a list of rooms to process, identify the room bounding polylines which contain those roomnames.
2. Offset each room outline outward. Find the door-points which are included inside each of these offset polylines. (The door-points will typically be slightly outside of the room polyline.)
3. Find any “obstacles” (furniture, columns, etc.) contained within the room polyline.
4. To start the shortest-path algorithm, create lines from each room vertex to the door point (these identify the initial “start” and “end” points for the algorithm).
5. Create lines from each vertex to every other vertex, including obstacle vertices and door points.
6. Cull lines which pass through obstacles or outside of the room polyline.



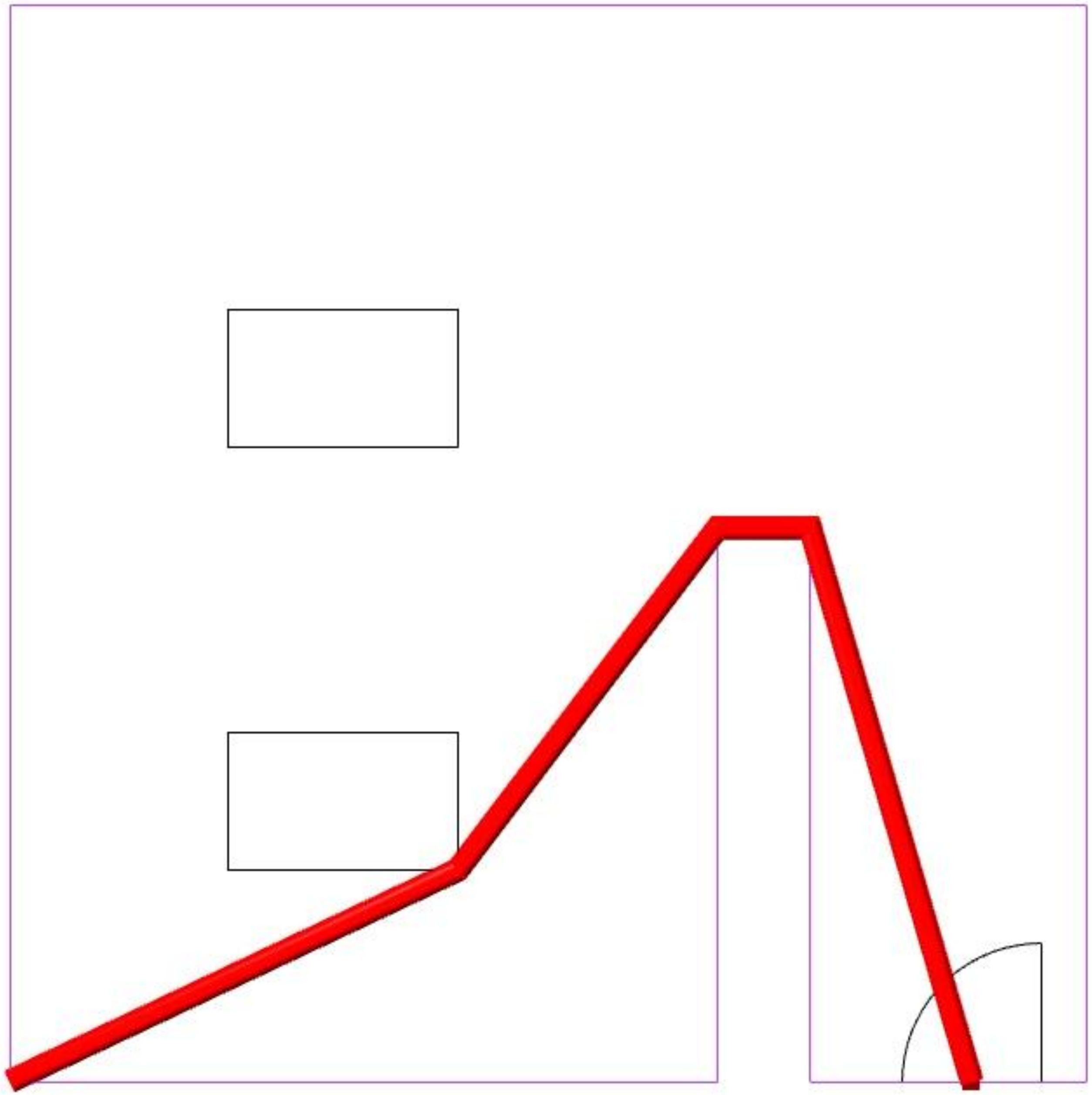
steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):

1. From a list of rooms to process, identify the room bounding polylines which contain those roomnames.
2. Offset each room outline outward. Find the door-points which are included inside each of these offset polylines. (The door-points will typically be slightly outside of the room polyline.)
3. Find any “obstacles” (furniture, columns, etc.) contained within the room polyline.
4. To start the shortest-path algorithm, create lines from each room vertex to the door point (these identify the initial “start” and “end” points for the algorithm).
5. Create lines from each vertex to every other vertex, including obstacle vertices and door points.
6. Cull lines which pass through obstacles or outside of the room polyline.
7. Use the shortest-path algorithm to find the shortest distance from each room vertex to the door.



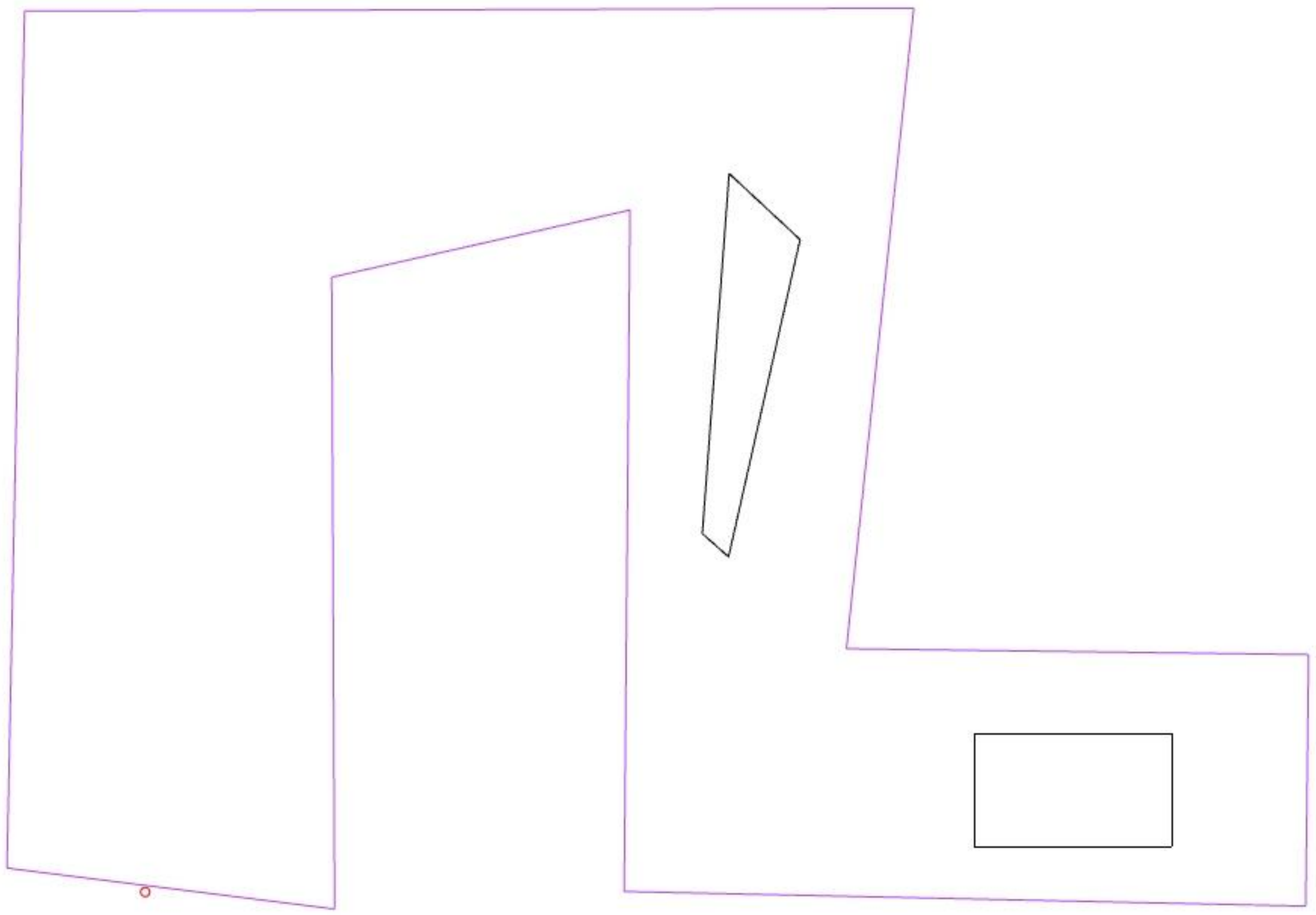
steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):

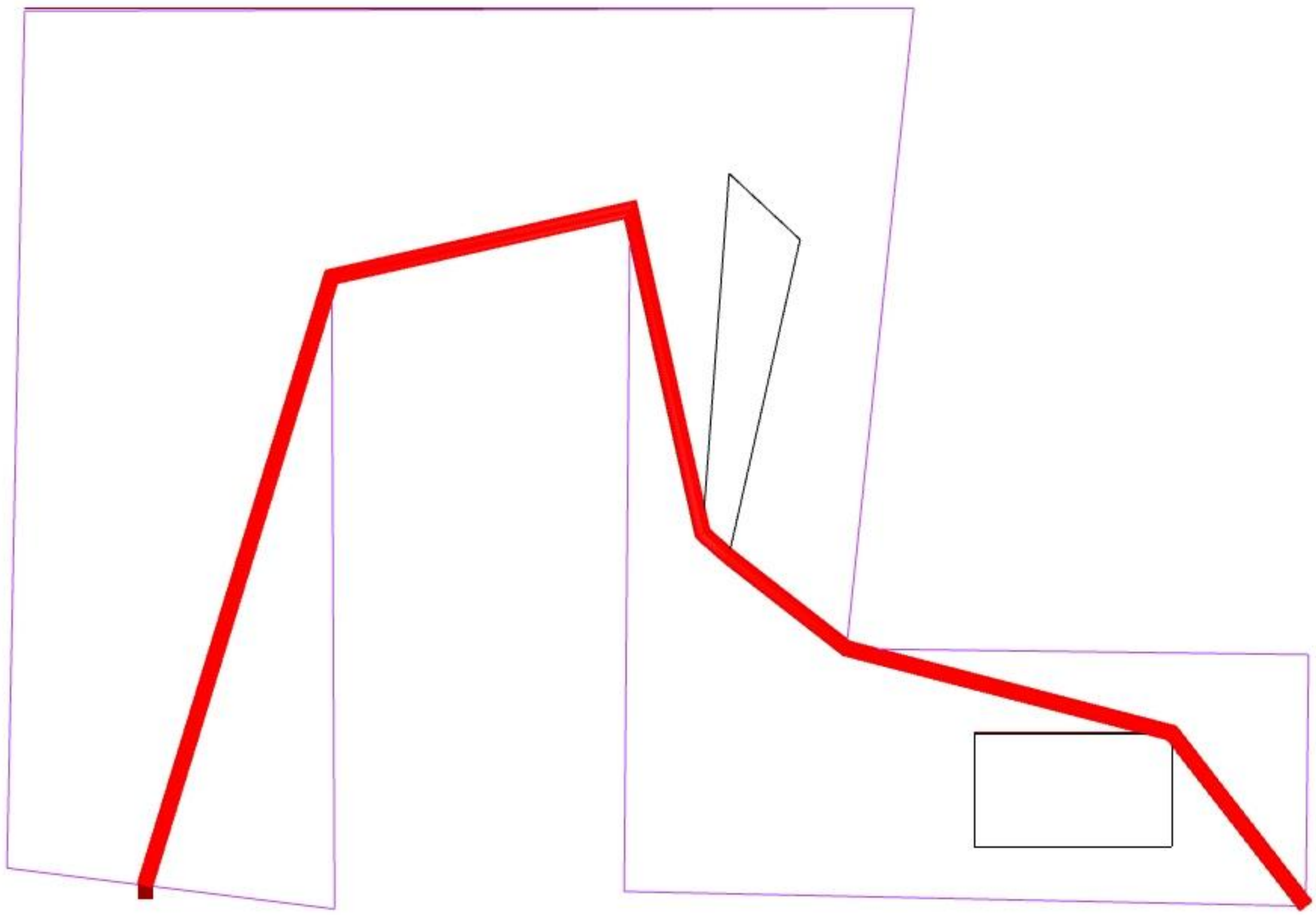
1. From a list of rooms to process, identify the room bounding polylines which contain those roomnames.
2. Offset each room outline outward. Find the door-points which are included inside each of these offset polylines. (The door-points will typically be slightly outside of the room polyline.)
3. Find any “obstacles” (furniture, columns, etc.) contained within the room polyline.
4. To start the shortest-path algorithm, create lines from each room vertex to the door point (these identify the initial “start” and “end” points for the algorithm).
5. Create lines from each vertex to every other vertex, including obstacle vertices and door points.
6. Cull lines which pass through obstacles or outside of the room polyline.
7. Use the shortest-path algorithm to find the shortest distance from each room vertex to the door.
8. Find the LONGEST of these shortest paths (this identifies the location in the room which is farthest from the door).



steps (based on Rhino-created geometry; geometry acquired from Revit is slightly different):

1. From a list of rooms to process, identify the room bounding polylines which contain those roomnames.
2. Offset each room outline outward. Find the door-points which are included inside each of these offset polylines. (The door-points will typically be slightly outside of the room polyline.)
3. Find any “obstacles” (furniture, columns, etc.) contained within the room polyline.
4. To start the shortest-path algorithm, create lines from each room vertex to the door point (these identify the initial “start” and “end” points for the algorithm).
5. Create lines from each vertex to every other vertex, including obstacle vertices and door points.
6. Cull lines which pass through obstacles or outside of the room polyline.
7. Use the shortest-path algorithm to find the shortest distance from each room vertex to the door.
8. Find the LONGEST of these shortest paths (this identifies the location in the room which is farthest from the door).

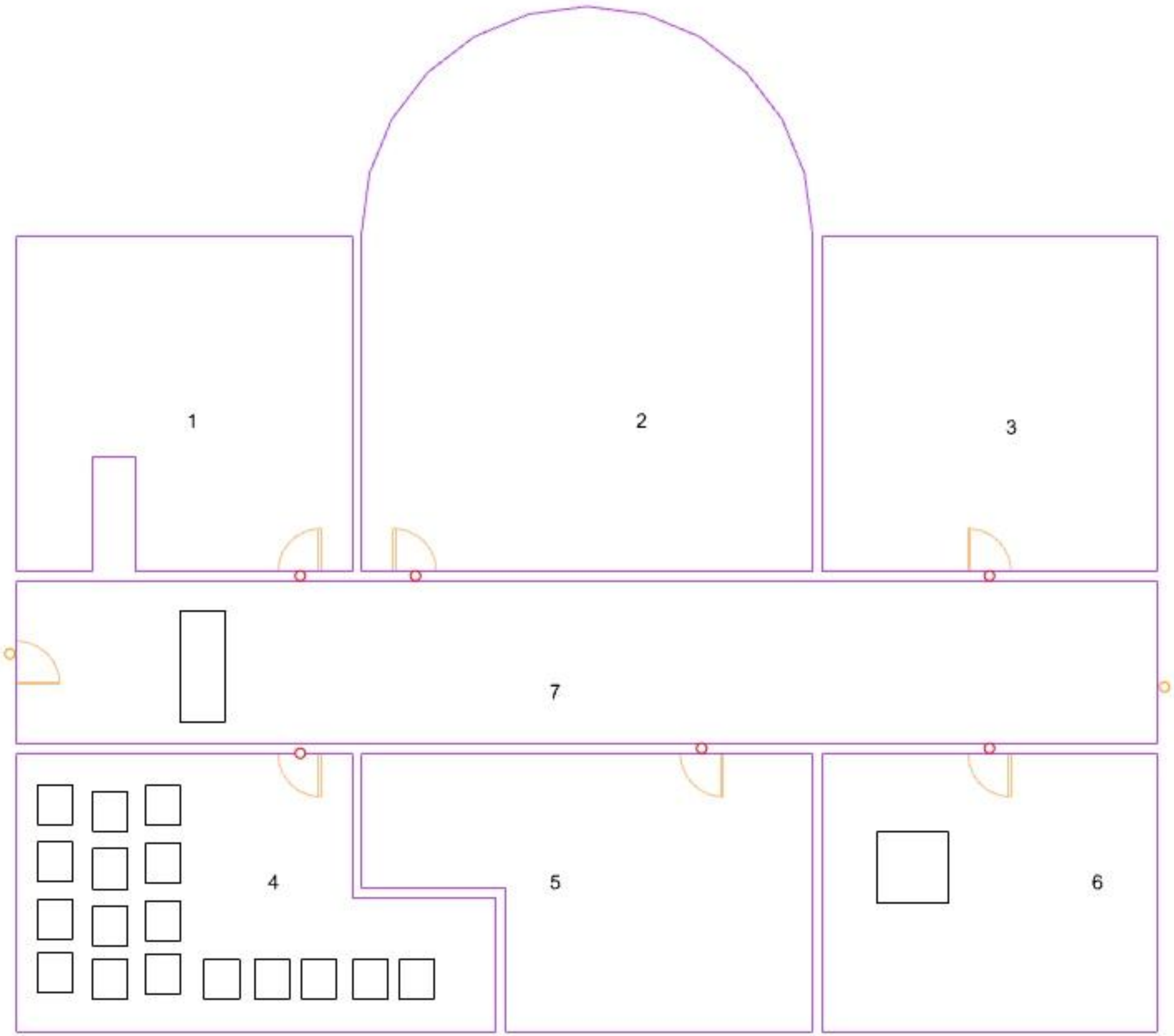




layers:

WALLS
DOORS
door-points
exitdoor-points
FURNITURE
ROOM-NUMBERS

There are two main components to the script: one to analyze regular rooms, and one for “corridors” which lead to an exit.

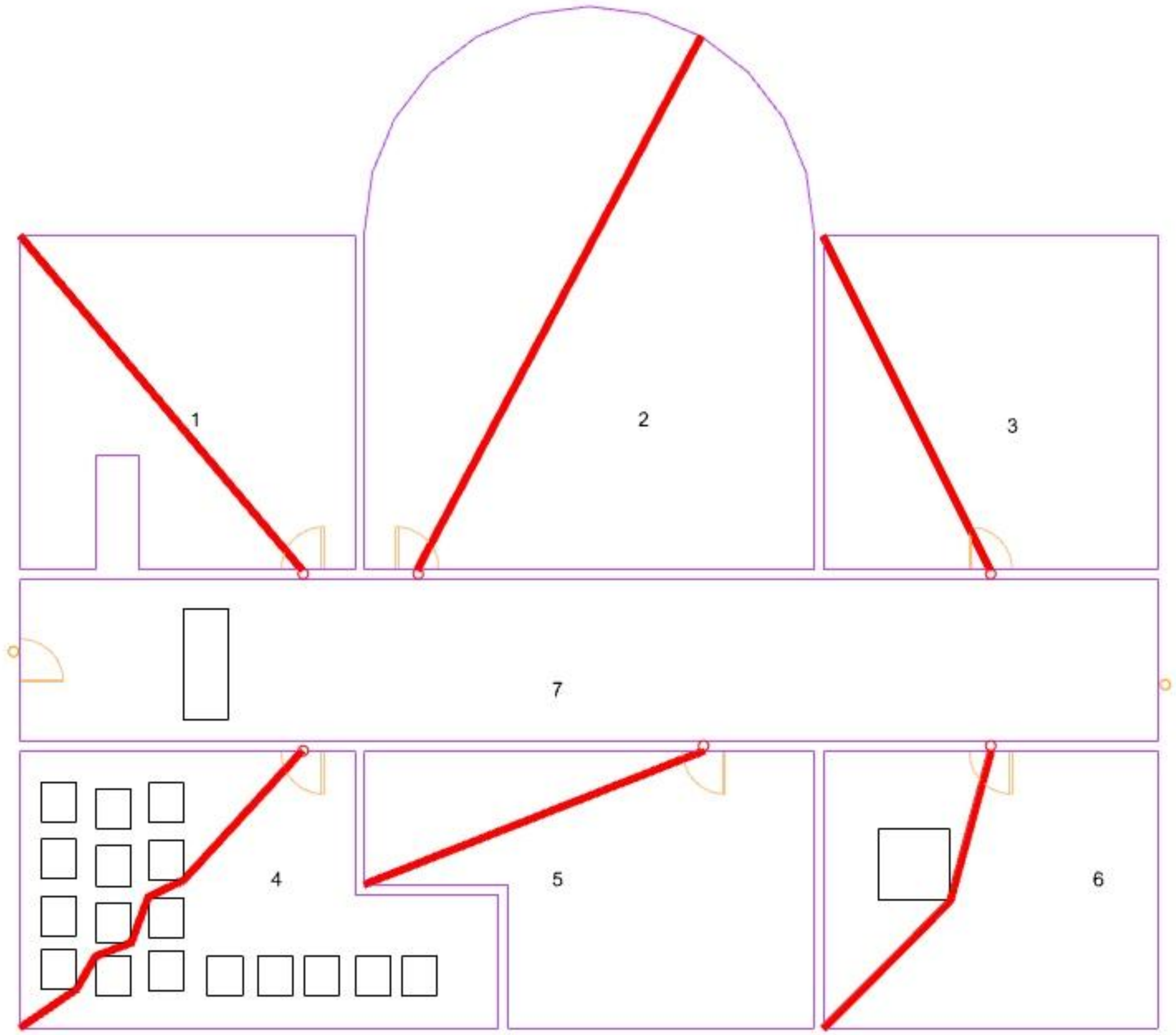


layers:

WALLS
DOORS
door-points
exitdoor-points
FURNITURE
ROOM-NUMBERS

There are two main components to the script: one to analyze regular rooms, and one for “corridors” which lead to an exit.

For regular rooms, the script associates doors with rooms; draws lines between the door and each room vertex, then uses the shortest-path algorithm for each line, computing a shortest path from each vertex. It then finds the longest shortest path.



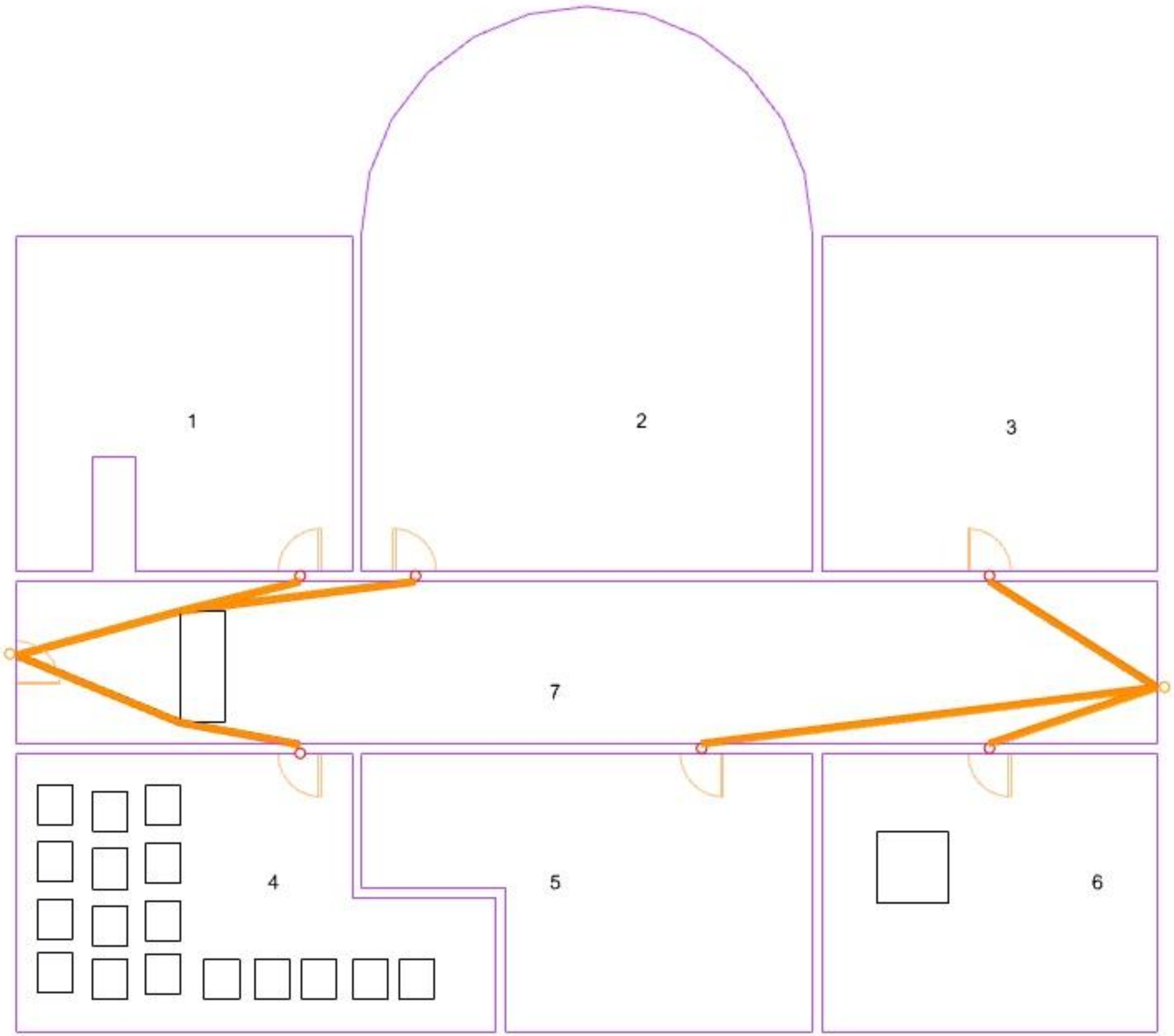
layers:

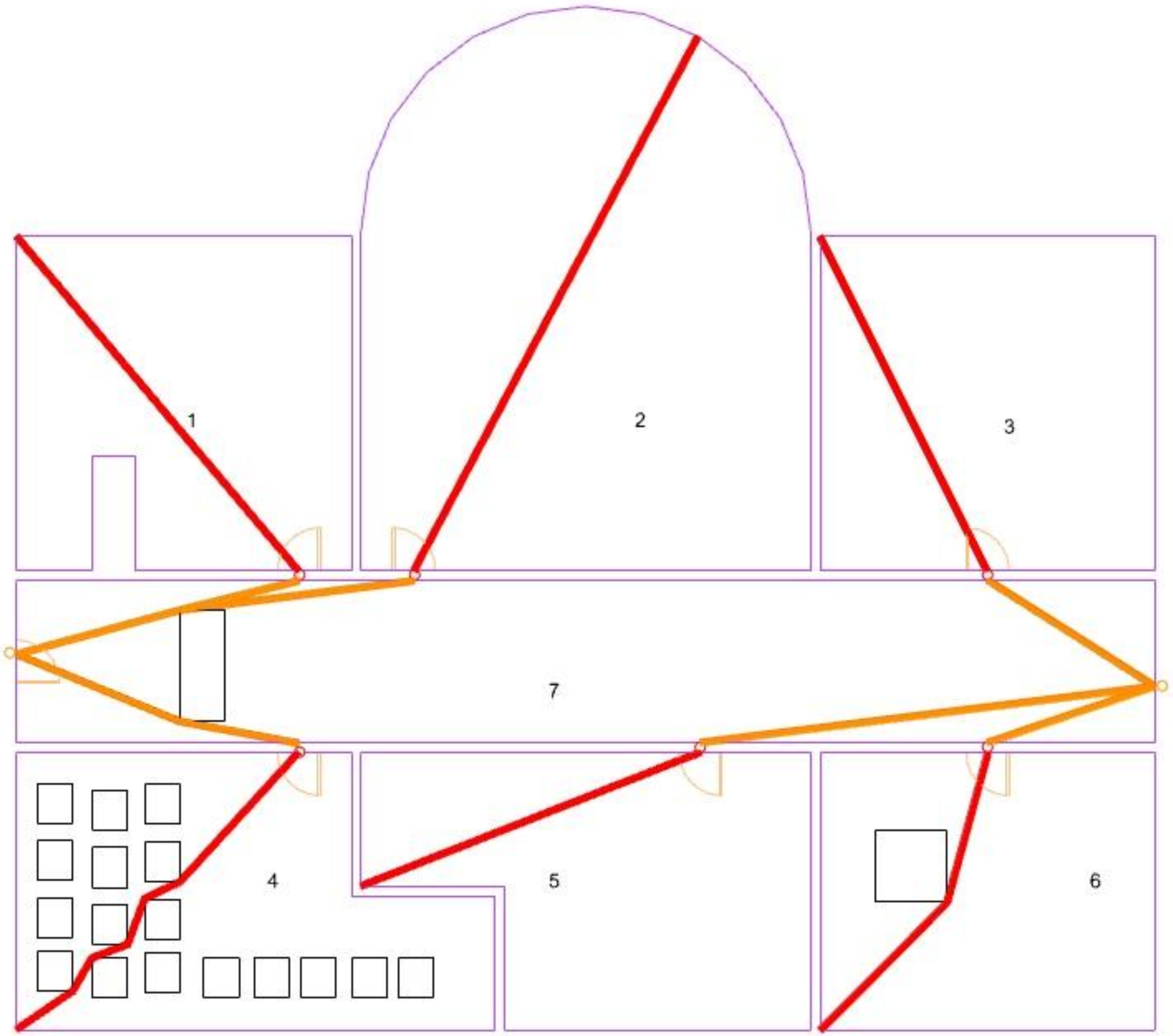
WALLS
DOORS
door-points
exitdoor-points
FURNITURE
ROOM-NUMBERS

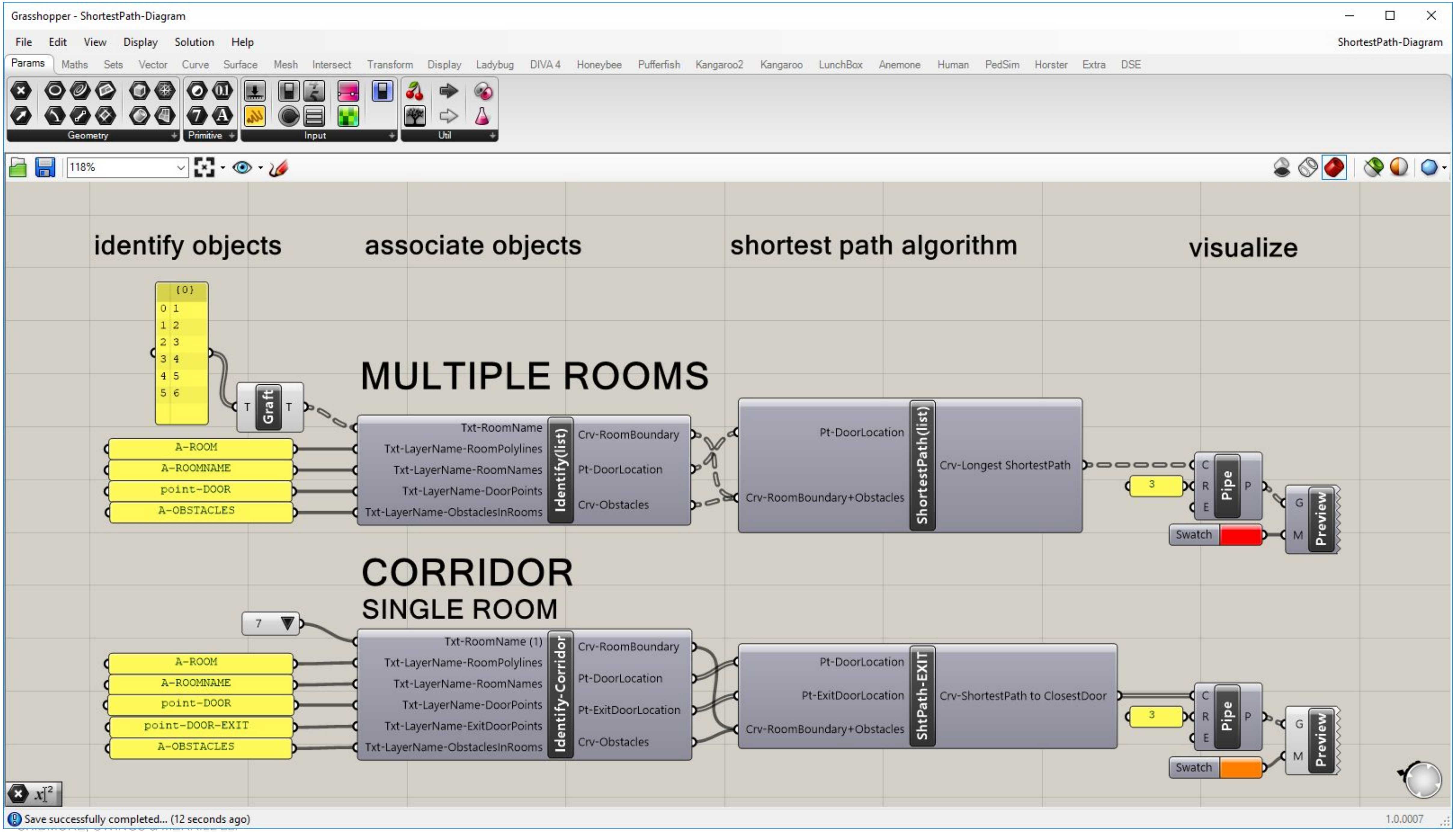
There are two main components to the script: one to analyze regular rooms, and one for “corridors” which lead to an exit.

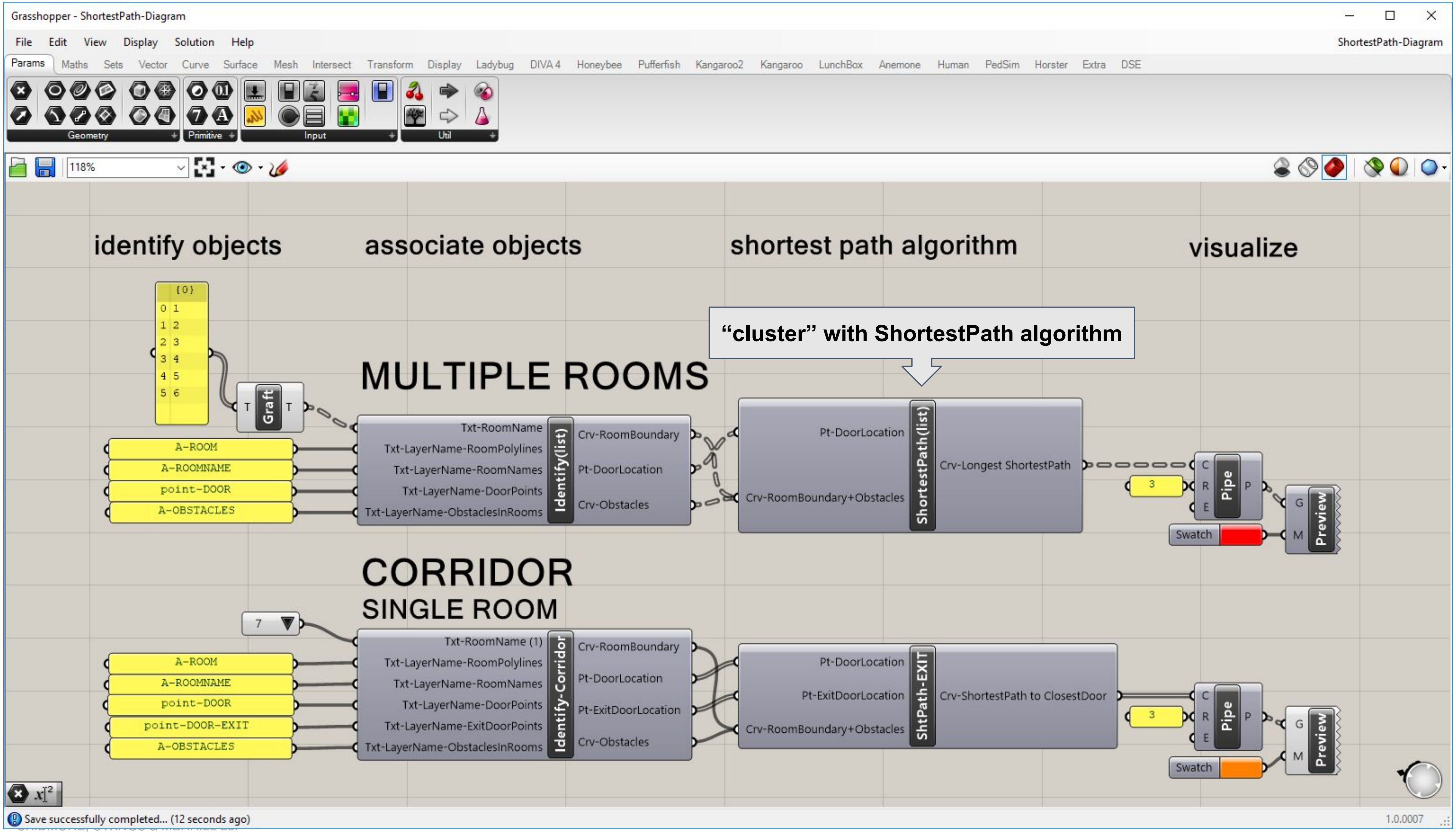
For regular rooms, the script associates doors with rooms; draws lines between the door and each room vertex, then uses the shortest-path algorithm for each line, computing a shortest path from each vertex. It then finds the longest shortest path.

For corridors, it finds the shortest path from each “room door” to each “exit door”. “Obstacles” are found by containment in each room.

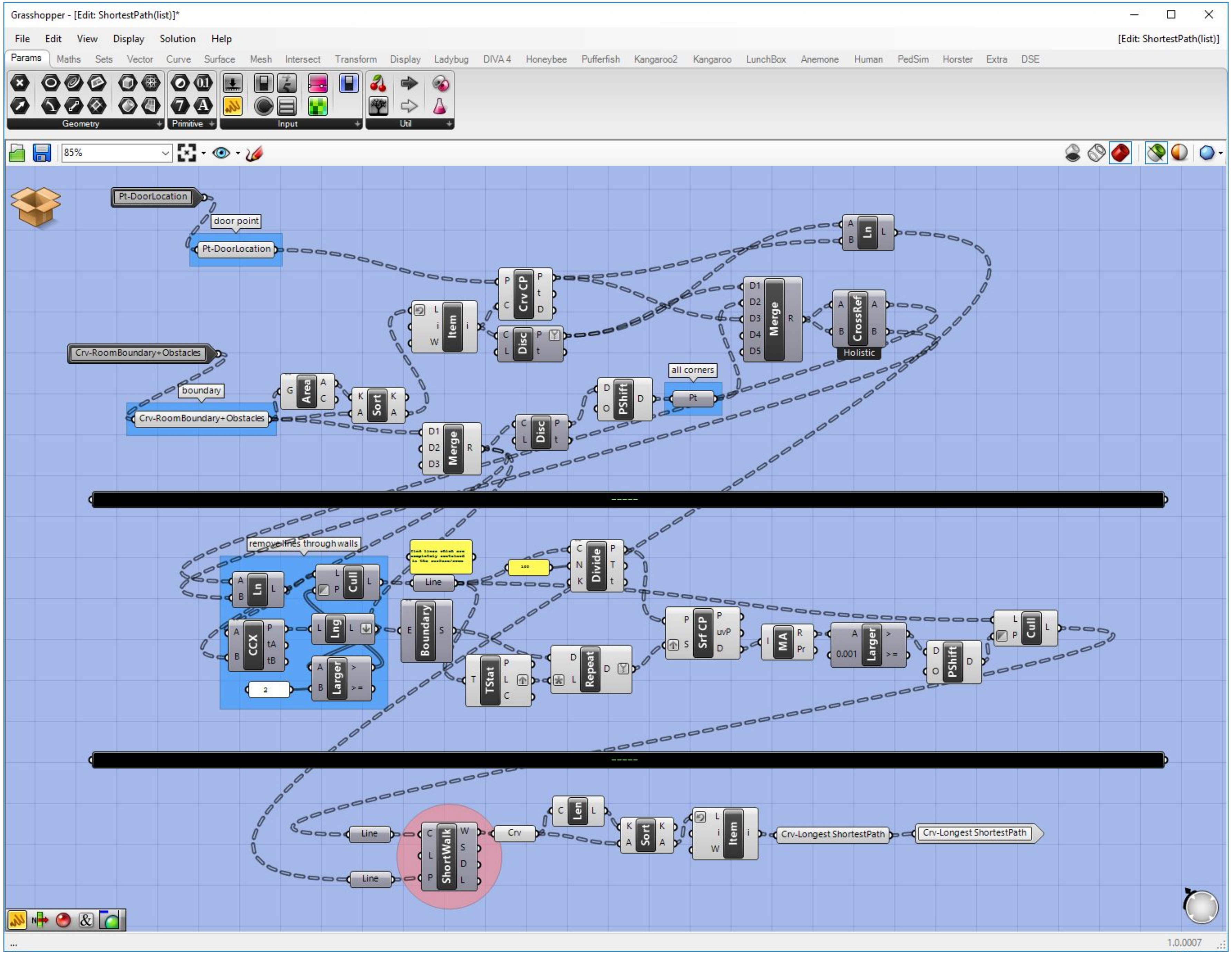



1
2
3
7
4
5
6


Grasshopper - ShortestPath-Diagram
ShortestPath-Diagram
identify objects
associate objects
shortest path algorithm
visualize
MULTIPLE ROOMS
Graft
A-ROOM
A-ROOMNAME
point-DOOR
A-OBSTACLES
Txt-RoomName
Txt-LayerName-RoomPolylines
Txt-LayerName-RoomNames
Txt-LayerName-DoorPoints
Txt-LayerName-ObstaclesInRooms
Identify(list)
Crv-RoomBoundary
Pt-DoorLocation
Crv-Obstacles
Pt-DoorLocation
Crv-RoomBoundary+Obstacles
ShortestPath(list)
Crv-Longest ShortestPath
Pipe
Swatch
Preview
CORRIDOR
SINGLE ROOM
A-ROOM
A-ROOMNAME
point-DOOR
point-DOOR-EXIT
A-OBSTACLES
Txt-RoomName (1)
Txt-LayerName-RoomPolylines
Txt-LayerName-RoomNames
Txt-LayerName-DoorPoints
Txt-LayerName-ExitDoorPoints
Txt-LayerName-ObstaclesInRooms
Identify-Corridor
Crv-RoomBoundary
Pt-DoorLocation
Pt-ExitDoorLocation
Crv-Obstacles
Pt-DoorLocation
Pt-ExitDoorLocation
Crv-RoomBoundary+Obstacles
ShtPath-EXIT
Crv-ShortestPath to ClosestDoor
Pipe
Swatch
Preview
Save successfully completed... (12 seconds ago)
1.0.0007



Grasshopper - ShortestPath-Diagram
identify objects
associate objects
shortest path algorithm
visualize
"cluster" with ShortestPath algorithm
MULTIPLE ROOMS
CORRIDOR
SINGLE ROOM
A-ROOM
A-ROOMNAME
point-DOOR
A-OBSTACLES
point-DOOR-EXIT
Graft
Identify(list)
Identify-Corridor
ShortestPath(list)
ShtPath-EXIT
Txt-RoomName
Txt-RoomName (1)
Txt-LayerName-RoomPolylines
Txt-LayerName-RoomNames
Txt-LayerName-DoorPoints
Txt-LayerName-ExitDoorPoints
Txt-LayerName-ObstaclesInRooms
Crv-RoomBoundary
Pt-DoorLocation
Pt-ExitDoorLocation
Crv-Obstacles
Crv-RoomBoundary+Obstacles
Crv-Longest ShortestPath
Crv-ShortestPath to ClosestDoor
Pipe
Preview
Swatch
Save successfully completed... (12 seconds ago)



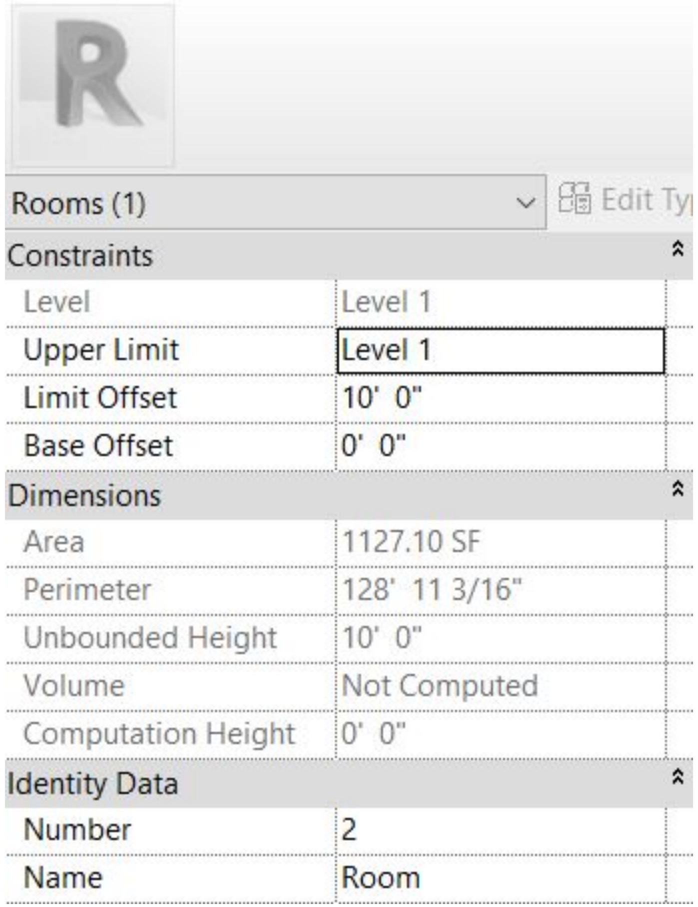
Grasshopper - [Edit: ShortestPath(list)]*
File Edit View Display Solution Help
[Edit: ShortestPath(list)]
Params Maths Sets Vector Curve Surface Mesh Intersect Transform Display Ladybug DIVA 4 Honeybee Pufferfish Kangaroo2 Kangaroo LunchBox Anemone Human PedSim Horster Extra DSE
Geometry
Primitive
Input
Util
85%
Pt-DoorLocation
door point
Crv-RoomBoundary+Obstacles
boundary
all corners
remove lines through walls
Crv-Longest ShortestPath

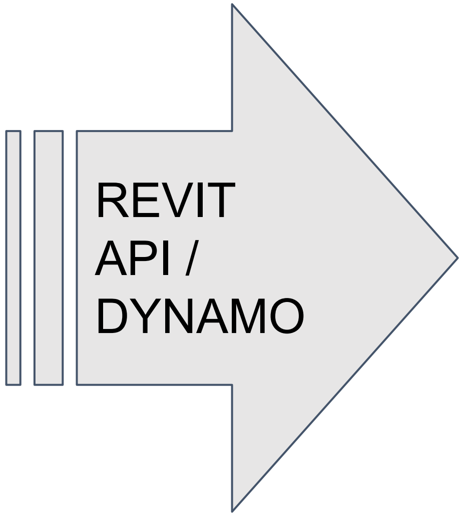
FORMATTING DATA FOR INTEROPERABILITY

*…**JavaScript Object Notation** or **JSON** … is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types …**

*https://en.wikipedia.org/wiki/JSON

FORMATTING DATA FOR INTEROPERABILITY





KEY-VALUE PAIR

ARRAY

{

"category":"Rooms",

"uid":"5b52def26b744e35a0484e5e3b6c18f40004f2c4",

"name":"Room 1",

"mark":"1",

"projectId":"1Vnj5QDPP2_gMJiD2jYtrG",

"levelName":"Level 1",

"location":**{**"**Z**":0,"**Y**":15.881751771605,"**X**":12.673566058023**}**,

"svgPaths":**[**"M0.666666666666667,0.666666666666667

26.3333333333333,0.666666666666667 6.3333333333334,

36 0.666666666666668,36

0.666666666666667,0.666666666666664Z"

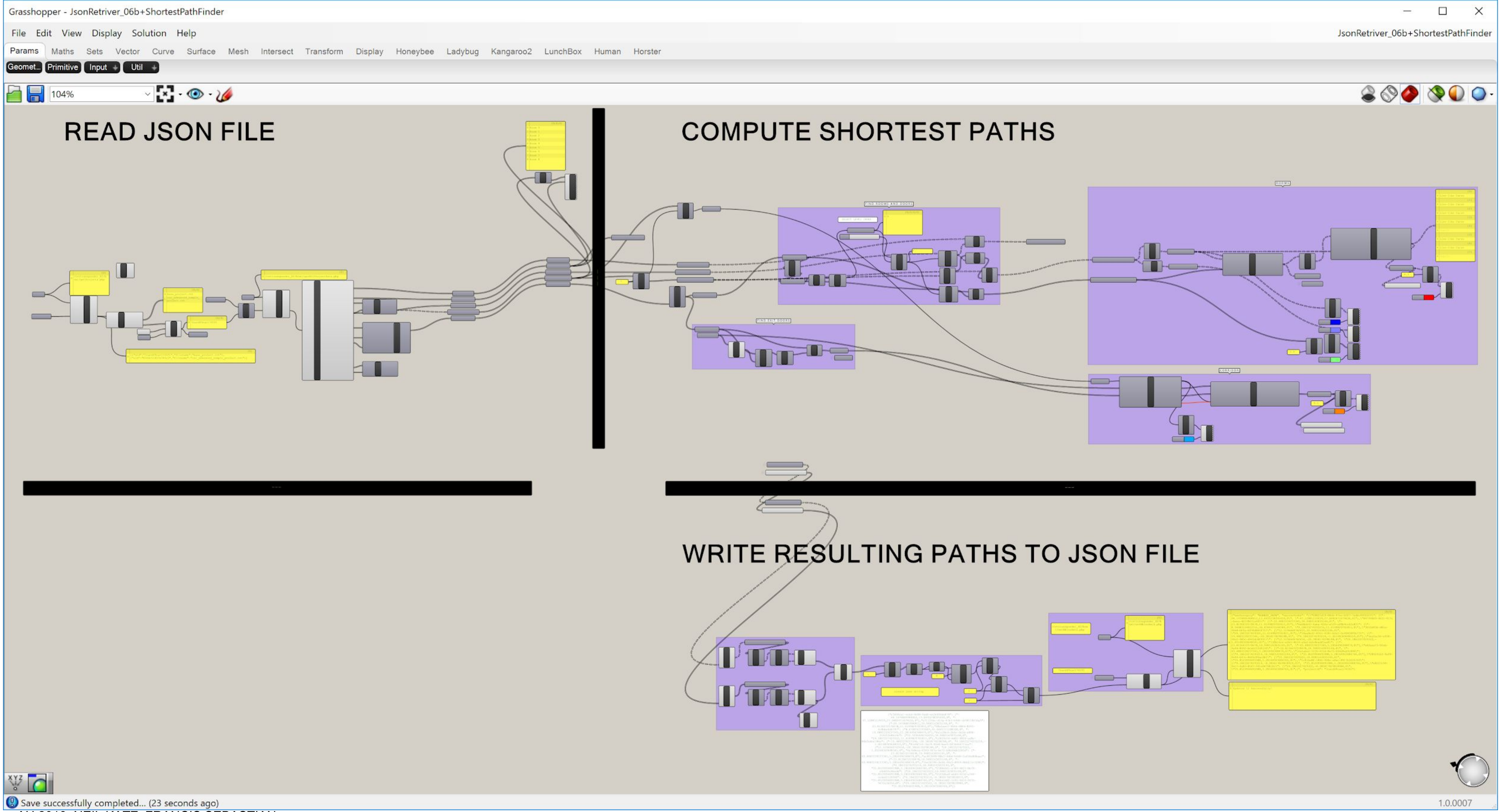
],

"properties":null

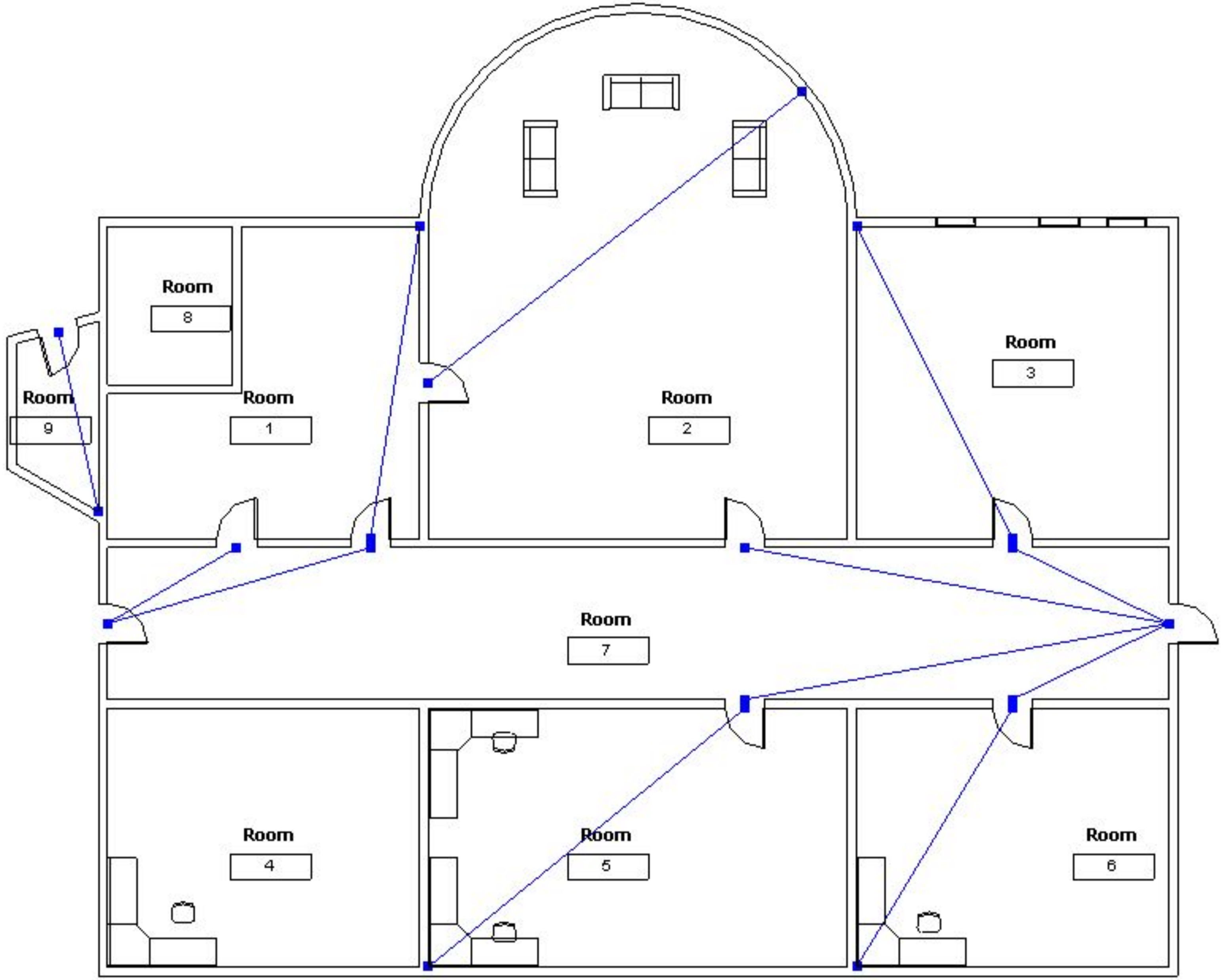
}

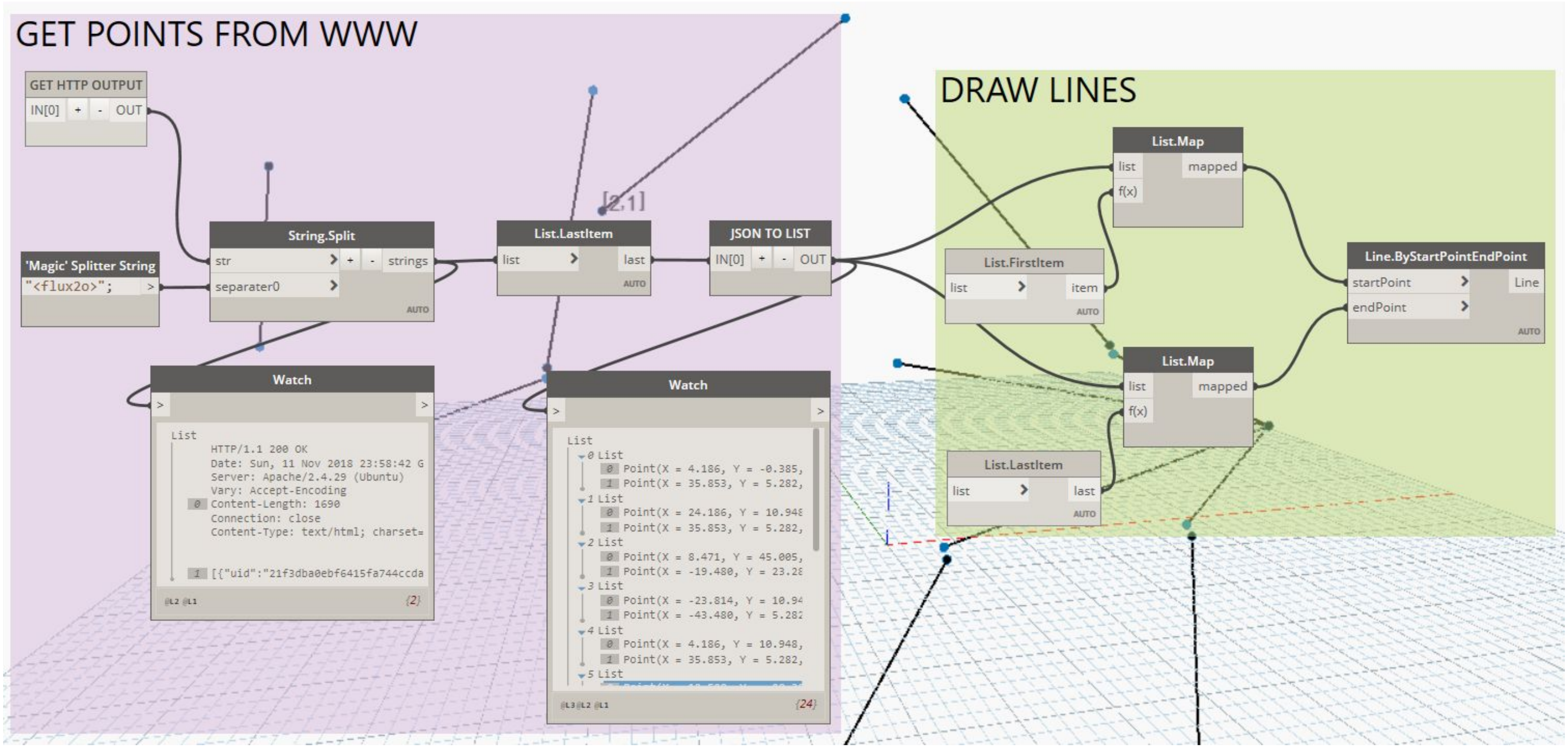
AU 2018; NEIL KATZ, FRANCIS SEBASTIAN

SKIDMORE, OWINGS & MERRILL LLP


Grasshopper - JsonRetriver_06b+ShortestPathFinder
File Edit View Display Solution Help
JsonRetriver_06b+ShortestPathFinder
Params Maths Sets Vector Curve Surface Mesh Intersect Transform Display Honeybee Ladybug Kangaroo2 LunchBox Human Horster
104%
READ JSON FILE
COMPUTE SHORTEST PATHS
WRITE RESULTING PATHS TO JSON FILE
Save successfully completed... (23 seconds ago)
1.0.0007

THE ROUNDTRIP



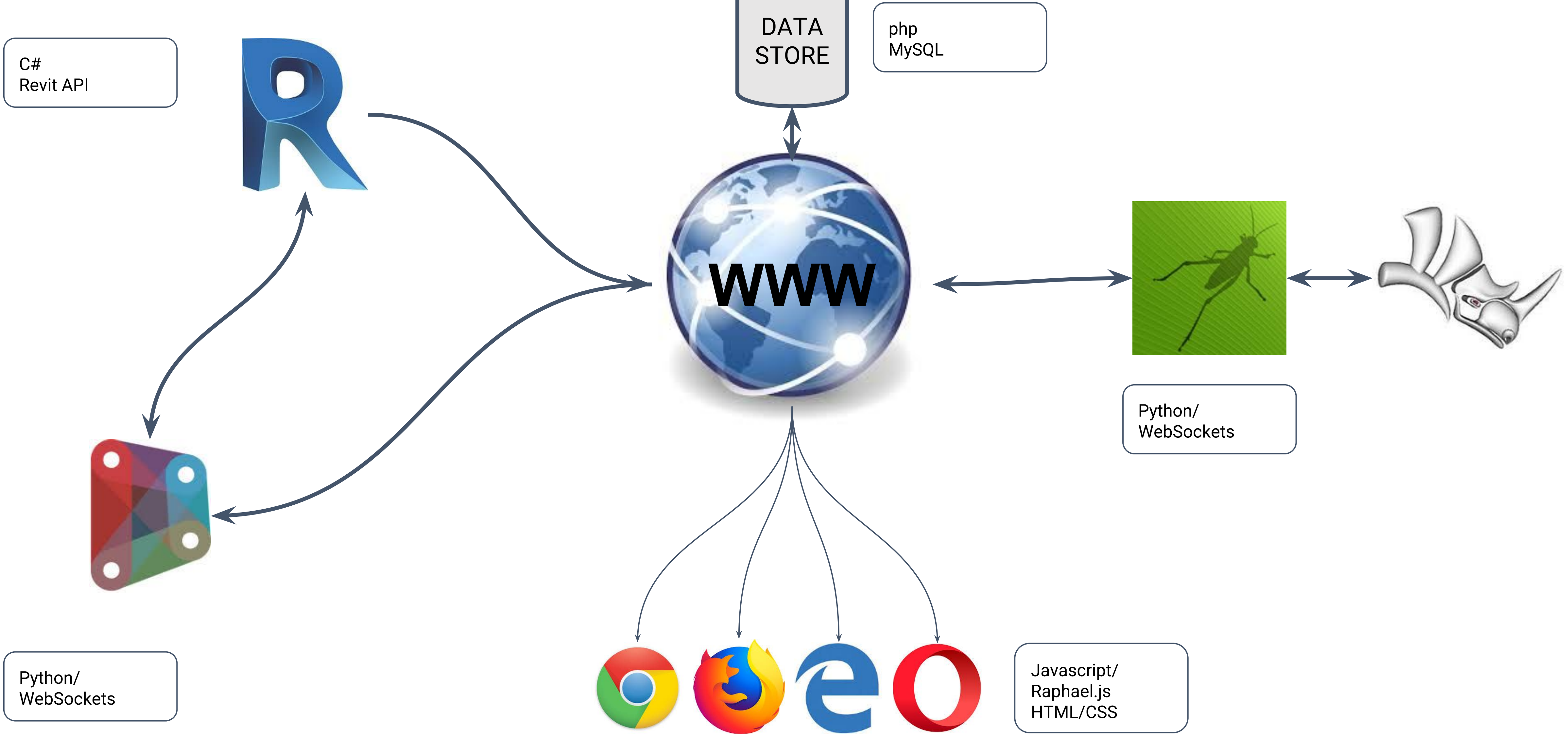

GET POINTS FROM WWW
GET HTTP OUTPUT
IN[0]
OUT
'Magic' Splitter String
"<flux2o>";
String.Split
str
separater0
strings
AUTO
List.LastItem
list
last
AUTO
[2,1]
JSON TO LIST
IN[0]
OUT
Watch
List
HTTP/1.1 200 OK
Date: Sun, 11 Nov 2018 23:58:42 G
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
0 Content-Length: 1690
Connection: close
Content-Type: text/html; charset=
1 [{"uid":"21f3dba0ebf6415fa744ccda
@L2 @L1
{2}
Watch
List
0 List
0 Point(X = 4.186, Y = -0.385,
1 Point(X = 35.853, Y = 5.282,
1 List
0 Point(X = 24.186, Y = 10.948
1 Point(X = 35.853, Y = 5.282,
2 List
0 Point(X = 8.471, Y = 45.005,
1 Point(X = -19.480, Y = 23.28
3 List
0 Point(X = -23.814, Y = 10.94
1 Point(X = -43.480, Y = 5.282
4 List
0 Point(X = 4.186, Y = 10.948,
1 Point(X = 35.853, Y = 5.282,
5 List
@L3 @L2 @L1
{24}
DRAW LINES
List.Map
list
f(x)
mapped
List.FirstItem
list
item
AUTO
List.Map
list
f(x)
mapped
List.LastItem
list
last
AUTO
Line.ByStartPointEndPoint
startPoint
endPoint
Line
AUTO

THE DEMO

we decided to be a bit cheeky...

flux2o.ml

TECHNOLOGY DEEP-DIVE



Autodesk Revit ===> WWW

REVIT API INTERFACE
IExternal Command
IUpdater

WEB INTERFACE
HttpClient
Async Await

JSON

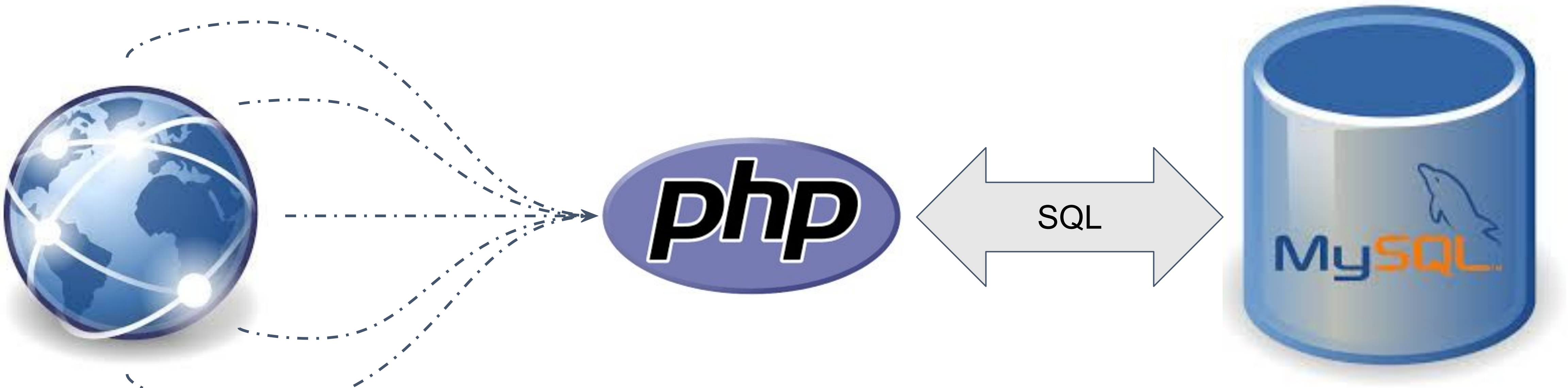
WWW ===> Database

SERVER OPERATING SYSTEM

ubuntu

WEB SERVER PLATFORM

APACHE
HTTP SERVER PROJECT



SERVER-SIDE SCRIPTING LANGUAGE

DATABASE PLATFORM

All Projects Index

uid	fileName

Revit Data

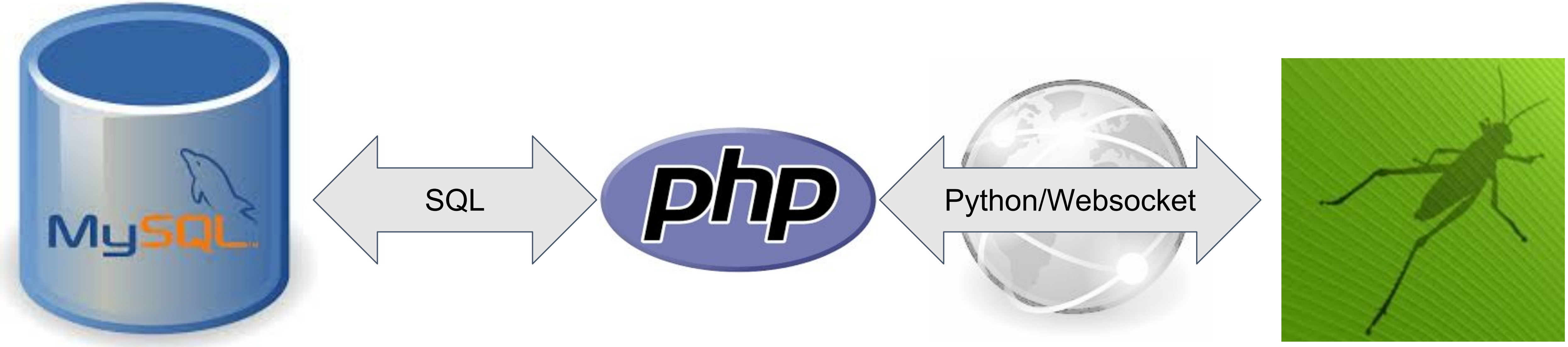
uid	category	levelName	rawjson

Grasshopper Data

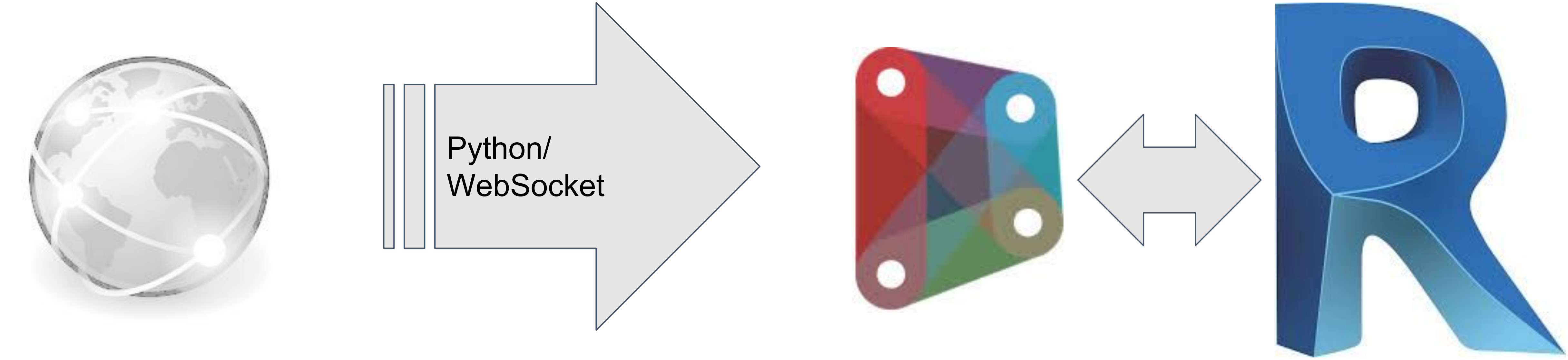
uid	pointjson

A DATABASE IS A COLLECTION OF TABLES

WWW ===> Grasshopper ===> WWW



WWW ===> Dynamo ===> Autodesk Revit



CONCLUSIONS

. use the best tool for the application!
    . they're probably not the same tools
    . one model / many tasks

One of my favorite definitions of “BIM”: a collaborative model, allowing many project participants to access and view the model in ways most appropriate for them. Someone comfortable with Grasshopper should have no problem to perform (and *automate*) egress analysis on a Revit model.

. strategies exist to write data from a Revit model in-the-cloud, on-the-fly, to a common format file (JSON)
. we can read and parse the JSON file to re-create the geometry in another environment
. new data can be pushed to JSON and incorporated into the Revit model

One example was shown today.

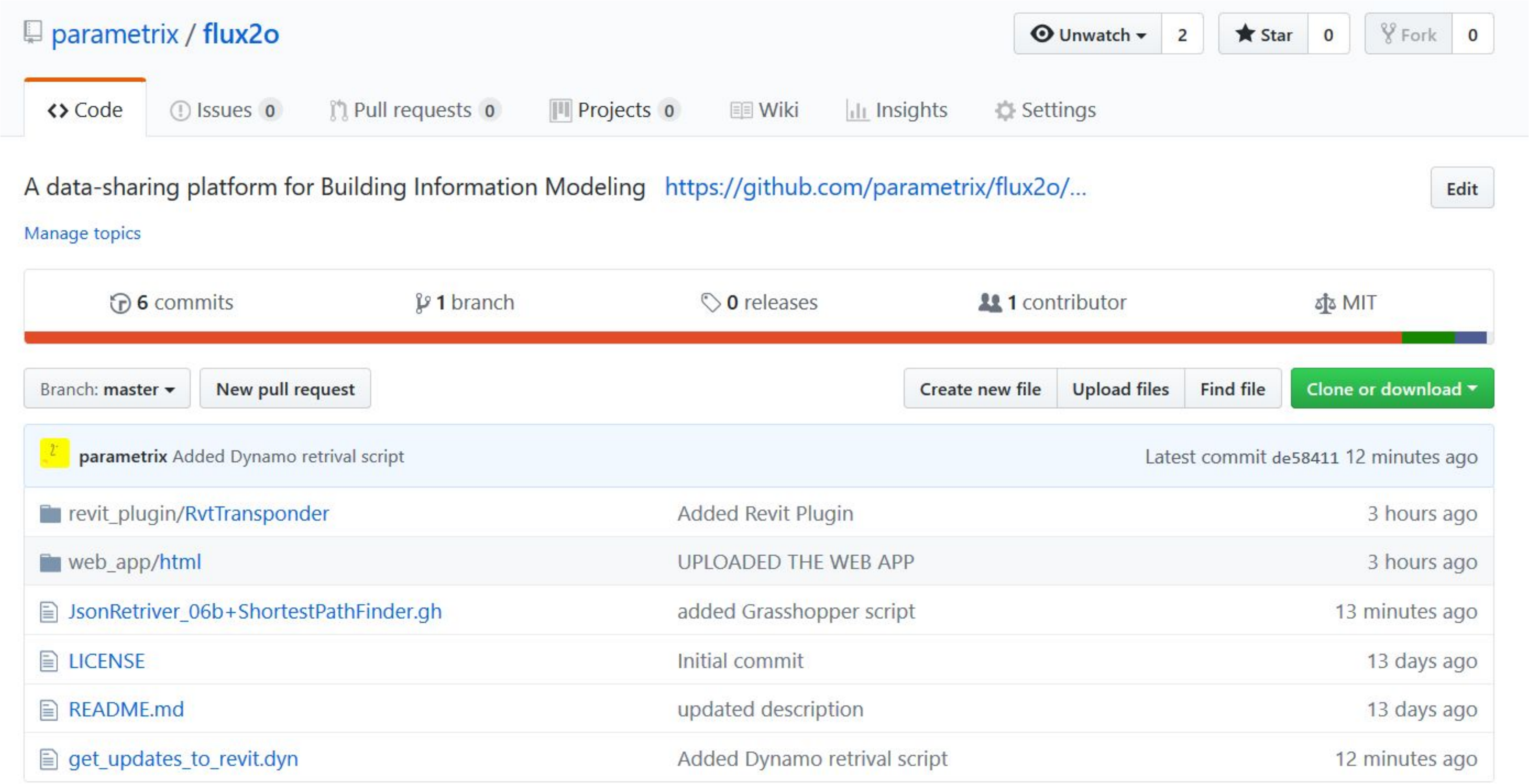
I have never completed a script. Every script does what it needs to do at the time it is written, and I always have every intention to go back and clean it up, and generalize it. That rarely happens. And when it does, that process never completes.

…

FUTURE DEVELOPMENT

Open-Source Project under MIT License:

https://github.com/parametrix/flux2o



THANK YOU!

Our sincere thanks to our colleagues at SOM who helped solve critical challenges along the way
TIMOTHY TAI
MAX HANEY

Online personalities who were a ready reference for most technical challenges
JEREMY TAMMIK, The Building Coder BLOG

Libraries we drew heavily from:
JSON.Net, by James Newton King
Shortest Walk, by giulio@mcneel.com
Socket API, various sources

QUESTIONS, COMMENTS, THOUGHTS...

AUTODESK®

Make anything.™