APPLIED GRAPH MACHINE LEARNING IN A THREEDIMENSIONAL ARCHITECTURAL CONTEXT

**Abstract.** Building on previous research on graph machine learning in architecture, this paper investigates topics that so far have been discussed only briefly or not at all. This includes data-generation and -representation, readability, understanding, accessibility and use of software. We primarily focus on understanding how the graph autoencoder deals with differently structured datasets as input, the look and distribution of the latent space and further the interpretation of the resulting, newly generated 3D models.   
The importance here lays mainly on the further understanding of how the graph autoencoder deals with differently structured datasets as input, the look and distribution of the latent space and further interpretation of the resulting, newly generated 3d models.   
We ask how to generate a dataset that, compared with previous results, will be distributed more homogeneously in the latent space to enable the graph autoencoder to produce geometrically logical and original results that follow the idea of space given in the dataset by improving the input rather than changing the architecture of the graph auto encoder. A new method of data generation as well as multiple options for data representations are tested and analyzed. Generated results are compared with each other and with the dataset for maximal understanding of the graph auto encoder.   
A second goal is improving the accessibility of the research. Instead of keeping the application only inside the Rhino/Grasshopper environment, we also use web-based tools to guarantee easy access for researchers and the public to understand the system further.

**Keywords.**  generative 3D architecture, generative graph machine learning, graph-based architecture, human-computer interaction, graph autoencoder, latent walk

1. Introduction

Here we present an extended and improved version of spatial generation with a graph autoencoder presented originally in Bauscher et al. (2024). The original paper presents a generative AI system the outputs new spatial configurations of architectural elements in three dimensions. The graph based autoencoder can read 3D buildings as training input and does not rely on any sort of conversion between 2D images or plans and a desired three-dimensional output in post. While this new approach shows high potential and is well rooted in the modern idea of the architectural design process, it also still has its weak points and open questions:

1. The used dataset consists of only four original buildings, that are each augmented a hundred times parametrically. As Aroyo et al. (2022) argue data might be the most important factor in successfully performing AI systems, and the slight geometrical augmentation of only 4 buildings is not ideal for training.
2. The conversion from 3D model to graph shows huge impact on the generated outputs and might not be perfectly suited for the task.

While these points offer room for improvement

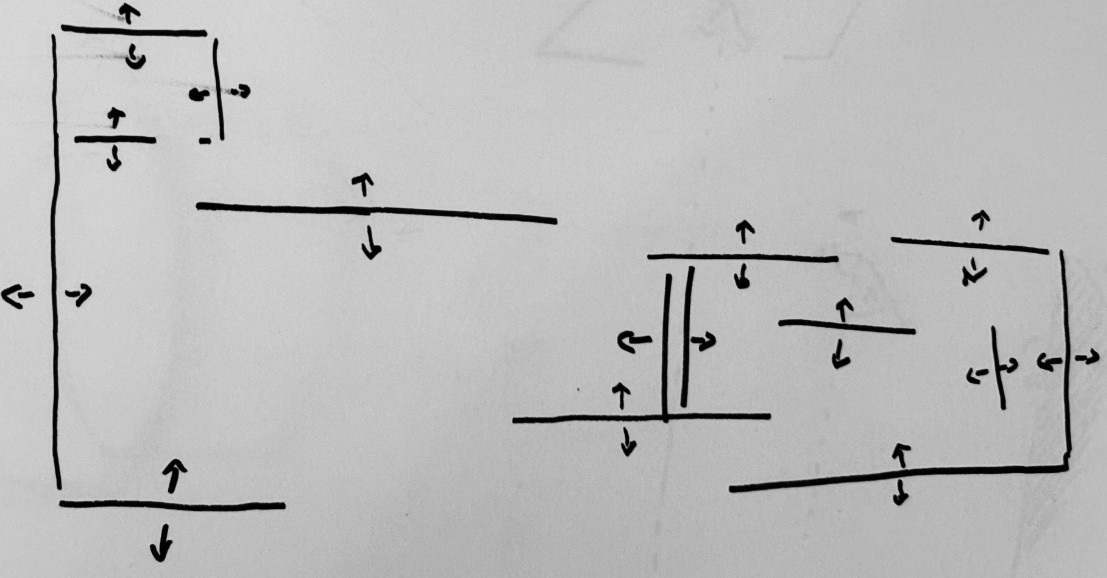
1. Original Methodology

## 1.1 DATASET

The custom dataset originally used consists of four modernist architectures, that have been individually remodeled parametrically in Grasshopper inside Rhino. That gave the opportunity to easily augment each building in its geometry to create more data for training the autoencoder. All houses are from the same architectural period, modernism, for the ease or remodelling and a geometric continuity across all data. The buildings are:

* Mies van der Rohe's Barcelona Pavilion (1929)
* Ray and Charles Eames' Eames House (1949)
* Mies van der Rohe's Farnsworth House (1951)
* Pierre Koenig's Stahl House (1960)

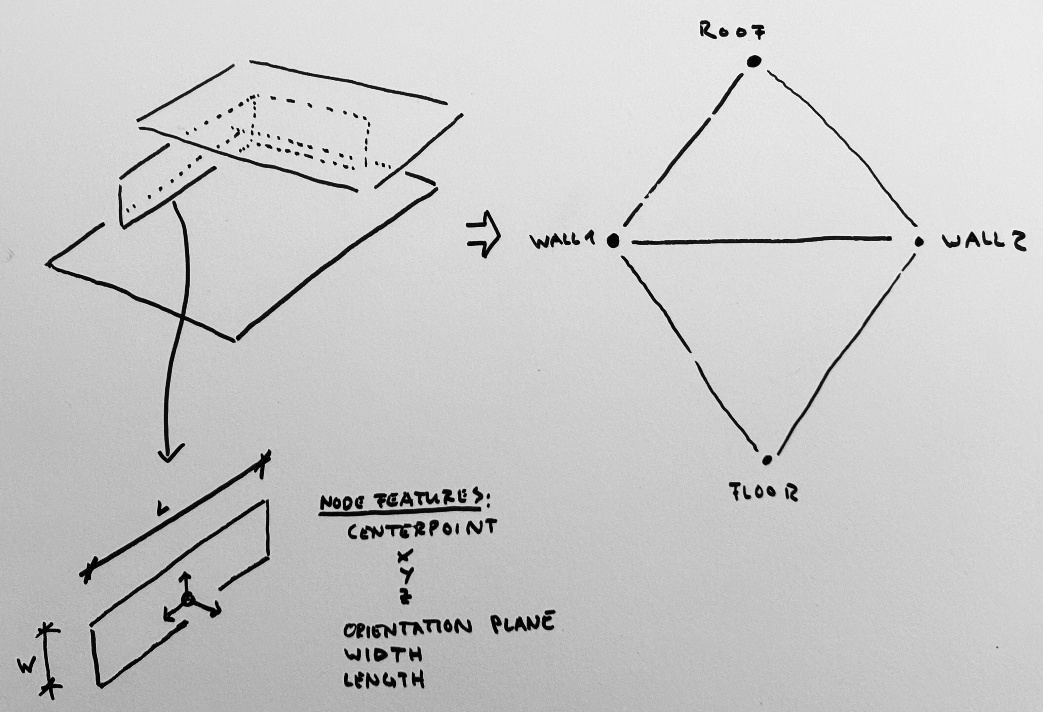
Those houses all follow the design principle of only using orthogonal walls, as well as mostly having full height openings for windows. Yet they vary greatly in size, proportion and location. They are also all very well documented, two of them being part of the case study houses series (Eames, Koenig) (QUOTE?) while the other being two of the best-known examples of modernist architecture (both Mies van der Rohe projects).

To avoid overcomplication in the 3D model the buildings were remodelled as a surface model. That means that walls, floors and ceilings are all seen as individual, rectangular surfaces without any thickness, while doors and openable windows are just empty left spaces. This supports the idea of trying to model space defining elements only, so there is no need for i.e. any structural considerations.

This approach fails in providing the autoencoder with a homogeneous dataset mainly due to the method of augmentation. Starting always from the original model, elements are only slightly moved and scaled randomly in one direction (Fig below), making the variation not being enough to allow for a smooth geometrical translation between the original buildings. The augmentations still always keep the same layout, element count, division between horizontal and vertical elements, overall size and ratio. The only value changing is geometrical proportion.

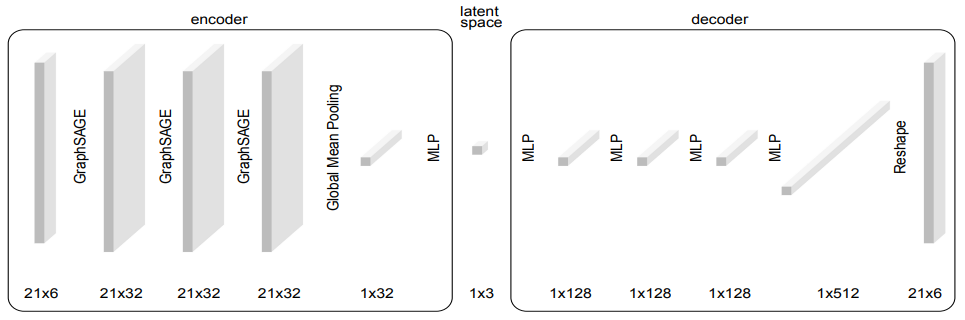
## 1.2 DATA REPRESENTATION

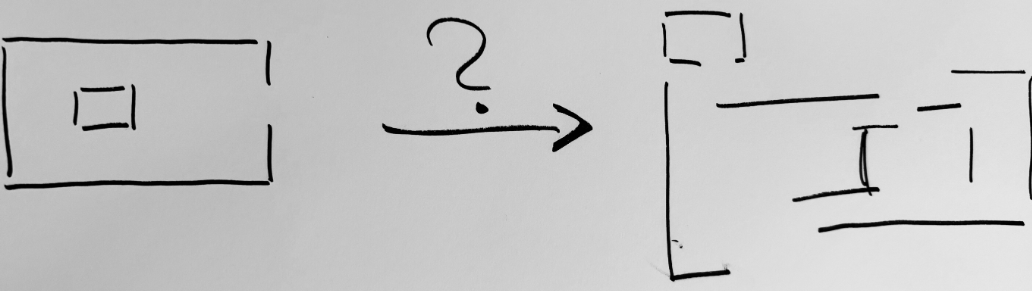
To make a 3D building machine-readable, it is converted into a graph. Graphs have been used for representing architectural space and buildings for a long time (Hillier, 1996) (Alexander, 1977) and are still used today in the databased environment of architecture (Rasmussen et al., 2020) (Elshani et al., 2022) (McGlinn & Pauwels, 2022). They can capture more information than most other data formats due to their non-discursive nature, thus provide the perfect basis for making 3D models of architecture machine readable.

Each surface-based building in the dataset is translated into an undirected graph, where all geometrical information is held in the node features. Each node represents one surface, and nodes are connected in the graph if they physically touch in the 3D model (Fig below). To encoder all necessary information, the node features contain the coordinates of the centre point, the orientation and the width and length of the surface. Here we have to point out that this specific method of conversion only allows for orthogonally positioned elements. The orientation value equals one of the three global planes (XY, XZ, YZ) and thus limits the geometrical options for input and output.

## 1.3 GRAPH AUTOENCODER

The auto encoder model (Fig below) is defined by encoder and decoder model. Here the encoder is a graph-based encoder as first described by Kipf and Welling (2014) consisting of message passing layers (Hamilton et al., 2018), while the decoder consists of linear layers that do not create or use any graph structure. That means all input data must have the same count of elements, because reconstruction the features though linear layers flattens the data into a one-dimensional vector with always the same length. While this poses some limitations to the system, the advantages of simplicity in implementation still outweigh them (Guo & Zhao, 2022).

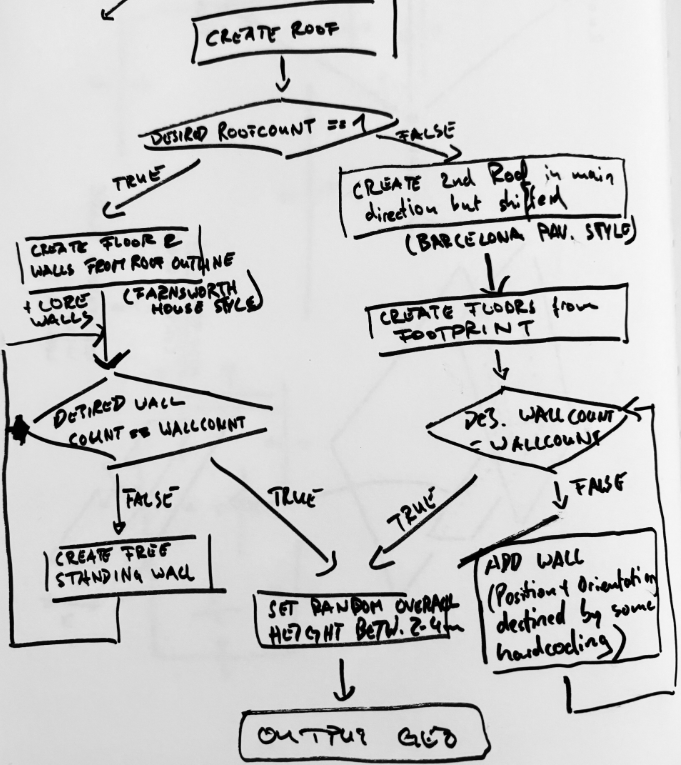


Another limitation is the chosen dimension of the latent space, 3, which restricts the performance of the model. Yet again, the simplicity and possibility of visualization outweigh the disadvantages, also considering that the model itself might be the least deciding factor on the performance of the whole system, and data collection and preparation are far more important (Jarrahi et al., 2023)

1. Adapted Methodology

## 2.1 DATASET

Aroyo et al. (2021) point out: “Data is potentially the most under-valued and de-glamorised aspect of today’s AI ecosystem” and “Benchmark datasets are often missing much of the natural ambiguity of the real world”. Both of these points can be applied to the original research on graph autoencoders in architecture. The second point is harder to approach due to the limiting number of three-dimensional datasets of buildings ready to use. The usage of a 2D floor plan dataset as they are widely available (de las Heras et al., 2015; Kalervo et al., 2019; Wu et al., 2019) seems to not only defeat the conceptual idea of the research project but also still pose multiple questions on the quality and origin of the data.

Therefore, the first point quoted can be addressed by further developing the existing dataset towards a more homogeneous landscape for better overall performance. To allow for a deeper involvement, we shrank the original number of houses from four to two, continuing development only with Mies van der Rohe’s Farnsworth House and Barcelona Pavillon. By choosing two architectures by the same architect, which differ in size, typology, proportion and layout but follow similar geometrical design principles, we open up the room for a more seamless transition between one another (Fig above).

Instead of starting with the original 3D model of the building for augmentation, we now use only the design principles applied to those buildings in an iterative algorithm (Fig below).

I.e. the Farnsworth House consists of that ring of facade panels (here: Glass) that is the same as the footprint of the roof and floor, while in the Barcelona Pavillon walls are always positioned in an offset of at least one meter in regard to floor and ceiling.

Through the iterative nature, not only boundary conditions (the original architecture slightly augmented) are captured, but also geometries that display an in-between solution. This results in buildings with different numbers of elements, as well as different layouts, and overall size and proportion.

The buildings are still model on the basis of architectural elements floor, ceiling and wall, which are all represented as rectangular and orthogonal surfaces due to the nature of the modernist style of architecture.

## 2.2 DATA REPRESENTATION

As the used conversion process from 3D model to graph and back has proved to be highly influential on the generated outcome from the AI system, we introduce now three options (Fig below) then can be compared against each other for more transparency and understanding.

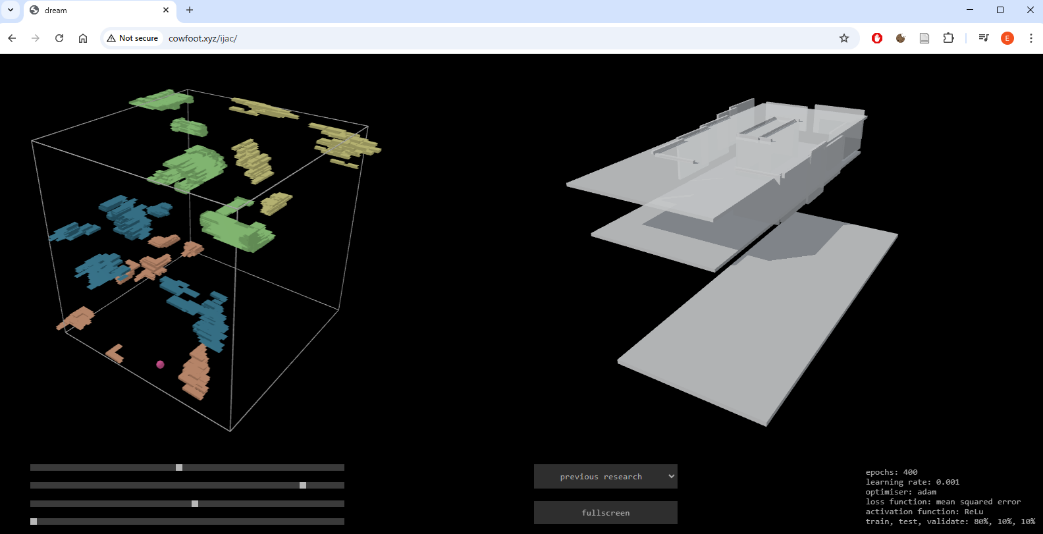
Option A is what was used before and generated controllable and valuable results, but with limitations. Each surface in the model thus node in graph is represented by its center point, width and length, and orientation in regard to the global world planes. While movement in X, Y, Z as well as changes in width and height is continuous, the rotation values are discrete and allow for three different cases.

Option B mostly copies option A but differs in the representation of rotation. Here, continues rotation around the global Z axis is allowed for more freedom in generation while still maintaining the rectangular definition of the surface.

Option C defines the surface by its corner points instead of width – height – center. This is the option with the most geometrical freedom, but also with the highest potential of creating faulty and illogical geometries if the concept of the corner points is not learned properly be the autoencoder.

## 2.3 GRAPH AUTOENCODER

The graph autoencoder model stays mostly the same as described in paragraph 1.3. The main limitation before was due to the decoder architecture, the buildings in the dataset must all be made up by the same number of surfaces, which now is the only change to the model. We introduce masking to the decoder, which now decodes the latent vector to be a one-dimensional vector as big as the input data with the greatest number of elements. If less elements have been input, we use a mask to decode only as many elements as needed. While this does not change the system of the training process, it does introduce a fourth variable to the generation process, where we can now specify the number of desired elements to be generated in addition to the three latent variables.

1. Results
2. Accessibility

Instead of relying purely on the Rhino and Grasshopper environment as done before, we use web-based tools to display the trained model quickly in the browser. Before, accessing the model, which is implemented using the Python machine learning framework PyTorch (Paszke et al., 2019) and its library PyTorch Geometric (Fey & Lenssen, 2019), was done inside Grasshopper using the HOPS component. While this works, the process is laborious, needs a lot of attention to make sure the right Grasshopper-Spaghetti are where they are supposed to be, and requires multiple software to run at the same time.

This is now streamlined, where the python script executing the training of the ML model saves the model, its performance statistics and latent space information into a specific location from where it can be accessed by the web-based application. This application (Fig below) is using three.js (*Three.Js – JavaScript 3D Library*, n.d.) to display generated 3d models and latent space, gives control for the latent space variables and lets the user quickly switch between the different trained models (which before was the hardest part to do in Grasshopper).

As already proved previously by the physical application involving a MIDI controller communication live with Grasshopper and HOPS, this improves general access and understanding for both developer and architect immensely and should be persuaded more by academic research for architectural applications.

1. Conclusion
2. References