544

SpaceChase

A Dynamic and Interactive Tool for Architectural Design Process

TUGRUL YAZAR[1], PELIN DURSUN CEBI[2], NILUFER KOZIKOGLU[3], OGULCAN UNESI[1] & MELIKE SENA ERDEN[2]

[1]ISTANBUL BILGI UNIVERSITY

[2]ISTANBUL TECHNICAL UNIVERSITY

[3]TUSPA ARCHITECTS

ABSTRACT

Architectural design, as a cognitive activity, has always been fed by architectural knowledge based on practice and theory, and it has been questioned how creative and generative design processes can benefit from scientific methods. Through inquiry and experimentation, the architect develops his or her ideas, constructs and analyzes the space, and continuously improves it. However, this is a subjective process. There are design tools and methods that provide objective criteria for the assessment of the design decisions and their potential for the designed space whilst iterating with feedback. This study intends to explain how scientific data might be functionalized to search for spatialities that have yet to be designed, and Space Syntax was analyzed to see if it could be included in such a process as a scientific approach. Following discussions on the characteristics of the architectural design process, the study focuses on the development and testing of SpaceChase for Grasshopper, a software plug-in that allows designers, architects, and students to build and analyze dynamic and interactive network structures. This experimental plug-in is intended to be used by architects as a generative and mind-opening design tool during the design process.

KEYWORDS

Architectural Design, Generative design, Real-Time Space Syntax Analysis, Space Syntax

1. INTRODUCTION

Space Syntax is a set of techniques that focuses on the concept of the network in architecture, attempts to decipher the potentials produced by spatial formations using graphic-theoretical tools, and aims to present a scientific, data-oriented, theoretical understanding of architecture and the built environment

(Hillier and Hanson 1984, Hillier 1996). The goal of Space Syntax, which exposes a theoretical understanding of how humans construct and use space, is to make the social knowledge buried in the space, the rules constructed on the space, and the meanings it generates visible and talkable. The method, which uses diagrammatic and mathematical tools to convert the attributes of spatial arrangements into a concrete and measurable form, has also piqued the interest of researchers who define architectural design as a research and discovery process (Dursun 2007). The main question here is how scientific instruments and processes, in addition to analytical knowledge, may be used in a generative and creative way in the production of space.

This study primarily creates a discussion on the dynamics of the architectural design process and searches how scientific instruments and processes such as Space Syntax might be applied in a generative and creative manner in the formation of space that goes beyond analytical knowledge. The following points about the architectural design process are deemed significant to emphasize:

1. Research and discovery in iterations: Architectural design is a process learned through research/discovery. Schön (1987) presents this activity as a kind of "making" (Schön 1987), which is largely learned and practised through "action and reflection". According to him "designing is a web of projected moves and discovered consequences and implications, sometimes leading to the reconstruction of initial coherence – a reflective conversation with the materials of a situation." (Schön 1987). Similar to Schön's idea, we can suggest that the architect/designer searches for possibilities in design by making sketch-like moves and tests and learns from what he/she designs.

2. The role of representations: Architectural design involves a series of moves through representations (Lawson and Dorst 2009) and exhibits a sequential process of description and redescription in the development of designs is presented (Oxman et al 1997). Lawson describes this process as "conversations with drawings". These conversations lead to a kind of creative discovery process which proceeds by creating and testing design ideas (Dursun 2007). Here, evaluations are important for generating new design ideas. However, evaluation is more of "a process of deliberation" than "a straight choice between alternatives" (Lawson and Dorst 2009). An architect uses his/her individual experiences as well as a wide range of science-based data and tools in this evaluation process. Based on these arguments we can suggest that architectural design takes place in a dynamic interaction process facilitated by the sketch between the designed and the designer. The process evolves with iterative experimentation, real-time feedback, and data.

3. The game-like cognitive constructs: Learning-by-doing procedures have a game-like quality. According to Yılmaz (2016), the game's value is determined by the experience it provides. It is well recognized that games play an active role in learning processes; the player learns from his or her experience in the game, may evaluate what he or she has experienced, and so can reflect the cause-and-effect links established in her or his life. Similar to this notion it can be suggested that the

architectural design process has a game-like quality, as reality is perceived through its representations and is shaped by particular rules/limitations.

4. Dynamic "modus operandi": Networks are dynamic forms in which relations are alive, in that they are in states of constant change. Exploiting this way of thinking in the early stages of architectural design makes it possible to keep the negotiation alive, which is important for a creative process. This approach also provides informative tools for architects as it permits designers to see potential in design and constitutes mediums for experimenting and probing (Kozikoglu and Dursun 2015).

Most Space Syntax studies, from a scientific standpoint, treat spatial configuration as an end product and place less emphasis on the process of making space, where space is in constant formation through the flux of the designer's decisions. This study aims to focus on the design process rather than the final product and to question how the existing Space Syntax toolsets can evolve into dynamic and interactive interfaces that feed the design at the moment and produce real-time information, instead of just developing an analytical understanding.

Space Syntax software tools have progressed from a simple yet groundbreaking tool produced by Paul Coates (Coates 2010) to explore spatial stacking by creating computational prototypes to software built by Nick Ship Dalton (1997) and Alasdair Turner (2007) for use in the urban and building scale. With three-dimensional VGA experiments developed by Varoudis and Psarra (2014) based on Turner's studies and real-time spatial analysis, including size, by Nourian (2016), the software has been enhanced to focus on diverse characteristics of space. The majority of the pioneering software tools in this research area have an analytical and descriptive character, focusing on the analysis of the existing spatial setup and attempting to interpret the meaning of the configurational structure. Recently, a new generation of interactive software tools has been developed, allowing reciprocal communication between the designer and the designed space (Nourian et. al 2013, Dursun et. al 2021). These are the tools that allow us to think and produce together by including Space Syntax computations into the design process.

SpaceChase, a software plug-in that allows designers and architects to build and analyze dynamic and interactive network architectures, is a recent Space Syntax software tool and the major subject of this study (Dursun et. al 2021, Kozikoğlu et. al 2020). This plug-in is intended to be used by architects as a generative and mind-opening design tool during the design process. In the scope of the work, the principles of the plug-in are presented and its value in software development studies has been demonstrated through its ability to distinguish itself from current software. It is suggested that the plug-in which has been tested in various workshops, can be an alternative to the Space Syntax toolkits by transforming from the descriptive to the generative, from the static to the dynamic and interactive, from the one describing the final product to the one looking for possibilities by including the personal narratives in the design process.

2. SPACECHASE FOR GRASSHOPPER

This study aims to externalize the stages of the development of a new Space Syntax tool, SpaceChase for Grasshopper, which is currently published and in active use, to express the novelties and notes from user experience, and to discuss the possibilities for further improvement. The development process was guided by the user feedback obtained in several workshops with students and professionals in academia, as well as through conferences and brainstorming sessions. The novelties this plug-in is expected to bring to the existing syntactic software are:

- Promotes the use of syntactic metrics during the design process, not after or before.
- Establishes a network map representation within CAD software, bringing the opportunity to strengthen its usage.
- Reveals an opportunity for designerly interpretation by not searching for optimum layouts.
- Implements an interactive and game-like design process allowing the revisions and different interpretations.
- Defines “contextual nodes” and models their effects on the relationship maps

SpaceChase for Grasshopper[1] is a plug-in that runs in Grasshopper[2], the parametric modeling interface of Rhinoceros. Rhinoceros is a computer-aided design (CAD) software developed for designers to create, analyze and visualize digital models. Grasshopper enables designers to perform algorithmic form-finding searches without the need for advanced coding skills.

2.1 Goals / Steps

The objectives of SpaceChase for Grasshopper are:

- Enabling Space Syntax metrics to architects in a well-known computer-aided design tool,
- Enhancing architectural design exploration by making the spatial networks dynamic,
- Encouraging architects to interfere with this dynamic process via simple user interaction,
- Contributing to the field by documenting the prototyping process of the tool.

The SpaceChase for Grasshopper had three major development steps:

- **Creating the initial network structure in Grasshopper:** This is the initial stage of the development process. It includes the experiments on how spatial nodes and relationships would be represented as a data structure. Preliminary studies suggested that it would be more efficient to develop an original structure from scratch, rather than using a ready-made network module.
- **Making network structures dynamic and interactive:** This is the original contribution of SpaceChase to the research field of Space Syntax, as explained before. After the definition of

[1] The plug-in is available at; spacechase.app and food4rhino.com/en/app/spacechase. A presentation of the toolset (in Turkish) can be accessed at: https://www.youtube.com/watch?v=bsdMINto4MI. A preliminary version of the toolset is presented at ECAADE 2020 conference. An short presentation of that version can be accessed at: https://www.youtube.com/watch?v=CLAbDoVif_Y.

[2] Grasshopper became one of the main components of the Rhinoceros software after version 6.

spatial networks in a digital environment, more tests and development prototypes were made to make them dynamic and interactive. Grasshopper's built-in Kangaroo components[3] were used primarily for this purpose. In future versions, it is aimed to implement an original infrastructure rather than Kangaroo to handle this step. Because Kangaroo is a tool for geometric optimization, which is not the purpose of SpaceChase.

- **Calculation and visualization of Space Syntax metrics:** This implementation was another milestone for SpaceChase. First, Dijkstra's (1959) shortest path calculation algorithm was implemented on network structures. This enabled the calculation of several Space Syntax metrics. The latest version of SpaceChase can calculate connectivity, mean depth, choice, control, and integration (relative asymmetry, real relative asymmetry). In addition, spatial relationship graphs (justified graphs) from a selected node can be generated.

2.2 Preliminary Studies

Grasshopper is based on visual algorithms that act similar to a network graph. The user places CAD commands as nodes on a canvas. These command nodes are called "components". Based on the dataflow coding paradigm, the Grasshopper interface utilizes a directional network system where users place components on a canvas and connect these components to create algorithms. Components have input and output parameters. Generally; Inputs such as numbers, alphanumeric characters, or coordinates are converted by components into geometric objects such as points, lines, and surfaces. Components connected in tandem form a chain and network structure and form the algorithmic expression of the geometric modeling process. By interfering with the steps and inputs of this structure, designers can quickly revise and derive their design alternatives. Grasshopper is developing as a community project. In addition to speeding up the geometric modeling process, it gains new features with new components developed by this community. Figure 1 shows a typical Grasshopper algorithm and its result. On the left side of Figure 1, the dataflow diagram of Grasshopper is shown. This diagram is created by placing and connecting task-specific components from a rich palette provided in the Grasshopper interface. The right side of Figure 1 shows the real-time visual result generated in the Rhino 3D CAD software's viewport.

The development strategy of SpaceChase consists of two major phases. The first phase is the development of the drafts, aiming at the goals outlined in the previous section. This is achieved by utilizing the native components of Grasshopper. The goal of these preliminary studies is to test the potential of dynamic and interactive network structures and to identify technical requirements before developing the plug-in. Several Grasshopper codes are created for this purpose. The following section describes some of the significant ones, along with the challenges, and the emerging qualities.

[3] Kangaroo is a component group running in Grasshopper. It is based on the geometric optimization of the *goal objects*. A real-time motion is displayed on the screen while the chosen goal objects are optimized synchronously. The resulting animation is open to the user's interference by a mouse.

The Grasshopper code shown in Figure 1 is the first attempt to explore the technical background of an interactive and dynamic network structure. Users can create simple networks by drawing the edges as lines. The motion is generated by the pushing/pulling forces between the endpoints of the edges. Kangaroo goal object named Clamp Length is used to keep the edge lengths within predefined limits. In the final part of the code, the nodes are visualized by Grasshopper's Metaball components. The system can be optimized by Kangaroo, ending the motion if it is not triggered by the user for a while. It was observed that geometric optimization should be prevented since the system is expected to remain in motion to enhance design exploration.

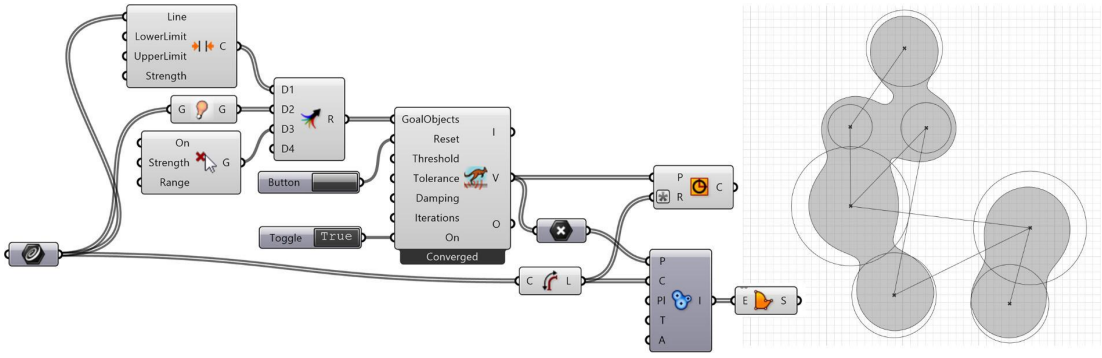

Figure 1. SpaceChase for Grasshopper Work-in-progress (WIP) Prototype #1 (October 2019). Left: Data flow diagram. Right: A screenshot showing the result.

After the observations and discussion on the initial prototype, the development process gained momentum on multiple paths. Prototype#2 includes the first innovations in data collection, contextual nodes, node weights, Space Syntax metrics, and user interface (Figure 2).

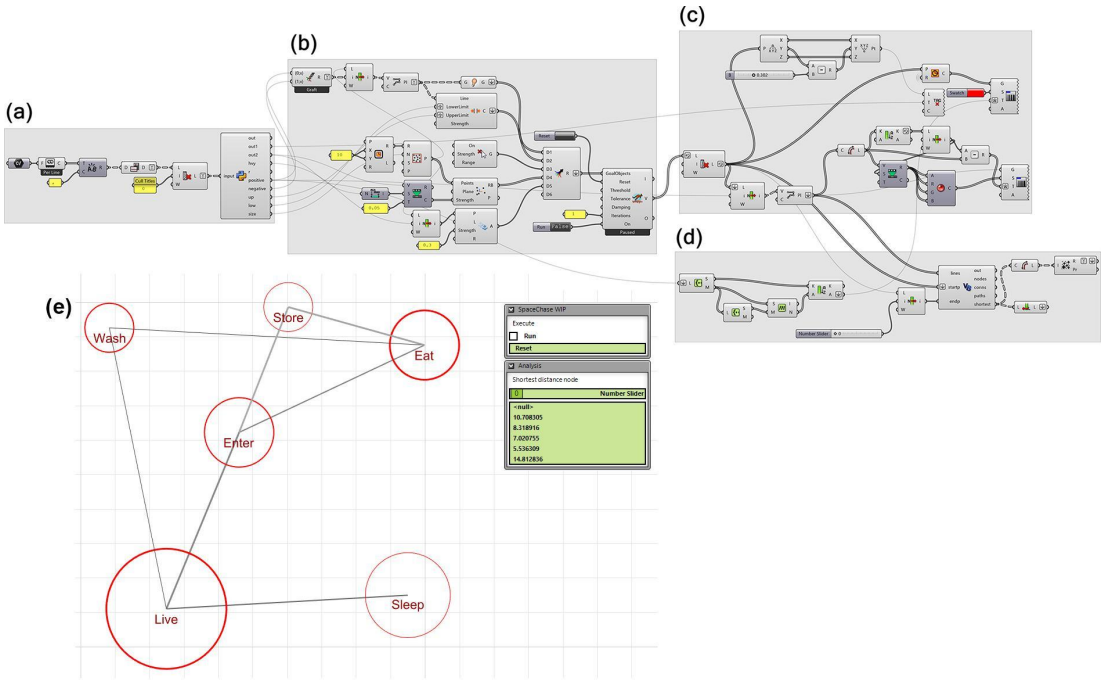

Figure 2. SpaceChase for Grasshopper WIP Prototype #2 (December 2019). Data flow diagram. (a): Reading the network structure from a data table. (b): Setup for dynamic and interactive network simulation, (c): Visualization of nodes and edges on the screen, (d): Shortest path calculation. (e): A screenshot, showing a sample graph and the graphical user interface.

Table 1. A sample data table used in SpaceChase prototypes

area	title	weight	positive relations		metrics		negative relations		metrics
30	Live	0	Live	Eat	0	10	Live	Sleep	5
10	Eat	5	Live	Wash	5	30	Eat	Sleep	15
5	Wash	0	Live	Sleep	20	30			
15	Sleep	3	Live	Enter	0	5			
10	Enter	10	Eat	Wash	0	10			
5	Store	5	Eat	Enter	5	20			
			Eat	Store	0	10			
			Wash	Sleep	0	5			
			Enter	Store	0	10			

Table 1 shows a sample data table used in Prototype #2. The CSV formatted table keeps the information about nodes and their relations. These data are then transferred to Grasshopper via the code group and the Python script shown in Figure 2-a and converted into Kangaroo goal objects shown in Figure 2-b. Two new goal objects are added in Prototype#2. Rigid Point Set ensures that the nodes tend to maintain their position with a certain tolerance. It was observed in the experiments that the network structure under geometric optimization can easily be thrown out of user control. Therefore, another goal object named Anchor Point fixes the positions of special nodes. These nodes, which will be described as contextual nodes in the future, were defined as immobile points in Prototype#2. The code group seen in Figure 2-c visualizes circles, points, and lines. The code group in Figure 2-d calculates the shortest paths between the nodes by utilizing Dijkstra's algorithm as a VB.NET script. This script is the basis for Space Syntax metrics to be implemented in the future. This prototype also utilized Grasshopper's user interface feature (remote panel). Users can drag the nodes on the Rhinoceros screen, control the simulation in the panel, and view the shortest paths. Since dragging the nodes doesn't change the network topology, the shortest path calculations are fixed. Similar to Prototype#1, the motion can stop if not re-activated by the user after a certain period.

After more improvements, Prototype#3 was developed to be tested with students. The code group shown in Figure 3-a imports a CSV data table and converts it into the appropriate format for SpaceChase. Unlike Prototype#2, this group does not use a Python script for better interaction with the other Grasshopper components. With the arrangements made in the code group seen in Figure 3-b, the constant, non-optimizing motion is ensured. In this prototype, the fixed points called contextual nodes are defined explicitly. These nodes aim to relate the network to the architectural context. Unlike spatial nodes in motion, the contextual nodes represent fixed qualities like directions, views, and vistas (north, south, scenery, panorama… etc.); contextually important and fixed objects and spaces (an old tree, an access gate, the neighboring building… etc.). This also opened possibilities for beta testing by enabling simple design scenarios. The code group shown in Figure 3-c enables new adjustments to better visualize the dynamic network structure. As another important improvement of Prototype#3, the code group shown in Figure 3-d calculates the connectivity values of the nodes.

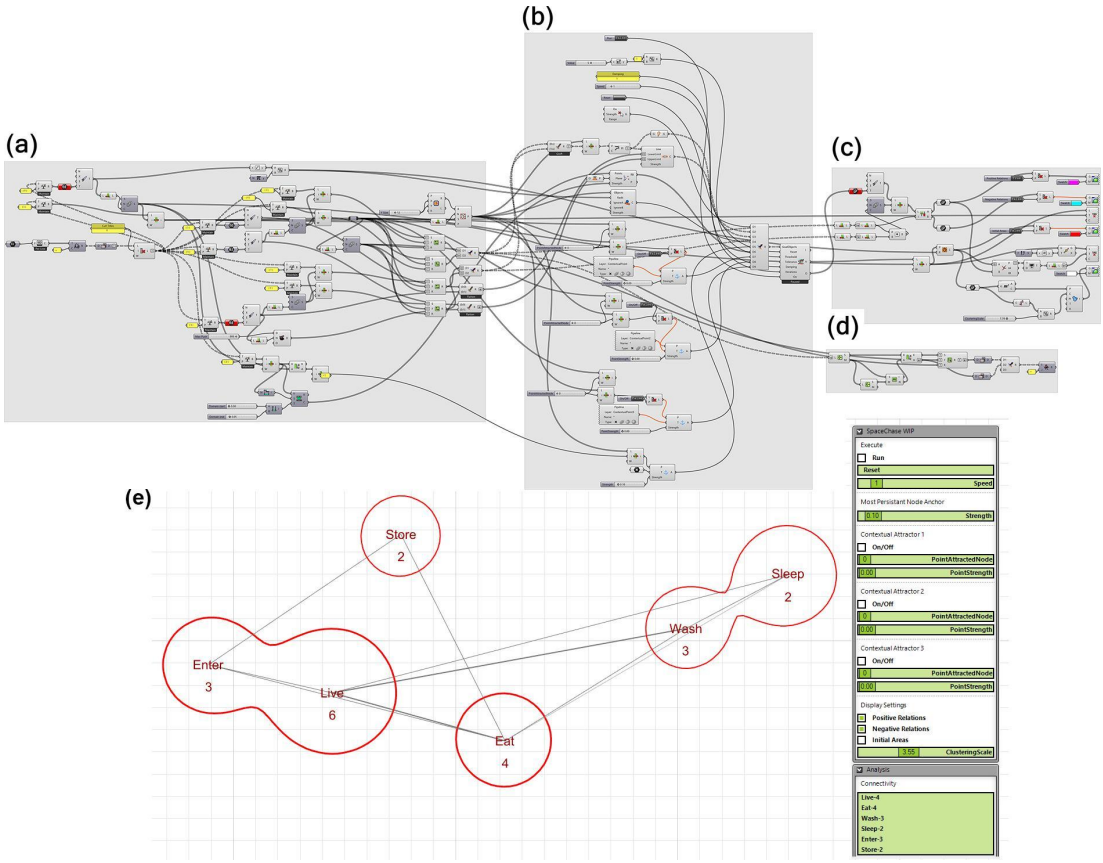

Figure 3. SpaceChase for Grasshopper WIP Prototype #3 (December 2019). Top: Data flow diagram. (a): Reading the network structure from a data table and organizing. (b): Setup for non-optimizing dynamic and interactive network simulation, (c): Visualization of nodes and edges on the screen, (d): Connectivity calculation. (e): A screenshot, showing a sample graph and user interface.

Prototype #3 was tested in the first student workshop held at Istanbul Technical University on December 5-12, 2019. In addition, a version between Prototype #2 and #3, was presented at the Education and Research in Computer-aided Architectural Design in Europe (ECAADE) 2020 conference (Kozikoğlu et. al 2020). As a result of these tests and feedback, the fourth and final prototype was created.

Prototype#4 represents the peak point of the improvements that could be made by utilizing only native Grasshopper components. The code groups are organized as clusters, ensuring that it is easier to understand. This prototype derives the network structure from the lines drawn on the Rhinoceros screen (Figure 4-a). In the student workshop, it was observed as a more flexible way of defining network structures. The obtained network data is directed to two clusters. The first one is the set of components called Dynamic Graph, seen in Figure 4-b. This cluster, like previous prototypes, acts as a data organizer between the network structure and the Kangaroo simulation. The second cluster calculates the depth, mean depth, and integration values (Figure 4-d). This experience in calculating Space Syntax metrics was important and encouraging in terms of showing that more metrics could be included in the next steps of the study. Another original aspect of this prototype is that it offers graphical interface features by using the Human UI components, which allows the creation of custom user interfaces (Figure 4-e). In this way, users can use all of the features of SpaceChase, even if they

do not know how to use Grasshopper. This graphical interface contains several parts (tabs) to control the modeling and analysis of the network structure. Prototype #4 was tested in a workshop on June 3-4, 2020.

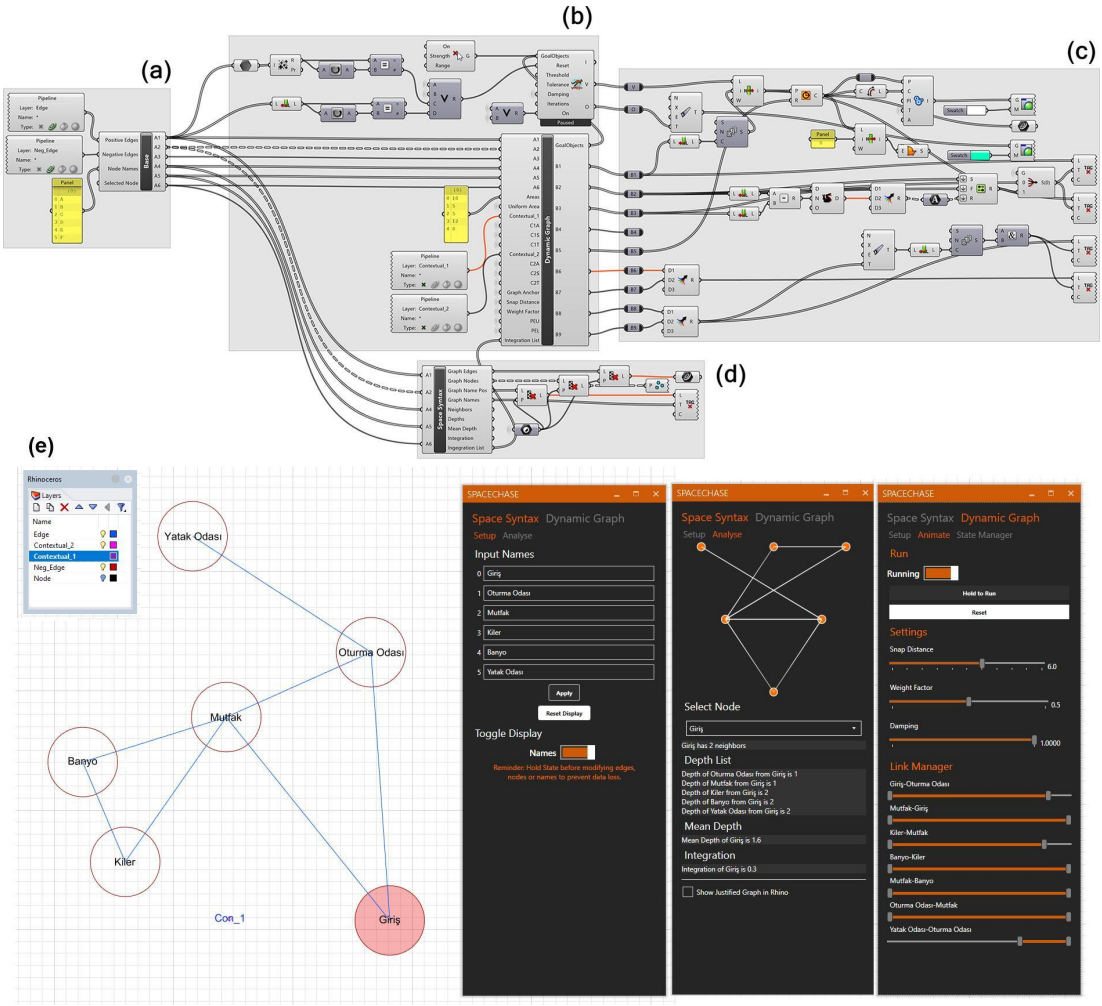

Figure 4. SpaceChase for Grasshopper WIP Prototype #4 (June 2020). Top: Data flow diagram. (a): Reading the network structure from the lines on the viewport. (b): Setup for dynamic and interactive network simulation, (c): Visualization of nodes and edges on the screen, (d): Space Syntax metrics calculation: Depth, Average depth, and Integration. (e): A screenshot, showing a sample graph and the screens of the graphical user interface.

Table 2 summarizes the features of these four prototypes. The lessons learned from the preliminary study, and the workshops can be summarized as follows;

Table 2. Comparison of the SpaceChase for Grasshopper prototypes

Part.	Network infrastructure	Dynamic and interactive simulation	Space Syntax metrics
#1	Network data is taken from the lines on the Rhinoceros screen.	The motion is not continuous. Kangaroo is used.	Not implemented
#2	Network data is taken from tables in CSV format. Nodes are randomly located.	The motion is not continuous. Kangaroo is used. A remote panel is used.	Shortest path
#3	Network data is taken from tables in CSV format. Contains context nodes.	The motion is smooth and continuous. Kangaroo is used. A remote panel is used.	Shortest path, connectivity
#4	Network data is taken from the lines on the Rhinoceros screen. Contains context nodes.	The motion is smooth and continuous. Kangaroo and Human UI are used.	Shortest path, connectivity, depth, mean depth, integration

- It was observed that the user interaction must be flexible, allowing different approaches when initializing a network structure. In the plug-in, it was decided to create networks both by Rhinoceros objects, and data tables. A generalized and flexible node definition should be developed to include these two inputs in the same network structure.

- Early studies have shown that dynamic and interactive network structures can benefit from Kangaroo components. When the duration of the research and the technical possibilities are evaluated, it was decided that it is not possible to develop an original algorithm that can replace Kangaroo shortly. Thus, this implementation is left to further research. It was decided to build the plug-in that interacts with Kangaroo and generates its goal objects.

- It has been seen that separate Grasshopper components can be developed that will calculate various Space Syntax metrics. Therefore, the user can modify the Grasshopper code to obtain the metrics required. This also raised the possibility of utilizing SpaceChase in different usage scenarios. Static Space Syntax analyses would be made without initiating a dynamic network.
- Prototype #4 can be viewed as a new user experiment, introducing its custom graphical interface. However, this approach limited the utilization of Grasshopper in different ways. In the continuation of the study, instead of deepening in this direction, it was decided to work within Grasshopper's canvas, and continue to use its strengths of data flow parametric modeling. Therefore, instead of creating secondary graphical interfaces, new Grasshopper components would be developed.

3. DEVELOPMENT OF THE PLUG-IN

The second phase of SpaceChase for Grasshopper is the development of a plug-in package consisting of new Grasshopper components. The plug-in was developed in the Python Language, by utilizing the Python Script component of Grasshopper. The codes are compiled into the Python plug-in format of Grasshopper (ghpy format). During the development process, different versions were created and tested. This section will describe three of these beta versions that can be seen as the final milestones of the process.

3.1 Beta Versions

The working principles of the plug-in are similar to the preliminary prototypes explained earlier. The major difference is that new special-purpose components and data types were developed instead of ready-made components. The first beta version of the plug-in consists of 13 components. The most important contribution of the first version of the plugin, 0.1.0, to the process is the definition of the new data types which carry mathematical information about the network structure throughout the data flow diagrams. These basic components of SpaceChase are as follows;

● Node Data (ND): This component contains data about spatial nodes. Experience from prototypes has shown that nodes can be created using different methods. Since each method has advantages and disadvantages, SpaceChase has defined multiple methods.

● Edge (E): Relationships between the nodes are called edges in the SpaceChase plug-in. The information they contain is the line objects of Rhinoceros. The quantity of the effects (push/pull) on the network is transmitted through the length of the lines contained in the edge objects.

● Nodes (N): This program object is created by a combination of Node Data and Edge objects and represents node points in the network structures produced by SpaceChase. It is used in calculating Space Syntax metrics and creating dynamic networks. Therefore, the first step in the workflow of SpaceChase is the definition of Node objects.

● Graph Data (GD): This is the network structure produced by the SpaceChase plug-in. After this program object is produced, it is used both in Space Syntax metrics and in the creation of dynamic network structures. It is an object in which the necessary information about the network topology is collected from Node Data and Edge objects, organized.

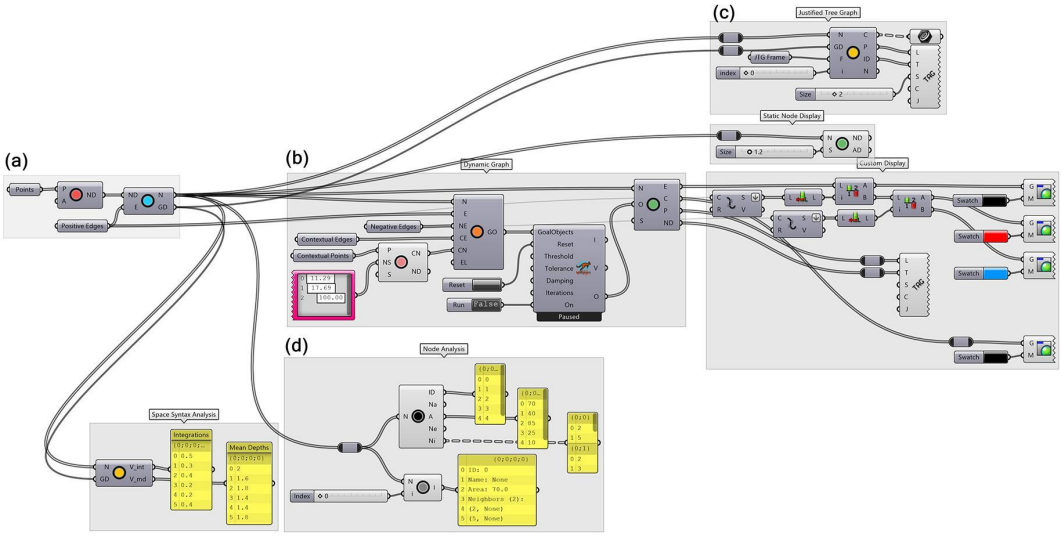

Figure 5. SpaceChase for Grasshopper Beta (0.1.0); Data flow diagram. (a): Reading the network structure from the lines on the viewport. (b): Setup for dynamic and interactive network simulation, (c): Visualization of nodes and edges on the screen, (d): Space Syntax metrics calculations.

Figure 5 shows a sample Grasshopper code made with the first version of the plug-in. The component, seen as a blue dot in Figure 5-a is responsible for collecting the information about the network (node positions, names, connections, etc.) and transforming it into data (Graph Data) that SpaceChase can read and use. After the SpaceChase network structure is created, this structure can be used in two different ways. The first one of these is to perform basic Space Syntax analysis and visualizations (Figure 5-d). The other way is to create the dynamic network structure and simulate and visualize it via Kangaroo (Figures 5-b and 5-c). These two usage patterns (component groups) can work independently from each other on the same network. This revealed some of the advantages and disadvantages discussed throughout the project. These usage scenarios are included in the latest version of the plug-in. On the other hand, the disconnection of Space Syntax metrics with the dynamic

network structures brought up the challenges of their integrated usage. The Syntactic Network Model developed in the latest version of the plug-in partially answered this question.

During the further development of the plug-in, no significant difference was observed in the user's screen. As a result of the further development and debugging, the second milestone of the process, version 0.2.5 has been created (Figure 6). This version contains 17 components under 8 component groups. The development led to the original codes that are transferring the mathematical underpinnings of Space Syntax metrics to the Python language. The accuracy of the Space Syntax metrics calculated by this version was compared and verified by using Syntactic software (Nourian et. al 2013a) on the same networks. This version has been tested at a workshop, held as part of the Education and Research in Computer-aided Architectural Design in Europe (eCAADe) 2020 Conference. During the workshop, some further improvements were made to the visualization capacity of the plug-in.

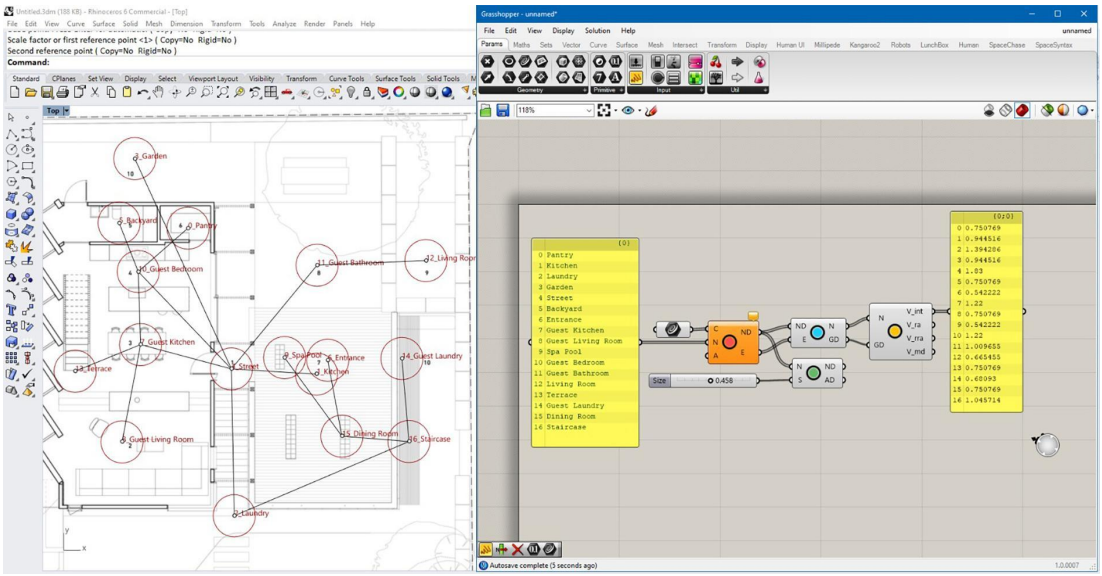

Figure 6. SpaceChase for Grasshopper Beta (0.2.5). (Left): A screenshot, (Right): A sample data flow diagram, calculating the integration values.

3.2 The Final Version

The latest version (0.3.3) of SpaceChase contains a total of 22 components in 7 component groups (Figure 7). This version minimized the errors and bugs and can also be considered a release version (1.0.0). It can be downloaded from the spacechase.app and food4rhino.com/en/app/spacechase.

In the last version of the plug-in, it was decided to enrich the data exchange with different sources and to add some interface features as a result of feedback from the workshops. The feature of extracting data from tables in .csv format, which is used from time to time since its prototypes and widely used in other network visualization software, has been added again. Thus, it became easier for the plug-in to exchange data with similar software (Cytoscape, Syntactic, Gephi, etc.) working on network structures, and it was ensured that the users of this software could learn and use SpaceChase easily.

Similarly, a component that reads the data in .json format has been developed, thus making it possible to transfer the network structures obtained in other digital environments.

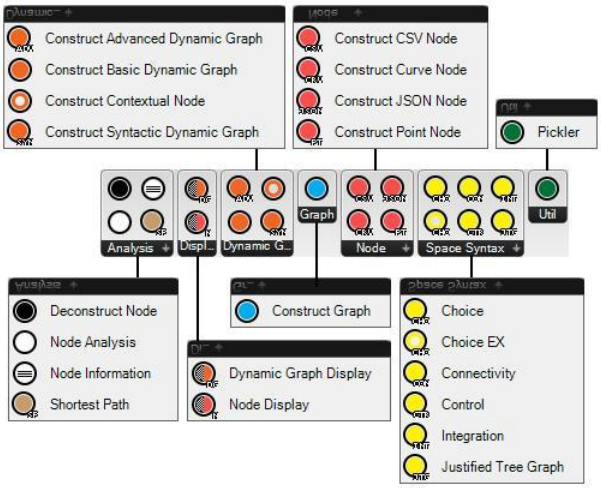

Figure 7. "SpaceChase for Grasshopper Beta" (0.3.3) (November 2020). The plug-in tab and the component groups as seen in the Grasshopper toolbar.

In this version, the dynamic network model has three separate utilizations. These three network models, divided into Basic, Advanced, and Syntactic, are explained in detail in the next section. The Construct Syntactic Dynamic Graph is a significant improvement on the previously mentioned issue regarding the incorporation of Space Syntax metrics into the dynamic network. Using this method (and component), the Space Syntax metrics specified by the user can also be included in the targets to be transferred to Kangaroo. Another innovation in the latest version relates to precise control of connections. In the last student workshops, it has been observed that the users do not have enough control over the motion of the dynamic network structure. It is possible to modify the effect of the contextual nodes on the network and the connection lengths between the nodes via Grasshopper, but this workflow was not flexible enough. Two new graphical user-interface windows have been added to the plug-in to strengthen user interaction. The first of these lists the connections in the dynamic network structure and allows the strength of each connection to be controlled by using sliders. The second window likewise makes it possible to more quickly check the pull effect of the contextual nodes.

3.3 The Usage Scenarios

This section summarizes how the latest version of the SpaceChase for Grasshopper plugin can be used[4]. Before starting to work with SpaceChase for Grasshopper, it is necessary to define the network structure. After drawing the nodes and connections as line objects on Rhinoceros, and transferring them to Grasshopper, then SpaceChase can be started and used. The node objects of SpaceChase are labeled as Node (N). The data (point coordinates, name, weight, etc.) of the nodes are defined as the

[4] A Windows-based computer must have Rhinoceros 6 (SR25 or later) installed to run SpaceChase for Grasshopper. The plug-in can also work under Rhinoceros 7. The latest version of the plugin, 0.3.3 (as of 10.01.2021), can be downloaded from https://www.food4rhino.com/en/app/spacechase. When the installation is finished, a Grasshopper tab named *SpaceChase* appears.

Node Data (ND) data type. To convert a point to Node Data (ND) that SpaceChase can read and process, components under the component group titled Node tab must be used. All of these components generate Node Data (ND) data using different inputs. Another main component of SpaceChase is the Construct Graph component, which takes Node Data (ND) and Edge (E) data and transforms them into a SpaceChase network structure (GD) (Figure 8-a). Using these initial components, a simple network structure can be modeled in Grasshopper.

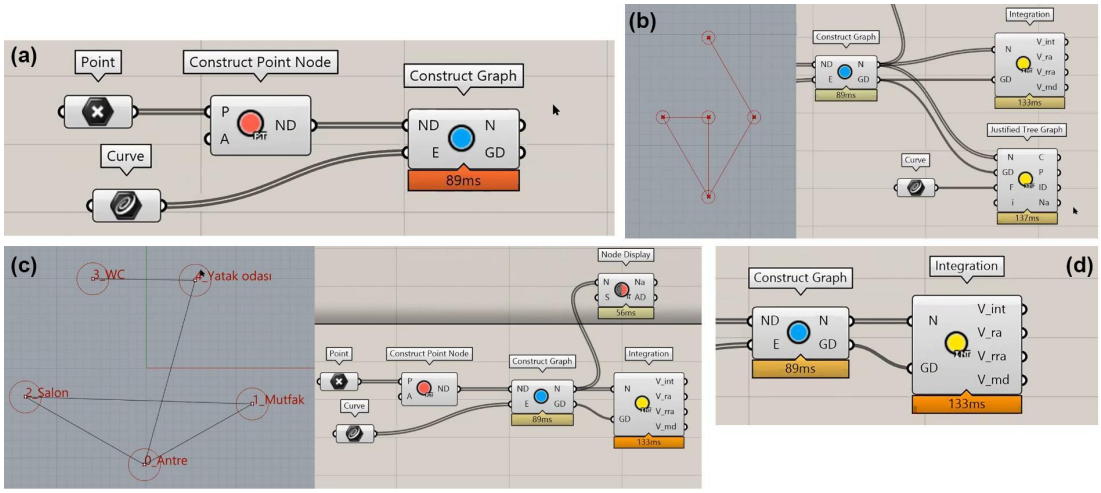

Figure 8. Some of the non-dynamic functionalities of the latest version. **(a):** Derivation of SpaceChase node (N) and network (GD) data from the node (ND) and link (E) data with the help of the Construct Graph component **(b):** Creation of the Justified Tree Graph by using the network data. **(c):** Visualization of nodes and **(d):** reading their integration values in Grasshopper.

After this initial stage, SpaceChase for Grasshopper offers two usages. If a dynamic and interactive simulation is not needed, the network structures created with the help of the components described above can be directly fed into the Space Syntax analysis. Components in the group titled Space Syntax can calculate the choice, connectivity, depth, control, and integration values of GD data type (Figure 8-b). In addition, a Justified Graph that visualizes the network structure can be created (Figure 8-c). One of the distinguishing features of SpaceChase for Grasshopper is that the analysis results and network visualization provide outputs that are compatible with Grasshopper. Thus, the user can continue using these outputs in the design and visualization processes.

The second usage of the plug-in is dynamic and interactive network structures. In the dynamic network structure, node points are expected to be in motion. Contextual nodes are used to control the simulation. These nodes are not included in the Space Syntax metrics. The negative (pushing) and positive (pulling) connections can be defined and modified. Negative connections tend to push the nodes away from each other. Connections with positive values tend to bring the nodes at both ends closer to each other. After these preparations, the dynamic simulation of the network structure can be started. The components in SpaceChase's Dynamic Graph component group offer solutions for different purposes. There are three types of dynamic network models;

● **Basic network model:** It is used for fast dynamic network visualization. This component generates a dynamic network model by calculating line and point data in terms of positive/negative relationships, lengths, and distances.

● **Advanced network model:** This network model provides the opportunity to intervene in calculations in the Basic model individually or collectively. It performs similar calculations but is coded from scratch to present the intervention possibility in a user-friendly way through graphical user interfaces. This is defined as a separate model from the first one because the preparation process requires more time compared to the first model.

● **Syntactic network model:** This network model performs the calculations at the discretion of the user in the second model with the help of the metrics selected as input to the component and offers a "syntactic" dynamic network visualization. The component calculates the dynamic network properties defined as weight and strength relatively within the system in return for the metrics selected by the user. It aims to present a visual representation of the Space Syntax metrics.

Based on the above setup, an advanced network model is explained below. This model utilizes the Grasshopper component labeled Construct Advanced Dynamic Graph. The inputs and outputs of this and other SpaceChase components are explained in Table 3. The GO output of this component must be connected to the Goal Objects input of the Bouncy Solver Kangaroo component. When the Bouncy Solver component is run, a dynamic network structure would be formed on the Rhinoceros screen.

The sample data flow diagram shown in Figure 9 consists of two main groups. The first group (Figure 9-a) includes the components in which the dynamic network is set up and simulated. The links visualized by the Dynamic Graph Display component seen in the upper-left corner of Figure 9-b include all syntactic and non-syntactic (negative, contextual) links. These connections can be visualized in different ways by Grasshopper that can separate this data list. After this connections and nodes can be visualized using Grasshopper components such as Custom Preview or Text Tag 3D. By continuing to work on this basic setup, it is possible to develop Grasshopper algorithms suitable for various design scenarios.

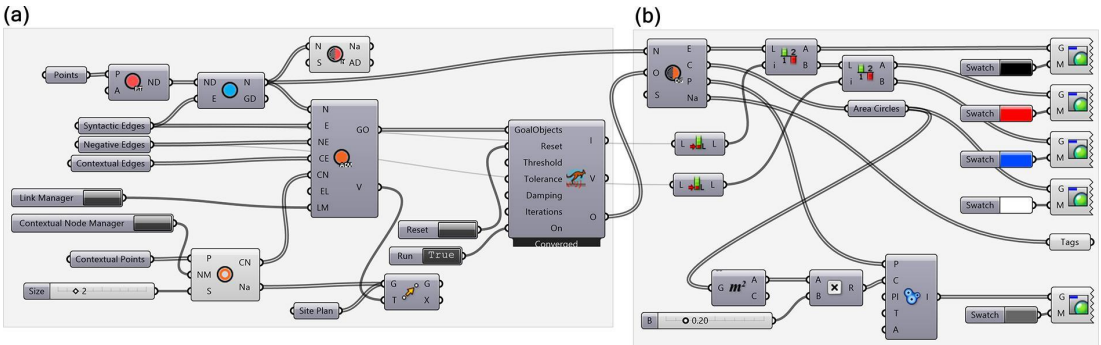

Figure 9. SpaceChase sample dynamic network setup. (a): dynamic network basic setup, (b): dynamic graph visualization group

Table 3. The SpaceChase components, their inputs, and outputs

Component name and function	Input(s)	Output(s)
Construct Point Node: Generates SpaceChase node data by using Rhinoceros point objects	**P:** Rhinoceros point objects (optionally containing name information). **A:** Areas (size) of the nodes (optional)	**ND:** SpaceChase node data
Construct Curve Node: Generates SpaceChase node data and edge data by using Rhinoceros line (curve) objects	**C:** Rhinoceros curve objects. **Na:** Labels of the nodes. **A:** Areas of the nodes	**ND:** SpaceChase node data. **E:** SpaceChase edge data
Construct CSV Node: Generates SpaceChase node data and edge data by using CSV file	**CSV:** CSV file address	**ND:** SpaceChase node data. **E:** SpaceChase edge data
Construct JSON Node: Generates SpaceChase node data and edge data by using a JSON file	**ND:** SpaceChase node data. **E:** SpaceChase edge data	**ND:** SpaceChase node data. **E:** SpaceChase edge data
Node Display: Displays SpaceChase node objects in Rhinoceros viewport	**N:** SpaceChase nodes. **S:** Text size	**Na:** Visible text objects. **AD:** Circles representing nodes
Construct Graph: Creates SpaceChase network structure	**ND:** SpaceChase node data. **E:** SpaceChase edge data	**N:** SpaceChase nodes. **GD:** Mathematical node objects
Construct Basic Dynamic Graph: Utilizes the basic dynamic graph model	**N:** SpaceChase nodes. **E:** SpaceChase ağ edges. **NE:** Negative SpaceChase edges. **CE:** (Positive) contextual edges. **CN:** Contextual nodes	**GO:** Kangaroo goal objects
Construct Advanced Dynamic Graph: Utilizes an advanced dynamic graph model, which enables more detailed user interaction	**N:** SpaceChase nodes. **E:** SpaceChase edges **NE:** Negative SpaceChase edges. **CE:** (Positive) contextual edges. **CN:** Contextual nodes. **EL:** Normalized edges (optional). **LM:** Link Manager interface toggle	**GO:** Kangaroo goal objects. **V:** A vector that transfers dynamic network nodes for better visualization
Construct Syntactic Dynamic Graph: Generates the dynamic graph integration with Space Syntax metrics	**N:** SpaceChase nodes. **E:** SpaceChase edges **NE:** Negative SpaceChase edges. **CE:** (Positive) contextual edges. **CN:** Contextual nodes. **EL:** Normalized edges (optional). **WGH:** "Weight" Space Syntax metrics. **STR:** "Strength" Space Syntax metrics	**GO:** Kangaroo goal objects. **V:** A vector that transfers dynamic network nodes for better visualization
Construct Contextual Node: Creates contextual node objects	**P:** Rhinoceros point objects. **NM:** Toggle that opens Contextual Node Manager window. **S:** Text size for the labels	**CN:** SpaceChase contextual node objects **Na:** Text objects for the labels of the contextual object
Dynamic Graph Display: Displays the dynamic network on Rhinoceros viewport	**N:** SpaceChase nodes. **O:** "GoalFunction" output of Kangaroo. **S:** Text size	**E:** Edge lines. **C:** Circle radii. **P:** Node points. **Na:** Text objects
Choice: Calculates the Space Syntax metric	**N:** SpaceChase nodes. **GD:** SpaceChase network structure	**V_cho:** Choice value
Choice EX: Calculates the Space Syntax metric without the first and last steps	**N:** SpaceChase nodes. **GD:** SpaceChase network structure	**V_cho:** Choice value without first and last steps

Connectivity: Calculates the Space Syntax metric	**N:** SpaceChase nodes	**V_con:** Connectivity value
Control: Calculates the Space Syntax metric	**N:** SpaceChase nodes	**V_ctr:** Control value
Integration: Calculates, mean depth, integration, relative asymmetry, and real relative asymmetry values	**N:** SpaceChase nodes. **GD:** SpaceChase network structure	**V_int:** Integration value. **V_ra:** RA value **V_rra:** RRA value. **V_md:** Mean depth value
Justified Tree Graph: Generates the diagram used in Space Syntax analysis	**N:** SpaceChase nodes. **GD:** SpaceChase network structure. **F:** The frame of the diagram. **i:** Index number of the node, from which the diagram will be generated	**C:** Lines of the diagram. **P:** Points of the diagram. **ID:** ID numbers of the points. **Na:** Labels of the points
Deconstruct Node: Creates Grasshopper point and curve objects from SpaceChase nodes	**N:** SpaceChase nodes	**P:** Point object. **C:** Curve object(s)
Node Analysis: Generates a list of data from SpaceChase nodes	**N:** SpaceChase nodes	**ID:** Indices of the nodes. **Na:** Labels of the nodes. **A:** Areas of the nodes. **Ne:** Labels of the links. **Ni:** Indices of the links
Node Information: Generates a report for a specific SpaceChase node	**N:** SpaceChase nodes. **i:** Index of the selected node	**I:** A text message containing information about the node
Shortest Path: Calculates the shortest path between two nodes	**N:** SpaceChase nodes. **GD:** SpaceChase network structure. **iS:** Index of the starting node. **iT:** Index of the target node	**SP:** Shortest path (line objects)
Pickler: An experimental component to be used in generating data readable outside of Grasshopper	**D:** Data to be recorded. **W:** Recording toggle **R:** Read command	**D:** Data read

The contextual Node Manager window can be used to control the effects of context nodes on the network. Similarly, the Link Manager window helps to explore different link weights. An important and unique capability that the Kangaroo provides to SpaceChase is that it allows nodes to be dragged with the mouse during simulation. Figure 10 shows this effect on a sample network.

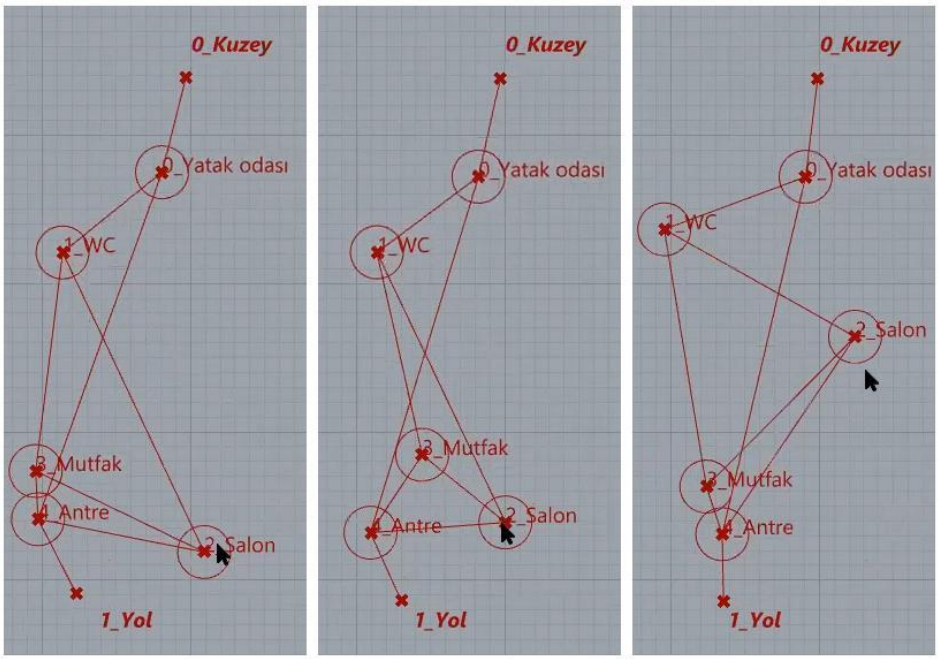

Figure 10. SpaceChase sample dynamic network interaction. The dots with a circle around them represent the nodes, and the dots without a circle represent contextual fixed nodes. The user drags one of the nodes and the rest of the network is actively responding to this transformation.

4. CONCLUSIONS

This section summarizes the limitations and future potentials of SpaceChase for Grasshopper, obtained from the observations and discussions at various workshops with students of architecture and young architectural designers, held in Istanbul Technical University in 2020-2021, ECAADE conference in Berlin, 2019, and the AURA, design, and education platform, Istanbul and as well as private sessions with practicing architects.

● **Flow and transformation of nodes:** While visualizing dynamic network structures, it has been a controversial topic that nodes that are close to each other can combine to act as a single node, thus enabling the transformation of the network topology on the fly. Metaballs were used for this purpose in the prototype studies. However, the algorithm behind Grasshopper's Metaball components cannot be fully controlled. This is not only a visualization limit but also an algorithmic constraint on the behavior of the nodes and their transformation in the process. In further research, the nodes that do not remain fixed can merge and transform, and it is possible to develop and use original metaball calculations in visualizations.

● **In-motion network calculations:** Another frequently discussed topic during the plug-in's development process is the possible effects of Space Syntax metrics on the dynamic network structure. Although a clear result has not been reached in this regard, in the latest version of the plug-in, it has been made possible to associate the motion with a selected Space Syntax metric by using the Syntactic Network Model component. However, since the initial network topology cannot be transformed by the system, it is not possible for the Space Syntax metrics to change during motion

and interact with the network structure mutually. In further studies, it is predicted that situations where the dynamic network structure connects and separates nodes, or disconnects and reconnects the nodes, can also be simulated. Thus, it may be possible for Space Syntax calculations to play a more active role in this process.

● **More spatial dimensions:** An issue expressed in the trial processes from the prototypes is that the network structures produced by the tool are two-dimensional. Since the network representation essentially expresses a relationality independent of dimensions, although there are no mathematical limitations on its representation in n-dimensional environments, it is predicted that the two-dimensional planes are the environment in which the relations between the nodes can be read most easily in static network structures, considering the user experience. However, this situation limits the possibilities in the dynamic network environment. Especially since the contextual nodes represent more grounded spatial information about an architectural context, the introduction of the third spatial dimension became a necessity. The workaround solution used in the workshops has been to define a vertical circulation element such as a "Staircase" as a nodal point. However, since the nodes affect each other as long as they are on the same plane, it becomes difficult to establish relationships in multi-level spatial systems.

● **Categorization and tagging:** Although there is no near technical limitation on the number of nodes that can be displayed on the screen at the same time, especially in dynamic network visualization, it is seen that this number can have a limit in terms of user perception and experience. In further studies, it is possible to add some local parameters to the nodal points and thus create a more useful tool for solving complex three-dimensional systems. These parameters can be in the form of tags, or they can be in the form of numeric values. Circulation elements may (dis)appear during the design process i.e. the software will in the future allow links to become nodes and spatialize and/or nodes diminish to links. Similarly, there is room for development in providing node groups such as a group of hotel rooms to shrink and expand on the go. In parallel, non-program rooms, in-between spaces, or halls with multiplex programs, etc. may appear as pop-up suggestions or remarks with template design networks fed into the software, as well as design regulations.

● **Unique simulation engine:** SpaceChase for Grasshopper uses Kangaroo geometric optimization engine in a different way than intended. In the current version of Grasshopper, this is considered to be the most (and probably the only) efficient way to model dynamic network structures in Grasshopper. It is also possible to use other optimization tools (such as Flexhopper) beside Kangaroo. A further work seen as a result of this work is the creation of SpaceChase-specific and unique simulation components to remove Kangaroo from the workflow. This development was not carried out within the scope of this project as it would require much deeper research and support. With the addition of such a capability in a future study, it is possible to say that SpaceChase will have much more advanced controls for activating syntactic networks. It can be predicted that this study will also be the beginning of an independent (stand-alone) software study that can work outside of Grasshopper and Rhinoceros.

● **Machine learning:** An advanced topic that this study opened up for discussion is the utilization of big data by tools such as SpaceChase. The SpaceChase plug-in, which was developed for analytical and observational purposes from the beginning of the study, has features that can lead to future studies in terms of processing and recording network data. The similarities between the network structures, which are the representation environment of SpaceChase, and artificial neural networks, which are the representation environment of machine learning, are remarkable.

According to recent studies, Space Syntax software may be used in the design process to inform designers not only about how the spaces they create at the end of the process will be evaluated, but also to make their original design ideas visible and debatable (Dursun et. al 2022, Kozikoğlu et. al 2021). The major goal is to create strong links between analytical, creative, and productive approaches. These experimental investigations, such as SpaceChase, are said to contribute new qualities to space syntax research while also providing scientific/creative tools for architects to interact in the design process.

ACKNOWLEDGMENTS

The study presented in this paper was supported by TÜBİTAK ARDEB 1001 scientific project number 119M082. More information about the project can be found at the spacechase.app website.

REFERENCES

Coates, P. (2010) Programming.Architecture. London: Routledge

Dalton, N. (1997) Pesh. UCL, London.

Dijkstra, E. W. (1959) ‘A Note on Two Problems in Connexion with Graphs’, Numerische Mathematik. Vol. 1, December 1959, pp.269–271.

Dursun, P. (2007) ‘Space Syntax in Architectural Design’, In: Proceedings of 6th International Space Syntax Symposium. ITU Faculty of Architecture, Istanbul, 12-15 June 2007, pp.056.1-056.12.

Hillier, B. (1996) Space is the Machine. Cambridge: Cambridge University Press.

Hillier, B., Hanson, J. (1984) The Social Logic of Space. Cambridge: Cambridge University Press.

Kozikoğlu, N., Dursun Çebi P. (2015) ‘Thinking and designing with the idea of network in architecture’, ITU A|Z, 12(3), November 2015, pp.71-87.

Kozikoğlu, N., Dursun Çebi, Aldemir Celen, A., (2021) ‘Mapping relations as a design strategy, physical attraction forces correlation for design thinking’, ITU A|Z, 18(3), November 2021, pp.597-610.

Kozikoğlu, N., Dursun Çebi, P., Yazar, T., Balaban, B., Üneşi, C., Erden, M.S. (2020) ‘Dynamic Architectural Canvas: Designing a Relational Mapping-based Architectural Design Tool’, In: Proceedings of 38th eCAADe Conference, Vol. 1, TU Berlin, Berlin, Germany, 16-18 September 2020, pp.229-238.

Lawson, B., Dorst K. (2009) Design Expertise. Oxford: Elsevier.

Nourian P., Rezvani, S. and Sariyildiz, I.S. (2013). ‘A Syntactic Design Methodology: Integrating Real-time Space Syntax Analysis in a Configurative Architectural Design Process’, In: Proceedings of the Ninth International Space Syntax Symposium, Seoul, 13 December 2013, pp.048.1-048.15

Nourian, P., Rezvani, S., Sarıyıldız, S. (2013) ‘Designing with Space Syntax, A Configurative Approach to Architectural Layout, Proposing a Computational Methodology’, Computation and

Performance, In: Proceeding of 31st eCAADe Conference, Vol. 1, Delft, the Netherland, 18-20 September 2013, pp.357-365.

Nourian, P. (2016) Configraphics: Graph Theoretical Methods for Design and Analysis of Spatial Configurations. Ph.D. thesis. Delft University of Technology.

Oxman R., (1997) 'Design by Re-representation: a Model of Visual Reasoning in Design', Design Studies, 18(4), pp. 329-347

Schön, A. D. (1987) Educating the Reflective Practitioner. San Francisco: Jossey-Bass Publishers.

Turner, A. (eds) (2007) 'New Developments in Space Syntax Software', In: 6th International Space Syntax Symposium, Istanbul Technical University: Istanbul, Turkey

Varoudis, T., Psarra, S. (2014) 'Beyond Two Dimensions: Architecture Through Three Dimensional Visibility Graph Analysis', The Journal of Space Syntax, Vol 5. pp.91-108.

Yılmaz, E. A. (2016) Oyunlaştırma. İstanbul: Abaküs.