The Polycentric Region, the Productive City, and the Adaptable Building: Reflections on the Implementation of an Urban Intervention

Oliver Dunkel

Diplomarbeit

The Polycentric Region, the Productive City, and the Adaptable Building: Reflections on the Implementation of an Urban Intervention

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Diplom-Ingenieurs/Diplom-Ingenieurin
unter der Leitung

Mladen Jadric, Ass.Prof. Arch. DI. Dr.techn.

Institut für Hochbau und Entwerfen
Hochbau und Entwerfen E253-4

eingereicht an der Technischen Universität Wien
Fakultät für Architektur und Raumplanung von

Oliver Dunkel
01326856

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Wien, am 30.05.2022

Abstract

The urgency of implementing urban planning measures to reduce greenhouse gas emissions is manifested by analyzing current climate status reports. The theoretical part of the present work provides an overview of the relevant topics of urban planning concerning this matter in Vienna. These are examined from the perspective of the region, the city, and the individual building.

The findings of this examination are tested on a conceptualization of an urban intervention located at the Währinger Gürtel. The implemented urban structure is designed at a large scale and adaptable to various uses. By outlining a hybrid structure which is partially consisting of wood, current trends in sustainable construction are taken into account. In addition, this structure is composed of a canon of serial building elements which are algorithmically calculated and pre-fabricated off-site. The aim is to save material and further the economy of the building. Sustainability concepts are implemented by employing reused scaffolding components as the primary building material for the open spaces attached to the facade. Due to the task chosen for the practical part, this thesis deals particularly with the three-dimensional design of spatial support structures as well as with the implementation of green facades.

Kurzfassung

Durch die Analyse aktueller Klimastatusberichte wird die Dringlichkeit der Umsetzung städtebaulicher Maßnahmen zur Reduktion von Treibhausgasemissionen manifestiert. Der Theorieteil der vorliegenden Arbeit bietet eine Übersicht über die diesbezüglichen Themenfelder der Wiener Stadtplanung. Zur Verdeutlichung werden diese aus der Perspektive der Region, der Stadt und des einzelnen Gebäudes betrachtet.

Die daraus gewonnen Erkenntnisse werden an der Konzeption einer städtebaulichen Intervention am Währinger Gürtel erprobt. Zur Umsetzung gelangt eine großmaßstäbliche, nutzungsneutrale Struktur im städtischen Raum, die den aktuellen Trend der emissionsarmen Konstruktion in Holz- bzw. Hybridbauweise aufgreift. Außerdem soll sich diese Struktur aus einem Bauteilkanon zusammensetzen, dessen serielle Elemente algorithmisch berechnet und abseits der Baustelle industriell hergestellt werden. Hierdurch kann Material eingespart und die Ökonomie verbessert werden. Um der Nachhaltigkeit Rechnung zu tragen, bestehen die an der Fassade angeschlossenen Freiräume aus recycelten Bauteilen – genauer gesagt aus ausgemusterten Baugerüstteilen. Bedingt durch die für den Praxisteil gewählte Aufgabenstellung, setzt sich diese Arbeit im Besonderen mit der dreidimensionalen Ausgestaltung räumlicher Tragstrukturen, ebenso wie mit der Umsetzung grüner Fassaden auseinander.

Table of Contents

Acknowledgments

I want to thank Mladen Jadric, Ass.Prof. Arch. DI. Dr.techn. for his mentorship over the last years and his exceptional support.

Furthermore, I am grateful to Dr. Robin Hodgson for proofreading my work.

1. Introduction

Vienna's average temperature will have changed drastically at the end of this century and will be comparable to the average temperature of cities like Marseille and Dakar. The actual amount of change will be determined by the measurements of climate protection taken and their effectiveness. Studies suggest that Vienna will be one of the most affected cities by the climate crisis in Europe.

The records show that the average temperature has risen by 2 degrees since the 1970s, even by 3 degrees in Vienna. The change is perceptible by Vienna's citizens when today's temperatures exceed 40 degrees and do not fall below 20 degrees during nighttime.

From 1961 to 1990, ten days of the year exceeded 30 degrees and therefore are accounted as heat days. While heatwaves usually lasted for five days on average during the last 30 years, this could be 28 days by the end of this century. From 2015 to 2020, the records show an average of 33 heat days. This drastic increase does not affect everyone's comfort in the

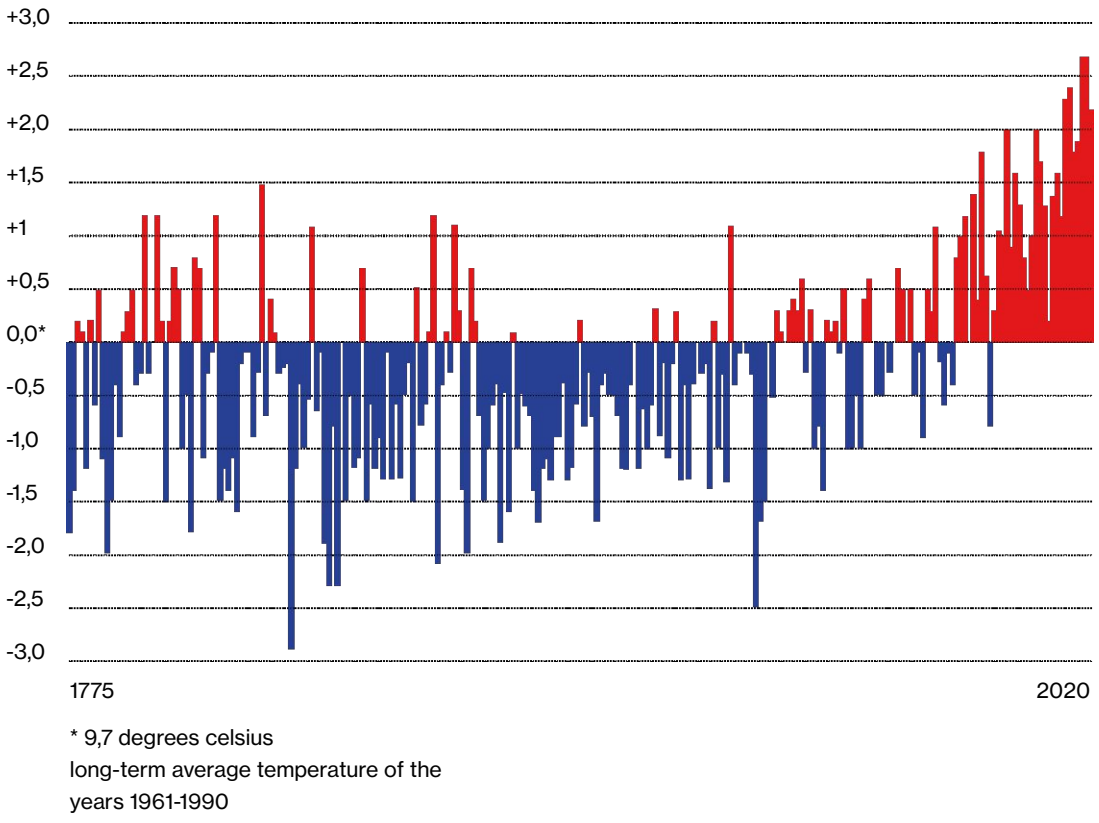


Figure 1: The deviation of the annual mean air temperature from the long-term average temperature of the years 1961 to 1990 in degrees Celsius.

same way. So is the impact on poorer households more significant due to smaller living spaces and rising expenses on air conditioning. Physically restricted persons and the elderly may suffer physical afflictions due to lower tolerance to high temperatures. When considering the impaired cognitive capabilities and productivity of city dwellers, heightened temperatures even lead to a declining economy and an overloaded health care system.[1]

The Paris Agreement dictates a maximal increase by 2 degrees, at its best by 1.5 degrees of the average temperature, based on pre-industrial times. Furthermore, it is agreed to reduce global greenhouse gas emissions by 55 % til 2050. The EU wants to commit to a more ambitious plan named the European Green Deal. Therefore, the EU aims to reduce 55 % by 2030, based on the emissions of 1990, pursuing CO2

1 Municipal Department 20 – Energy Planning, Wiener Klimafahrplan: Unser Weg zur klimagerechten Stadt (Vienna: Municipal Administration, March 2022), 16, https://www.wien.gv.at/umwelt-klimaschutz/pdf/klima-fahrplan-lang.pdf.

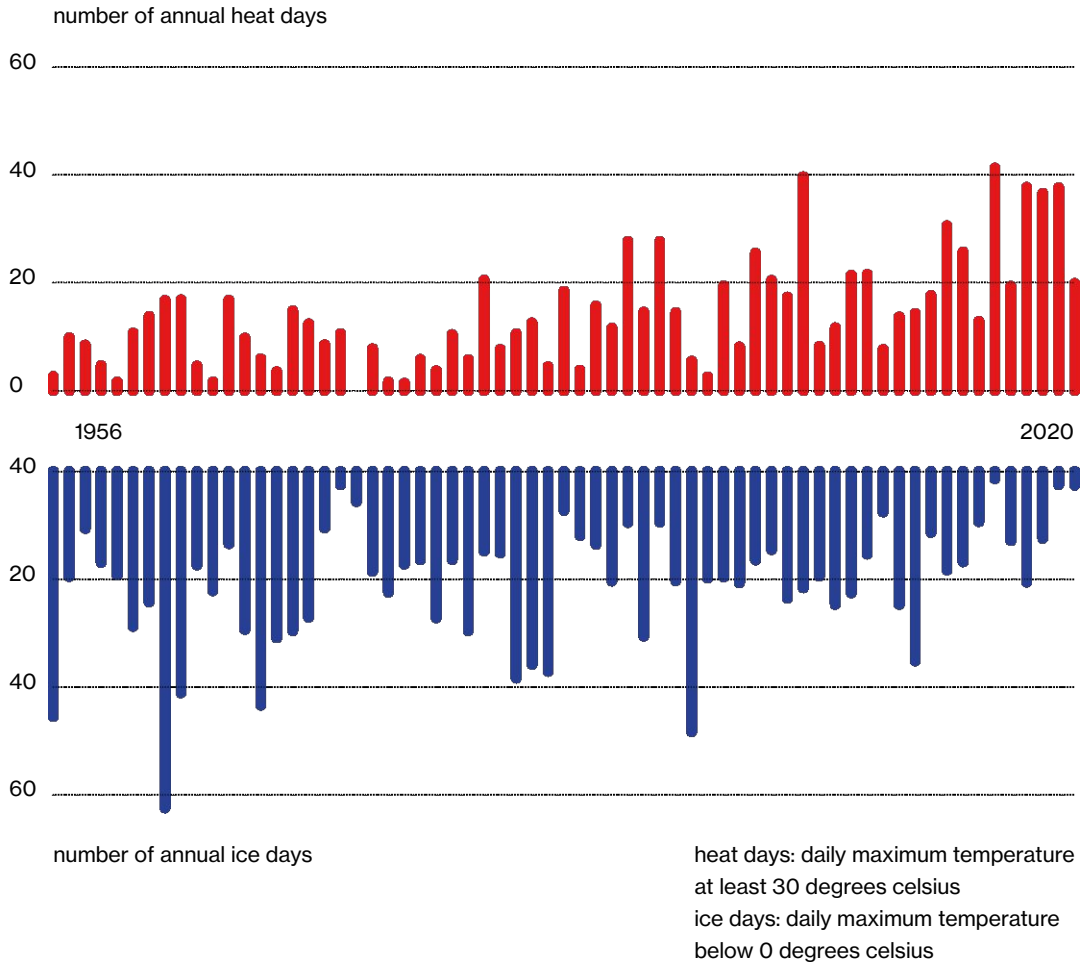


Figure 2: A visualization of the observable climate change.

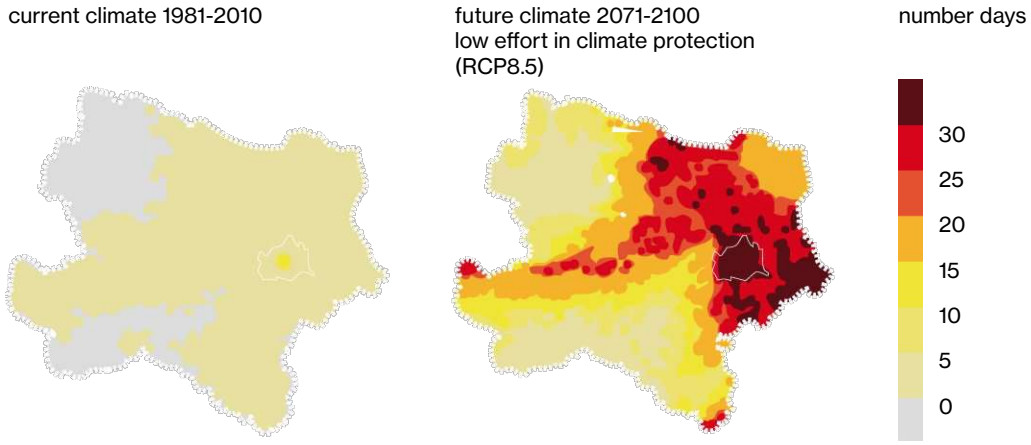


Figure 3: The map shows the average number of tropical nights in a year when measurements for climate protection implemented are of low ambition. A night counts as a tropical night if the daily minimum temperature is not below 20 degrees.

neutrality by 2050. On the national level, the Austrian government pursues CO2 neutrality and renewable energy supplies of 100% by 2030.

Apart from climate protection, the emphasis lies on climate adaptation and the implementation of a circular economy.[1] The number of dwellers in Vienna will reach 2 millions before 2030. Recent developments, such as the war in Ukraine, will contribute to the growth.[2] Vienna will reduce its material footprint concerning consumption per head by 30 % until 2030, 40% until 2040, and 50 % until 2050.[3] Vienna will reduce its local energy consumption per head by 30 % until 2030

1 Municipal Department 18 – Urban Development and Planning, Smart Klima City Strategie Wien: Der Weg zur Klimamusterstadt, (Vienna: Municipal Administration, February 2022), 19, https://smartcity.wien.gv.at/wp-content/uploads/sites/3/2022/03/scwr_klima_2022_web-neu.pdf.

2 Municipal Department 18, Smart Klima City Strategie Wien, 16.

3 Municipal Department 20, Wiener Klimafahrplan, 32.

and by 45 % in 2040 compared to 2005.[1]

Vienna aims to implement a circular economy as well as a sharing economy. The high consumption rate of materials is one of the main factors concerning greenhouse gas emissions. Future products will be more durable and repairable. Reusing materials and products contributes to building a circular economy. If not possible, products and materials are transformed into secondary raw materials. Waste re-enters the raw material cycle by recycling. Only a tiny proportion of waste is recycled today when excluding thermal recycling. Thermal recycling is not preferable because of the permanent loss of materials.[2] The CO2 emissions of the transport sector per head will decrease by 50 % until 2030 and 100 % until 2040. An additional challenge is the implementation of a nearly zero-emissive commercial traffic by 2030 in the inner city. Vienna will not only be fully accessible by public transport but also be a walkable city, meaning the daily necessities of its city dwellers will be at a close distance, more precise in 15 minutes walking distance. Shortly, each citizen will reach a high-quality green space at a maximum walking distance of 250 m. By 2030 the number of private-owned cars will be down to 250 per 1000 city dwellers. Therefore it will be possible to reduce the parking infrastructure while converting the freed space into green infrastructure.[3]

In particular, measures concerning

1 Municipal Department 18, Smart Klima City Strategie Wien, 40.

2 Municipal Department 18, Smart Klima City Strategie Wien, 34.

3 Municipal Department 18, Smart Klima City Strategie Wien, 41.

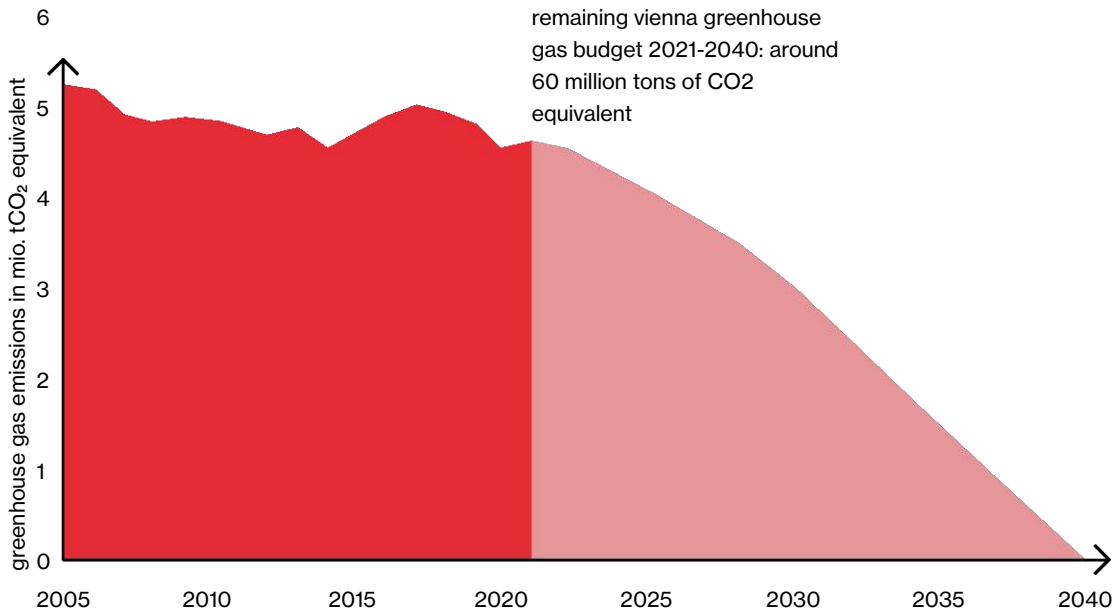


Figure 4: The trajectory of greenhouse gas emissions in line with targets.

the building industry include using a building envelope for energy production and as an additional space for greenery. Applied shading devices protect against solar radiation to prevent summerly overheating. Therefore strategies of passive cooling are sufficient. In a best-case scenario, structures will be widely demountable and re-useable, or at least separable in their raw material components by 2030. More precisely, by 2040, 70 % of a structure, such as building elements, products, or materials, will be re-useable.[1]

The characteristic of city-friendly mobility is its availability for all city dwellers. A city design ensuring walkability shows a higher density of its districts. Daily necessities like workspaces, schools, healthcare facilities, food suppliers, green spaces, and more are reachable by foot. Furthermore, a diversified street scenery invites to stroll and roam. Traffic design within the accessible areas prioritizes citizens' safe and comfortable usage. More precisely, streets have to stay cool during summer, and infrastructures for active mobility like walking and cycling have to be spacious. For Vienna, that means planting more trees and greening facades.

On the one hand, the evaporation of the greenery cools the street scenery; on the other hand, fully grown trees shade open space. Vienna will extend its cycle path network while closing gaps and widening its sidewalks. Noncommercial open spaces intended for having a rest are provided evenly in the future cityscape.

Commercial vehicles will be electrified, including courier, delivery, and transport services. Concepts in logistics like the last mile are to be implemented in urban design.[2]

Commercial and residential buildings are designed to be long-lasting and efficient in material usage. Vienna minimalizes its material footprint by considering structures as material storage for the city. By generating layouts that are adaptable to different kinds of users, the lifetime expectancy of buildings can be extended.[3]

1 Municipal Department 18, Smart Klima City Strategie Wien, 42.

2 Municipal Department 18, Smart Klima City Strategie Wien, 54.

3 Municipal Department 18, Smart Klima City Strategie Wien, 60.

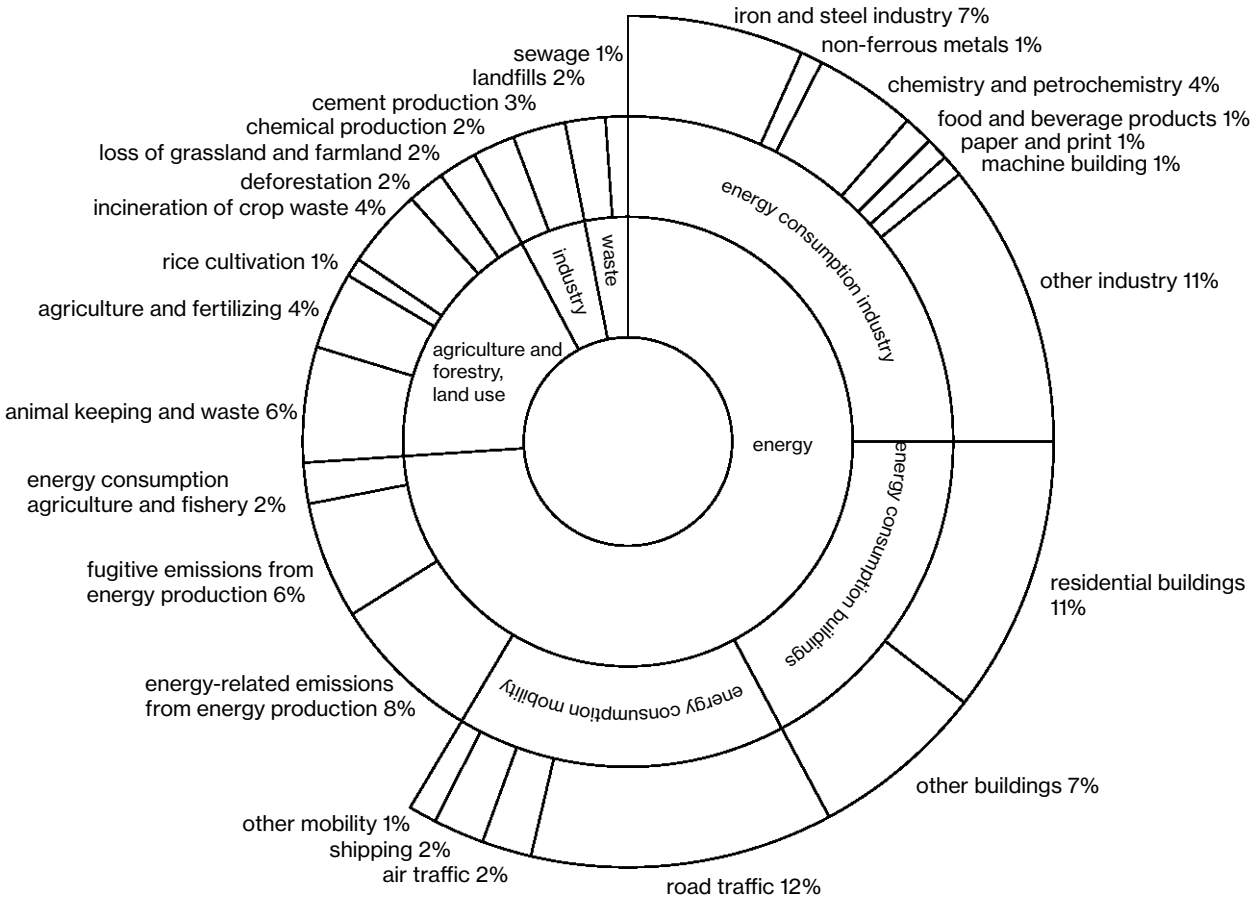


Figure 5: The contributors to global greenhouse gas emissions in 2016.

2. The Polycenric Region

In Europe, urbanization manifests primarily in movements from small towns to the major cities and from rural areas to suburbias. This process causes gentrification, meaning that middle and low-income households have to move to the city's outskirts as a result of rising rents. Furthermore, the risk of overloading a cities' infrastructures increases due to the rise of traffic density.[1]

Low- and middle-income households have to relocate as a result of rising rents, if a city center is gaining more and more significance. The social fabric eventually segregates. Therefore Alexander Christoph suggests the concept of decentralization, meaning the development of additional secondary centers. It is essential to equip these secondary centers with a catchment area inhabiting a sufficient number of dwellers to support them. These secondary centers can adopt unique features to become of higher regional significance and attract dwellers living outside their catchment area. For example, specialized departments of a city's primary health care infrastructure could be spread among these centers, providing maximum diversity.[2]

1 Mathias Mitteregger et al., AVENUE21. Automatisierter und vernetzter Verkehr: Entwicklungen des urbanen Europa (Berlin: Springer Vieweg, 2020), 18.

2 Christopher Alexander et al., Eine Muster-Sprache: Städte, Gebäude, Konstruktion, 2th ed., trans. Hermann Czech (Wien: Löcker, 2011), 61-65.

2.1 The Car-Friendly City

The *Athens Charter*, published in 1933, on the one hand, and the *Leipzig Charter*, published in 2007, on the other hand, are contradictory milestones in the European cities' early history of traffic and mobility politics. The era between these dates underwent a radical paradigm shift and consists of three phases:

1. The substantial increase in private car ownership due to a general growth in wealth and pervasive enthusiasm concerning the automobile.

2. Arising doubts about the idea of a car-friendly city, caused by the city dwellers' evident distress – the number of private-owned cars climaxes as a result of this.

3. Finally, a decrease in the number of private-owned cars, pursuing the ideal of a liveable city.[1]

The extensive destruction of many European cities during World War 2 provided the opportunity to redesign the very structure of the cities. Built upon the idealism of the *Charta of Athens*, the concept of the structured and broken-up city developed by Göderitz, Hoffmann, and Rainer in 1957, provides the underlying idea for this mission. When the book *Die Autogerechte Stadt* written by Reichow and Hans Bernhard, was published in 1957, city planners reasoned that a car should encounter no hinderances on its way. Now the remnants of traditional cities were at risk of being demolished by the urge to separate a city's functions.

In the late 1960s, the high motorization rate among city dwellers led to urban sprawl. In the late 1960s and early 1970s, a countermovement started. In 1965, Alexander Mitscherlich published the book *Die Unwirtlichkeit der Städte*. He criticized the sole focus on functionalism in modern city planning.[2]

In Vienna, city planning politics aimed at implementing the ideal of modernism. So the Executive City Councillor for Urban Planning, Franz Novy, argumented in 1946, that the structure of the liberal and medieval town is not only wrong but also even dangerous and that it is not suitable to form a habitat for a future society. The new urban order has to follow the concept of separation of functions. Therefore it changes the urban life inevitably.[3]

1 Mitteregger et al., AVENUE21. Automatisierter und vernetzter Verkehr, 25.

2 Mitteregger et al., AVENUE21. Automatisierter und vernetzter Verkehr, 23.

3 Rudolf Scheuvens et al., "Wien: polyzentral: Forschungsstudie zur Zentrenentwicklung Wiens," (Werkstattbericht Paper No 158, Municipal Department 18 – Urban Development and Planning, Vienna, 2016), 48, https://www.wien.gv.at/stadtentwicklung/studien/pdf/b008475.pdf.

The Wiener Gürtel partially circles the central city district and is Vienna's busiest main road. In the second half of the 20th century, the traffic axis was known for its red-light-milieu and heavy traffic. Since then, the axis built on the former foundations of the fortified town walls has been reactivated.

Other essential traffic axes are the Ringstraße – the circular main street surrounding the first district and housing government buildings, museums, and various historical sites – and the radial streets leading the traffic in and out of the city. Accelerated by the motorization rate, these arterial roads have lost their original multifunctional character and became single-purpose traffic axes.[1]

One of the most critical aspects of city dwellers who take part in the city's economic, social, and cultural life is mobility. A dense mixed-use urban tissue leads to accessibility by foot or by bike. Furthermore, dense population enables the operation of high-ranking public transport in an economical way. The private vehicle becomes obsolete. On the contrary, the urban sprawl provokes a rise in motorization and an interspersing of spacious monofunctional structures.[2]

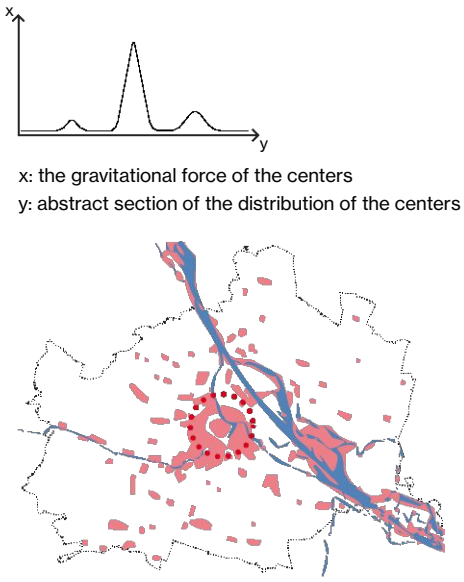


Figure 6: The pedestrian city around 1850. The half-hour radius on foot can be seen.

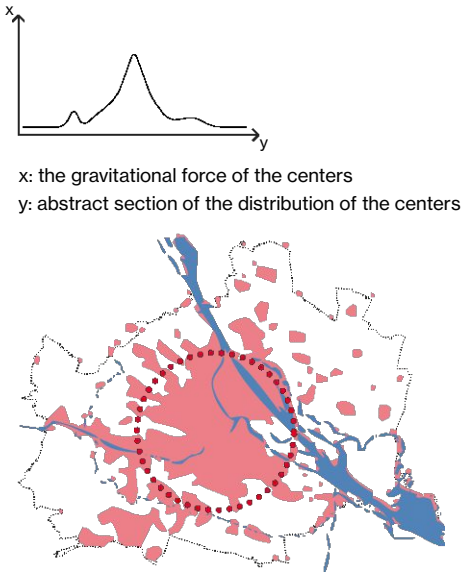


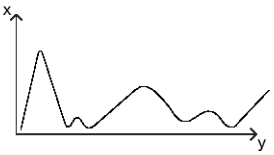
Figure 7: The tram city around 1910. The half-hour radius by the tramway can be seen.

1 Scheuvens et al., "Wien: polyzentral," 80.

2 Scheuvens et al., "Wien: polyzentral,"

Above all, in cities, the potential of the dwellers themselves is great to actively shape the future of mobility. A suggested strategy is to avoid the need for transportation, especially when it is environmentally harmful, by implementing modern ways of communication in our daily life that make encounters in person obselete. In those cases, where transportation is mandatory, the strategy is to avoid using a car.

The use of public transport, biking or walking is to be preferred. Last but not least, progressive change in the design of public space and the functional structure of the city should be advocated. City districts need to be compact to provide the daily necessities to their inhabitants within walking distance. Furthermore, concepts like the 'Last Mile' in the transport and delivery sector or the 'Superblock' in traffic planning ought to be implemented fast.[1]



x: the gravitational force of the centers
y: abstract section of the distribution of the centers

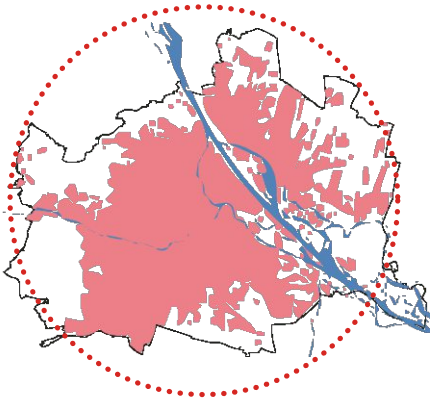
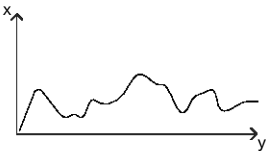


Figure 9: The car city around 1970. The half-hour radius by car can be seen.



x: the gravitational force of the centers
y: abstract section of the distribution of the centers

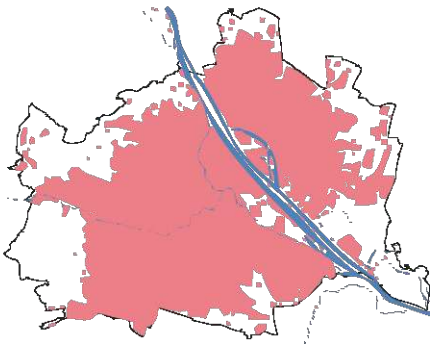


Figure 8: The walkable city in 2015. Continuing to build the city of short distances.

78.

1 Mitteregger et al., AVENUE21. Automatisierter und vernetzter Verkehr, 18.

2.2 The Revival of the Medieval Town

City centers are heterogeneous structures merging a variety of functions that define a city. Historically, one of the most important of these functions is the provision of daily goods. Today the characteristics of centers are more diverse. Since the increase of the dwellers' mobility, the centers of the cities began to specialize in niches and moved towards the outskirts. The formerly dense and walkable net of sub-centers began to vanish and its small stores managed by their owners were replaced by shopping centers in the suburbias. Other examples of specialized centers are office centers, entertainment centers, and education centers. In the recent past, international brands adopted the newly formed vacancy in the historic centers.

Today there is a renaissance of the traditional city center. Even shopping malls are now to be found on high frequented shopping streets. If done thoughtfully, this could even support the remaining small businesses by complementing their mix of offers. At the same time, with the upcoming e-commerce and delivery service, profound changes are recognizable.

While in general, commercial areas are continuously declining, more dwellers desire original products found in shops with a unique assortment. Therefore the former dense supply structure has to be rebuilt to support the concept of the walkable city.[1]

Nowadays, the holistic net of centers is more multifunctional than ever, but isolated structures are highly specialized. With the radical change of mobility, starting with the evolving public transport and later on reinforced by the private motorized transport, city dwellers are more attracted to extensive assortments than to their accessibility in walking distance.

Within the last few years, initiatives (e.g., *STEP 2025*) started, which are aiming at bringing back the formerly dense and therefore walkable net of centers by scattering their multifunctional fragments in the urban tissue. Thereby, city-dwellers' preferences should progressively shift from being a car owner to using public transport.[2]

The benefits of an ideal city center unfolds to its full potential when the following characteristics are in place:

1.) Centers are multifunctional and frequented by different user groups at different times.

2.) Centers are including retail,

1 Scheuvens et al., "Wien: polyzentral," 11.

2 Scheuvens et al., "Wien: polyzentral," 17.

various services, social infrastructure, cultural institutions, recreation parks, leisure centers, educational and other public facilities.

3.) Centers either already have or should be developed at various transfer points or axes of public transport and are therefore well accessible.

4.) Centers have unique aesthetic features in architecture and/or are of specific historical or cultural significance.

5.) Centers offer a variety of high quality open spaces.

6.) Centers are also fully accessible and traversable by foot or by bike.

7.) Centers offer non-commercial zones useable by a heterogenous group of users at all days a week.[1]

According to the spatial model of the *STEP 2025*, there are three different kinds of center structures: the metropolitan center, the main center, and the quarter center. They have to be evaluated in a hierarchical descending order, regarding their offer for the neighborhood and their significance for other parts of the town.

Metropolitan Centers have a significance on a supra-regional level and a public value far beyond the city borders. They have the highest-ranked retail and service offer, such as jewelers' shops and attorneys' offices. Furthermore, they are tourist hotspots. Their offers are highly specialized.

Main centers have a significance on a regional level and are attracting dwellers of the neighboring city districts as well. Their offers are moderately specialized.

Quarter Centers are of a significance for their close neighborhood. Their retailers offer a broad assortment of goods beyond the daily needs of the dwellers. By being close to metropolitan and main centers, their services complement the more prominent centers' offers.[2]

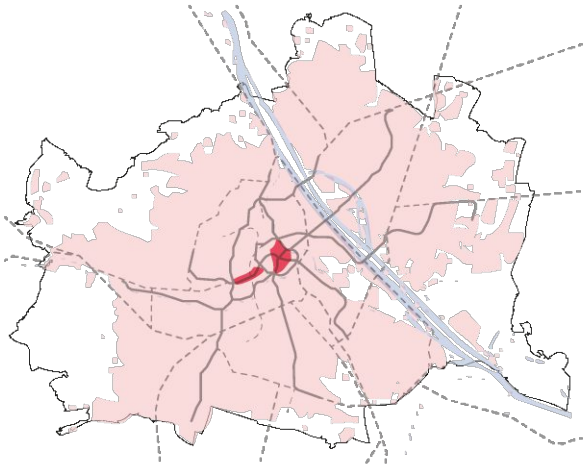
On this background their are certain future challenges regarding Vienna's centers to be observed:

1.) An increase of value regarding open spaces amongst urban society is recognizable. User groups want to actively take part in the creation and organization of recreational spaces.

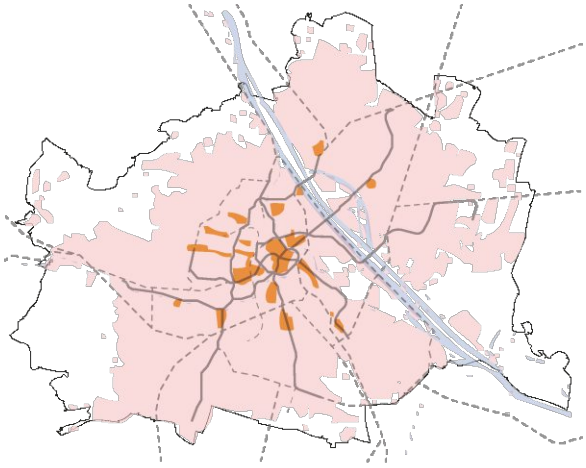
2.) An increased need for sufficient public transport and better accessibility by foot and by bike as well as new mobility concepts like vehicle sharing and on-demand-mobility.

1 Municipal Department 18 – Urban Development and Planning, "Mittelpunkte des städtischen Lebens: polyzentrales Wien," (Werkstattbericht Paper No 158, Municipal Department 18 – Urban Development and Planning, Vienna, 2019), 18, https://www.wien.gv.at/stadtentwicklung/studien/pdf/b008563.pdf.

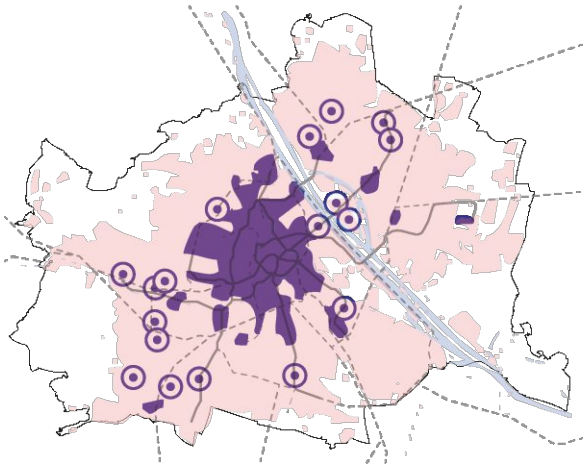
2 Municipal Department 18, "Mittelpunkte des städtischen Lebens," 30-31.



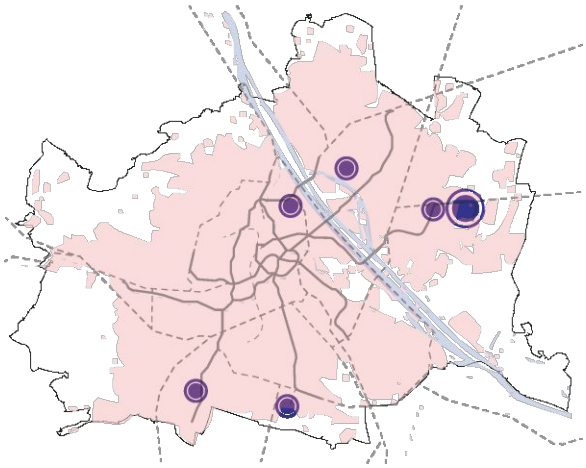
metropolitan centers



main centers



quarter centers / central areas



new centers

Figure 10: The Viennese spatial model of the polycentric urban structure.

3.) The need for developing new city centers by the expansion of high-ranking public transport systems. Only in the more significant urban expansion areas new centers will be sustainable.

4.) Mitigating against critical climatic conditions such as the urban-heat-island-effect and decreasing the city dwellers motorization rate by realizing the walkable city.(cf. *Smart City Strategy*)

5.) Social, demographic, and economic changes are altering the needs of user groups. An aging society has a greater need for non-commercial zones and recreational areas.

6.) Caused by the progressing digitalization, the money spent in local retail facilities decreases, despite rising population figures. Gastronomy and the offer of recreational activities are playing a vital part in supporting the local retailers.[1]

To achieve the politically intended walkable city, city planners have to completely change city center structures again, as they have done with the development of the car-friendly city as well. Therefore, a city center's multifunctionality is to be reestablished by offering a mix of goods and services, including sufficient integration of the public transport system. Furthermore, close-by living should be enabled to generate the needed user frequencies.[2]

Developing each of these structures as a single monofunctional project of its own could result in lifeless homogenous city districts. This leads to an increase in motorized individual transportation since only a mix of uses supports the development of walkable cities similar to those of the Viennese Gründerzeit.[3]

1 Municipal Department 18, "Mittelpunkte des städtischen Lebens," 23.

2 Scheuvens et al., "Wien: polyzentral," 22.

3 Silvia Forlati et al., Mischung: Possible!: Wege zur zukunftsfähigen Nutzungsmischung, 2th ed. (Vienna: Institut für Architektur und Entwerfen, Abteilung Wohnbau und Entwerfen, Technische Universität Wien, 2017), 15, https://smartcities.at/wp-content/uploads/sites/3/Mischung-Possible-Auflage-2-2-5.pdf.

3. The Productive City

The idea of the spatial separation of living and working was implemented in the industrial city by Tony Garnier in 1929 and is a predecessor of the *Athens Charter*. Because most artificial cities were incorporating the principles of the *Athens Charter*, Christopher Alexander asks in *A City is Not a Tree*:

"Is this a sound principle? It is easy to see how bad conditions at the beginning of the [last] century prompted planners to try to get the dirty factories out of residential areas. But the separation misses a variety of systems which require, for their sustenance, little parts of both."[1]

Christopher Alexander describes his idea of an ideal system with an example: a news rack, a traffic light, and a sidewalk are fixed parts of a system. When city dwellers have to wait at a red traffic light, they turn to the news rack because they have nothing else to do.[2]

Mixed used structures are more sustainable than monofunctional structures in the long term because they are more easily adoptable when economic and social changes occur. These ultimately lead to a higher ecologic value of these structures.[3]

To be seen as a mix of uses, the interlacing of structures for living and non-living functions has to be the catalyst of actual interactions. This is for example the case, when a broad spectrum of daily necessities of their occupants is within a walkable distance.[4]

Christopher Alexander's definition of a neighborhood resembles the Viennese Grätzl. Some common characteristics are the relatively small area covered and, therefore, a limited number of inhabitants. Furthermore, areas of reduced traffic or no traffic are required for such neighborhoods to be formed. One of the most crucial differences between those two concepts is in the definition of their border. Christoph Alexander's neighborhood has a strongly defined border, emphasized by a gateway. This gateway is not closed, in contrast to the concept of the gated community; instead, it serves as a symbolic entrance. The Viennese Grätzl has no generaly accepted border at all. The extention of a Grätzl is perceived differently and depends on personal perspectives.[5]

1 Christopher Alexander, "A City is not a Tree," in Design After Modernism: Beyond the Object, ed. John Thackara (New York: Thames & Hudson Ltd, 1988), 67-84.

2 Christopher Alexander, "A City is not a Tree," 67-84.

3 Forlati et al., Mischung: Possible!, 22.

4 Forlati et al., Mischung: Possible!, 23.

5 Christopher Alexander et al., Eine Muster-Sprache, 85-89.

Christopher Alexander differentiates between heterogeneous and homogeneous city populations. He suggests forming a homogeneous society by scattering subcultures evenly within the urban fabric. With this, a process of mutual adjustment is initiated. Necessary traffic infrastructure and high emissive industry are placed between these subcultures, forming a soft border implementing Christopher Alexander's idea of an enclosed neighborhood. By inserting commercial clusters in the urban fabric and positioning workplaces either next to or within residential buildings or even residential units, the forming of mixed-use neighborhoods is accomplished.[1]

Christopher Alexander's vision of an urban fabric formed by a mosaic of subcultures does not correspond with contemporary urban design methods. Today's experiences show that socially diverse city districts tend to segregate and form accumulations of subcultures anyway. Therefore it is appropriate to strengthen the social mix with most effort to eventually achieve an acceptable result.[2]

Industrial production is a significant part of the Viennese economy because of its above-average value added. Therefore it sustains and creates prosperity for its city dwellers. Employees in the Viennese industry are well paid and jobs are accessible by workers of lower qualification.

Communal services like the power generation in Freundenau, the wastewater treatment in Simmering, or the waste incineration in the Spittelau are essential in the value-added chain and, therefore, part of the productive city.[3]

Simultaneously, with Vienna's population growth, more employments of decent conditions also for lower qualified employees have to be generated in the nearby future. These should be settled not just in the service sector but also in the manufacturing sector. The Viennese city planning guidelines for a productive city aim at repositioning the industry and manufacturing sectors' role as an indispensable part of the Viennese economy. This goal is noticeable in a paradigm shift in city planning across borders, more precisely going from the post-industrial city to the productive city.[4]

1 Christopher Alexander et al., Eine Muster-Sprache, 45-59.

2 Eva Winroither. "Analyse: Warum Aspern Blau wählte," diepresse.com, October 13, 2015, https://www.diepresse.com/4842937/analyse-warum-aspern-blau-waehlte.

3 Municipal Department 18 — Urban Development and Planning, "Produktive Stadt," (Werkstattbericht Paper No 171, Municipal Department 18 — Urban Development and Planning, Vienna, 2017), 24, https://www.wien.gv.at/stadtentwicklung/studien/pdf/b008500a.pdf.

4 Municipal Department 18, "Produktive

It is noticeable that city dwellers are increasingly using their spare time to offer services or self-produced goods in the form of self-employment or as part of a sharing economy. This trend is a catalyst for strengthening city inhabitants' resilience and is to be supported by providing workshops, small retail spaces, or open workspaces.[1]

An open workspace is a facility that houses small companies like start-ups and individual entrepreneurs. The needed infrastructure like restrooms, meeting rooms, printers, and others are shared among the tenants.

A provider manages the facility. Potential tenants have several options of membership to get access to the facility. In this concept, the community and the access to affordable working spaces play an important role.

This concept enriches the walkable city's idea because its tenants can often live in close distance. Furthermore, this enterprise can be easily developed within existing buildings as it can adapt to a variety of different floorplans.

A mixed-use approach seems to be prone to evoke conflicts between city dwellers. Some experts might raise the question: do we even want an urban city? If we ask city dwellers, the answer is yes, but do they really understand the implications? A discrepancy between imagination and reality is observable. Nevertheless, there are measures to soften upcoming conflicts. These can be e.g., citizen participation and careful planning.[2]

Justified by the substantial increase of population, every economic activity contributing to the city's added-value and job creation has to be supported. Despite the achievements in technology, not every economic activity will be sustainable for being integrated into a mixed-use city district; more precisely, a counter-productive mix has to be avoided by all means.[3]

An approach would be to arrange the mercantile structures linearly, preferably at high frequented streets; or to concentrate them, to form a commercial area within the quarter. Both strategies have the advantage of keeping the remaining part of the development area free for the allocation of social housing.

1 Municipal Department 18, "Produktive Stadt," 43.

2 Forlati et al., Mischung: Possible!, 51.

3 Municipal Department 18, "Produktive Stadt," 29.

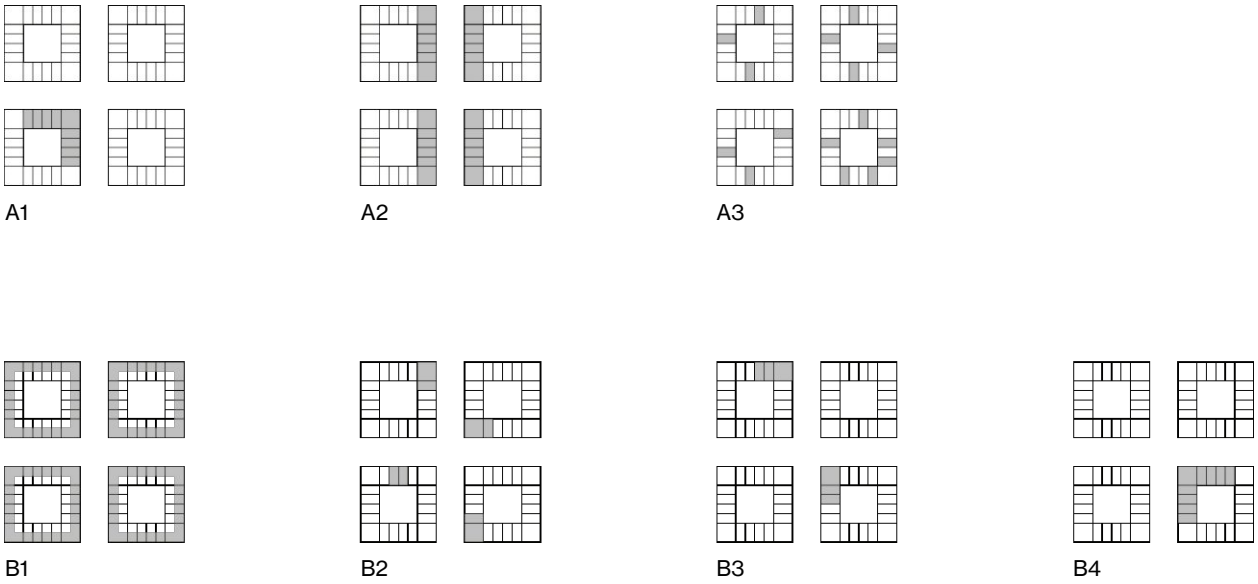


Figure 11: Possible spatial principles by which non-residential uses are distributed.

Another concept would be to spread the business activities among the city quarter.[1]

A decent mix of uses can be implemented on various levels: starting with the most extensive scope, the quarter district itself, followed by the block structure, the floor level, and the single residential unit.[2]

A) An arrangement of non-residential uses in mixed-use city quarters could be:

1.) The non-residential uses are clustered within an area of their own.

2.) The non-residential uses are arranged linearly.

3.) The non-residential uses are evenly scattered among the quarter district.

B) The scope of distribution of non-residential uses in mixed-use city quarters could be:

1.) The construction site:
Each construction site covers a certain percentage of non-residential uses wanted for the city quarter.

2.) The construction field:
The non-residential uses are concentrated on a subpart of a construction field, therefore excluding the remaining developments in this field from the implementation of a mixed-use approach.

3.) The sub-quarter:
The non-residential uses are concentrated on a subpart of the city quarter, therefore excluding the

1 Forlati et al., Mischung: Possible!, 28.

2 Forlati et al., Mischung: Possible!, 42.

remaining developments in the sub-quarter from the implementation of a mixed-use approach.

4.) The quarter:

The non-residential uses are concentrated like a focal point within a city quarter, therefore excluding the remaining developments in the quarter from the implementation of a mixed-use approach.

It seems, the most radical approach to a mixed-use urban design would be the distribution of non-residential uses at the scope of all construction sites within the constructions fields. This would mean that for the whole quarter, no development area is excluded from the implementation of a mixed-use approach.

At the other end of the scale, all non-residential uses would be concentrated on a single construction site or construction field, leading to the effect that for the rest of the developments there is no need for the implementation of a mixed-use approach. If this part of the quarter is within a walkable distance to the residential developments, a walkable city's goal can still be reached.

C) The arrangement of non-residential uses inside structures could be:

1.) as layers:

Each floor is monofunctional.

2.) as plates:

Non-residential uses constitute the base on which residential units are placed on top. The roof-surface of the base could also be used as a recreational area for the residents.

3.) as sprinkles:

Non-residential units are scattered all over a structure.

4.) as function-neutral structures:

The load-bearing structure of a building is designed to support non-residential and residential uses.[1]

1 Forlati et al., Mischung: Possible!, 97-100.

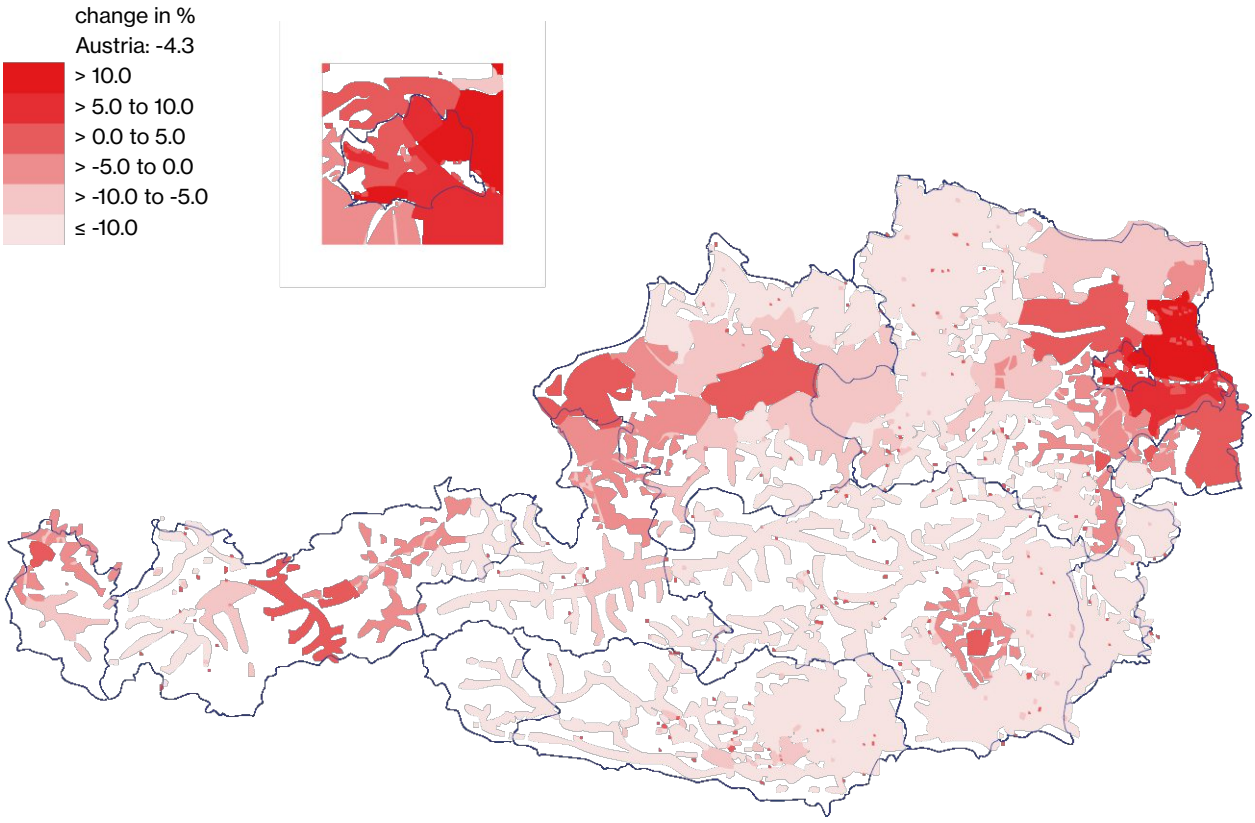


Figure 12: Population development in the age group 20 to 64 years from 2021 to 2050.

The size range of commercial units is an essential factor when planning affordable spaces. It starts at over 5000 m² for hypermarkets and ends at below 20 m² for individual enterprises. Larger units have the advantage of being easier manageable in the long term due to tenants with a more reliable financial stability. Small units, on the other hand, further the personal relations with the inhabitants of the residential structures and thus support the emergence of a mixed-use quarter.[1]

The necessities of the city dwellers' daily lives are within walkable distance if they are obtainable within a range of 300 to 500 meters.[2]

The city of short distances relieves the infrastructure from personal motorized transportation and public transportation and enables a more efficient usage of these two.[3]

An economic activity that manufactures material assets has to be associated with the productive sector according to the *ÖNACE 2008*, the

1 Forlati et al., Mischung: Possible!, 28.

2 Forlati et al., Mischung: Possible!, 27.

3 Forlati et al., Mischung: Possible!, 22.

classification system for economic activities.

The functional concept of the productive city embedded in the Viennese urban development plan *Step 2025* includes an extended portfolio of economic activities in the productive sector.

Automotive trade and repair, cleaning, construction industries, energy providers, environmental technology, maintenance services, passenger and freight transport, postal and courier services, rental of tools and vehicles guarding, repair, storage, waste disposal and recycling, and wholesale trade are to be included as well.[1]

By integrating the industry into the city, the delivery distances as well as the delivery times are shortened. Furthermore, the proximity between power generators and consumers allows for a more efficient use of the supply grid and an easier implementation of sustainable smart grid strategies, like power supply decentralization or energy-sharing.

By interlacing functions in dense city districts, expertise and knowhow in various fields is accumulated, supporting an open knowledge transfer environment, that helps enterprises to act innovatively and sustainably.

Because of the climate change, environmental regulations are expected to be more restrictive in the foreseeable future, leading to rising prices in the transportation sector. This provides industries, which are close to their business partners with a competitive advantage.

Those industries can also more easily implement the circular environment concept, which is especially important when dealing with resource scarcity.[2]

Apart from the fields of business that are present in the city structure, it is also important to look at the jobs the city dwellers are occupied with, to determine how flexible a city structure can and has to be.

According to the Statistik Austria, in Vienna in 2012, most people were employed in the following fields:

1.) knowledge and technology-based segments such as electronics and IT equipment (approx. 10,000 employees - with leading companies such as Siemens, Kapsch, TTTech, or IBM)

2.) food and fodder production (approx. 7,500 employees - e.g., traditional Viennese companies such as Manner or Ankerbrot, but also Felber, Wiesbauer, Ottakringer, and others)

3.) pharmaceutical sector (approx. 4000 employees - e.g., Octapharma,

1 Municipal Department 18, "Produktive Stadt," 23.

2 Municipal Department 18, "Produktive Stadt," 24.

Boehringer Ingelheim)
4.) vehicle manufacturing (e.g., General Motors, MAN, Bombardier)
5.) mechanical engineering (e.g., Hoerbiger, Schiebel Antriebstechnik, Merten)

In contrast to the other federal states of Austria and to other major European cities, Vienna has a high employment rate in the above-average paid knowledge and technology-based segments. Except for the food industry, the low-tech industry is of minor importance in Vienna (e.g., leather, textiles, and clothing).[1]

Vienna's employee structure has therefore to be considered ideal for the implementation of further measures of the current urban structuring – at the end of which the transformation into a walkable city shall be accomplished.

The productive city does not only consist of areas of a mix-used structure and city dwellers who have the ability and affinity to live in such structures. They also have to be able to work there.

Thus, there is a definite demand for different types of production places:

1.) Monolithic sites with a larger parcelling. This is especially of interest for large international companies.
2.) Clusters, i.e. small spatial structured business parks that can accommodate highly specialized companies and embedded co-working spaces that are also serving as reserves for job growth.
3.) Hubs and depots, providing the logistic space for the progressively growing e-commerce companies – alongside other businesses of smaller or larger sizes, either positioned in the outskirts or in the city center(s). Such companies usually have storage spaces with low spatial requirements.
4.) Infills, i.e. small scale structures integrated in mixed-use city quarters. They are particularly of interest for innovative enterprises that want to profit from the synergies with their surrounding context.[2]

1 Municipal Department 18, "Produktive Stadt," 45.

2 Municipal Department 18, "Produktive Stadt," 52-53.

Concerning the location of these different types of production places within the city, there are three different types of operational zones to be distinguished:

1.) Industrial-commercial areas:

This type of operating zone usually has a surface area over five hectares and is exclusively reserved for economic activities. These are predominately industries of the primary or secondary sector (*ÖANCE 2008*) which are not readily suitable for spatial interlacing with enterprises of the tertiary sector and certainly not with housing.

The reasons for this are the pollutant and noise emissions, the traffic volume, or the spatial extent of the buildings. Retail should not be established in these precious areas because of the continuing trend of vanishing sales areas in the cities caused by the growing e-commerce sector. The only exception should be made for the local supply providing necessities for workers and employees.

Furthermore, unrestricted operational use should be ensured at all day times, small parcellations should be offered, and the land price level should be affordable. Concerning accessibility, a connection to the highway system is needed, and the availability of public transport is essential.[1]

2.) Commercial mixed areas:

This type of operating zone is a traditional commercially used area, usually located within the center of the city. It is well accessible by individual as well as by public transportation. Its central location is desirable for densification and for mixing with other uses such as housing. There are multistorey high compact structures to accommodate the full-range of different uses, which are generating synergies among each other – imaginably, every kind of spatially compact commercial uses such as crafts, on-demand manufacturing and such. Not more than 50 percent of the above-ground structure is used for residential uses, in which common spaces (e.g., circulation, garbage room, bicycle parking) are already included. A local supply for its inhabitants has to be integrated.[2]

3.) Integrated individual sites:

This type of operation zone is often positioned within residential areas. They are highly connective with their surroundings, implying advantages for employees, customers, and suppliers.[3]

1 Municipal Department 18, "Produktive Stadt," 66.

2 Municipal Department 18, "Produktive Stadt," 76.

3 Municipal Department 18, "Produktive Stadt," 87.

Architectural solutions are often categorized as hardware and the city's way of governance as the software in urban design. Since there are many architectural solutions available, the most output and the best result can be expected by improving the planning processes.[1]

For initializing a mixed-use and creating a sustainable neighborhood, giving the commercial mix an overarching identity is beneficial. Simultaneously a specialization in the offer within the city quarter can be pursued, which gives the neighborhood a relevance to the region. Examples include branding the district as a creative, healthcare, or innovation quarter.[2]

The wohnhausfonds_wien has building sites for state-subsidized housing construction available. A so-called Bauträgerwettbewerb (project developer competition) ensures sustainable use of these sites and grants state subsidies for its winners.

However, for building sites of private owners, subsidies are possible, too. For this, the Grundstückbeirat (property advisory board) has to prove that the projects will be developed on certain standards in four categories, namely ecological and economic aspects as well as architectural, and social sustainability.

In the year 2011, Vienna has started a smart city process, in the course of which a guideline was formulated, the so-called *Smart City Wien Framework Strategy*. Its purpose is to ensure the sustainable transformation of the city in the medium and long run. In contrast to other cities' smart city strategies, Vienna is also taking CO2 emissions and social aspects into account.[3]

In 2010, the Klima- und Energiefonds has started the Smart-Cities-Initiatives. The focus had been primarily on technical solutions, but this has changed since the beginning. Now it is more on developing solutions that enable cities and their dwellers to change their lifestyles to support a sustainable future. Today, holistic strategies are preferred that work on improvements in energy balance and the dwellers' satisfaction in urban areas. Key elements are affordable housing, the promotion of mixed uses, and giving city dwellers the chance to participate in project developments at an early stage.[4]

1 Forlati et al., Mischung: Possible!, 15.

2 Forlati et al., Mischung: Possible!, 81-96.

3 Forlati et al., Mischung: Possible!, 17.

4 Forlati et al., Mischung: Possible!, II.

Figure 13: Building 20, 18 Vassar Street in Cambridge, Boston

4. The Adaptable Building

During World War II, the MIT erected a structure housing a laboratory in only six months. Rumor has it that the MIT graduate Don Whiston designed it within a single afternoon. Researchers used the space to conduct extensive research on radar technology to support the war effort. The volume was designed in the shape of five fingers and had only three floors. The roof was equipped with a variety of strange-looking devices. Since the building was constructed of wood, it violated the city's fire regulations. However, by claiming it to be only temporary, the authority eventually issued a building permit. Yet in fact, it was demolished not before 1998. This building met the needs of the MIT researchers for so long, just because of its adaptability.

Its structure was the strongest on the MIT campus. It was able to support up to 756 kg/m². The researchers were able to implement additional wiring by themselves, as the cable trays suspending from the ceiling were easily accessible. Due to the building's improvised aesthetics, no one felt obligated to ask for permission when an experiment required minor alterations to the layout.[1]

Former users described it as follows: "Windows that open and shut at the will of the owner!" (Martha Ditmeyer) "The ability to personalize your space and shape it to various purposes. If you don't like a wall, just stick your elbow through it." (Jonathan Allan) "If you want to bore a hole in the floor to get a little extra vertical space, you do it. You don't ask. It's the best experimental building ever built." (Albert Hill) "One never needs to worry about injuring the architectural or artistic value of the environment."(Morris Halle) "We feel our space is really ours. We designed it, we run it. The building is full of small microenvironments, each of which is different and each a creative space. Thus the building has a lot of personality. Also

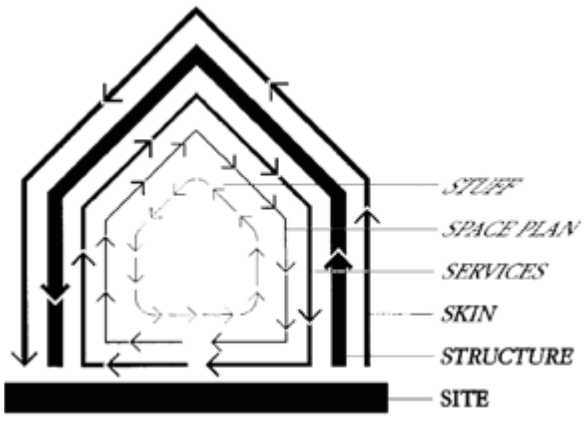


Figure 14: Stewart Brands Shearing Layers

1 Stewart Brand, How Buildings Learn: What Happens After They're Built (New York: Penguin Books, 1995), 3/4-6/14, https://www.kobo.com/at/de/ebook/how-buildings-learn.

it's nice to be in a building that has such prestige." (Heather Lechtman)[1]

Such an adaptable building design facilitates the change of short-lived layers without affecting the structure of the long-lived layers.[2] Stewart Brand extends and revises the four major layers of a building (shell, services, scenery, and set), originally defined by Frank Duffy as the four S. Brand now differentiates between six layers of a building (site, structure, skin, services, space plan, and stuff). Each layer has a different life expectancy and needs replacement at a different rate. Brand describes this as "shearing layers of change". And he adds: "Because of the different rates of change of its components, a building is always tearing itself apart."[3] Brand argues that the only eternal layer is the site. The structure has a life expectancy of 30 to 300 years but usually is demolished after 60 years. The reasons for this are usually adjustment difficulties that occur with changing demands.

Inhabitants will not alter the structure of a building because it is expensive and unsafe. The skin has a life expectancy of 20 years. It needs replacement due to a change in technical requirements, material fatigue, or just because it is out of fashion. Contemporary buildings require a variety of internal systems, such as different kinds of wiring, plumbing, sprinkler systems, HVAC (heating, ventilating, and air conditioning), elevators, and escalators, just to name a few. The technical components of these systems need replacement every 7 to 15 years. If this is not feasible, a structure might be demolished. The space plan is very inconsistent regarding its operational lifetime – it differs between 3 and 30 years. A store's layout might need an update every three years, while residential floor plans might last for over 30 years. The shortest-lived layer is the stuff. Things like chairs, desks, kitchen utensils, lamps, and so on are used daily and are soon out of fashion[4]

1 Stewart Brand, How Buildings Learn, 3/6/14.

2 "Manifesto Open Building," OpenBuilding.co, accessed May 10, 2022, https://www.openbuilding.co/.

3 Stewart Brand, How Buildings Learn, 2/3/19.

4 Stewart Brand, How Buildings Learn, 2/1-3/19.

One example that illustrates the separation of layers particularly well, is a remarkable contribution made to the IBA Hamburg. This building has been erected as a solitaire in the IBA Hamburg area, precisely Wilhelmsburg. Investment costs for investors and residents had been reduced by the contribution of the later inhabitants during the construction process. The construction was accomplished in two phases. First, a reinforced on-site concrete structure of five floors was erected. Storage rooms, staircases, and balcony breastwork included. Then, settlers built their individual apartments with Ytong elements themselves, while the balcony railing served as a fall protection. Although the project was a success, some aspects had to be reconsidered. The outer shell was too complex to be assembled by the settlers themselves. With the project of such kind it would be better to add the outer shell to the basic construction and leave only the assembling of non-isolated walls to the settlers.[1]

1 Andrea Benze, Julia Gill, and Saskia Hebert, Serieller Wohnungsbau: Standardisierung der Vielfalt (Berlin: Senatsverwaltung für Stadtentwicklung und Umwelt, 2013), 74-78, https://www.stadtentwicklung.berlin.de/staedtebau/baukultur/iba/download/studien/IBA-Studie_Serieller_Wohnungsbau.pdf.

Figure 15: Settlers Self-Build Housing, IBA Hamburg 2013, BeL
Figure 16: Basic Construction, IBA Hamburg 2013, BeL

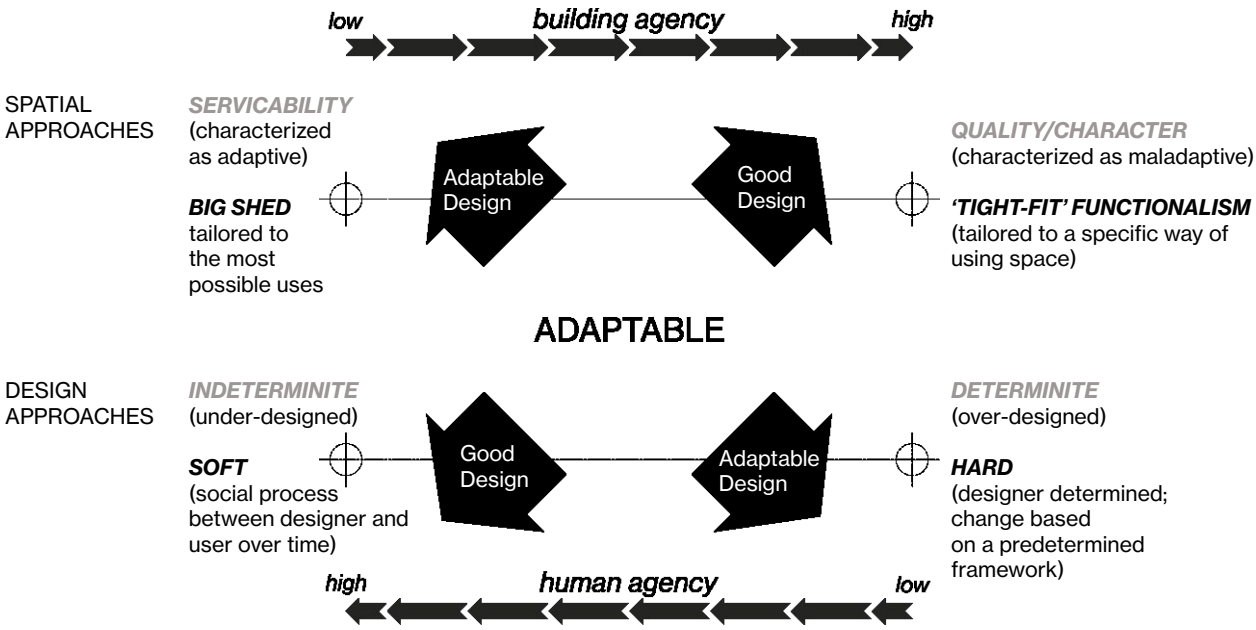


Figure 17: Different approaches towards adaptable design.

Many people consider the concept of adaptability as expensive. The construction industry and furniture stores advertise expensive, supposedly flexible products which are never used or are even functioning properly (e.g., responsive facades, moveable partitions, transformable furniture). Additionally, if a building proves itself to be highly adaptable over time, it is often regarded as a good design by accident.[1]

A later adaptation of a building's structure is only feasible if this is already taken into account in the design process. A key concept in doing so is the use of dry joints. Depending on the desired intervention in a buildings structure, it may just be sufficient to remove individual walls, but not each construction system supports this.

In elaborating this, Richard Roger-Bruno categorizes various demountable systems in: site-assembled-kits of parts, factory-made 3D modules, and hybrids. To simplify this list it is easier to differentiate just between:

1.) post and beam

1 Robert Schmidt III et al., "What Is The Meaning Of Adaptability In The Building Industry?" (conference paper, 16th International Conference on "Open and Sustainable Building": Proceedings of the international conference jointly organized by CIB W104 - Open Building Implementation and TECNALIA, Bilbao, May 2010), 236, https://www.irbnet.de/daten/iconda/CIB17955.pdf.

2.) slab and column

3.) panels

4.) space modules

The concept of post and beam supports flexibility in three axes. Walls and ceilings are infills with no structural relevance. A big challenge is a need for stiff structure joints.

Therefore the concept of slab and column might be a more obvious choice. By integrating a stiff core and rigid ceiling elements, structures can be braced more efficiently. Horizontal flexibility is kept unaffected to a great extent. Walls are just infills that can be removed or added at any time.

Yet, the use of load-bearing panels reduces flexibility in all three axes.[1] However, this might be more than adequate for residential structures and is advantageous in terms of soundproofing.[2]

The space module is very much in trend in the residential building sector. It is favorable regarding the on-site construction time, which is significant when building in busy city districts. While offering great flexibility during the design process, altering the layout without exchanging the modules once a structure is completed, might not be feasible.[3]

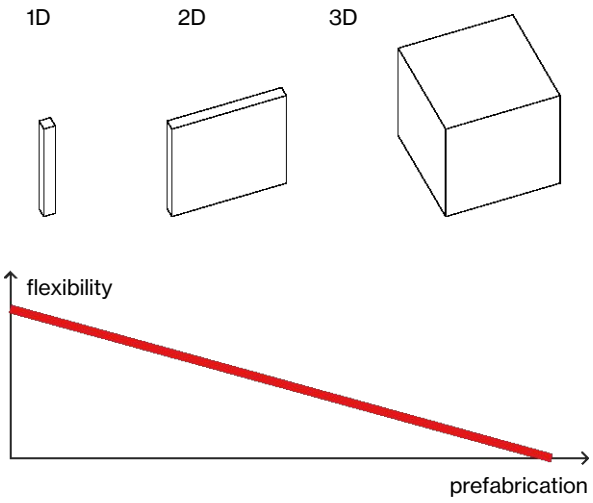


Figure 18: flexibility versus prefabrication

1 Richard Roger-Bruno, ""IFD" Systems = Open Building "Plus"" (conference paper, 16th International Conference on "Open and Sustainable Building": Proceedings of the international conference jointly organized by CIB W104 - Open Building Implementation and TECNALIA, Bilbao, May 2010), 257-259, https://www.irbnet.de/daten/iconda/CIB17955.pdf.

2 Hans Drexler et al., Holz: form- und kraftschlüssig: Entwicklung eines Voll-Holz-Bausystems mit form- und kraftschlüssigen geometrischen Verbindungen (Stuttgart: Fraunhofer IRB Verlag, 2020), 16, https://dgj.eu/dgj/wp-content/uploads/2020/03/res022_BBSR_HFKS_Abschlussbericht-LAYOUT-_2020.02.13_pr_5.4.pdf.

3 Roger-Bruno, "IFD" Systems = Open Building "Plus", 257-259.

Lacol Arquitectura Cooperativa uses the concept of load-bearing panels to design a communal housing project in Barcelona. It is located on the edge of Can Batlló, a former industrial zone. The layouts of the apartments are designed to be adaptable by leaving gaps between panels which can be used for doors if needed. If not needed, they can be shut using lightweight panels. In addition, folding screens are used to divide single rooms.

The structure is made of concrete and CLT (cross laminated timber). The staircase acts as a core and consists of 30 cm thick slabs that withstand the lateral forces. The timber used is pinus insignis, a local species growing in the Basque region. To reduce costs and CO_2 emissions, the inhabitants decided to share an extensive common infrastructure. The building has a courtyard covered by a greenhouse. In wintertime, this serves to capture solar energy. In the summer, it is opened and provides a refreshing breeze, utilizing the chimney effect.[1]

1 George Kafka. "Sustainable building, sustainable living: La Borda, Barcelona by Lacol," architectsjournal.co.uk, June 23, 2020, https://www.architectsjournal.co.uk/buildings/sustainable-building-sustainable-living-la-borda-barcelona-by-lacol.

Figure 19: La Borda social housing complex in Barcelona, Lacol arquitectura cooperativa, A
Figure 20: La Borda social housing complex in Barcelona, Lacol arquitectura cooperativa, B

5. Conclusion

It has become apparent that the advent of the automobile drastically expanded the daily radius of action of city dwellers. Therefore, they are willing to travel ever greater distances to run errands or to get to work. Centers have had to specialize in their offerings to provide a rich assortment of products and services that attract residents throughout the region. In order to realize the politically desired city of short distances, the centers have to diversify their offerings once again to meet the daily needs of residents within walking distance.

By implementing the concept of the productive city, quality jobs are provided closer to the homes of the workers and employees, and thereby further reversing the separation of functions proclaimed in the Athens Charter. Furthermore, close synergies are created between commercial and non-commercial uses, which ultimately helps to reduce CO_2 emissions and to increase productivity.

The sustainability and operational lifetime of a building depend largely on its flexibility. In other words, the floor plan of a building has to be adaptable to the changing needs of the tenants. This is more likely to be achieved when planners design buildings and spaces to be use-neutral. This means to dimension them spaciously and to choose a construction method that allows for changing floor plans afterwards. In any case, the implementation of separation layers according to Stewart Brand, that are providing the users with a certain freedom of design regarding the inner layers, is to be preferred over the use of expensive furnishing products such as portable furniture systems, room dividers, and such like.

The Vienna General Hospital is a cornerstone of primary medical care in Austria. The present project complements the offer of the district to create a center that is not only providing medical care but also a recreational area of regional and local importance. The center's offer is extended to create an urban space that meets the challenges of a warming climate. The flows of the individual motorized traffic, and those of the subway will be either moved underground or bridged. Thus, the site will also be accessible on the east-west axis, i.e., crosswise to the Währinger Gürtel.

The project contributes to the realization of the pedestrian city. The neutral-use structure allows the accommodation of a wide range of commercial uses, which, with the help of the surrounding area, creates a diverse offer that meets the daily needs of the residents in its vicinity. The establishment of an

inner-city logistics center will not only create high-quality jobs in the productive sector. Furthermore, it will also make it possible to supply the city's residents with everyday goods in a CO2-neutral manner.

The proposed structure is based on a flexible grid that subdivides itself according to a specially coded algorithm, creating cells with a wide-span width for commercial uses and cells with a narrow-span width for non-commercial uses. Cells with a wide-span width are attributed to commercial uses as their construction would be too expensive for affordable housing. Non-commercial uses are accommodated in cells with a narrow-span width to build affordably. These narrow cells are highly flexible by removing walls to form continuous spaces that also allow for commercial uses if needed, such as home offices, offices or start-ups, and studios.

6. The Project

Figure 21: Volkswagen AG financed the former CargoTram in Dresden.

Figure 22: The freight tramway in Amsterdam is loaded.

Figure 23: The Streetscooter model drives electric and can be configured by start-ups as desired.

Figure 24: The cargo vessel Zulu is fully powered by hydrogen and is set to operate on the Seine in Paris. A

Figure 25: B

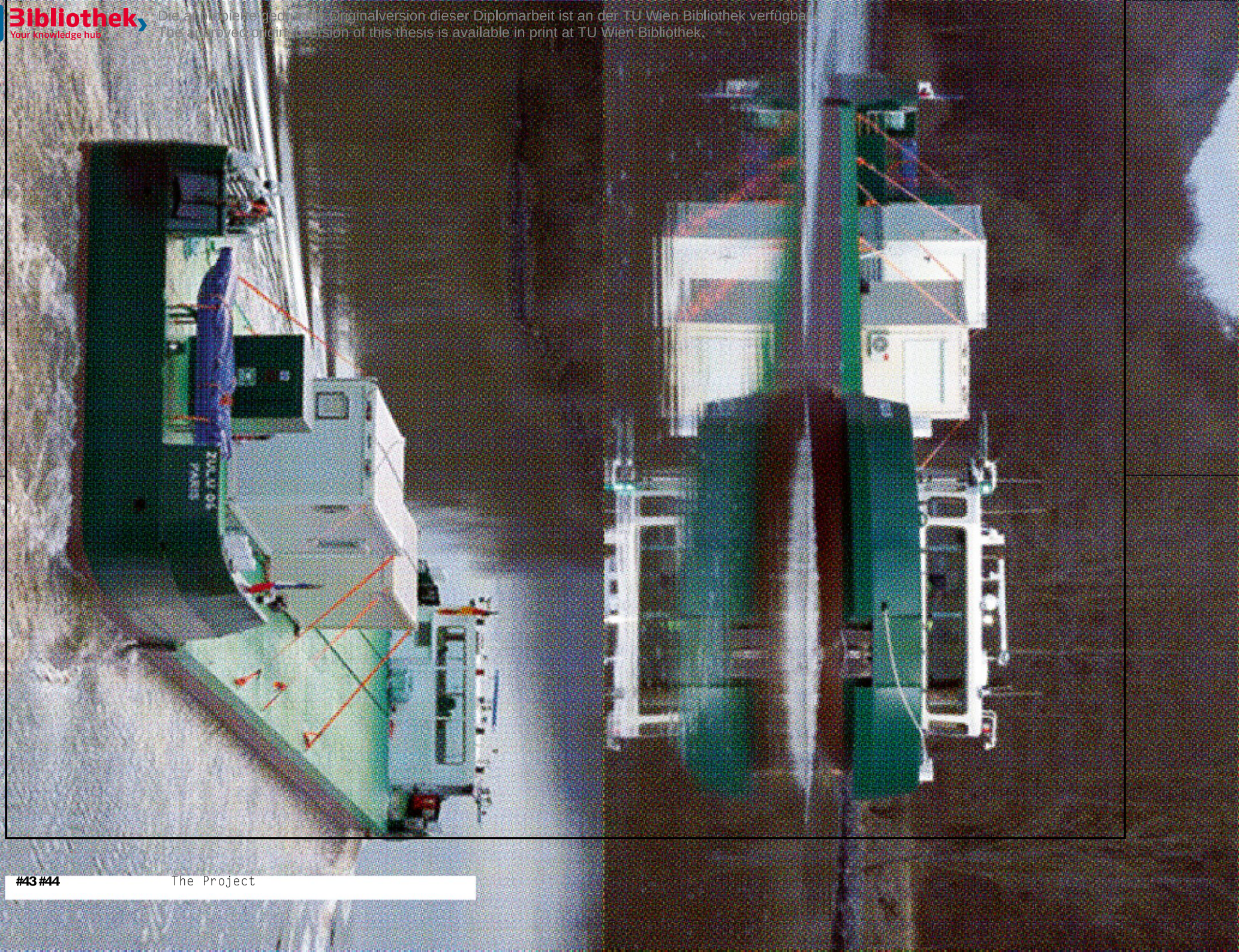


Figure 26: The EAV's cargo bike is used to operate on the last mile in the transport and delivery sector.

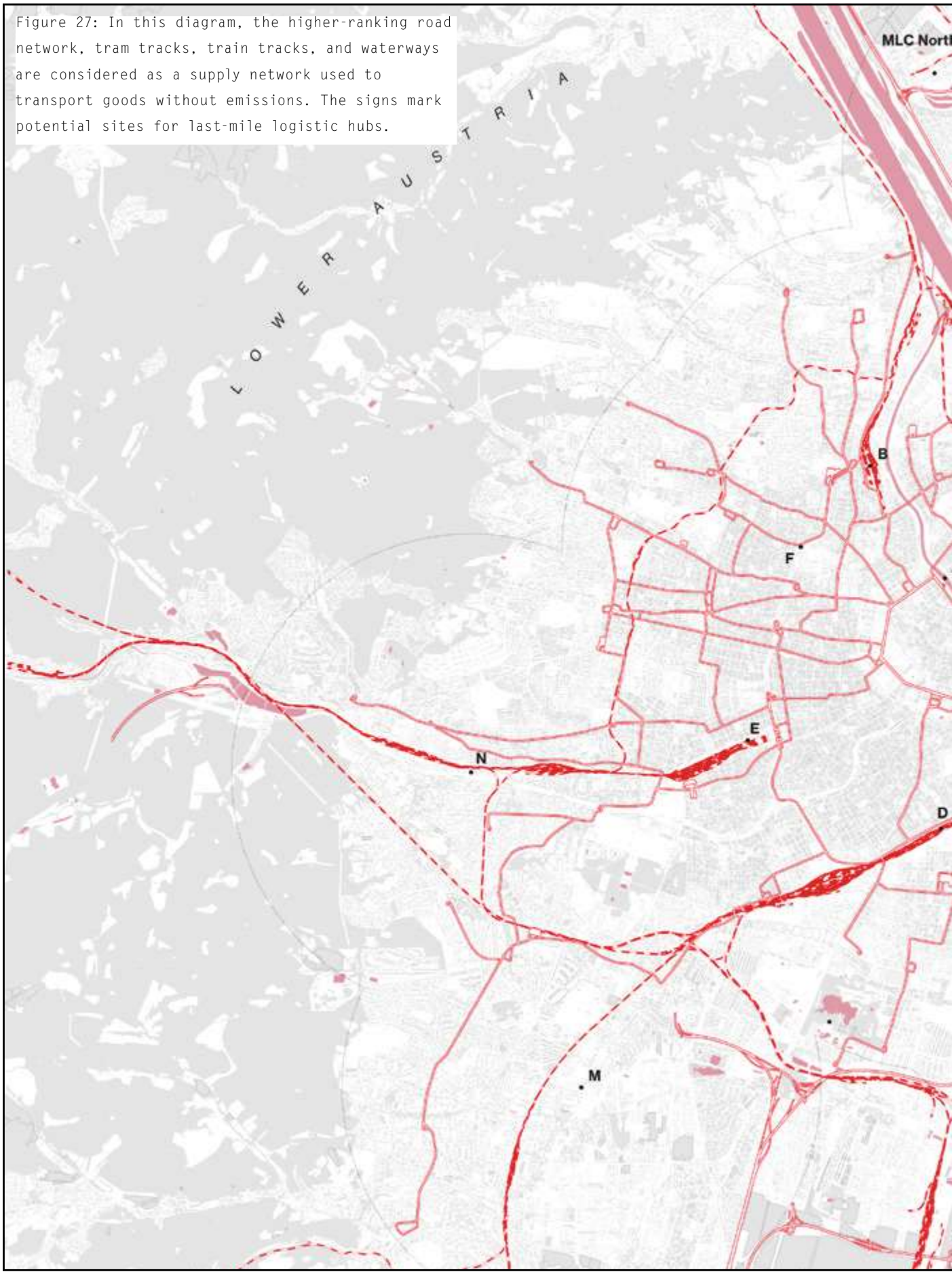


Figure 27: In this diagram, the higher-ranking road network, tram tracks, train tracks, and waterways are considered as a supply network used to transport goods without emissions. The signs mark potential sites for last-mile logistic hubs.

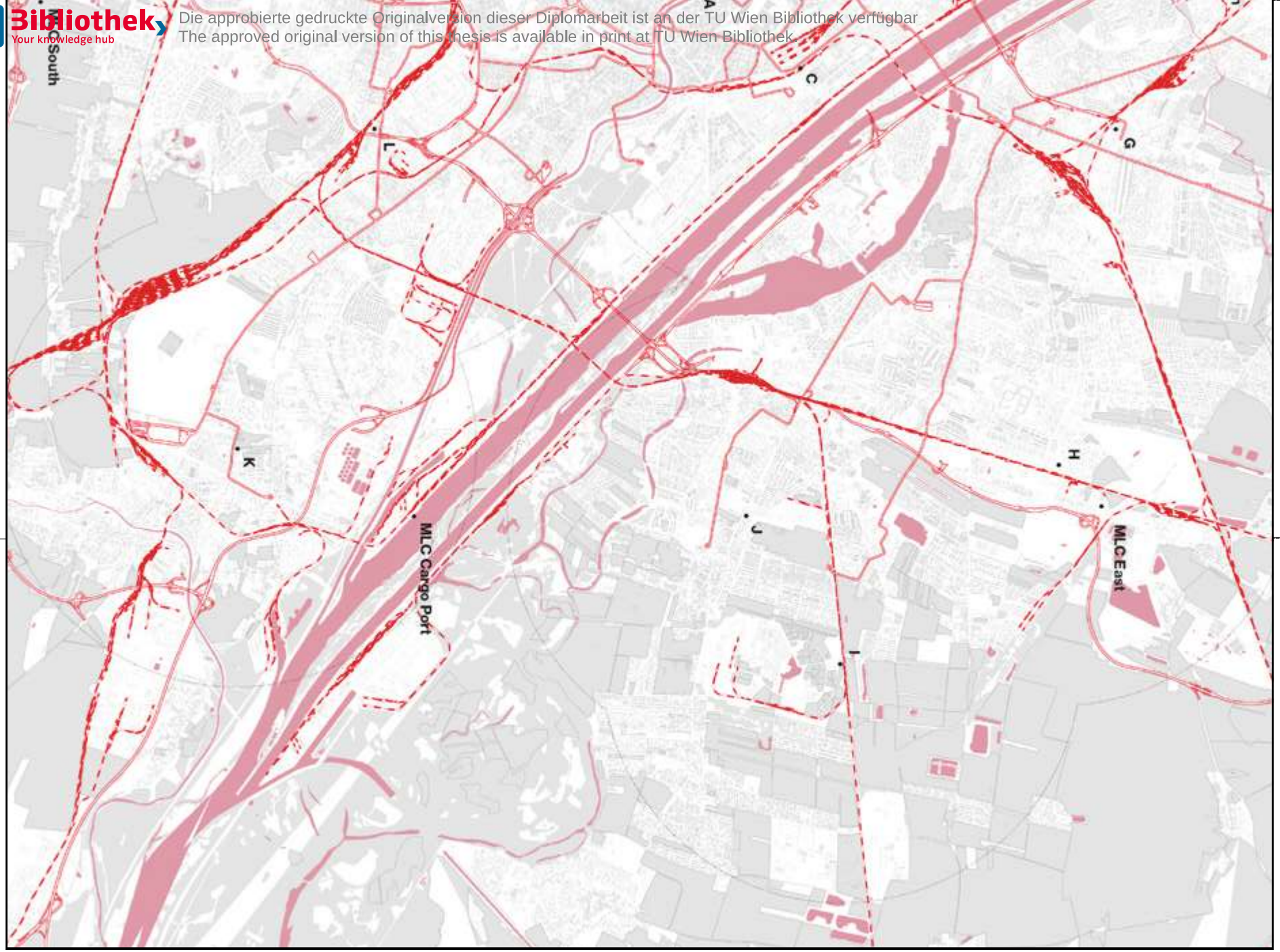
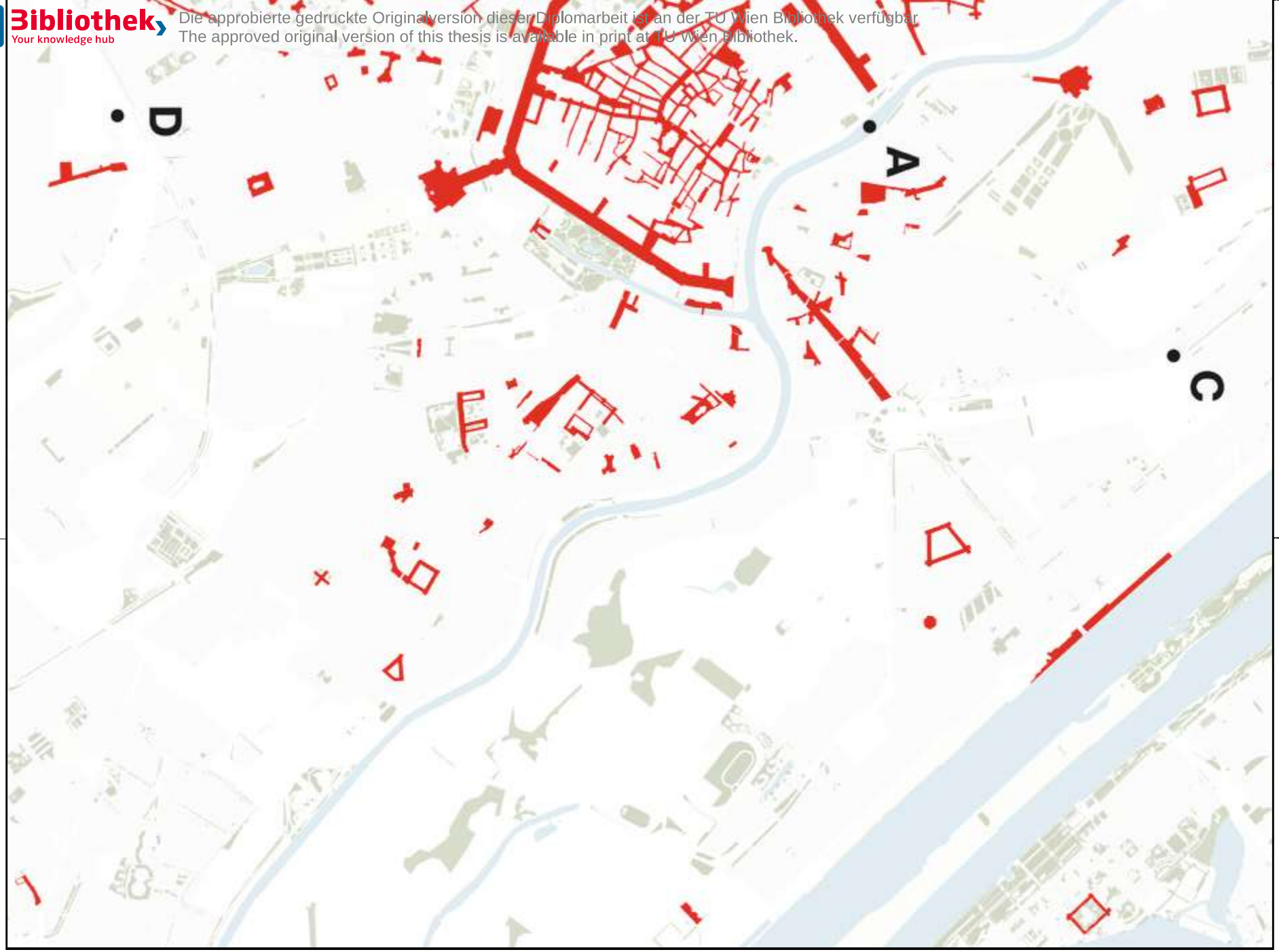


Figure 28: Distribution of public places in the urban fabric.



The Project

Initial site, scale 1:5000

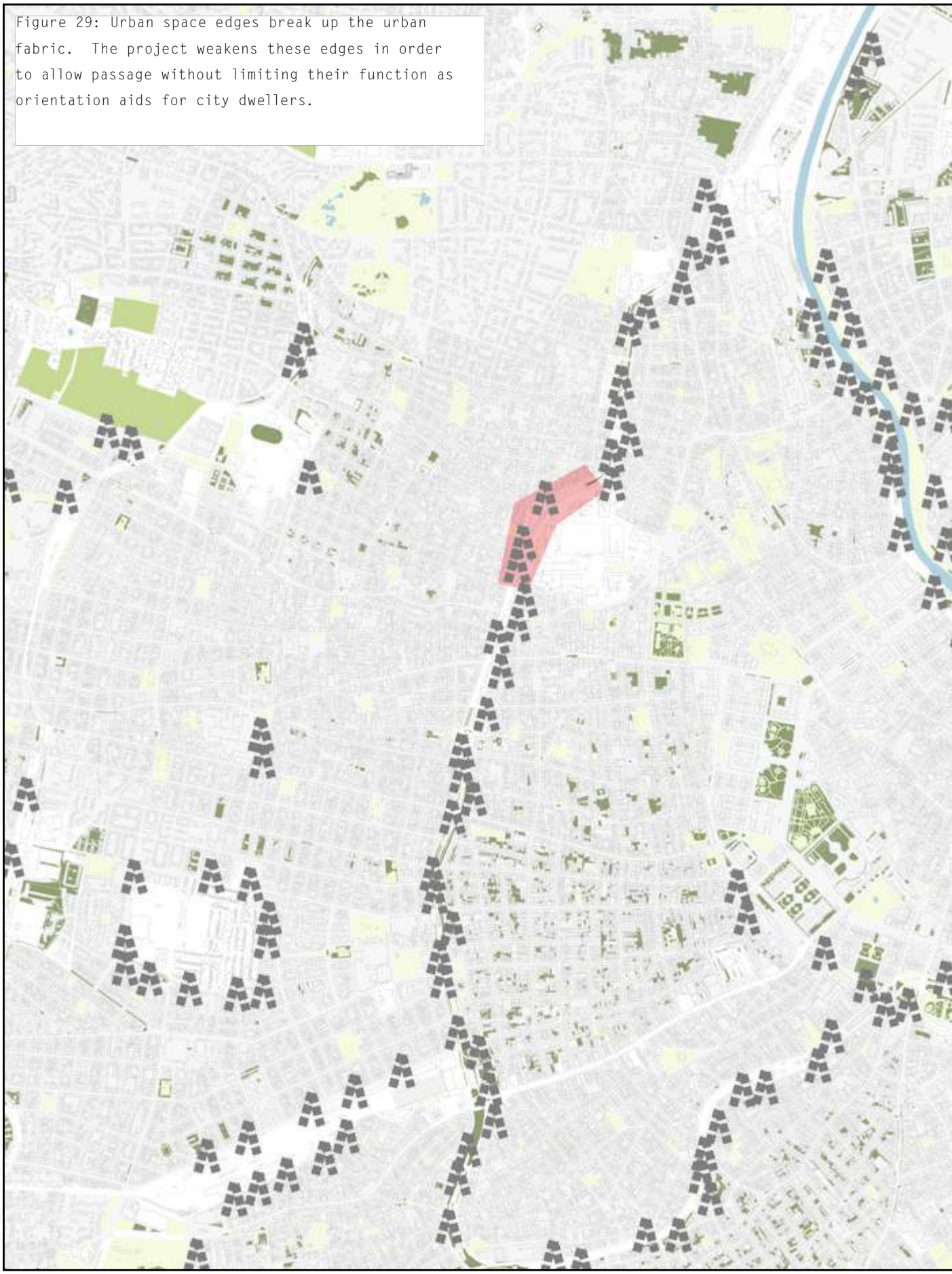
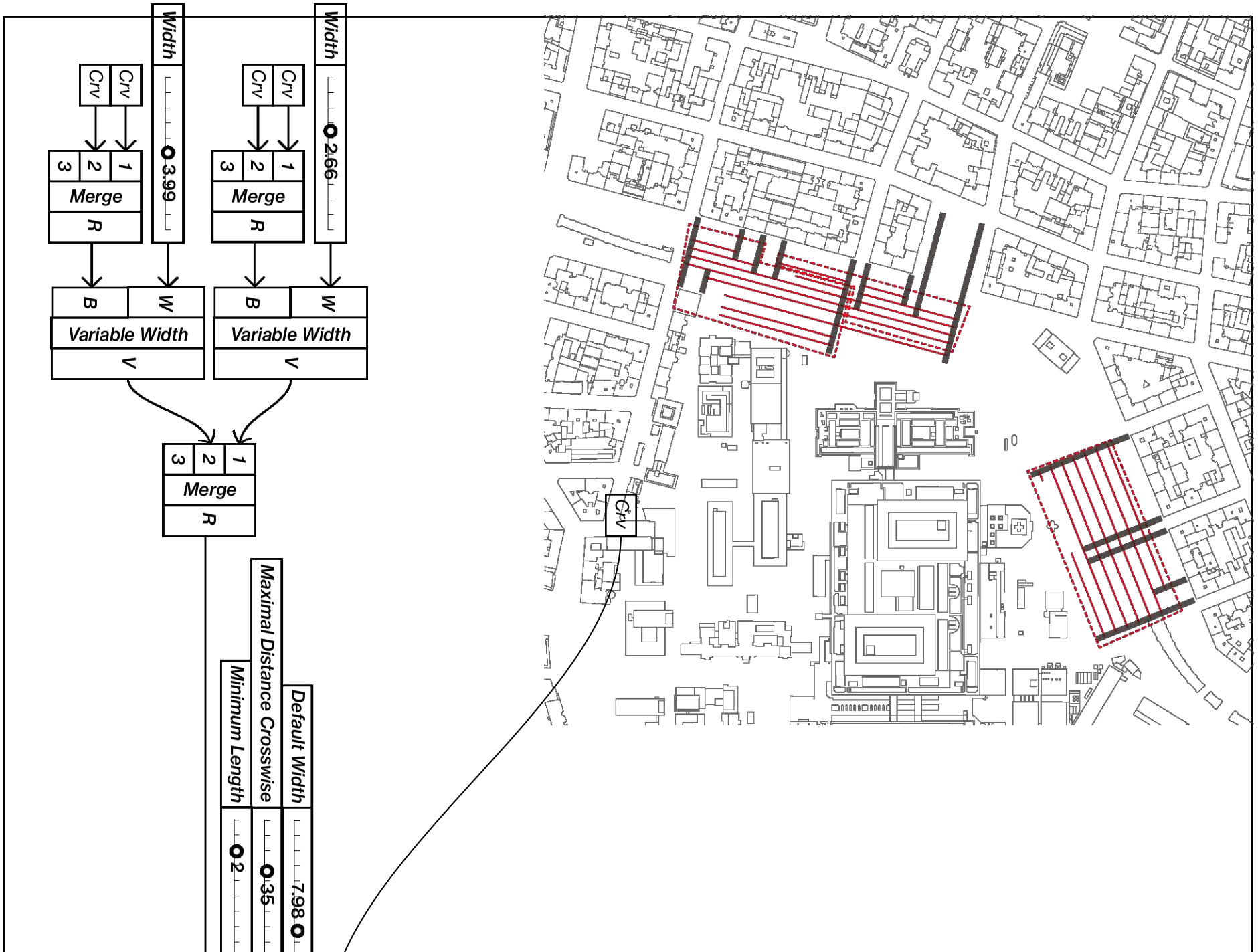
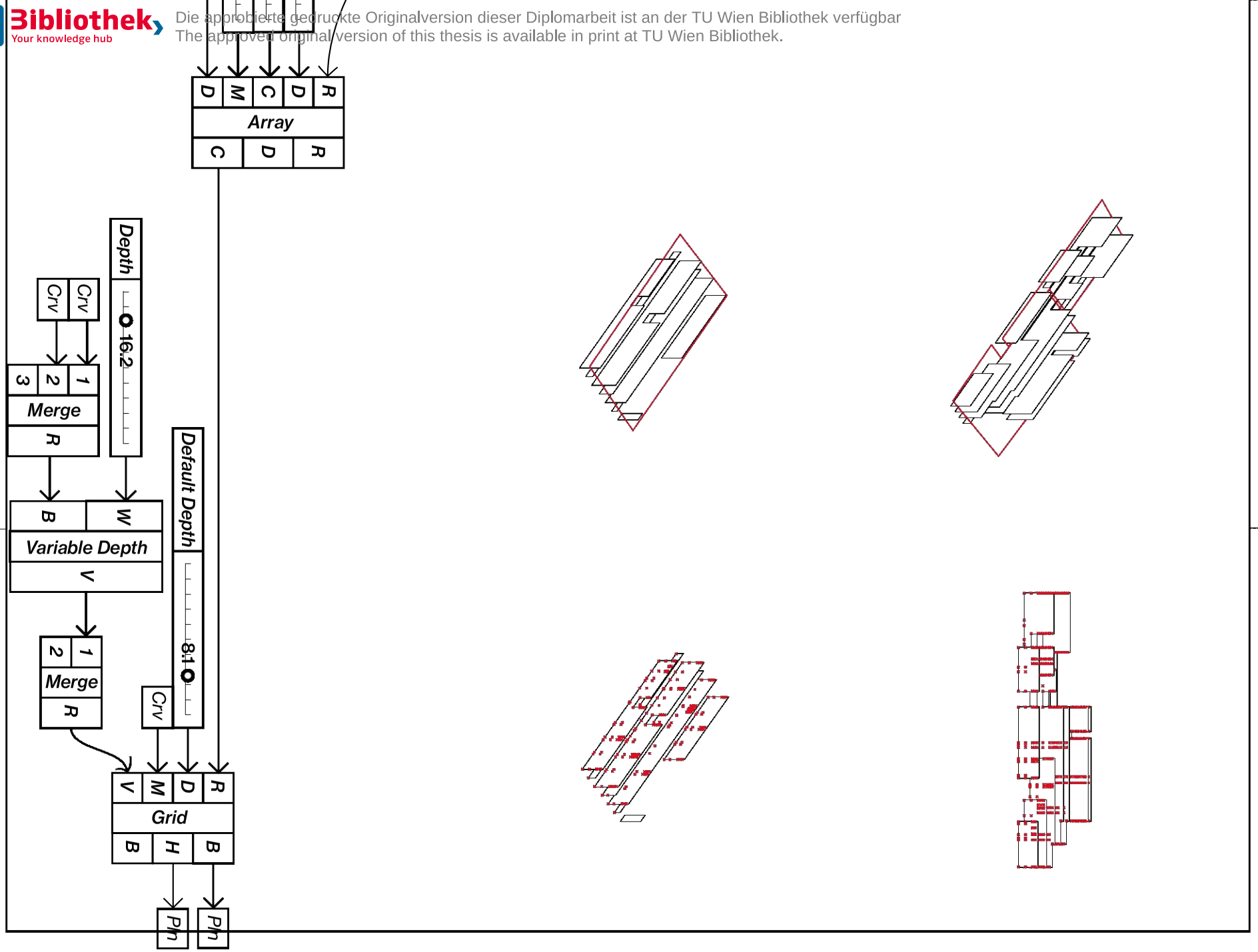
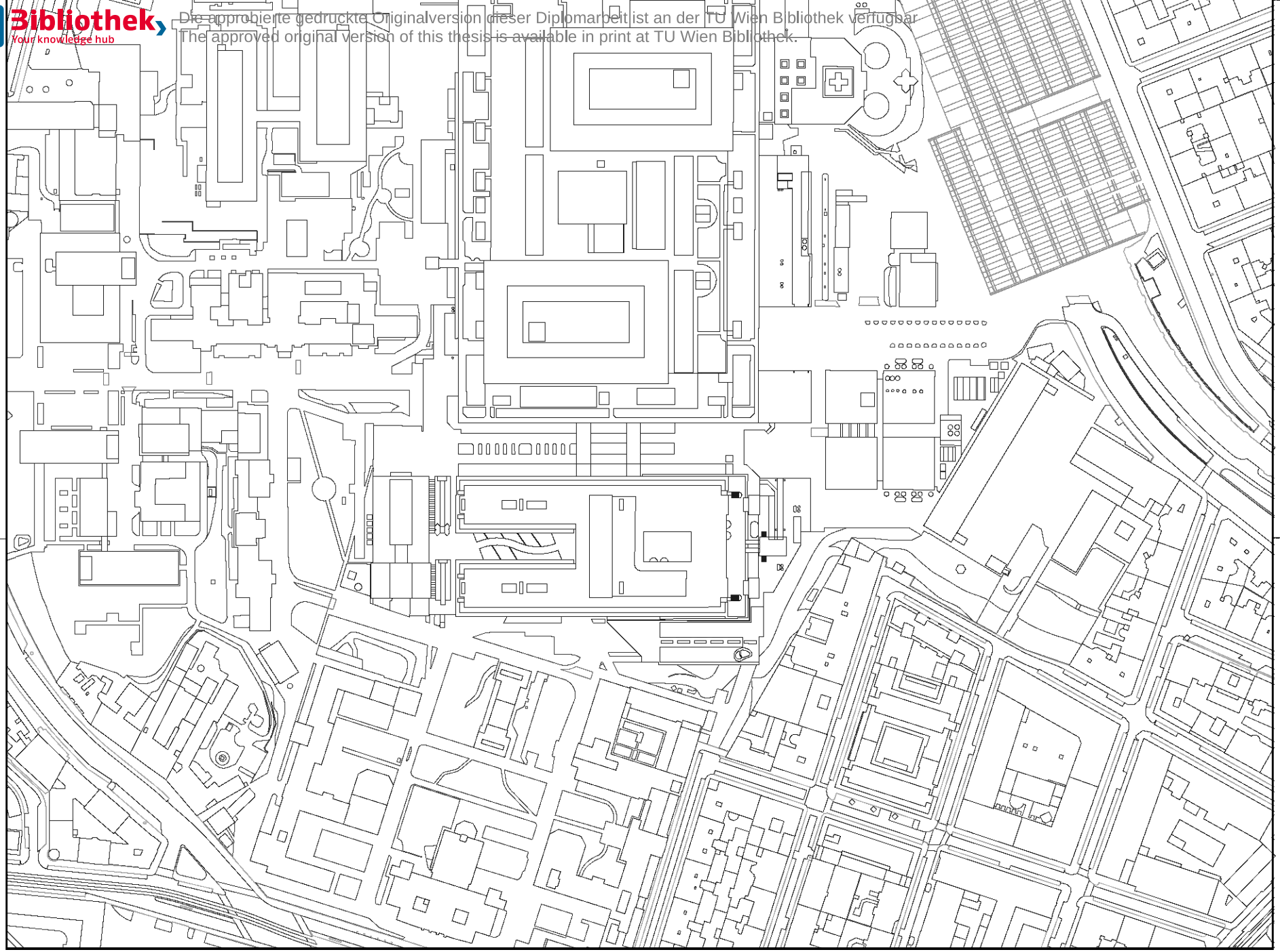


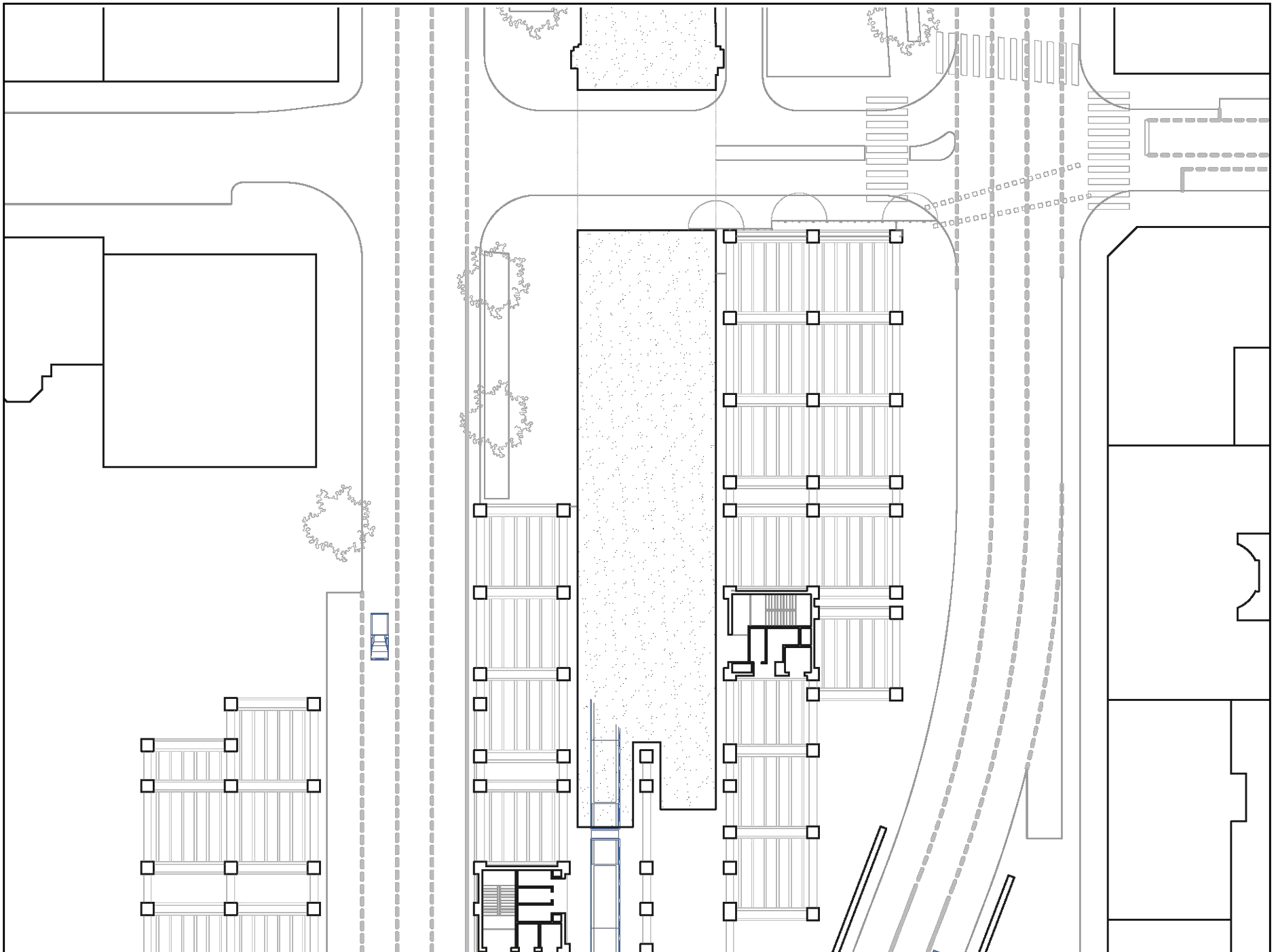
Figure 29: Urban space edges break up the urban fabric. The project weakens these edges in order to allow passage without limiting their function as orientation aids for city dwellers.

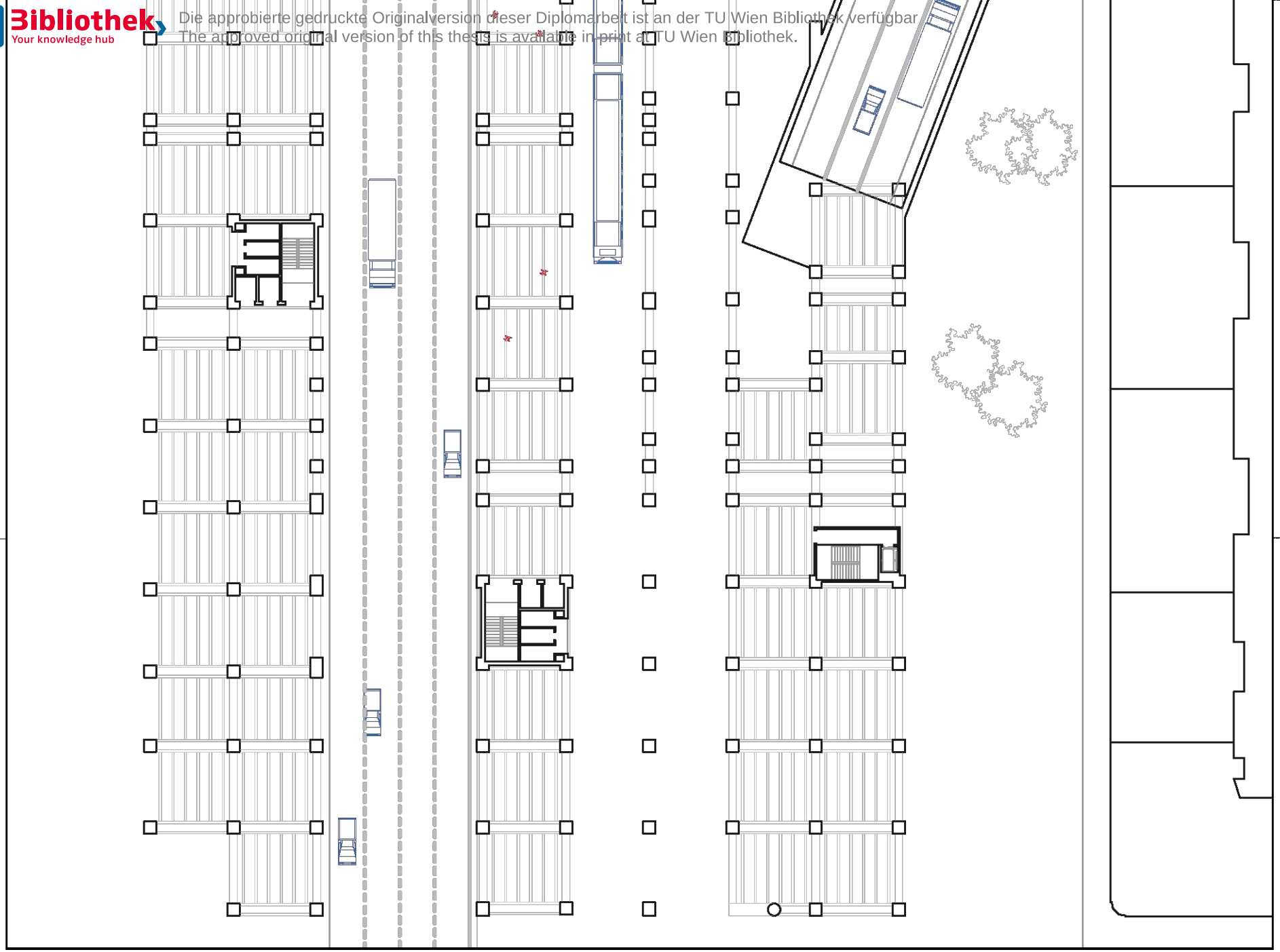


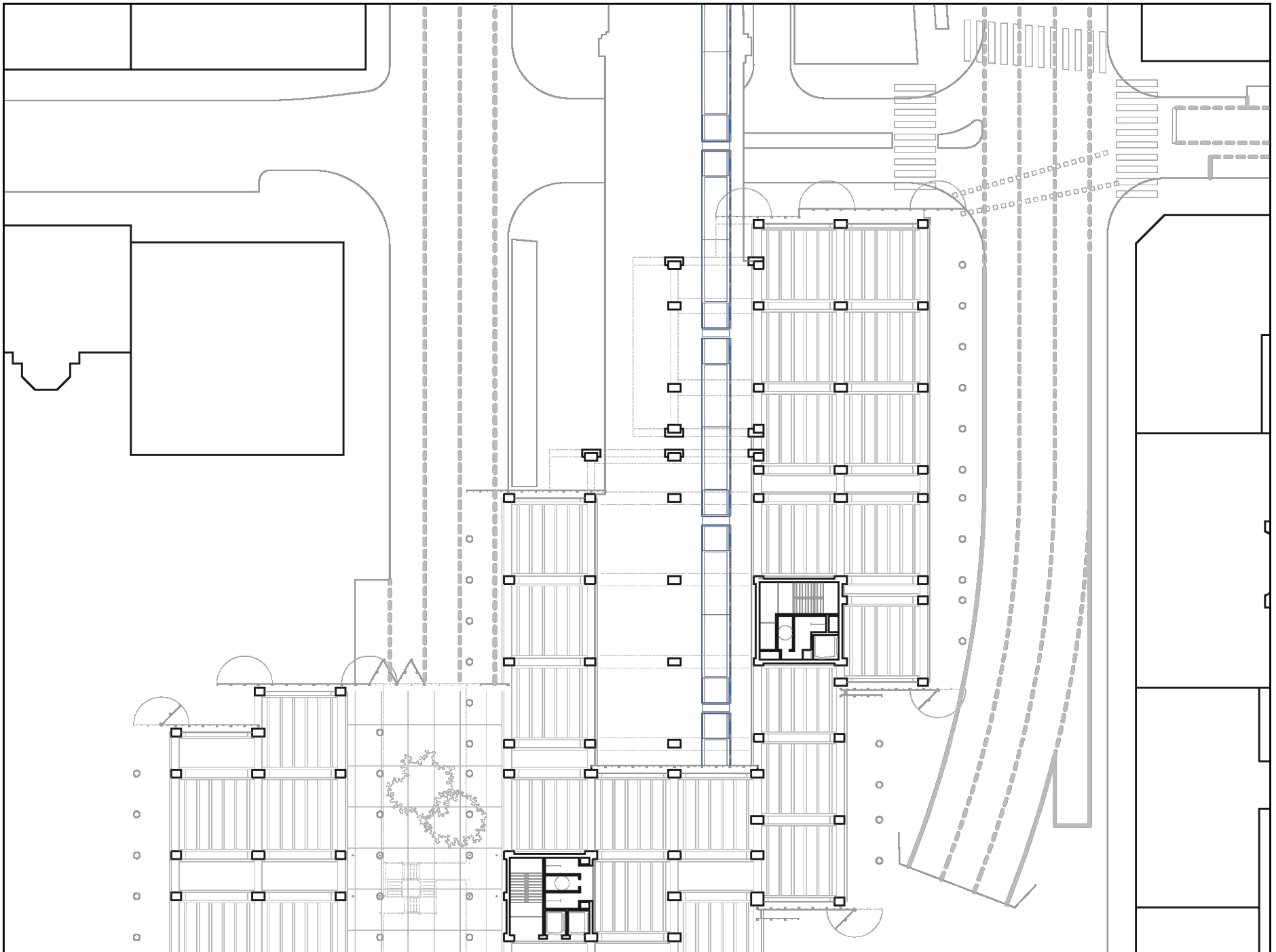


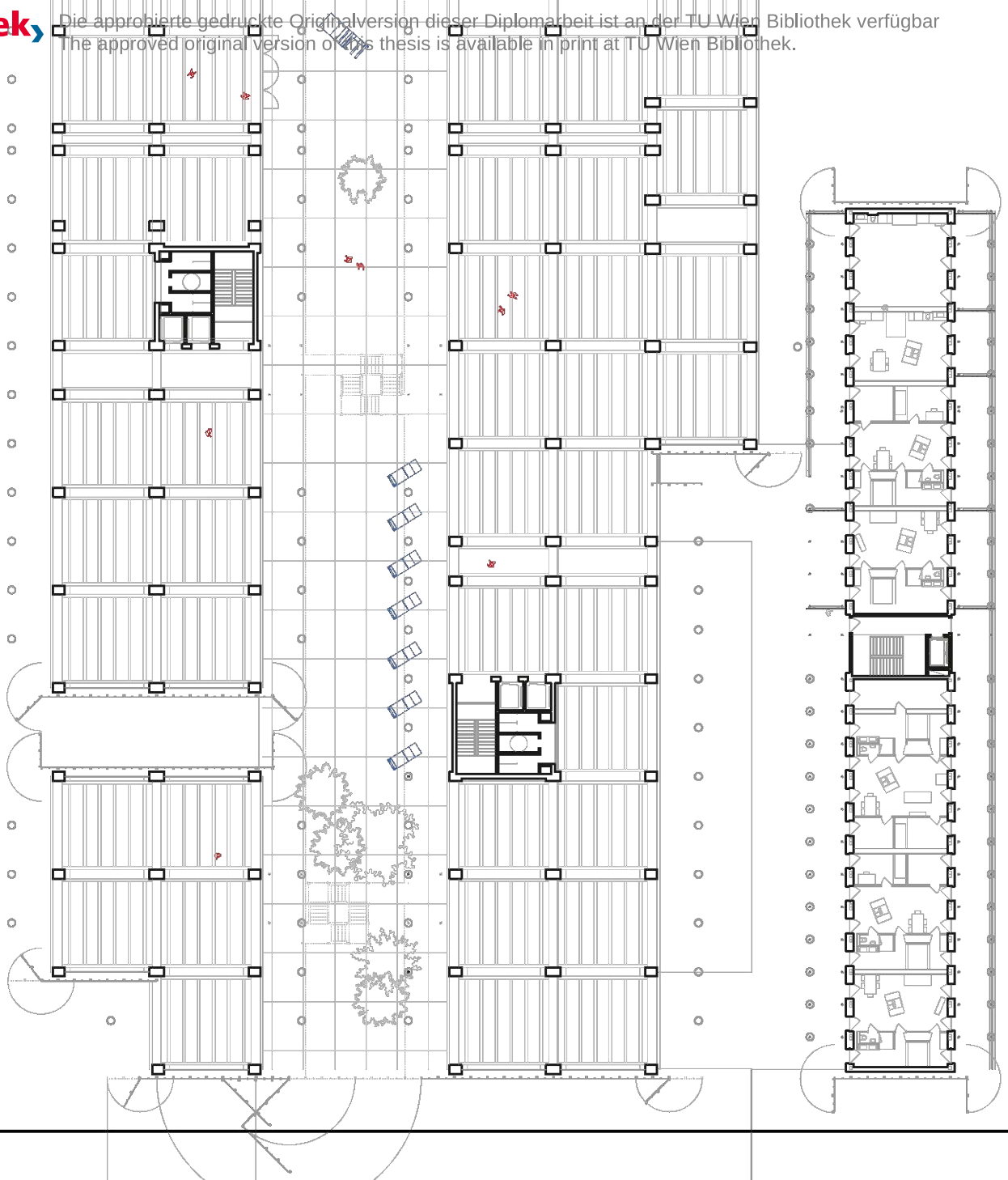


#57#58 The Project site plan, scale 1:2500

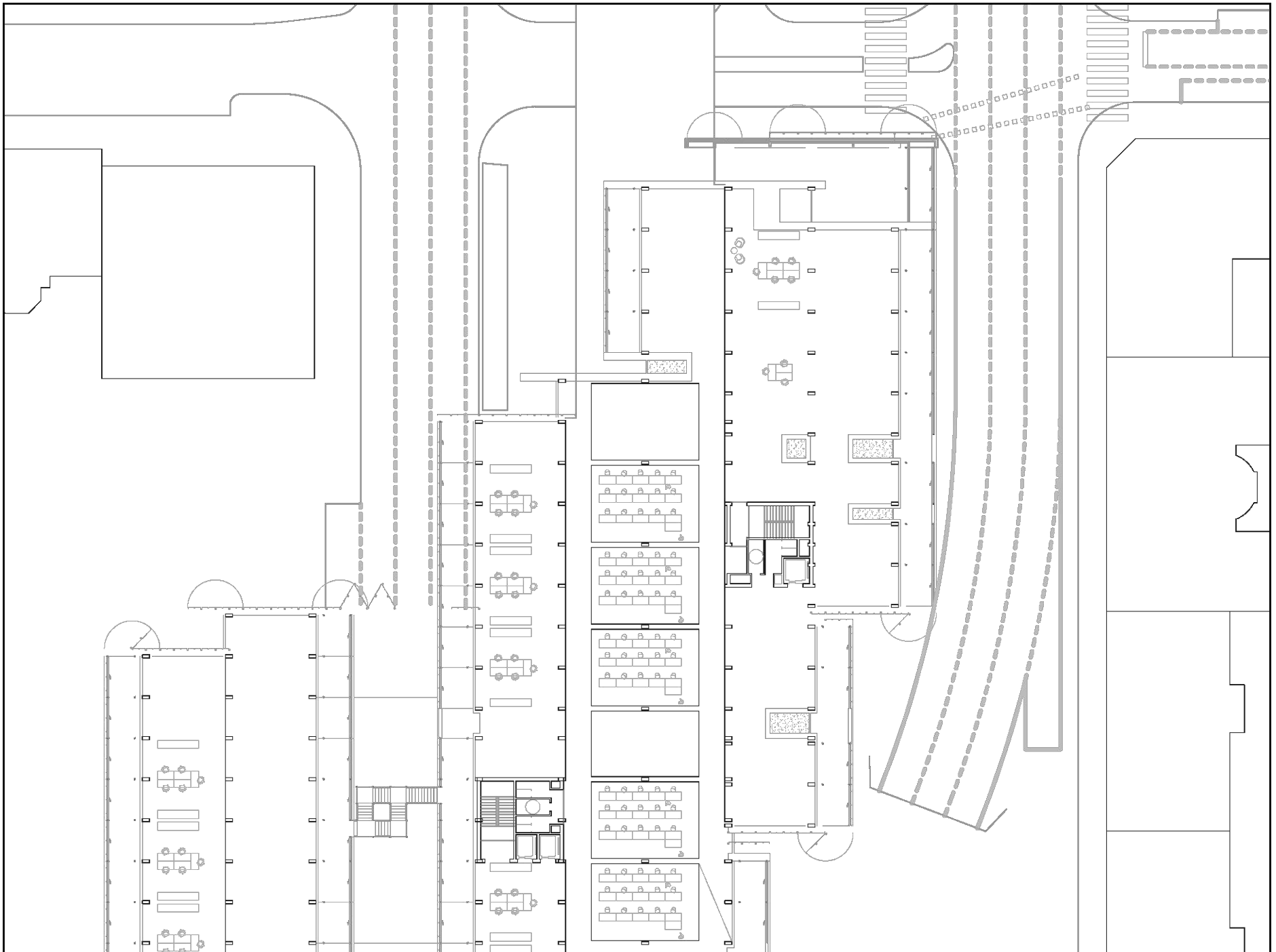


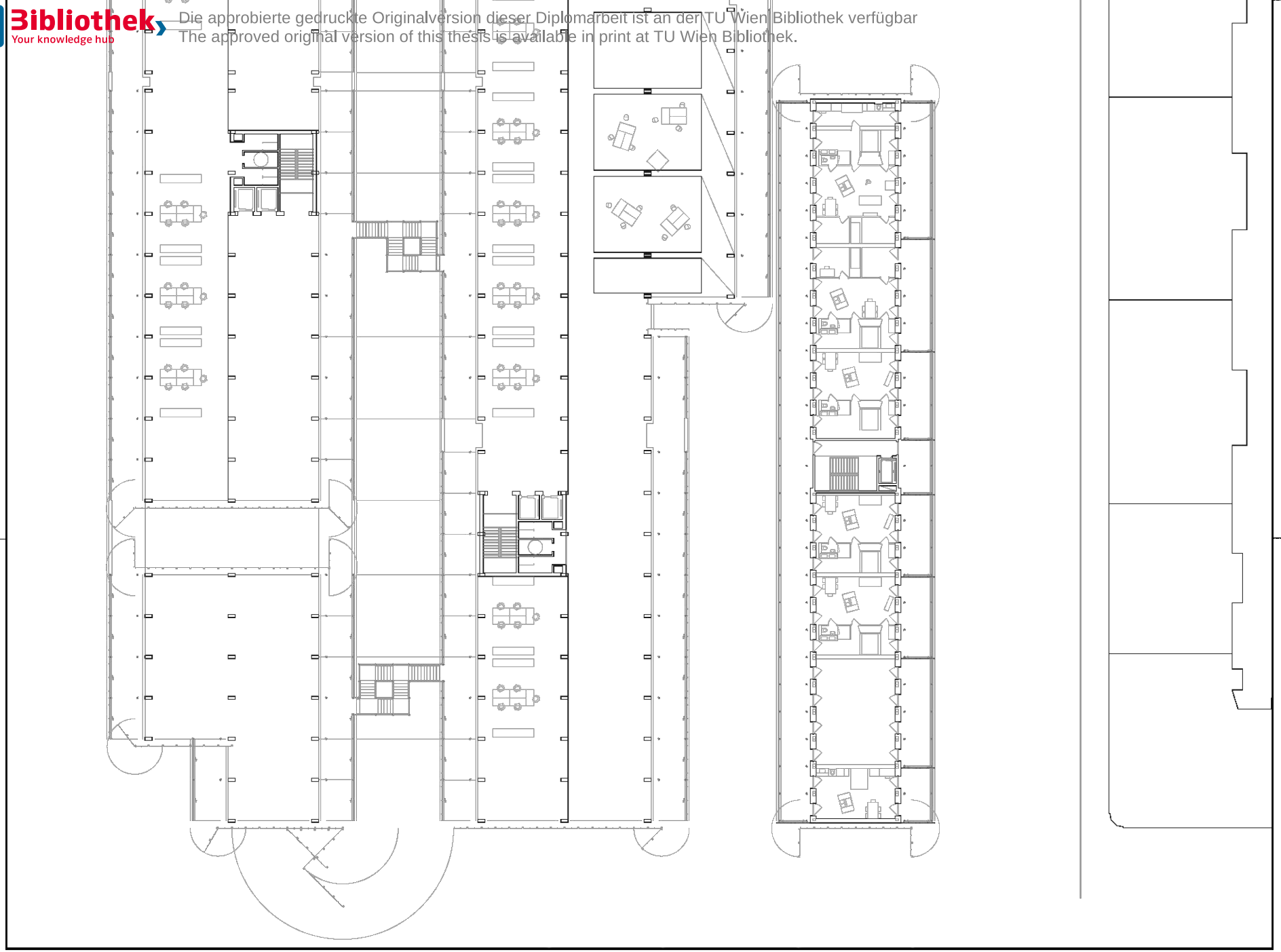




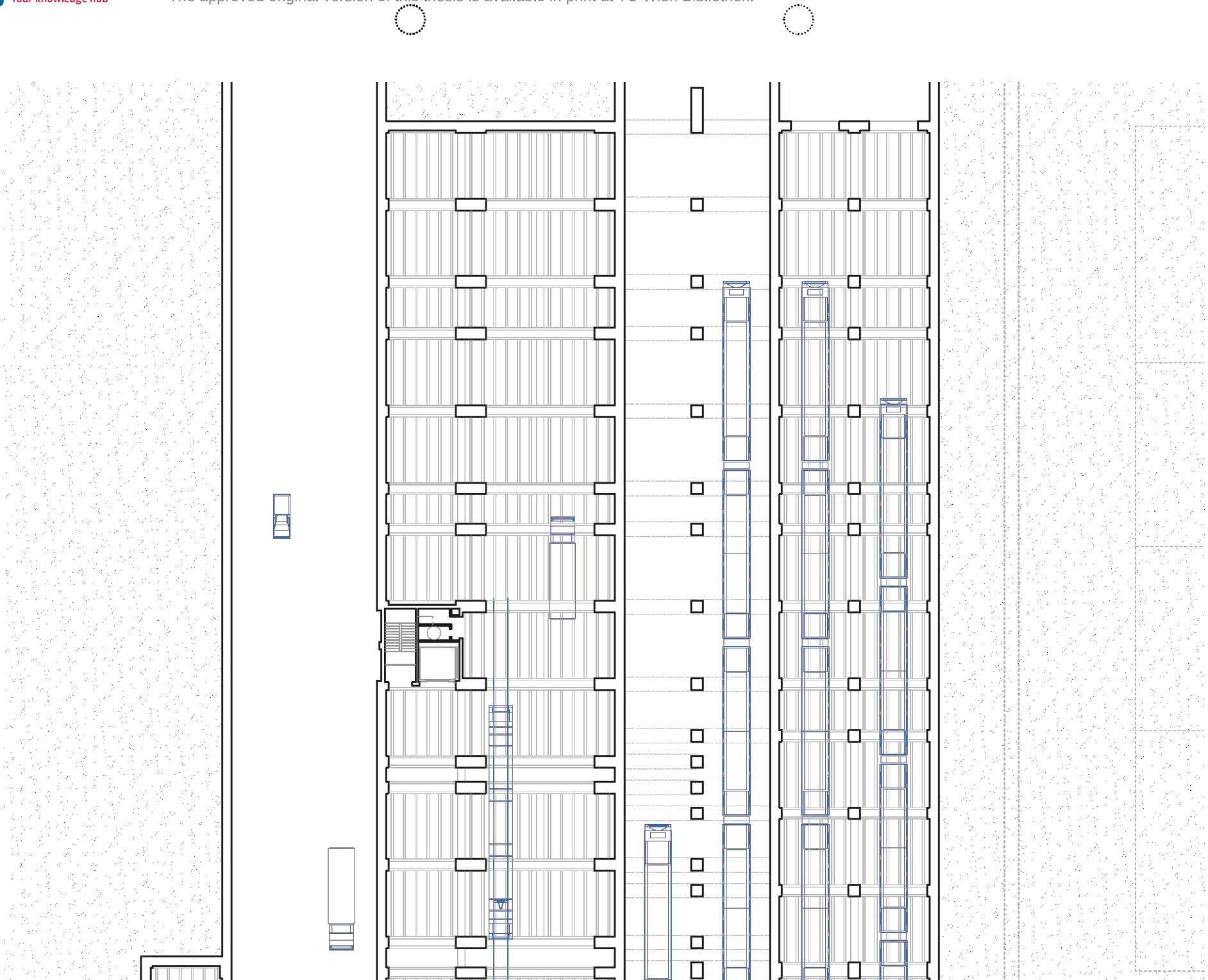


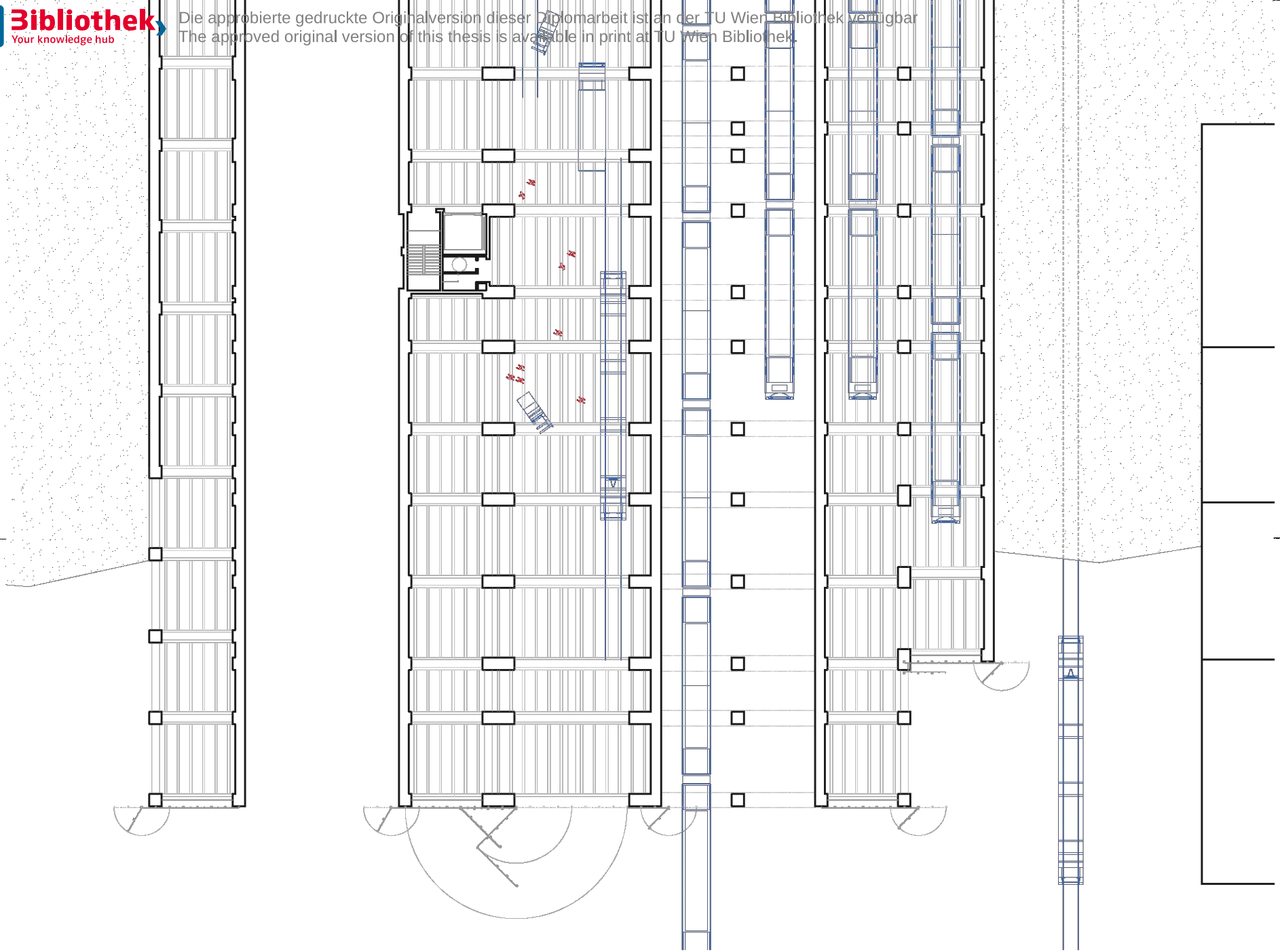
#61#62 The Project ground level, scale 1:500



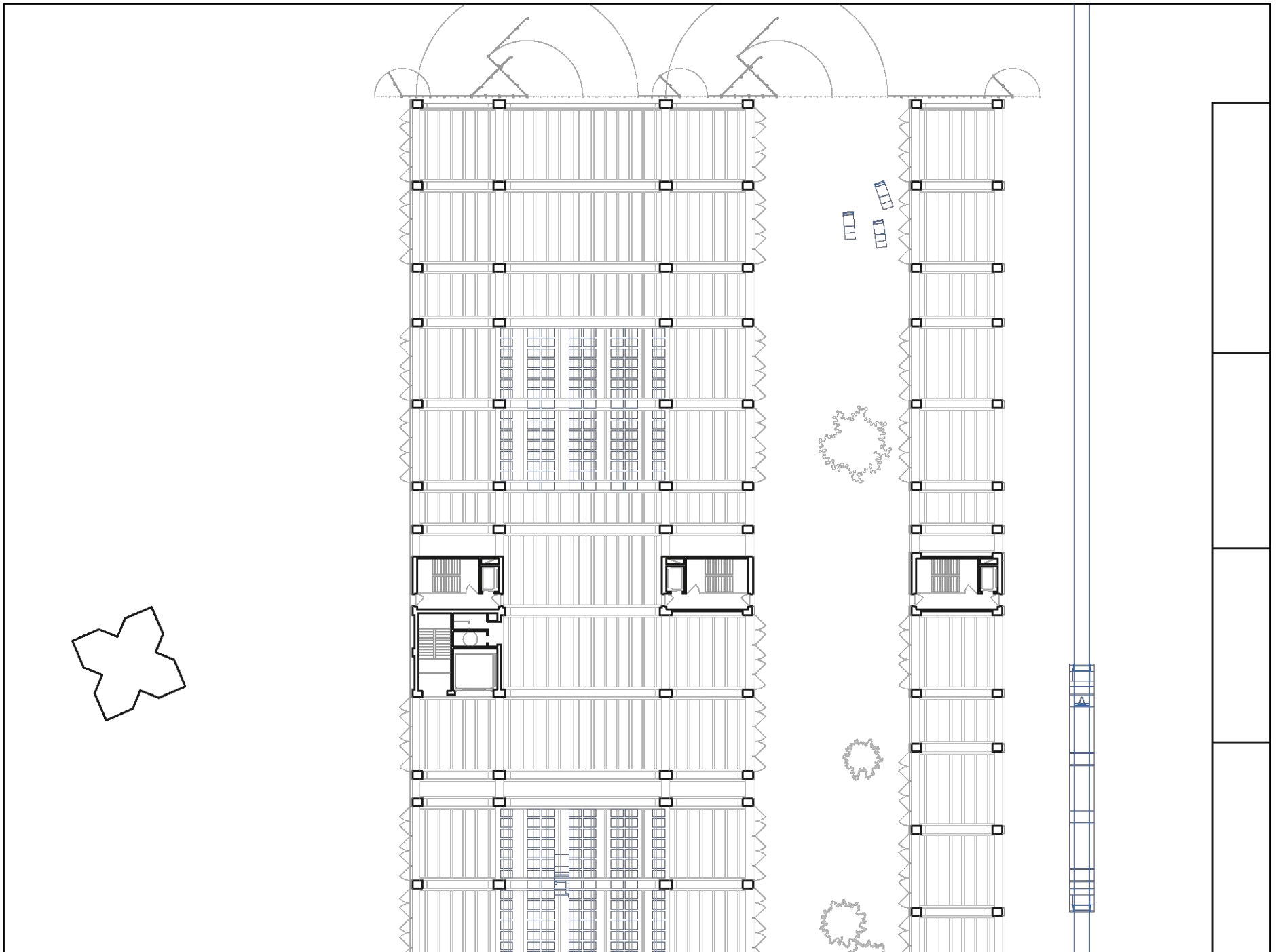


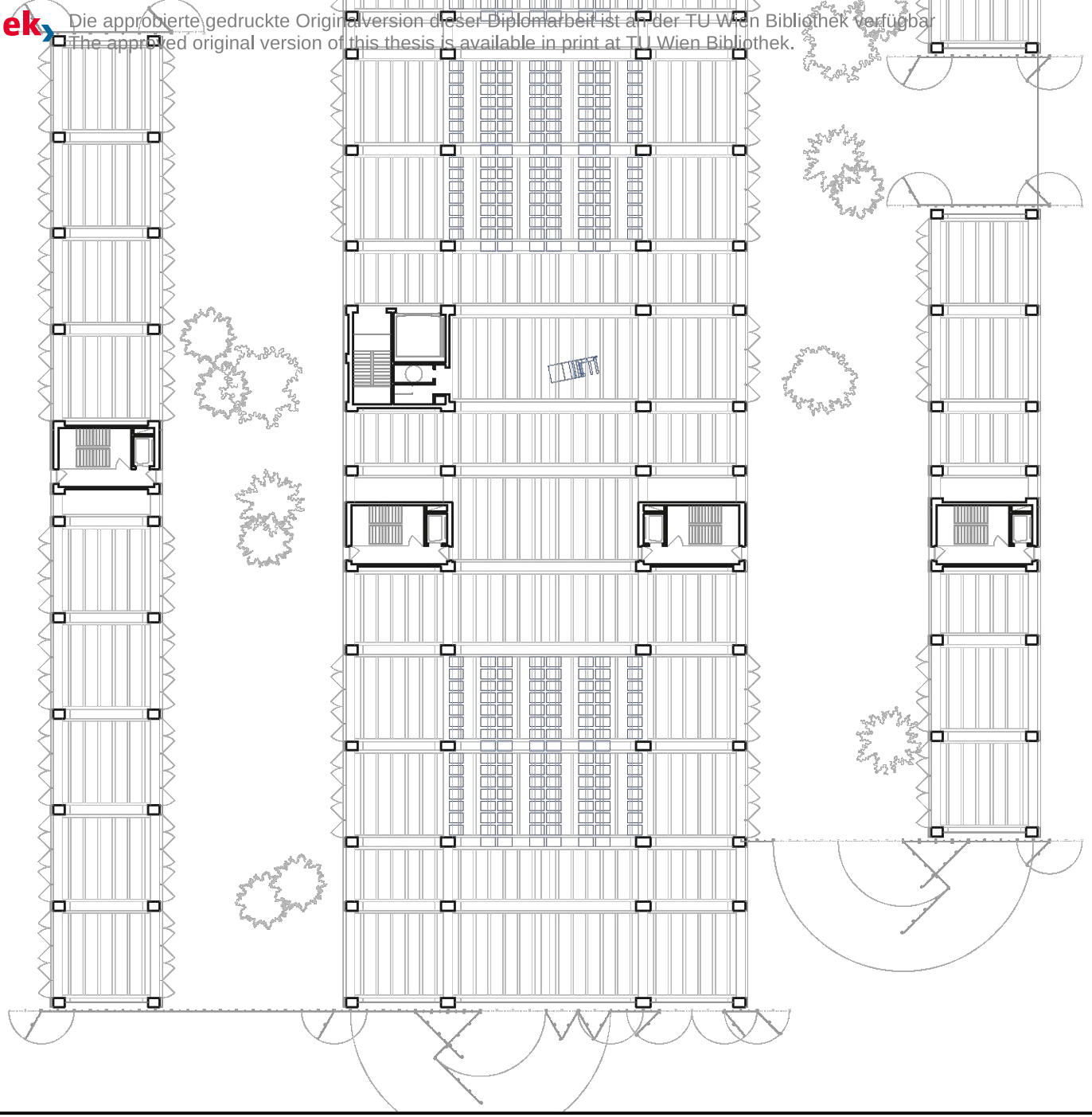
The Project upper level, scale 1:500



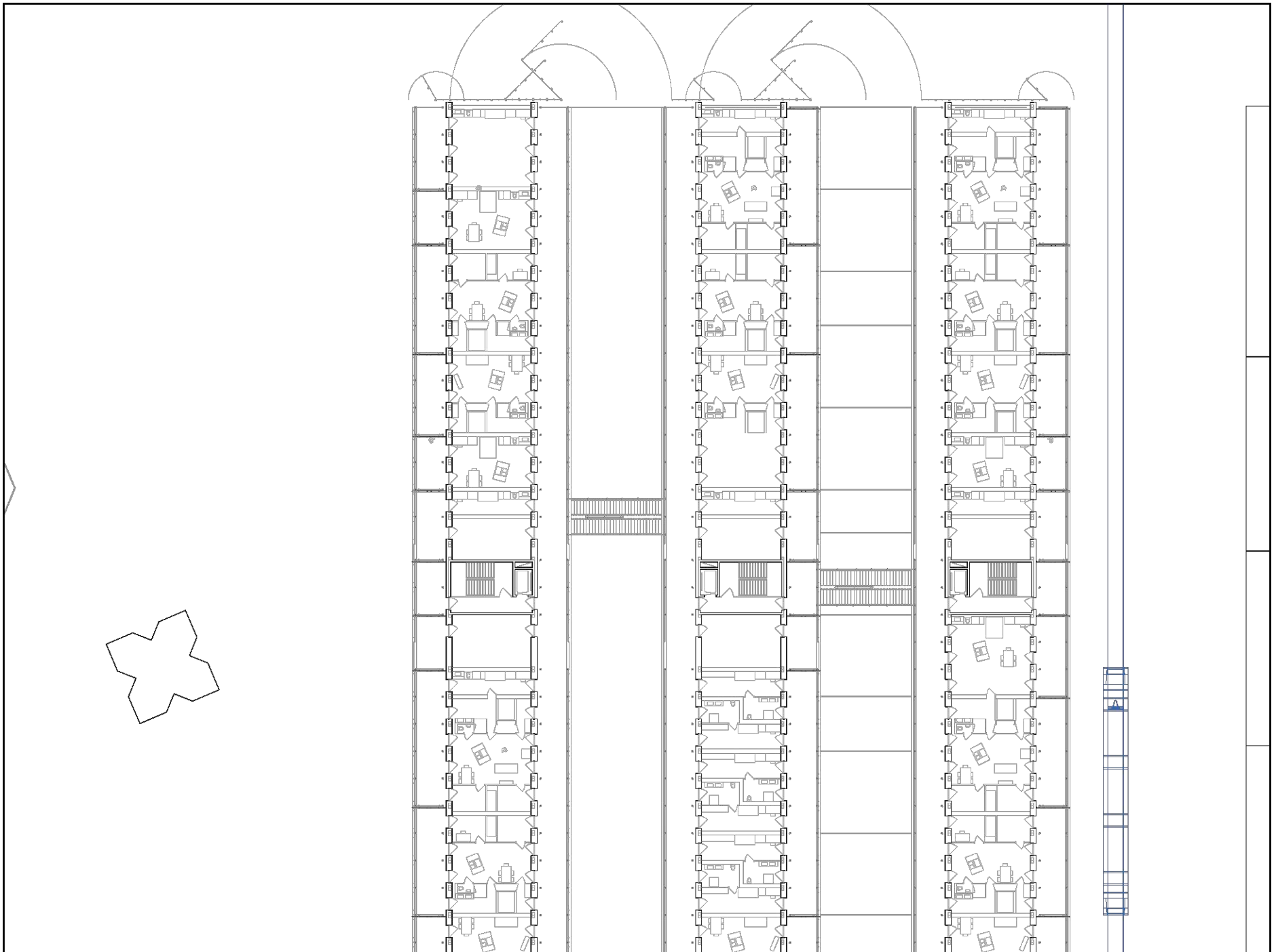


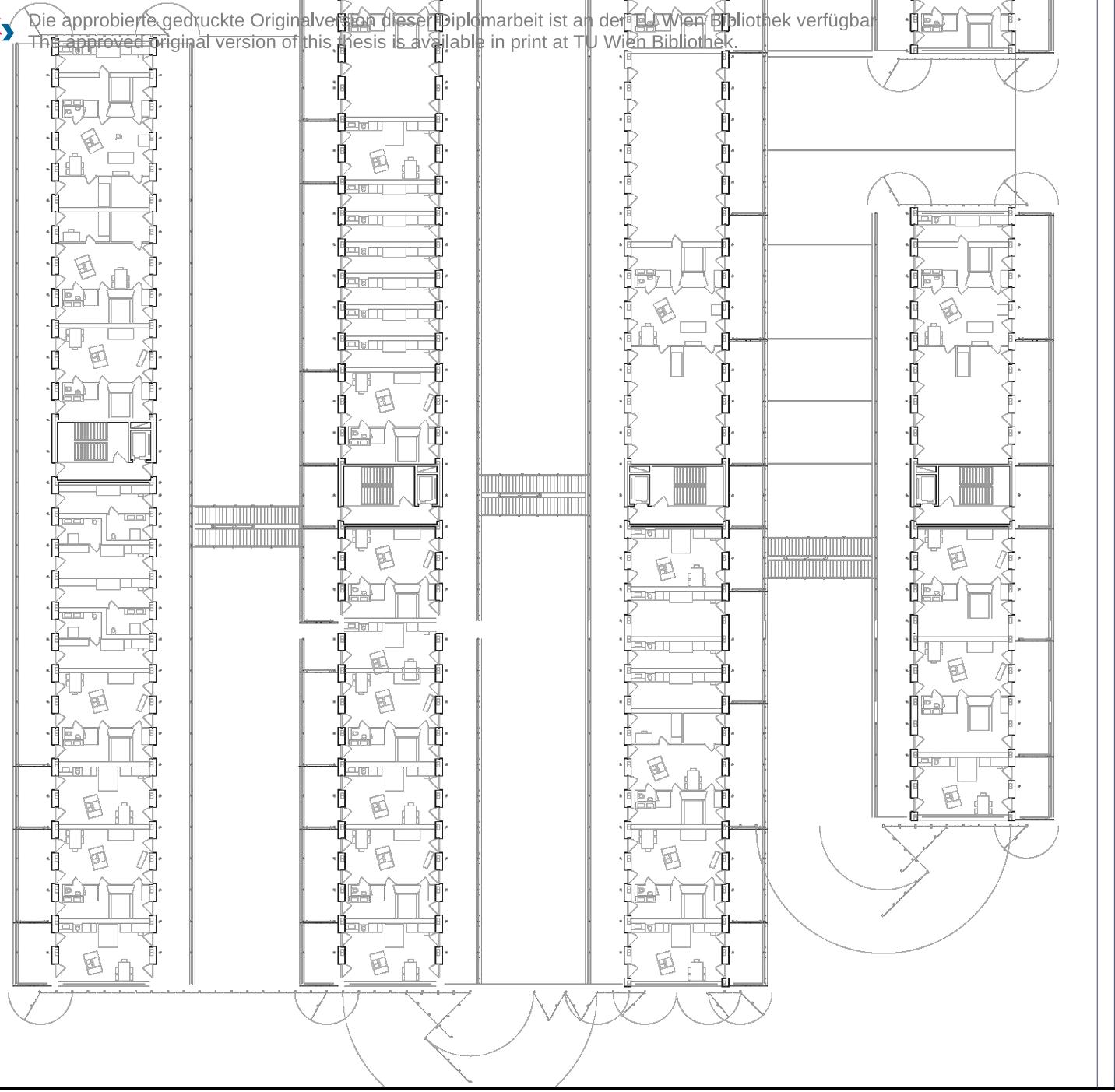
#65 #66 The Project lower level, scale 1:500

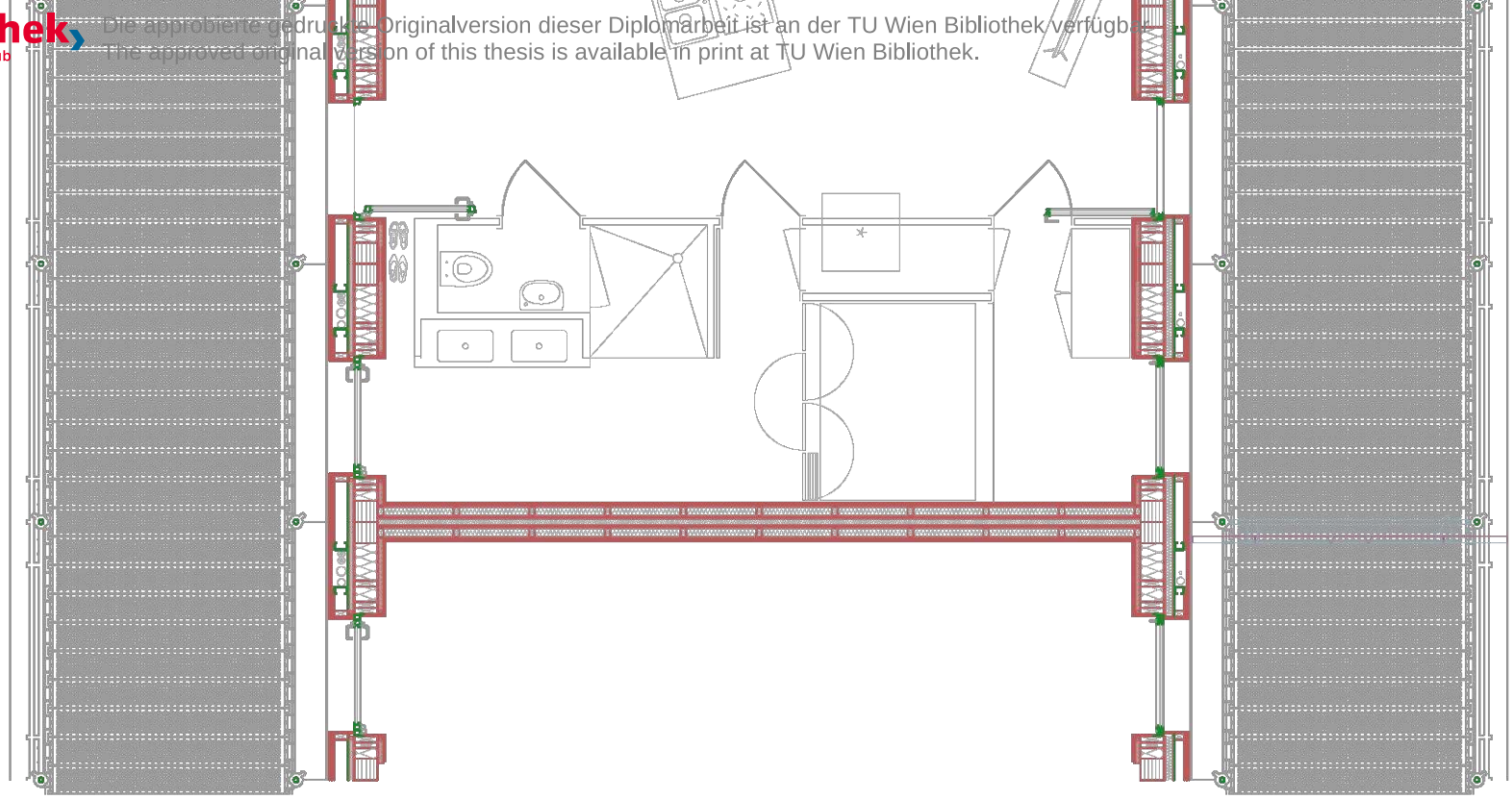


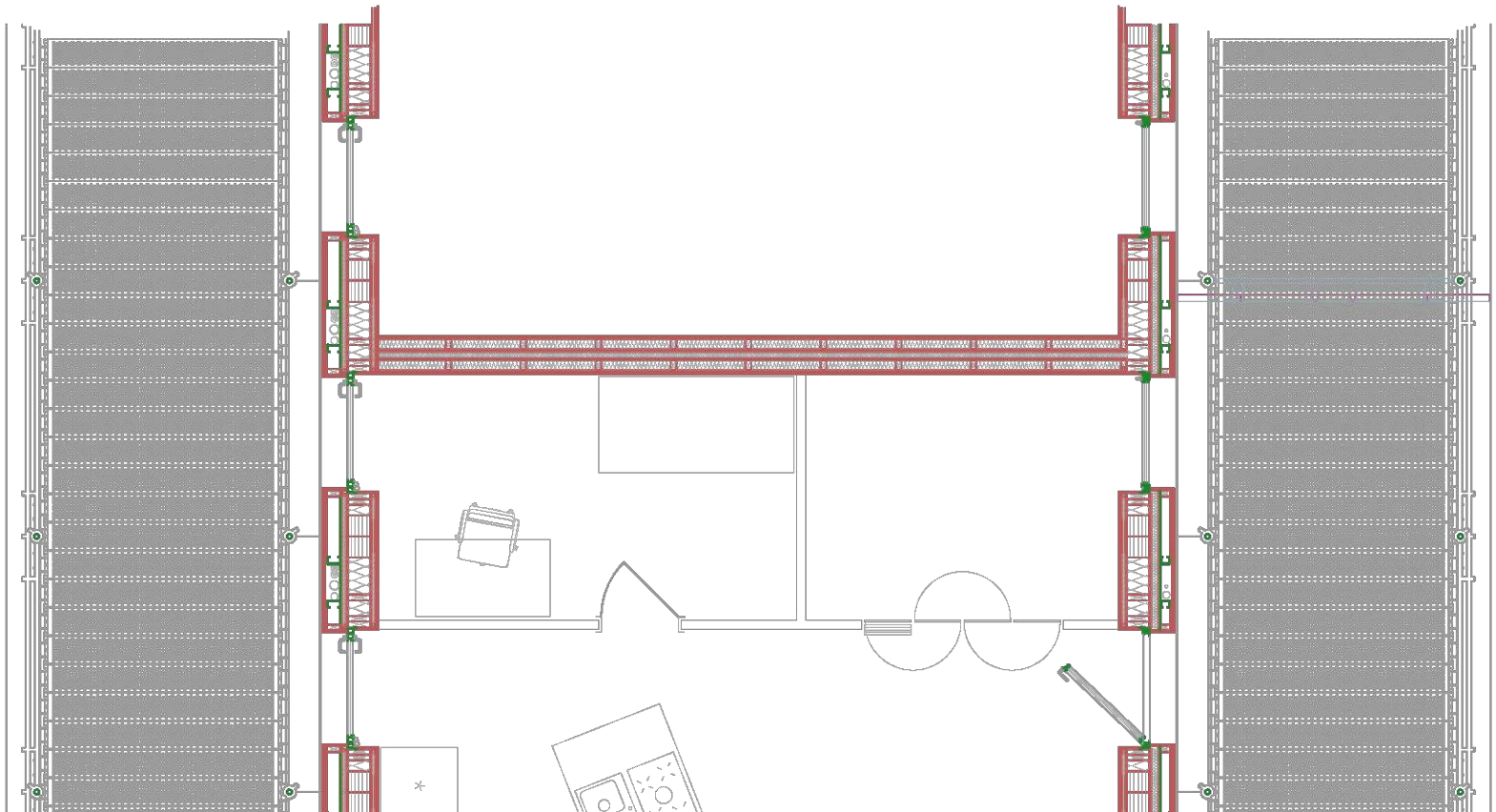


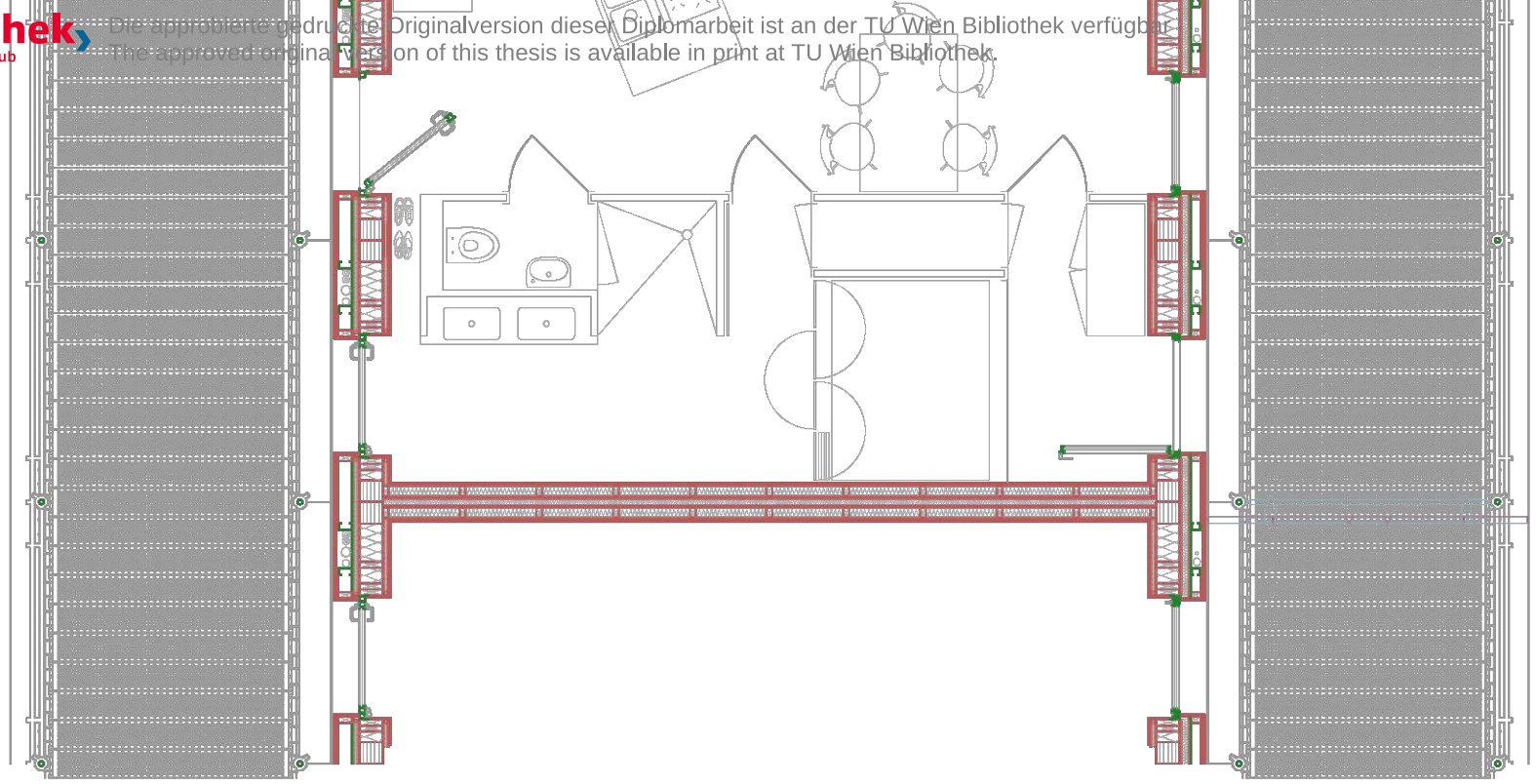
The Project ground level, scale 1:500

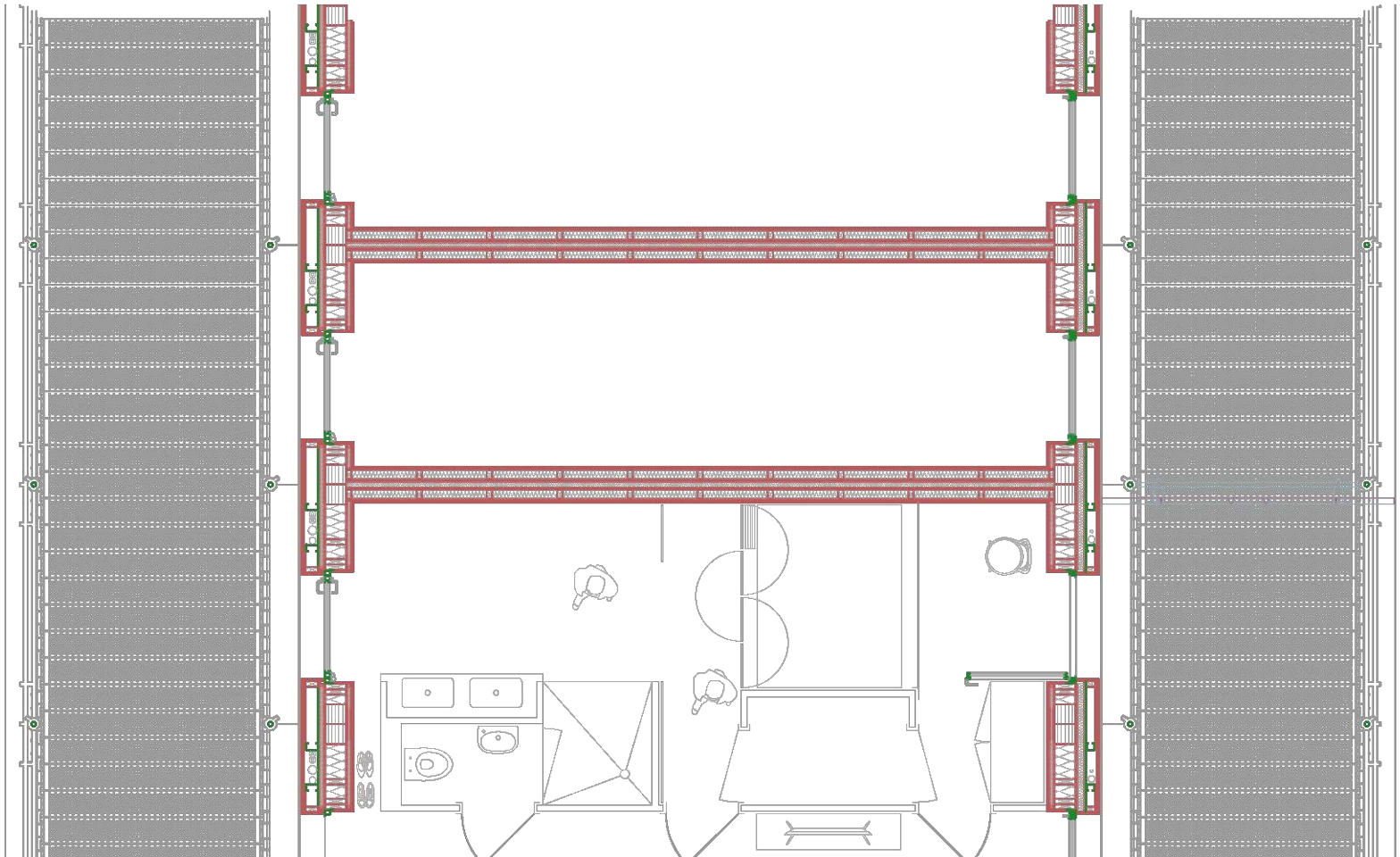


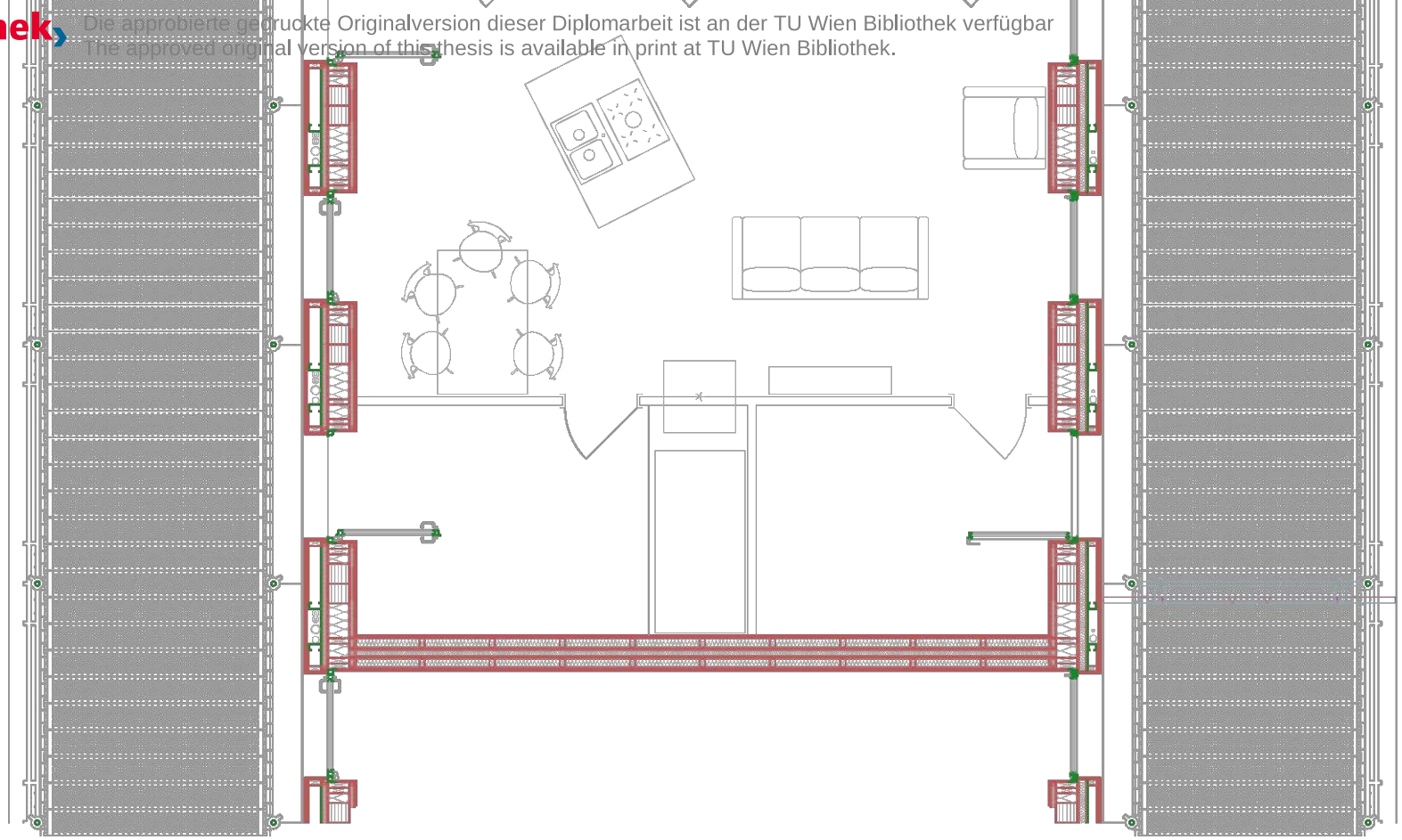


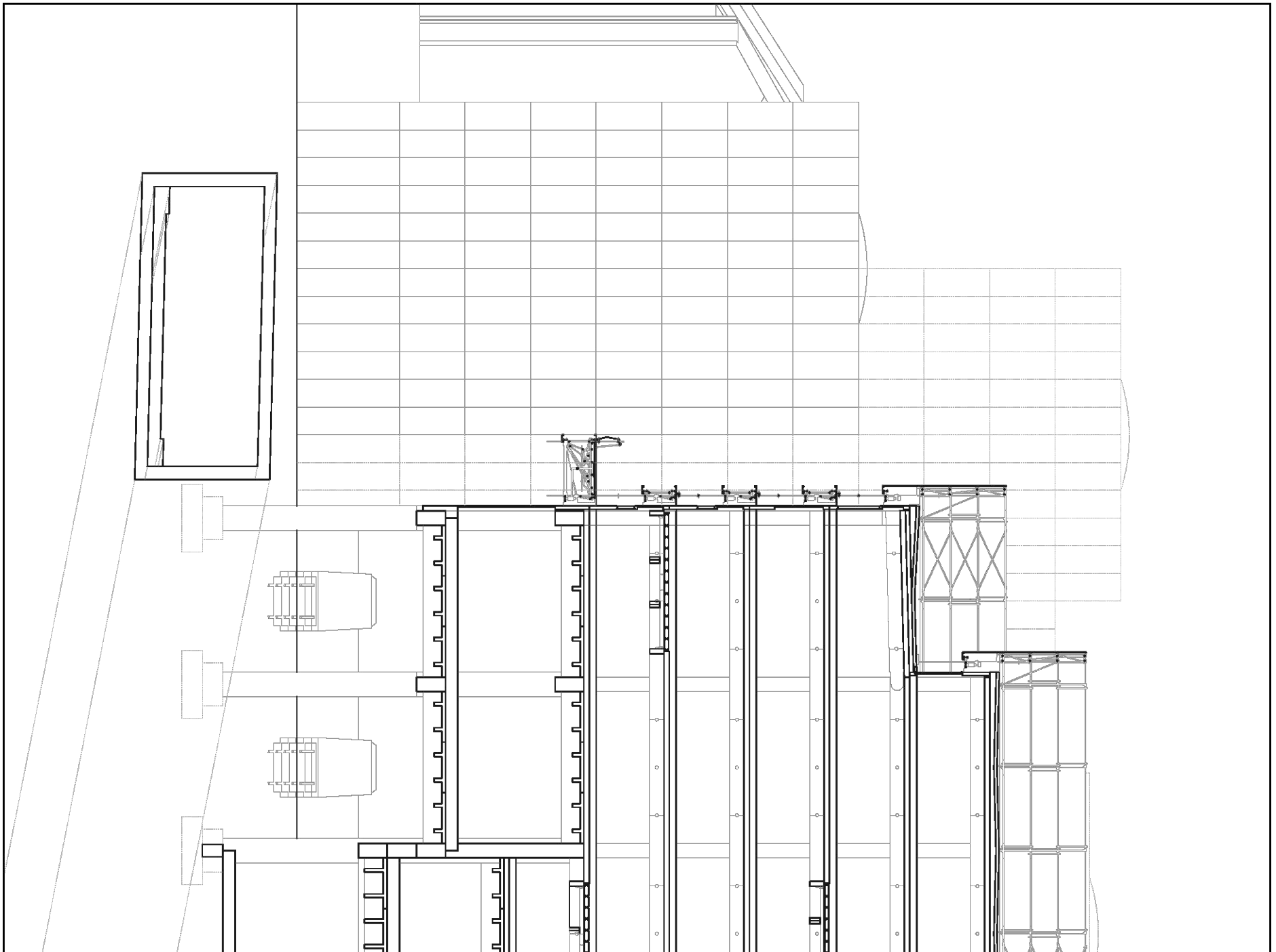


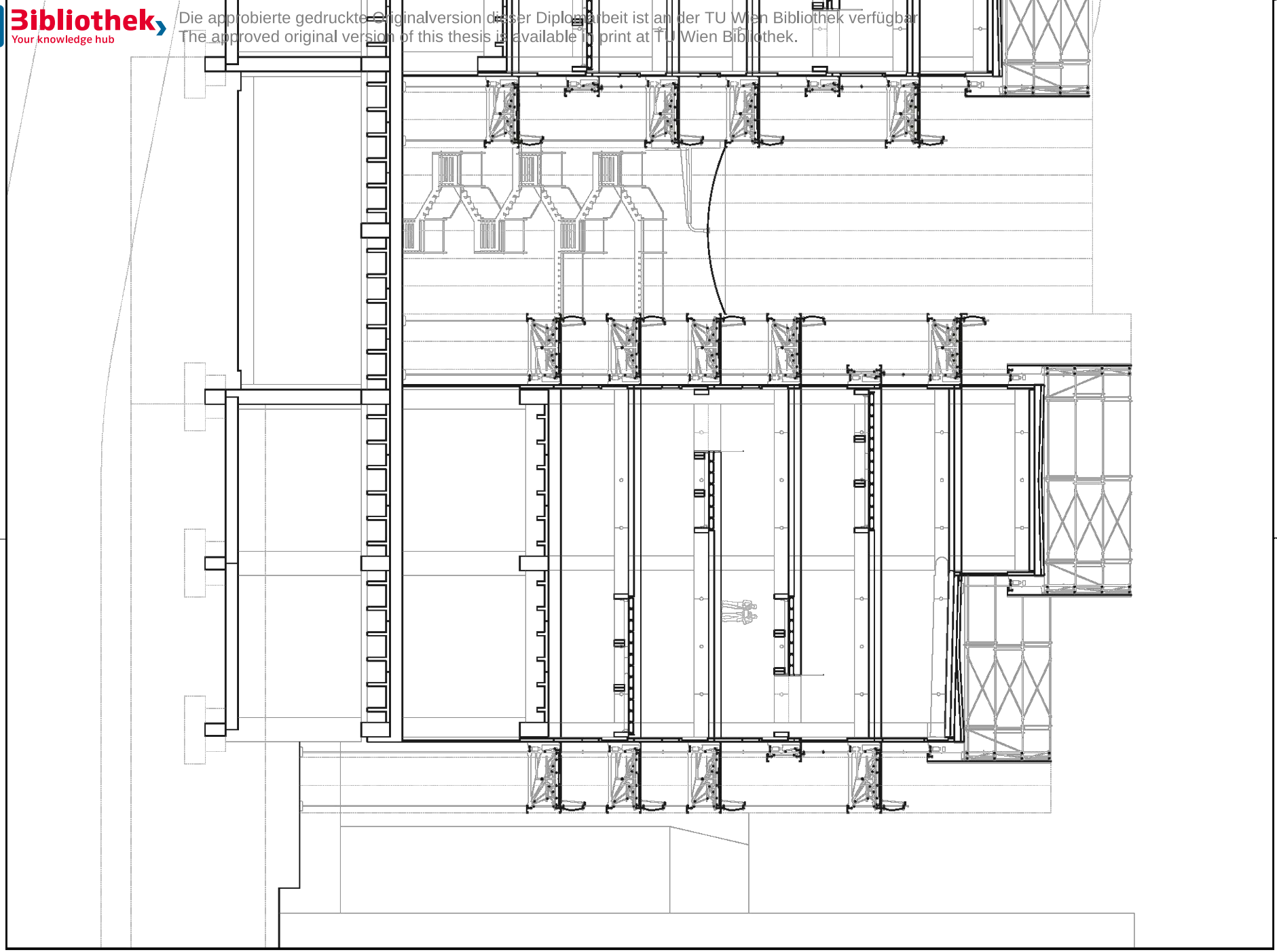




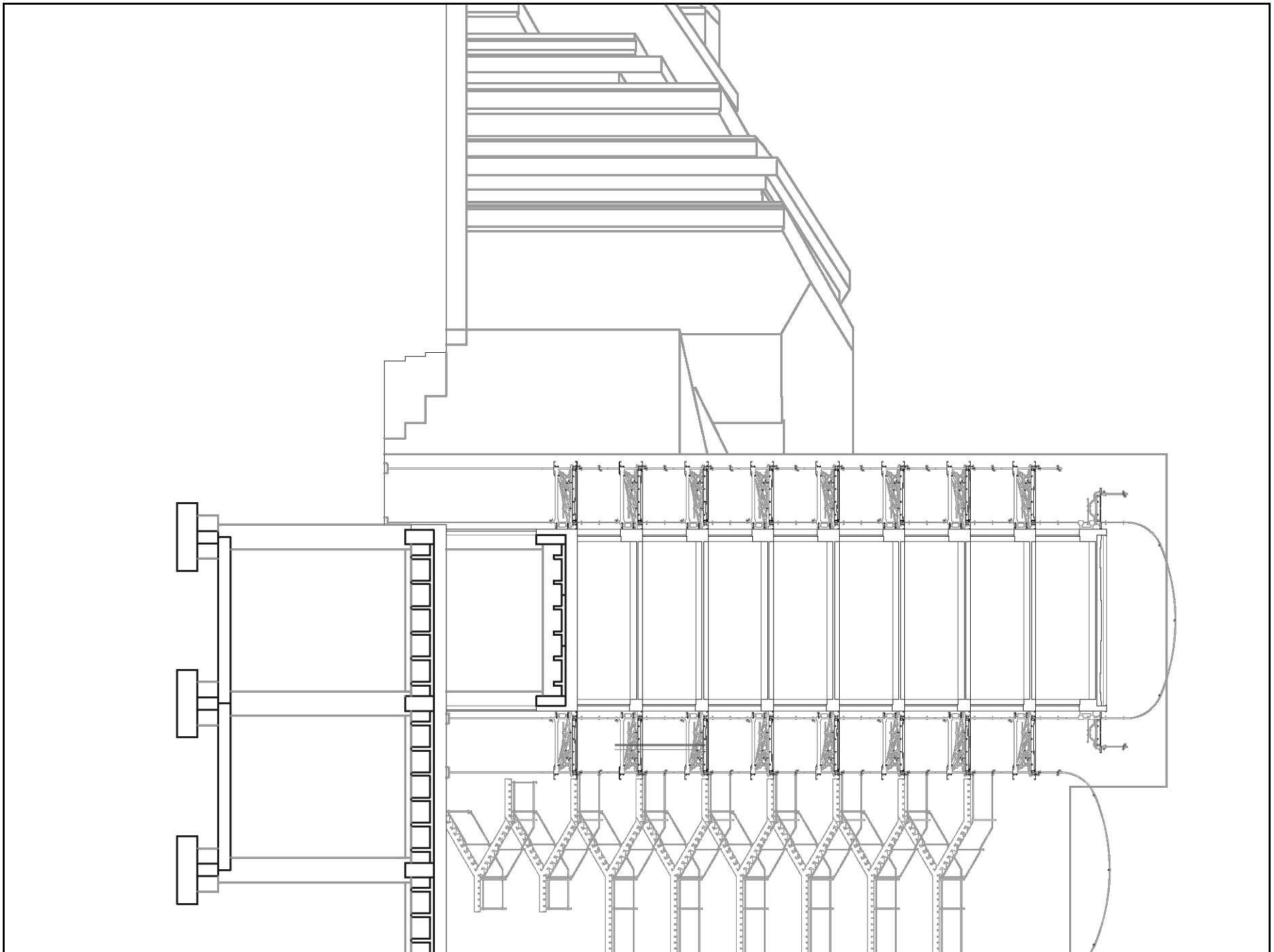


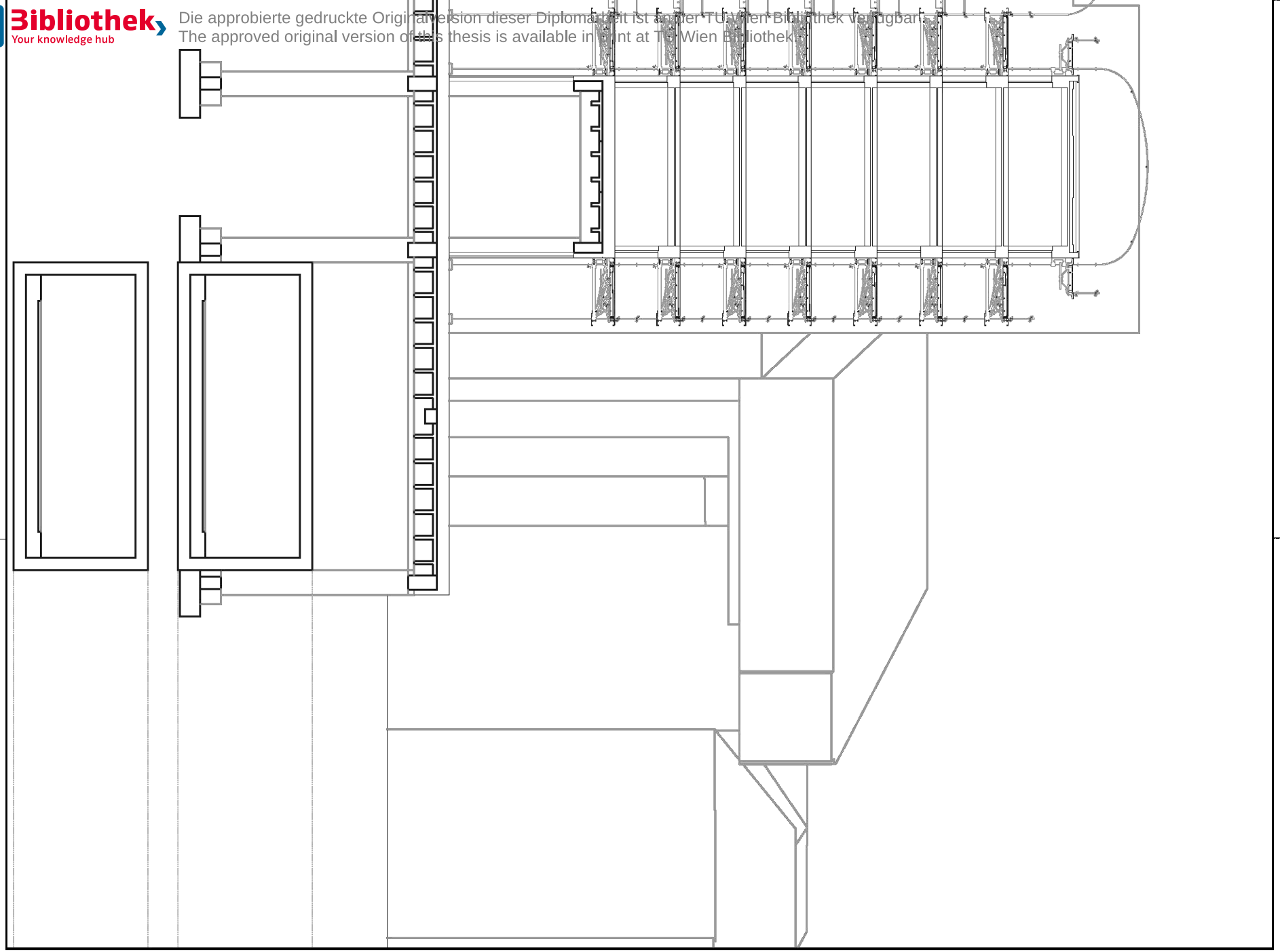




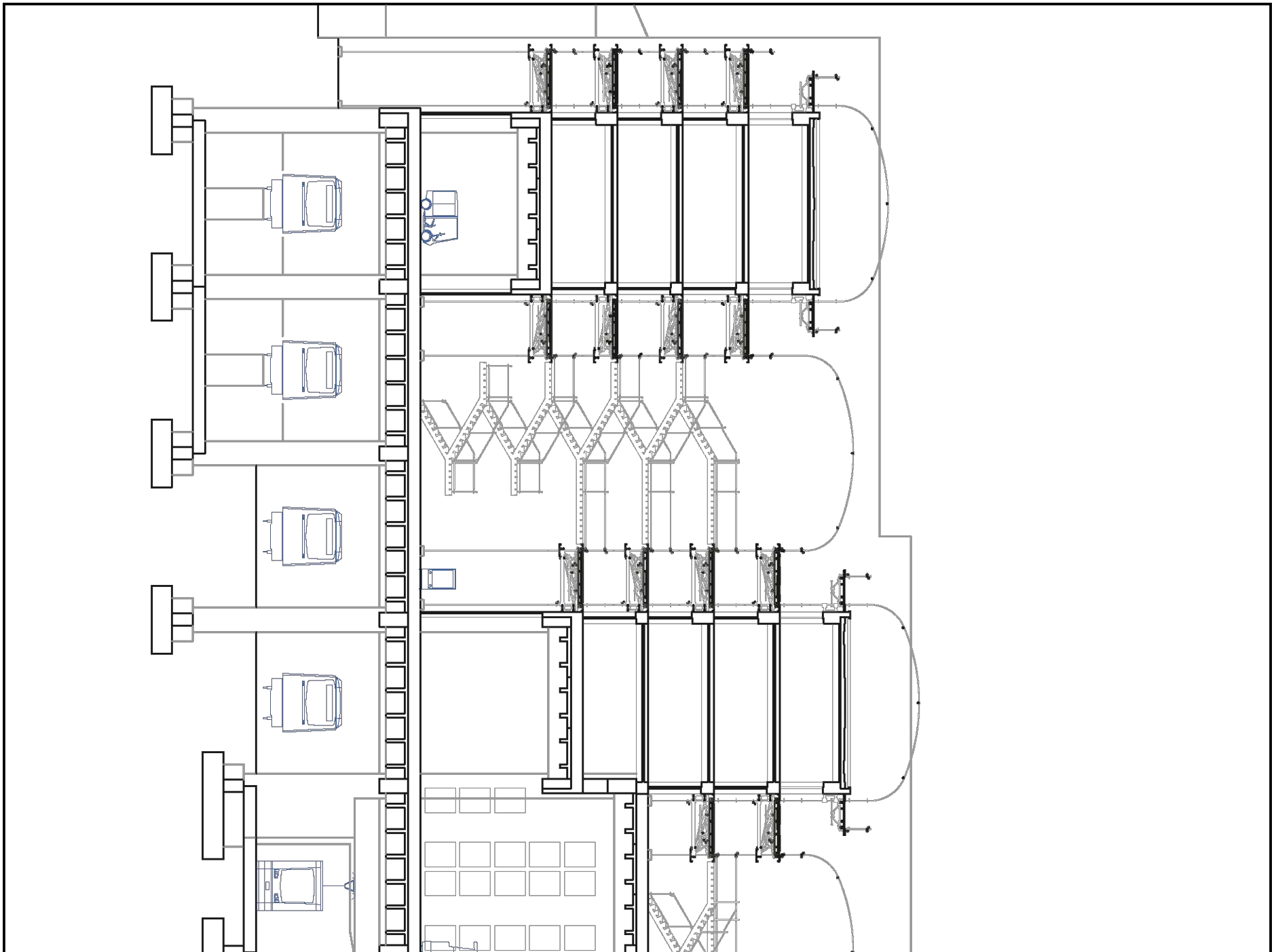


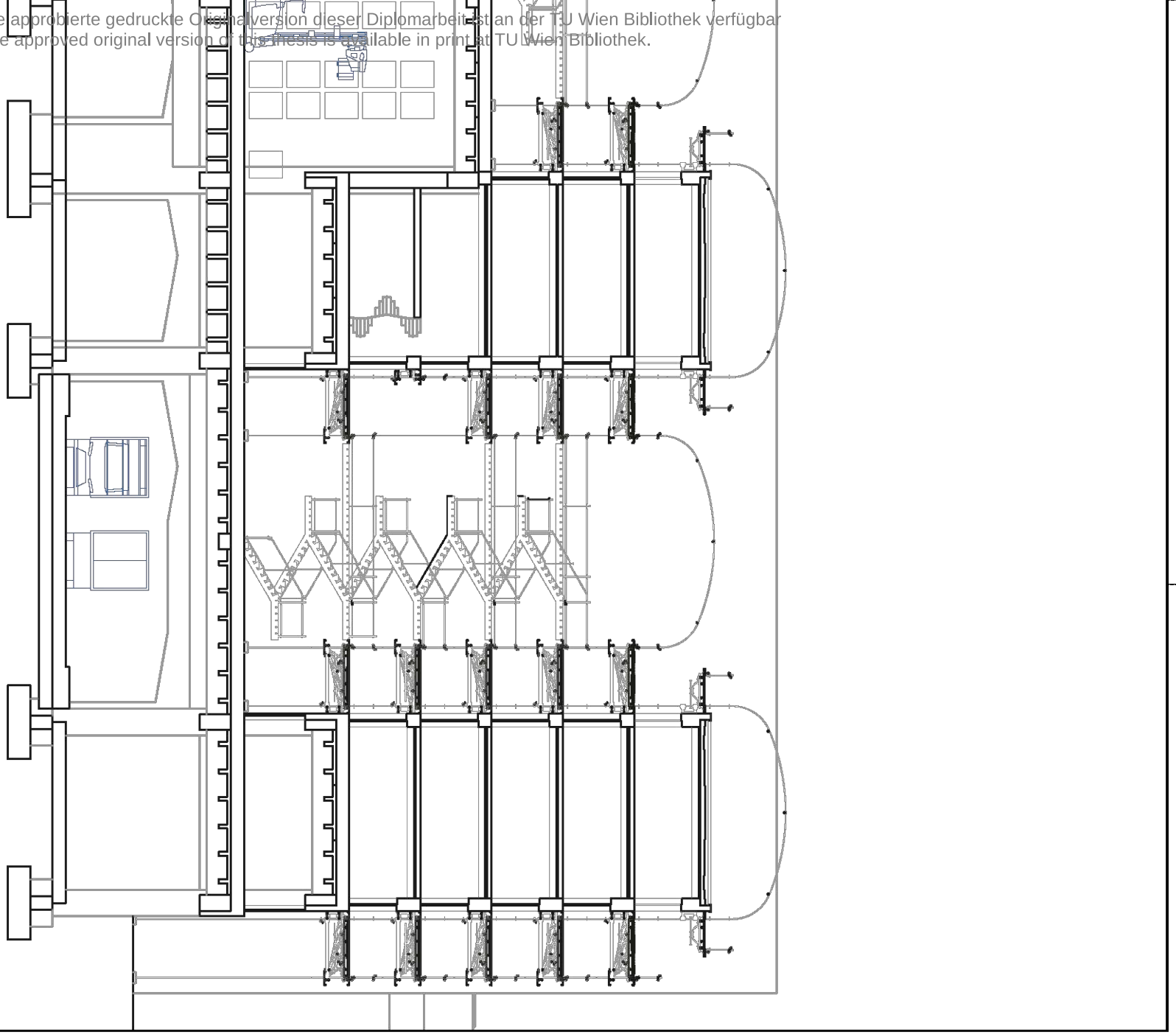
section A-A, scale 1:250



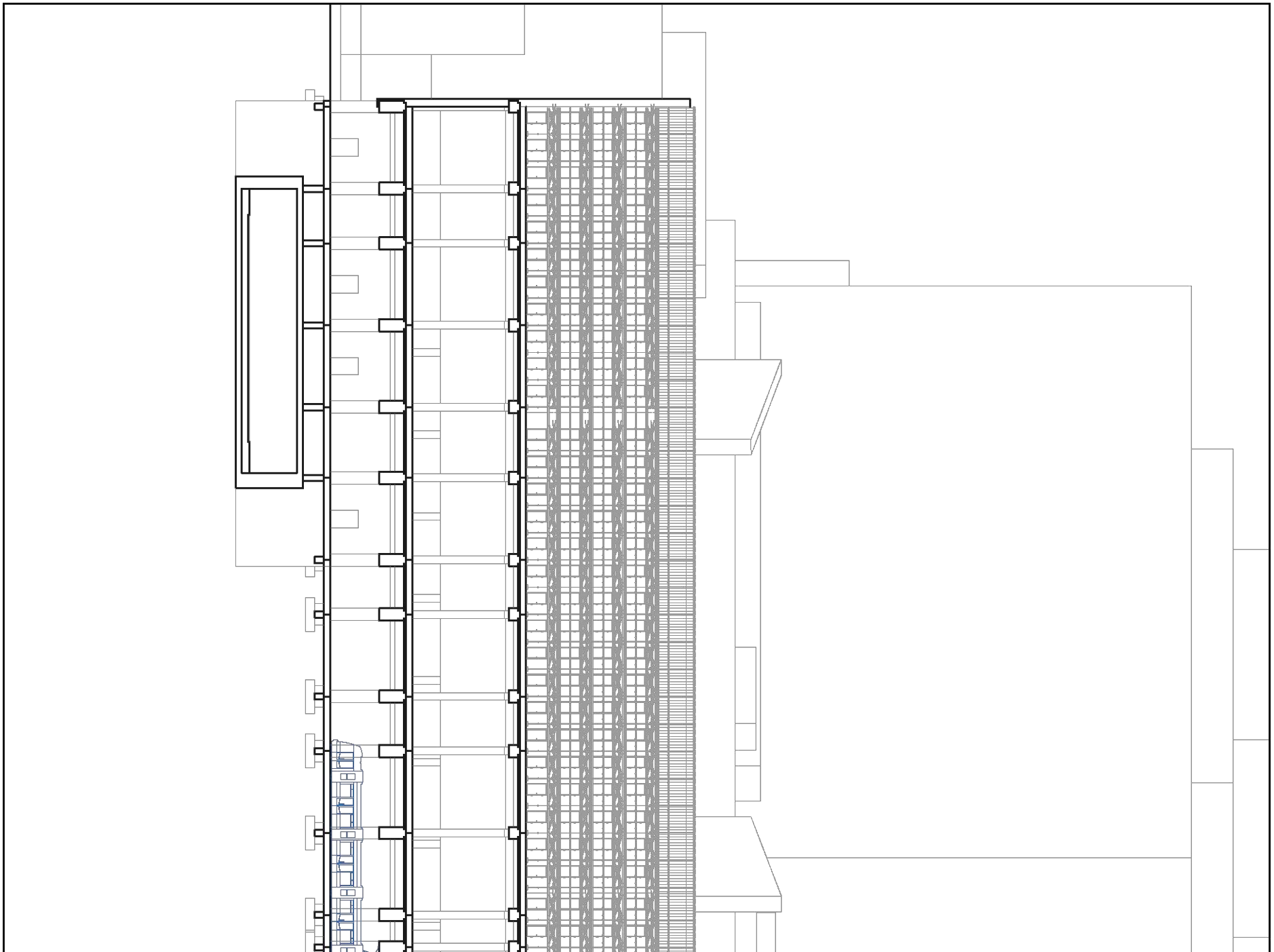


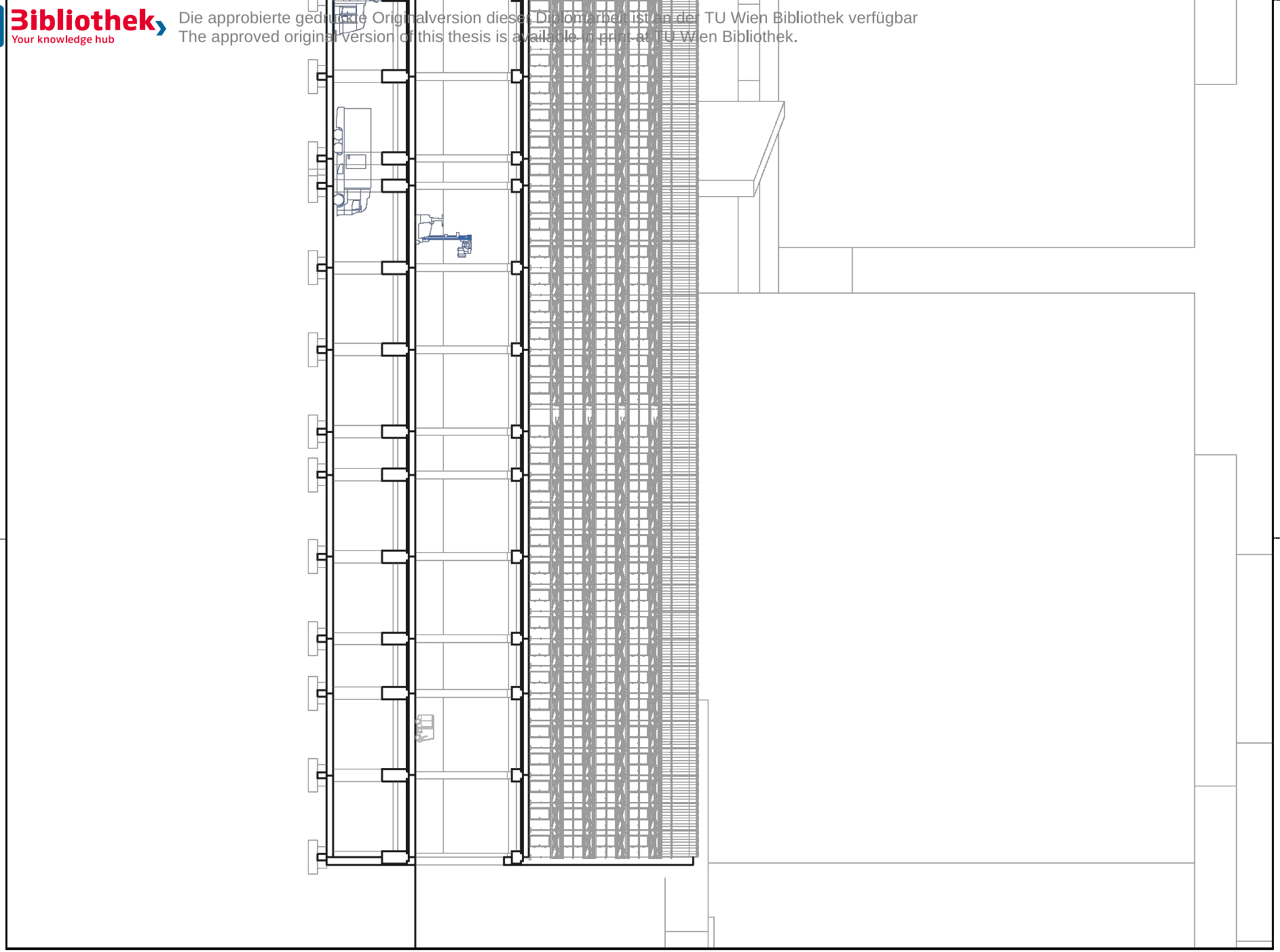
section B-B, scale 1:250



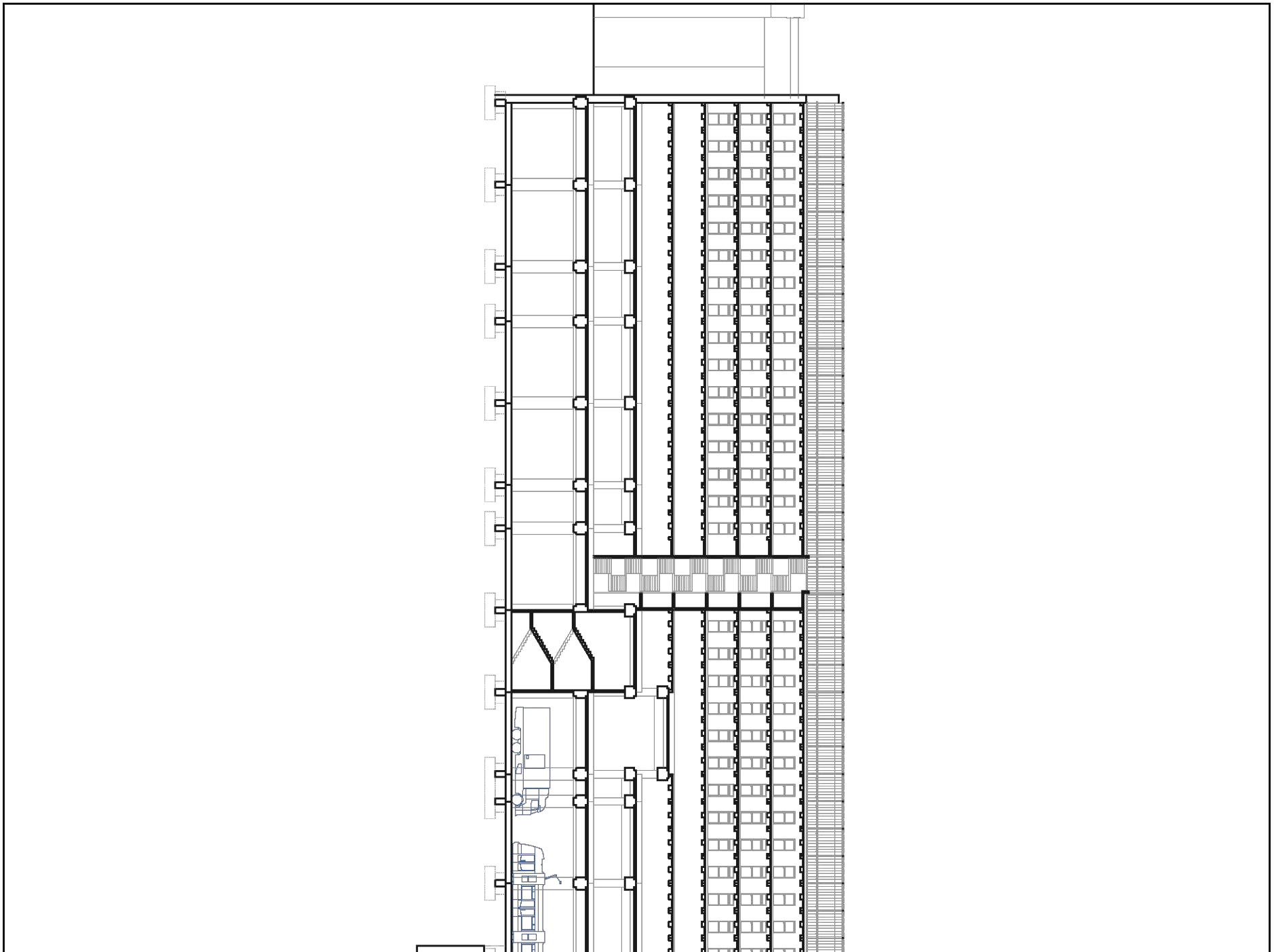


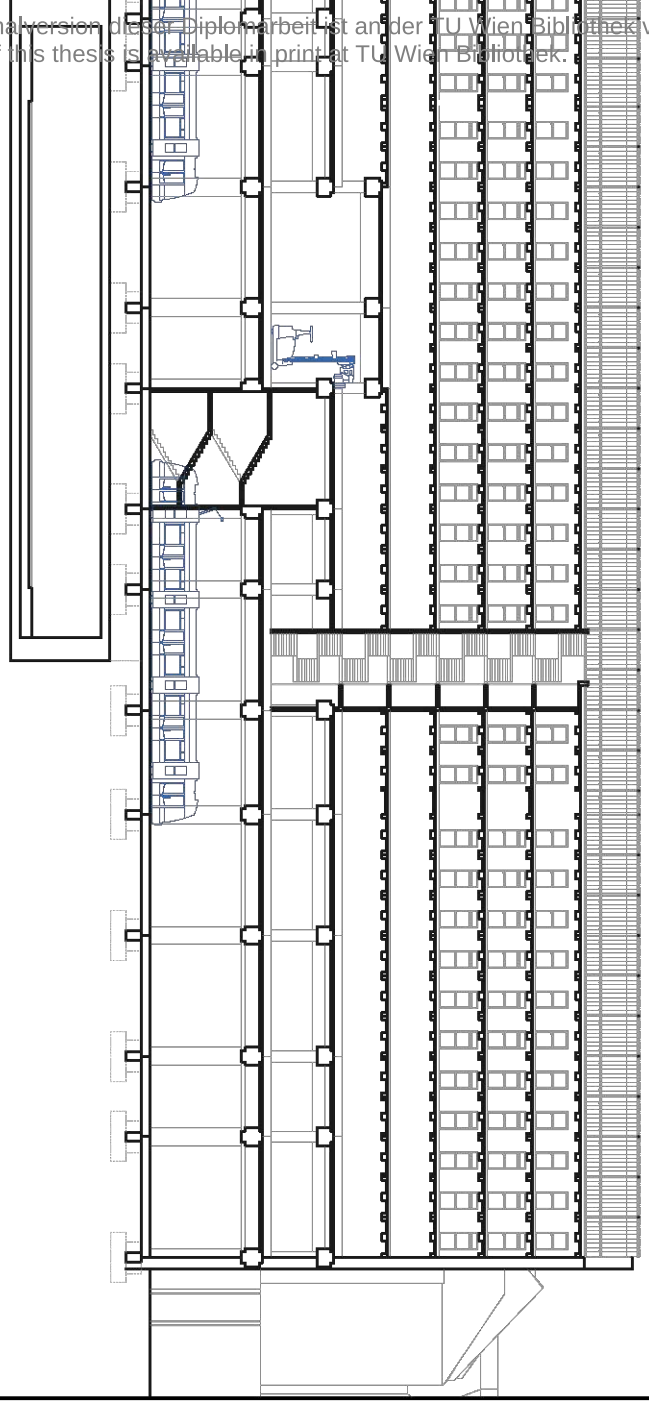
section C-C, scale 1:250



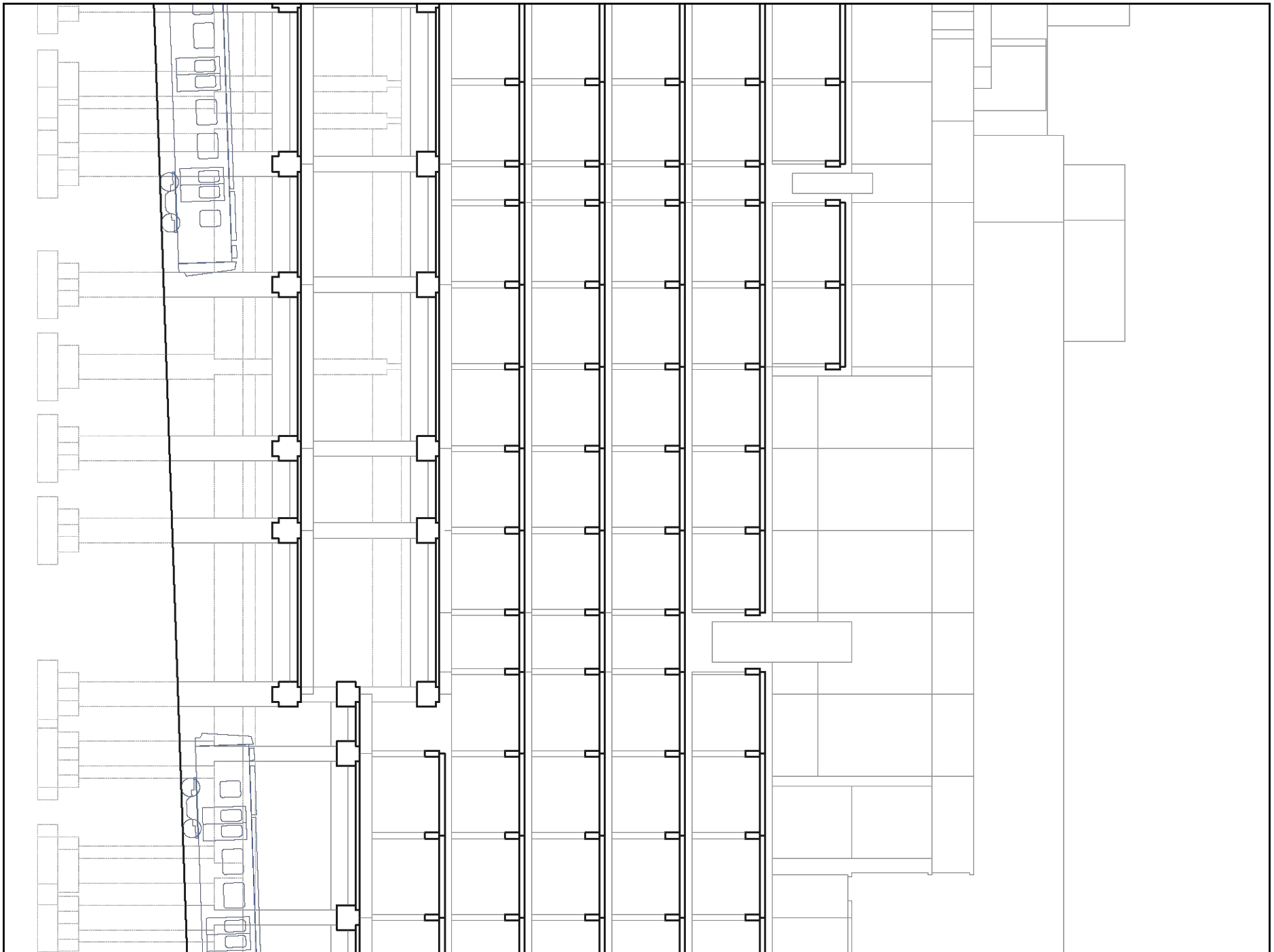


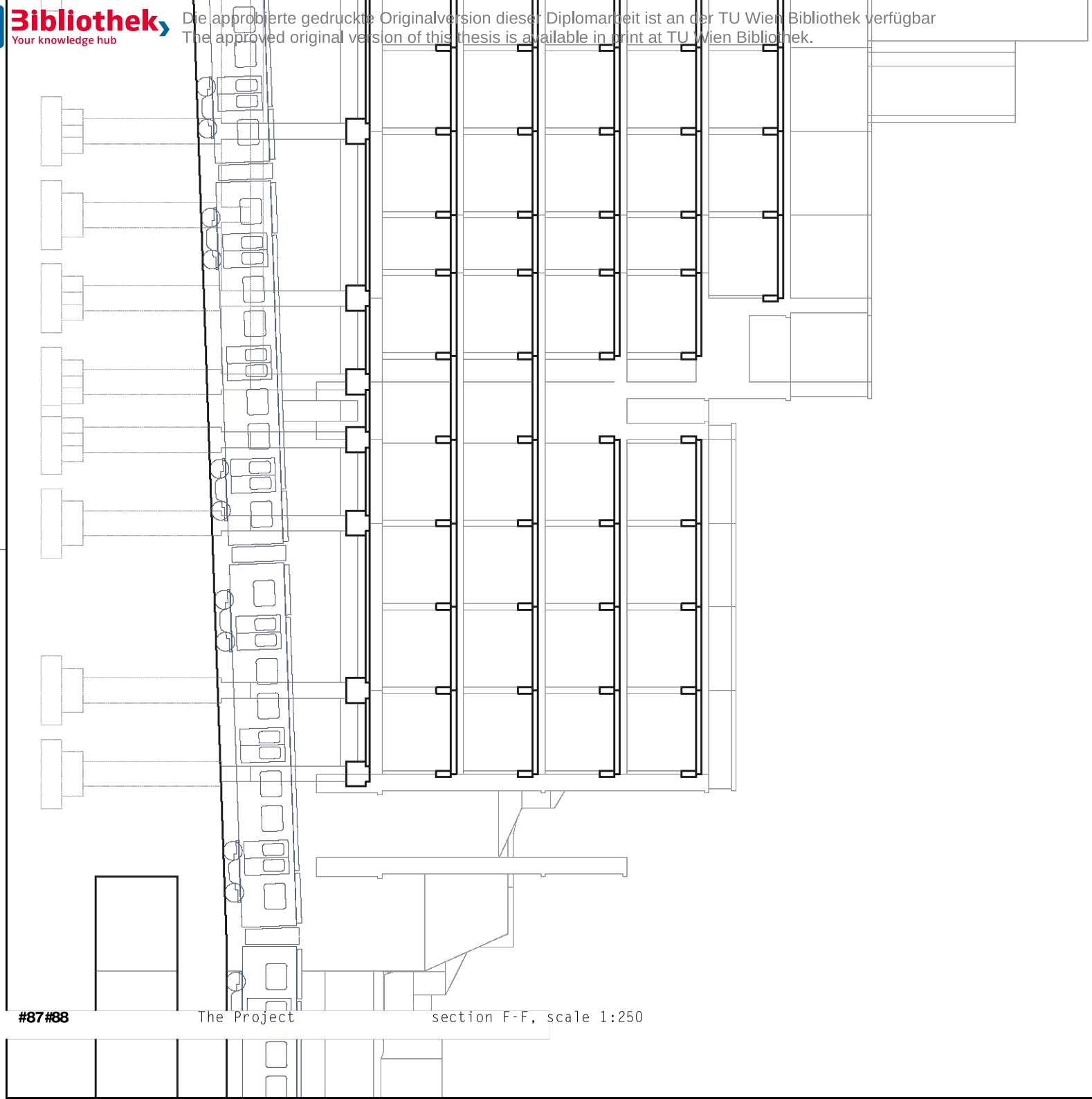
section D-D, scale 1:250



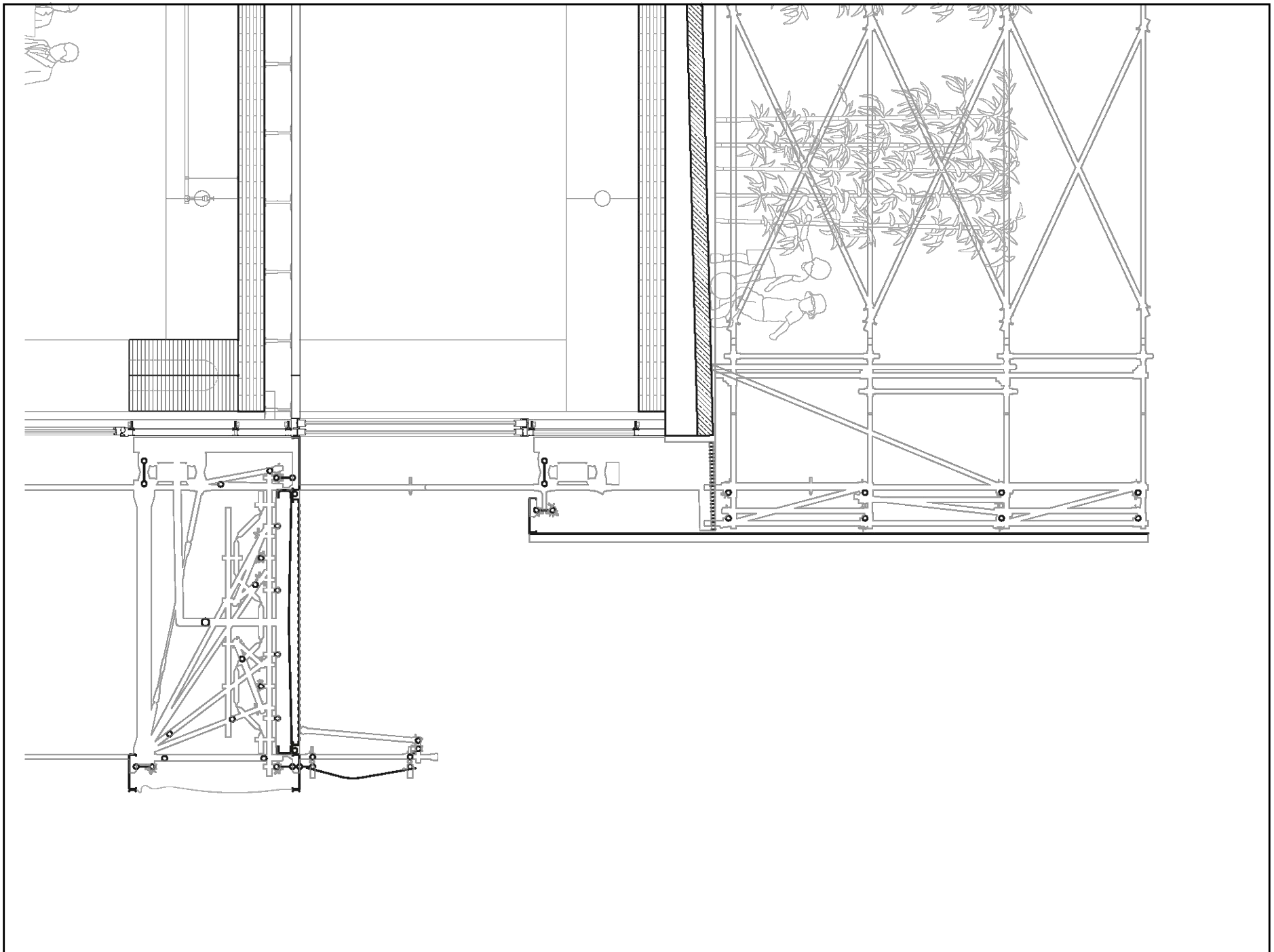


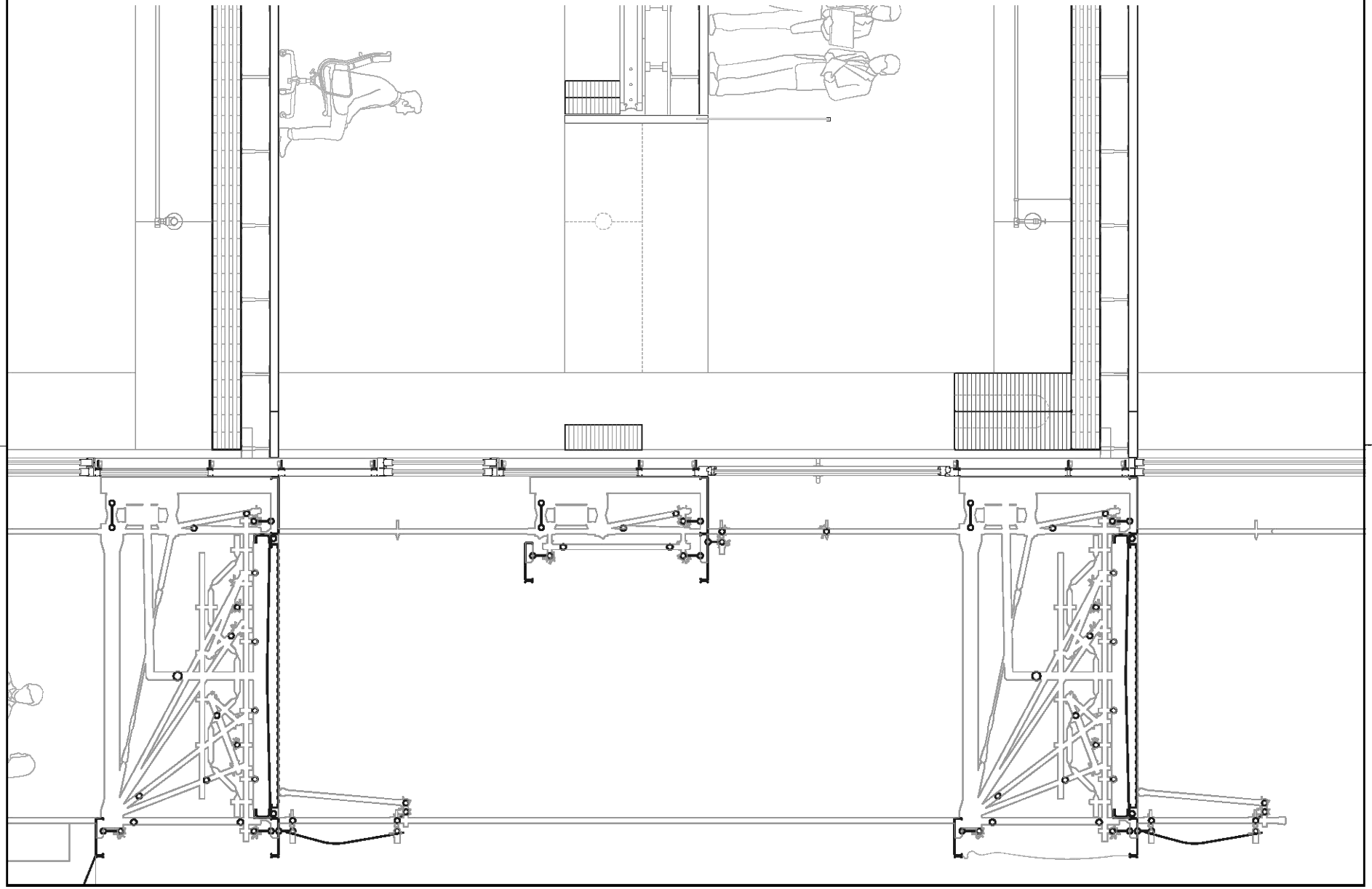
section E-E, scale 1:250



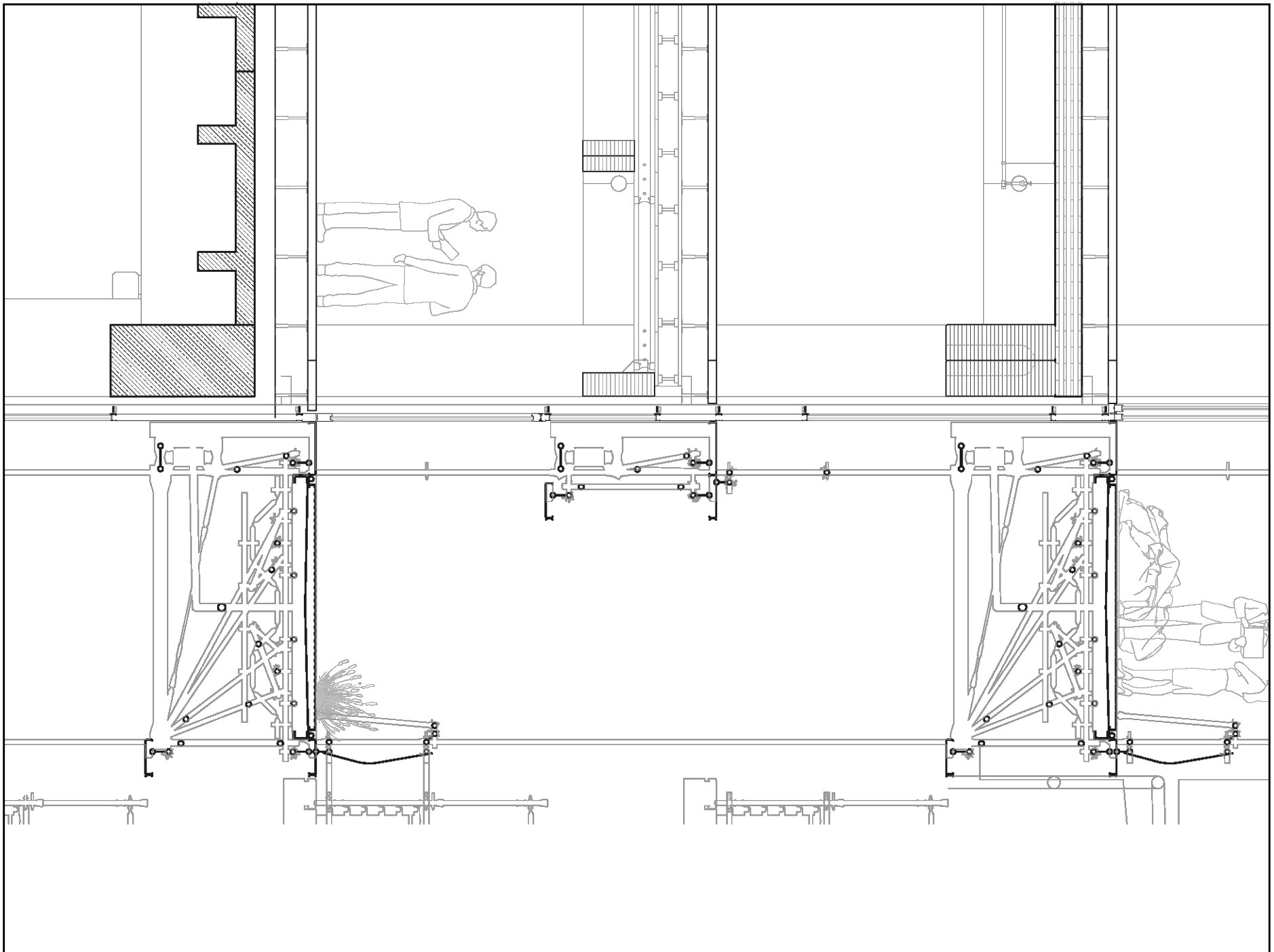


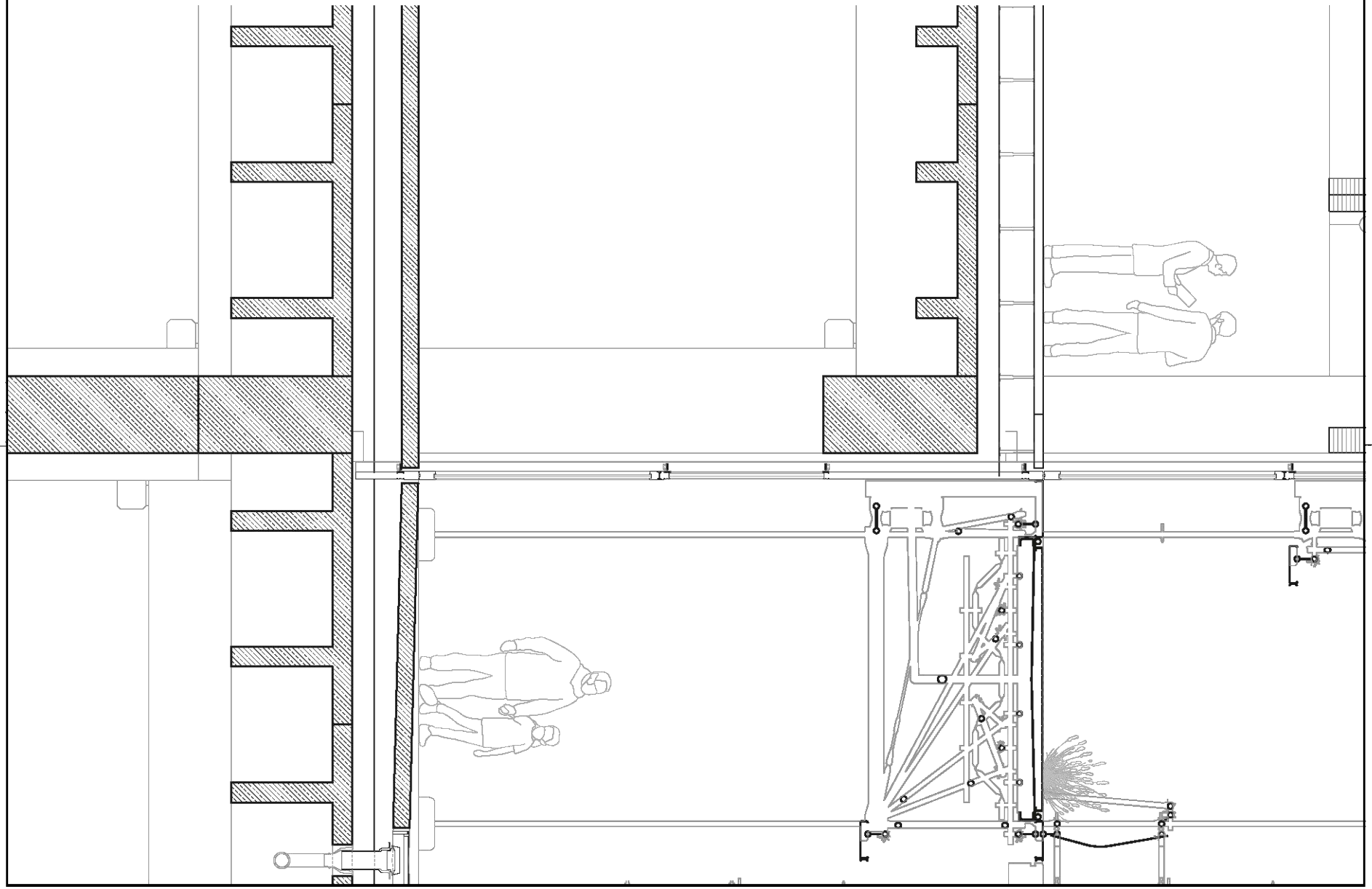
section F-F, scale 1:250



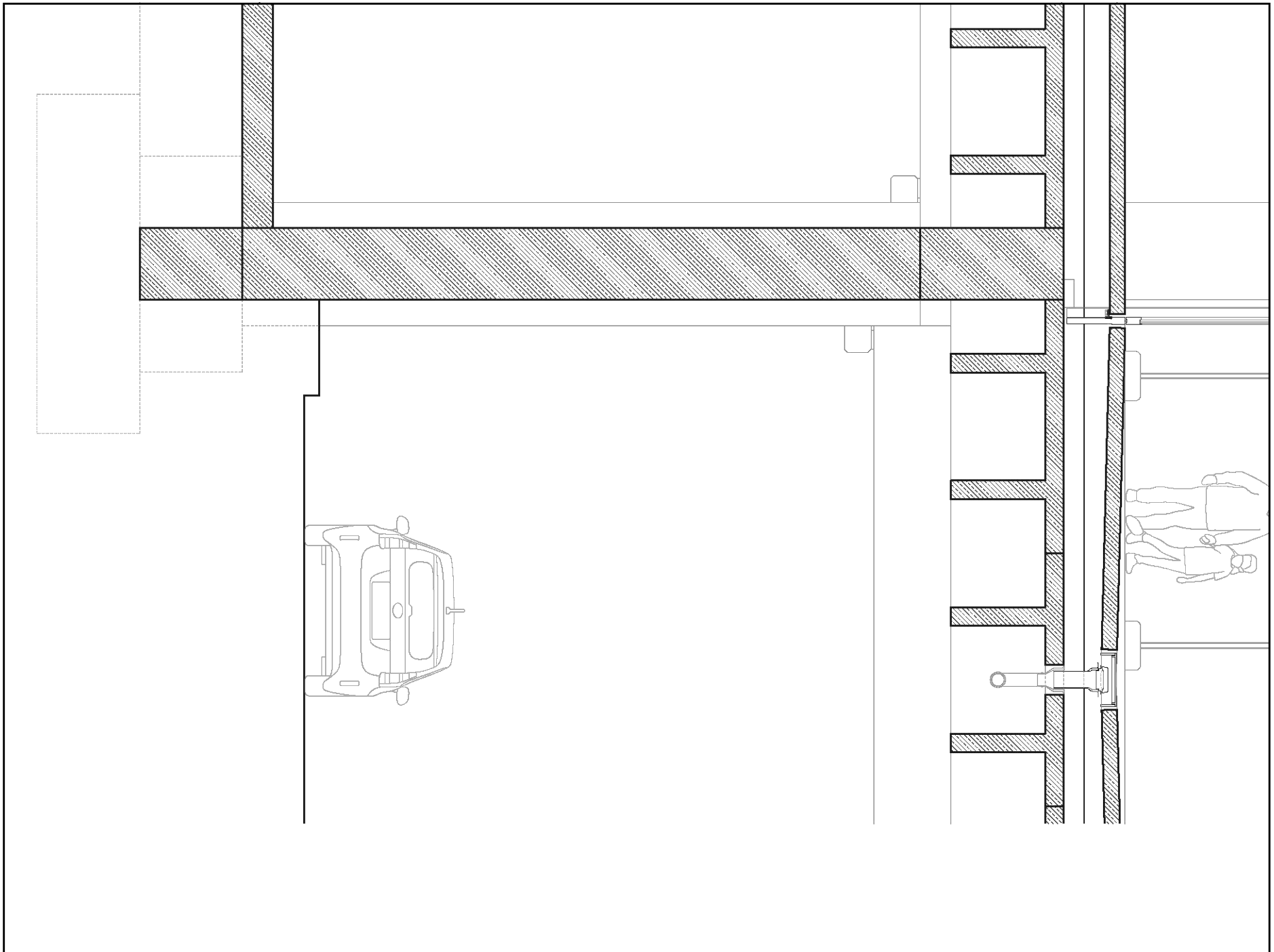


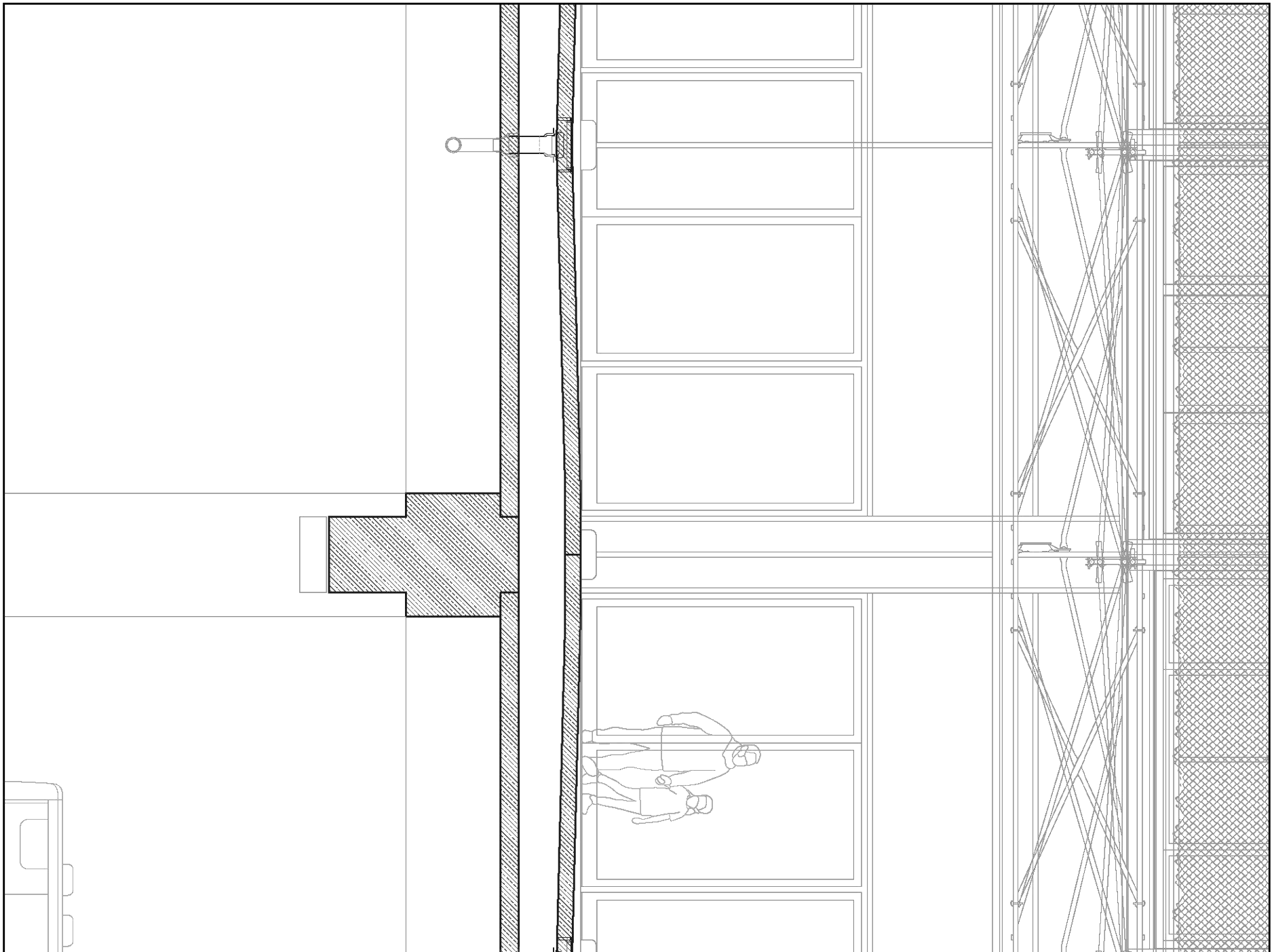
facade section A-A, scale 1:50

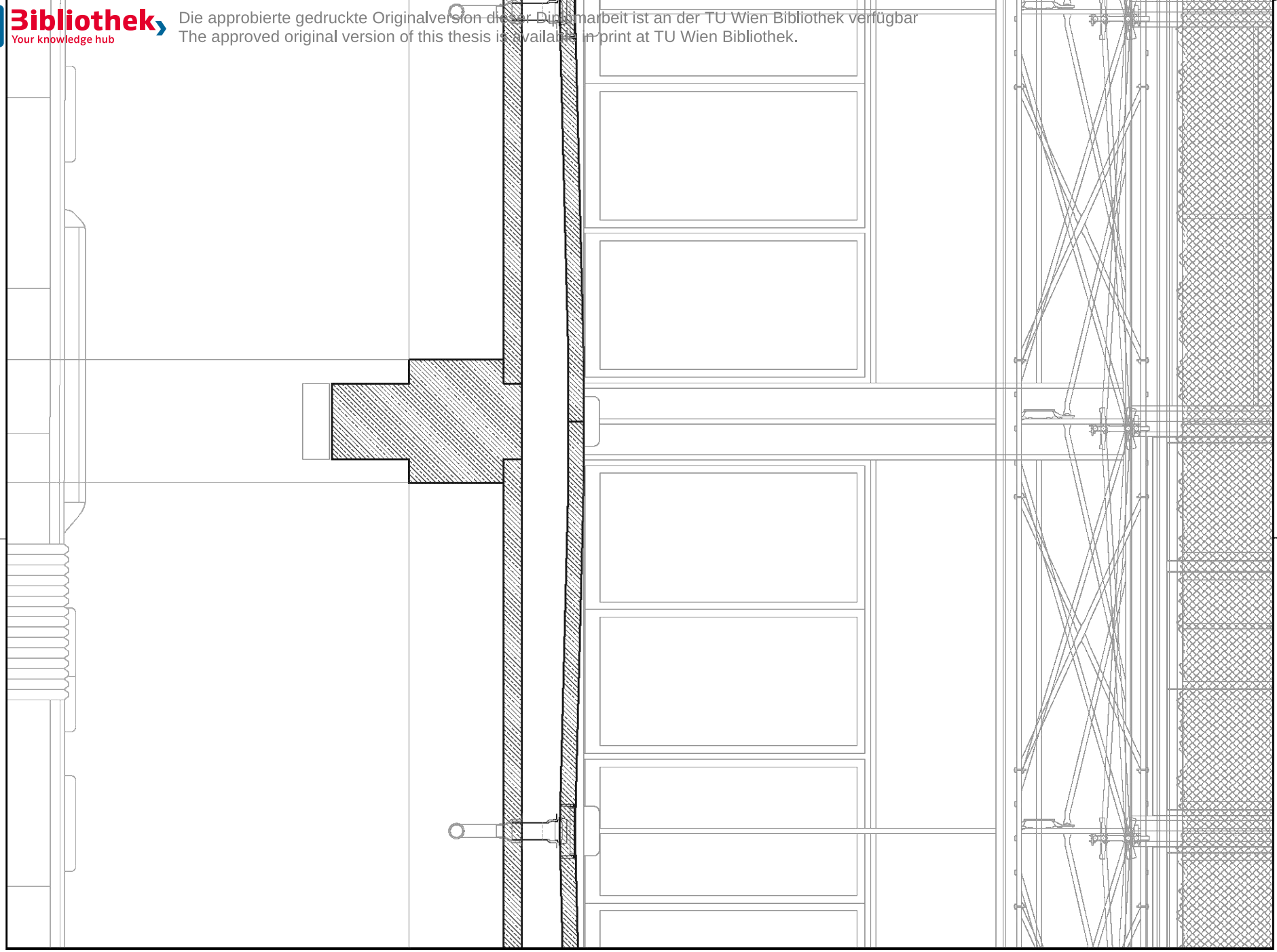




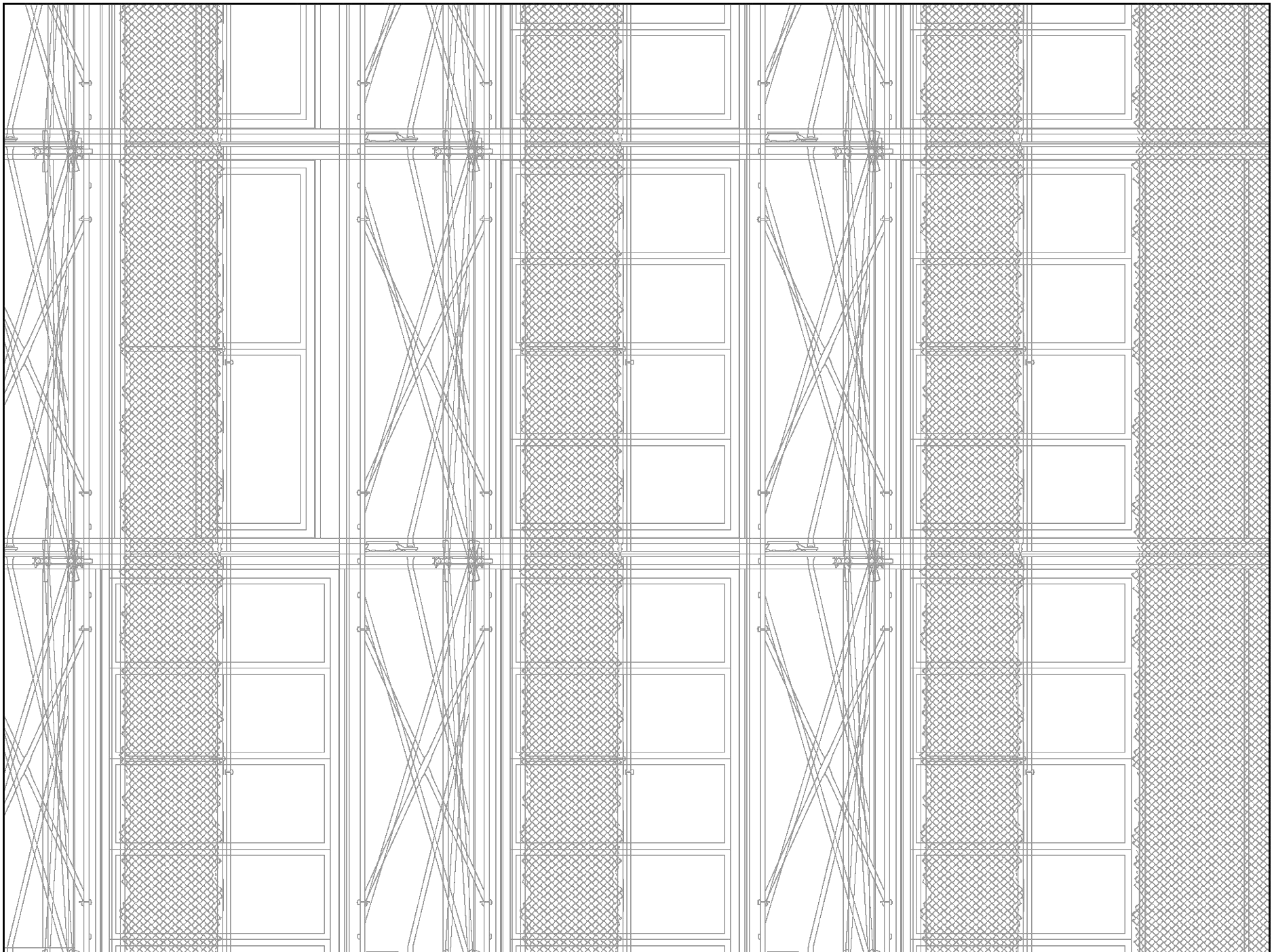
facade section A-A, scale 1:50

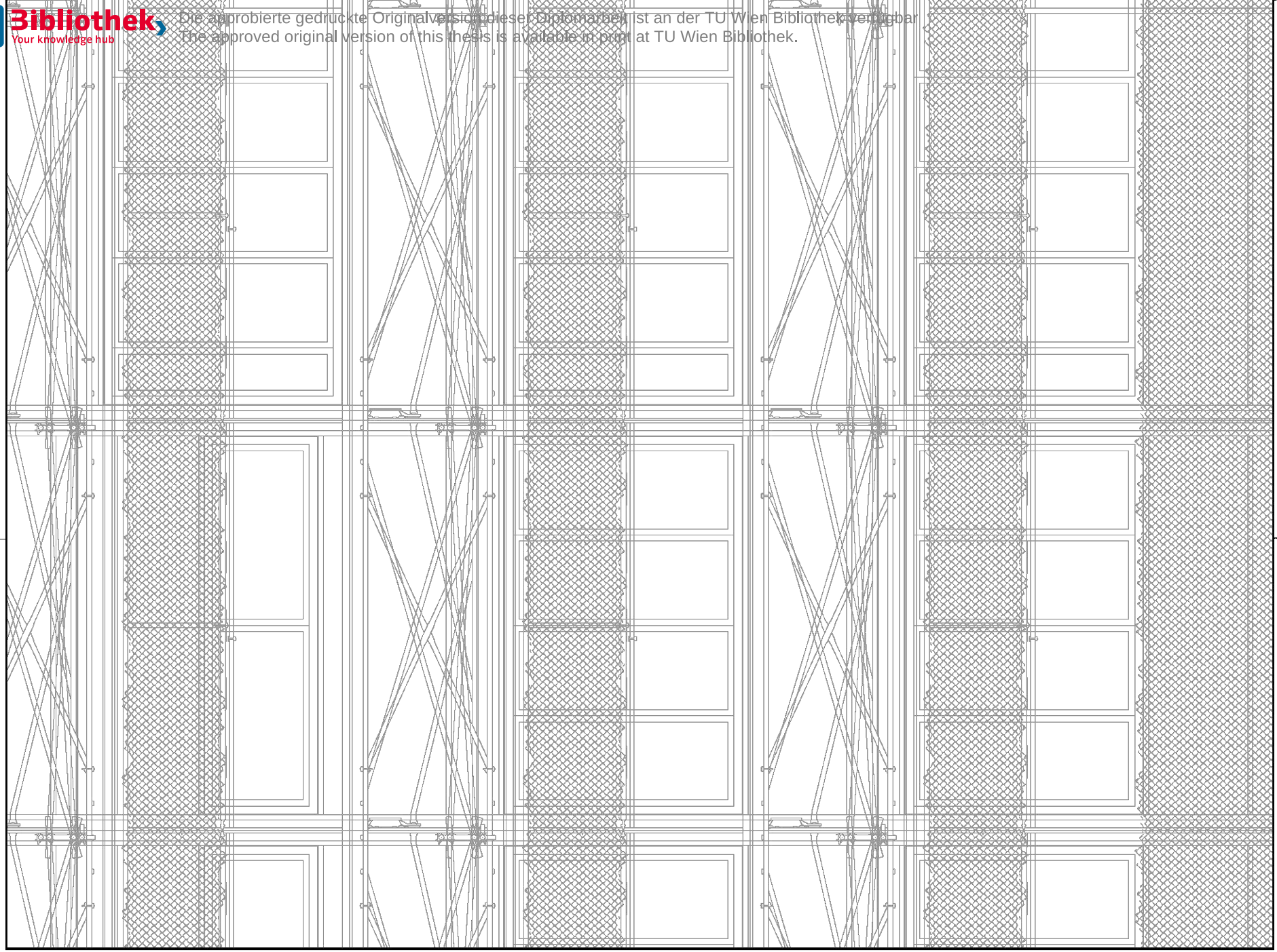




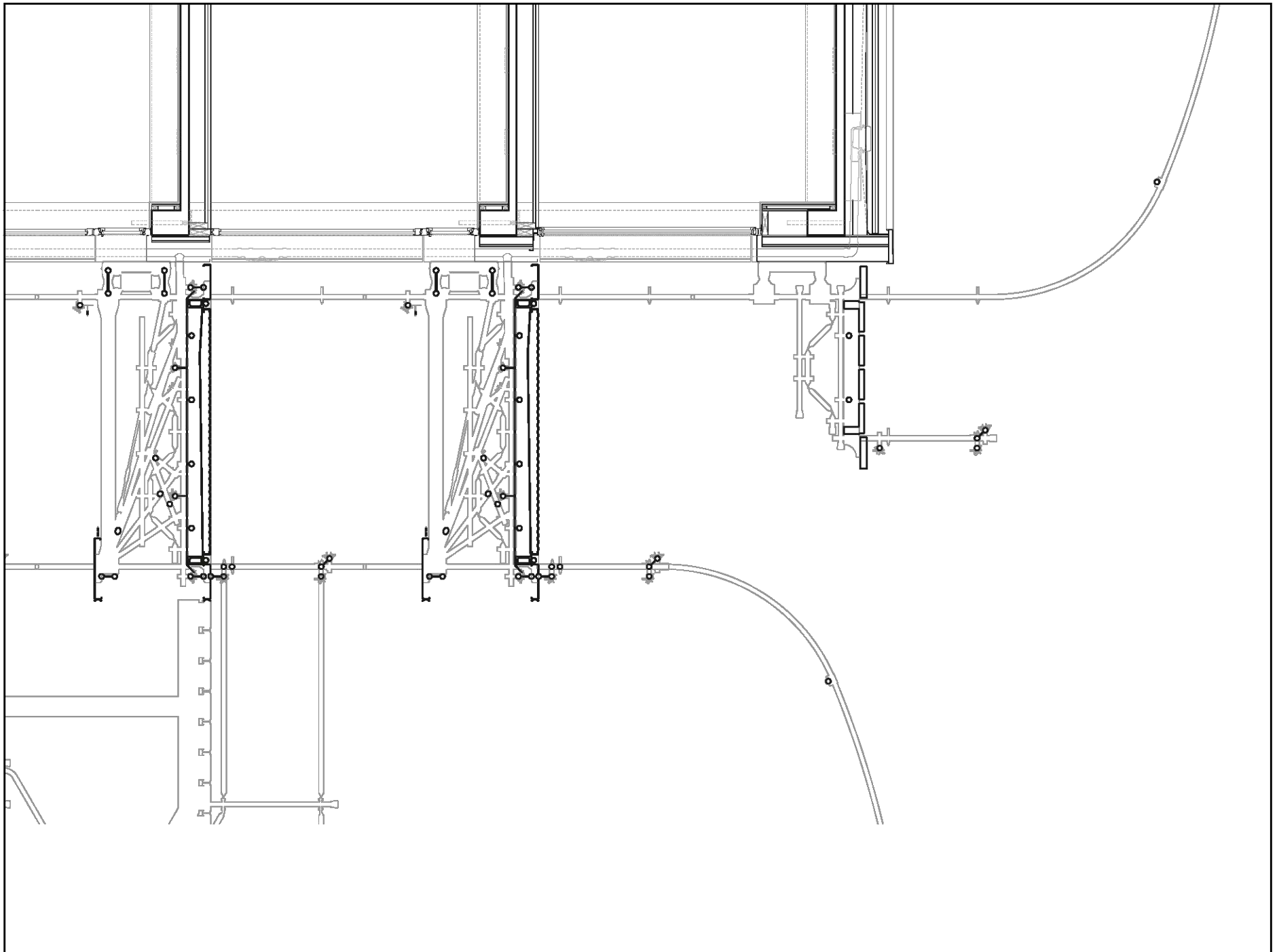


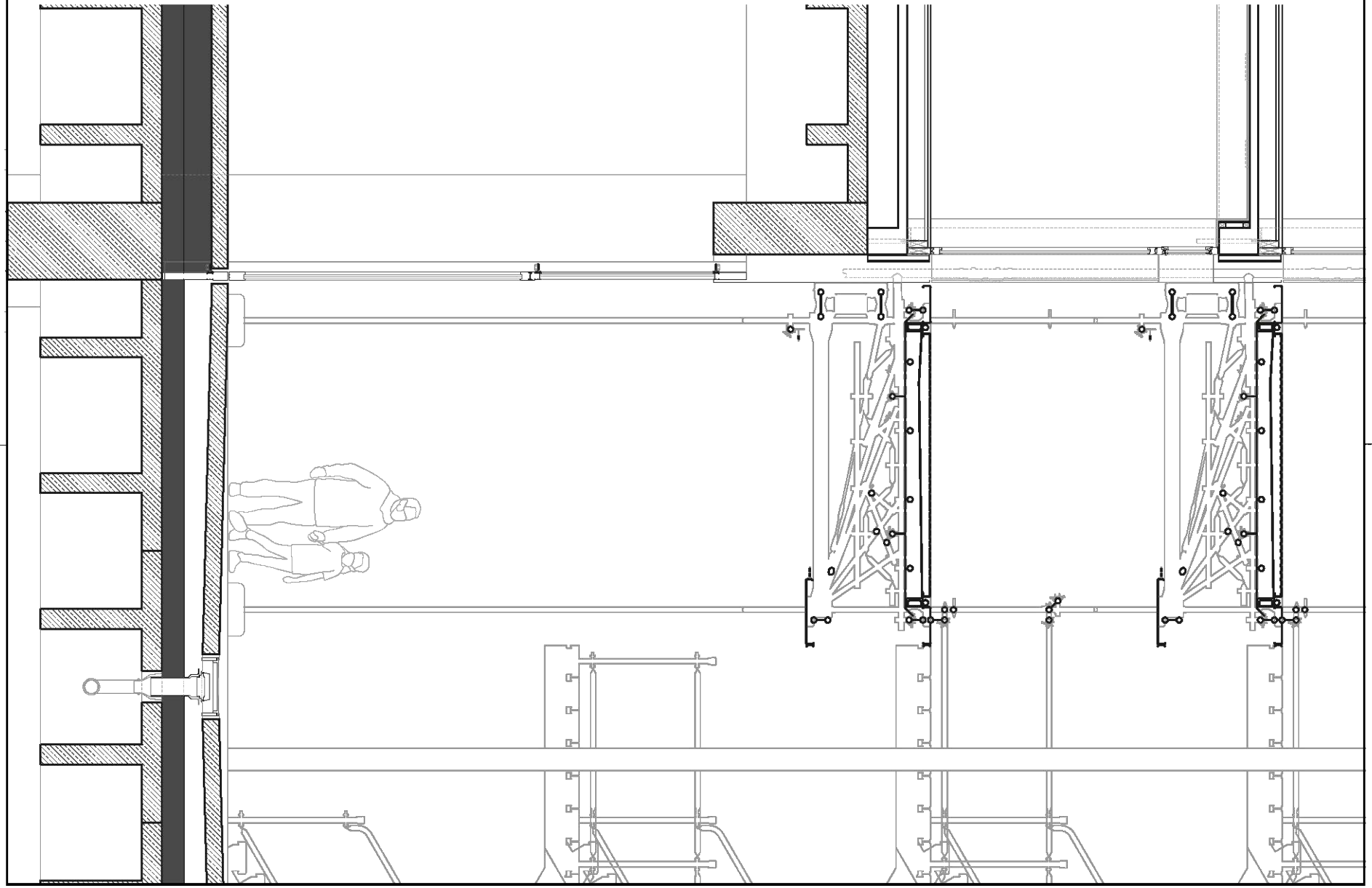
elevation, scale 1:50



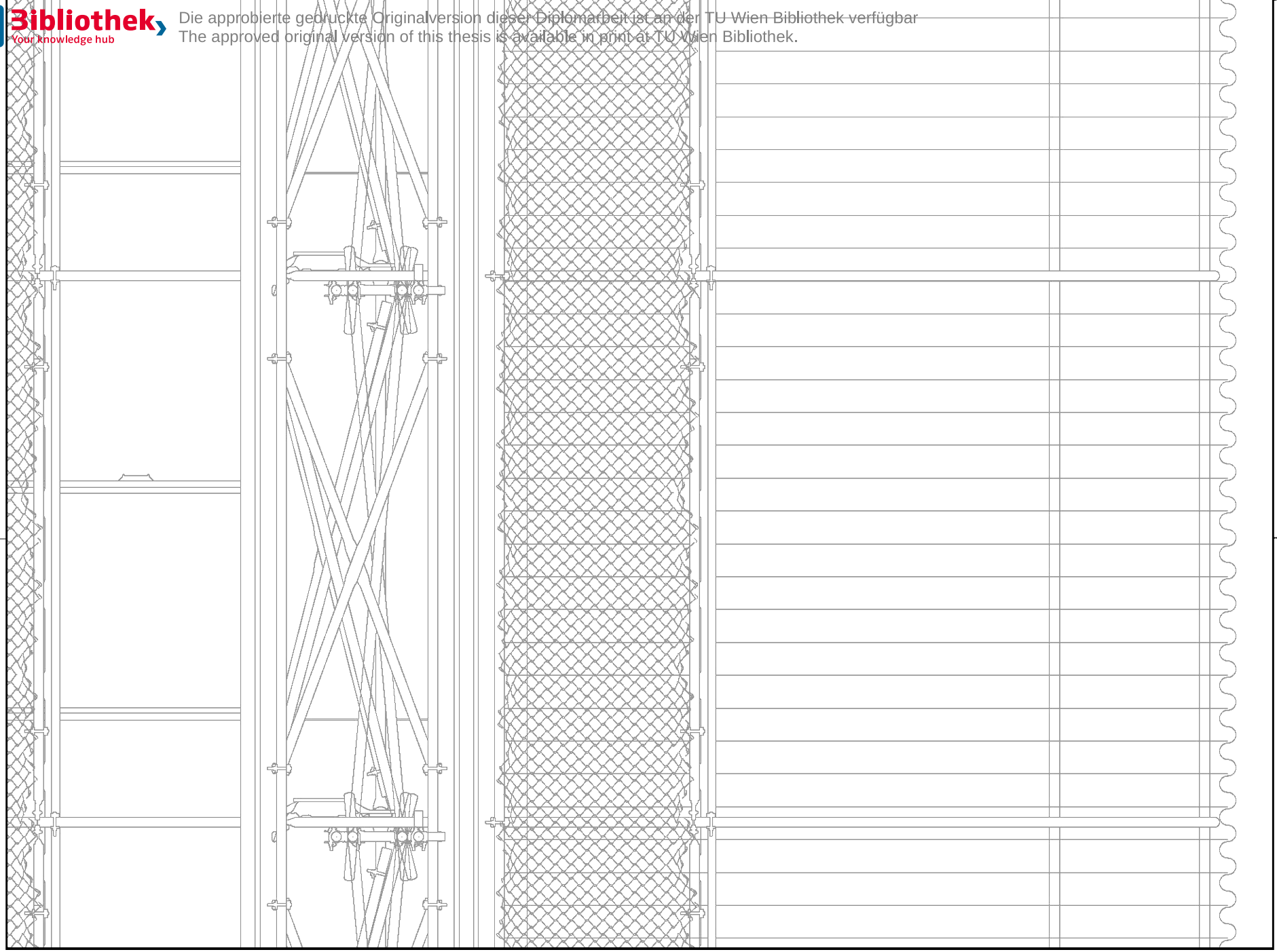


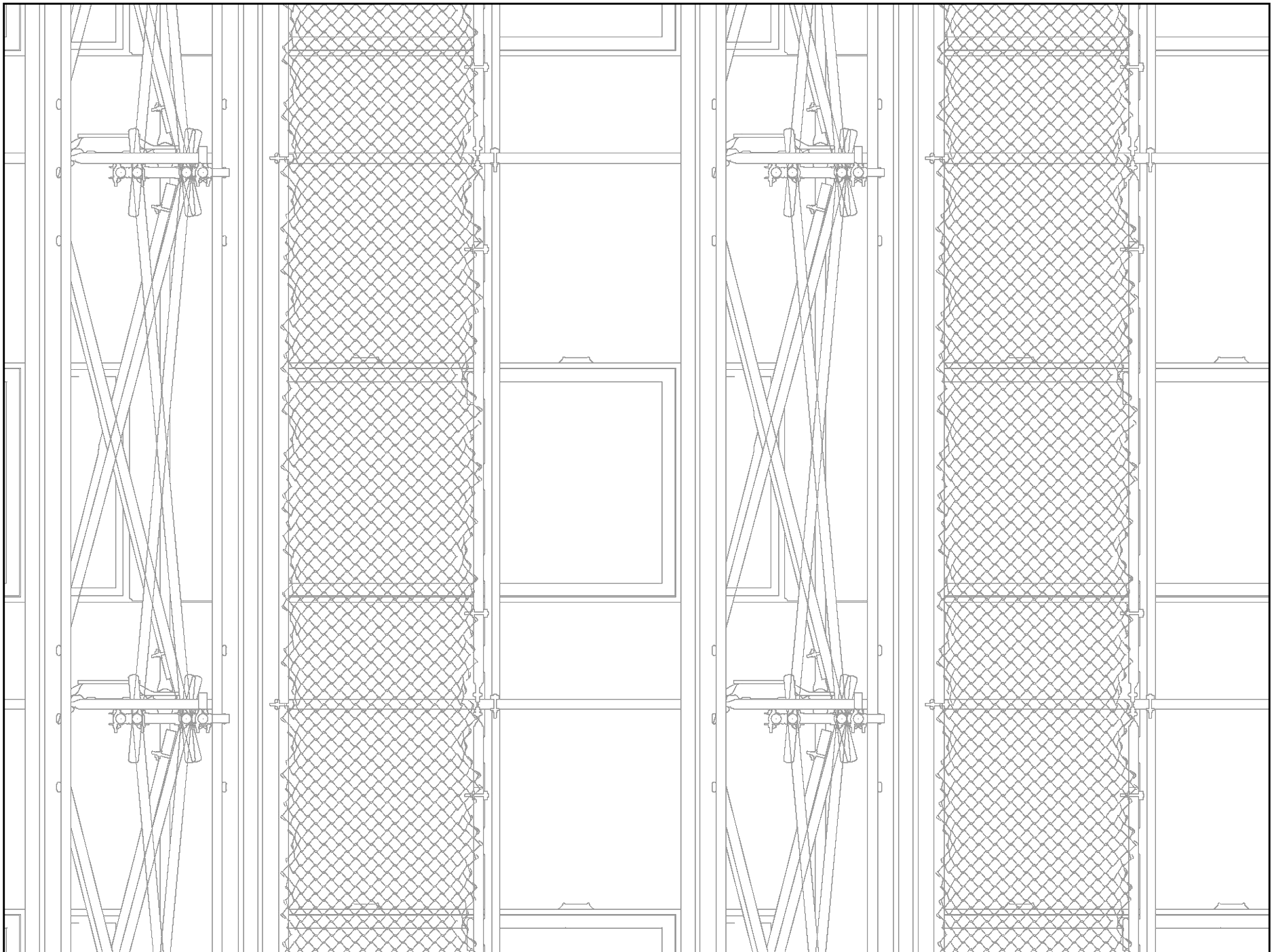
The Project elevation, scale 1:50

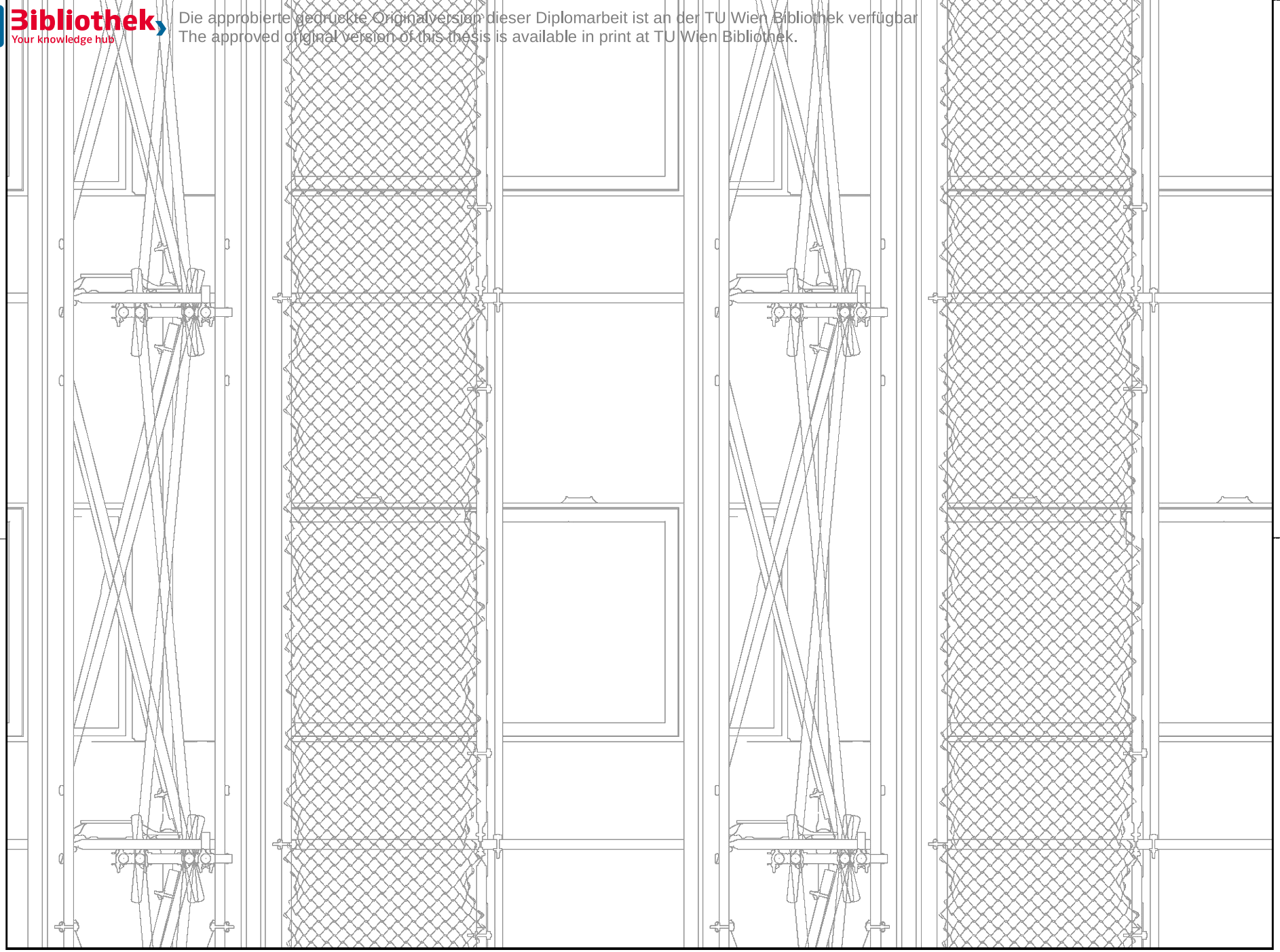


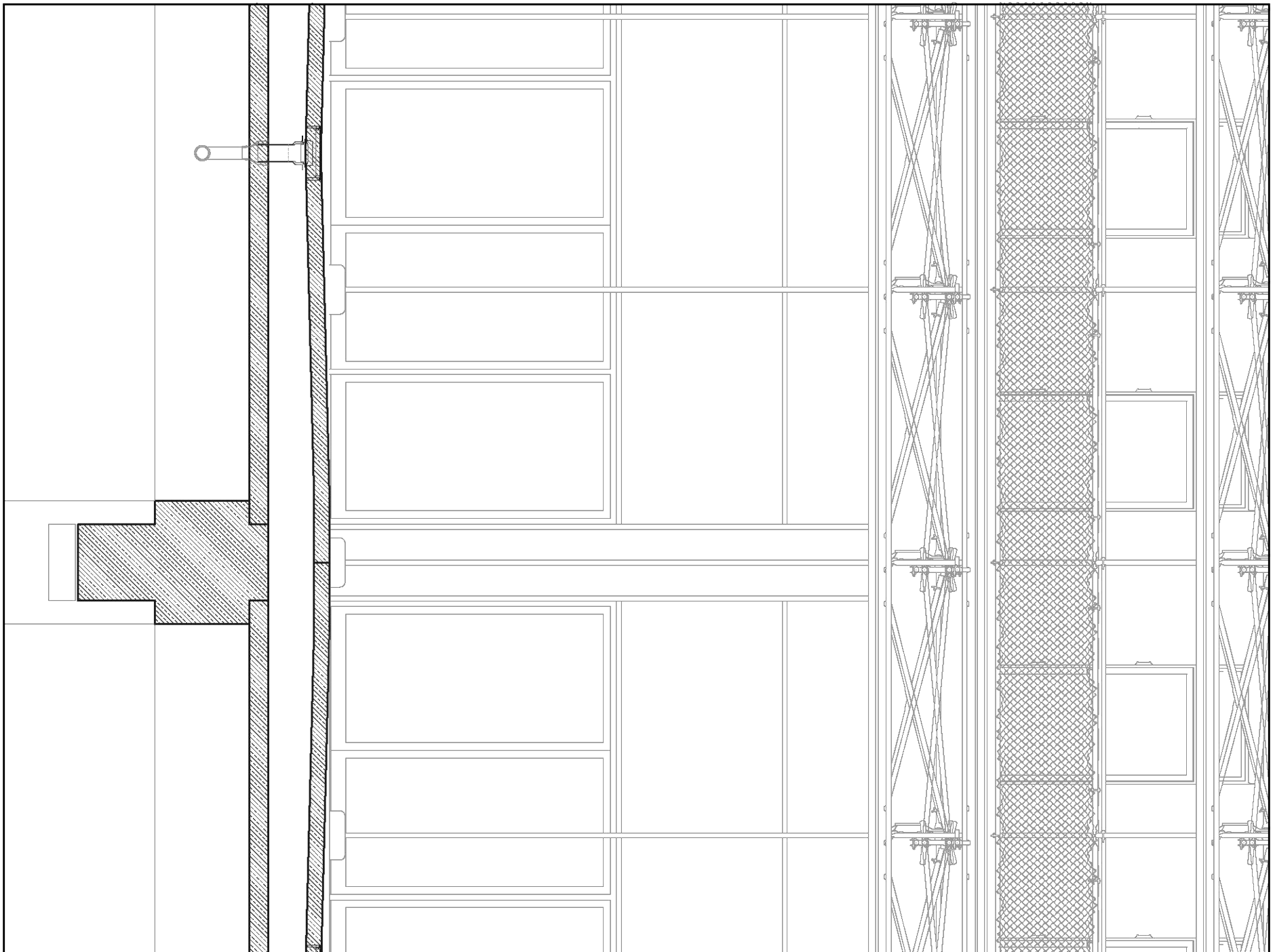


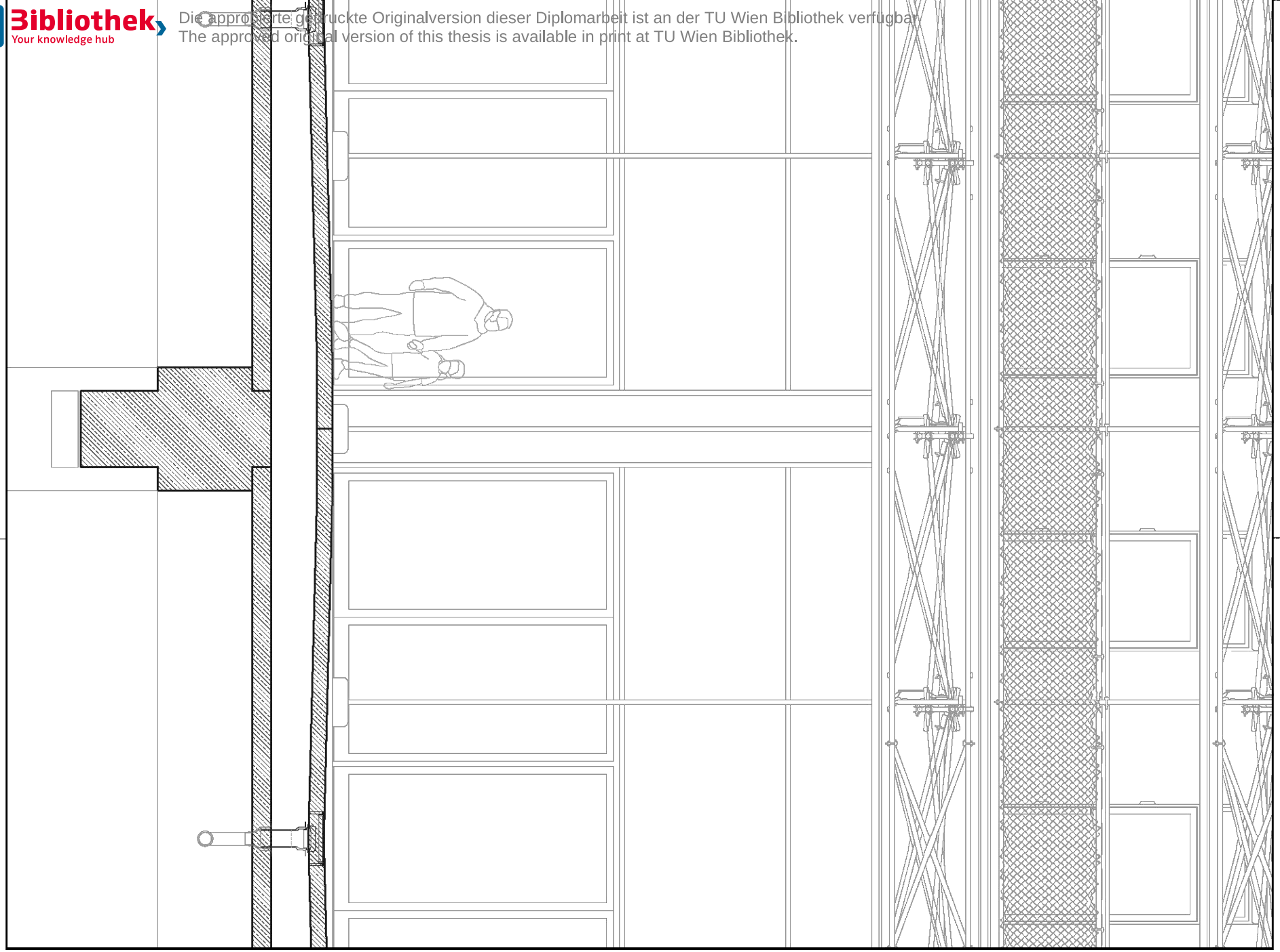
#99 #00 The Project facade section B-B, scale 1:50



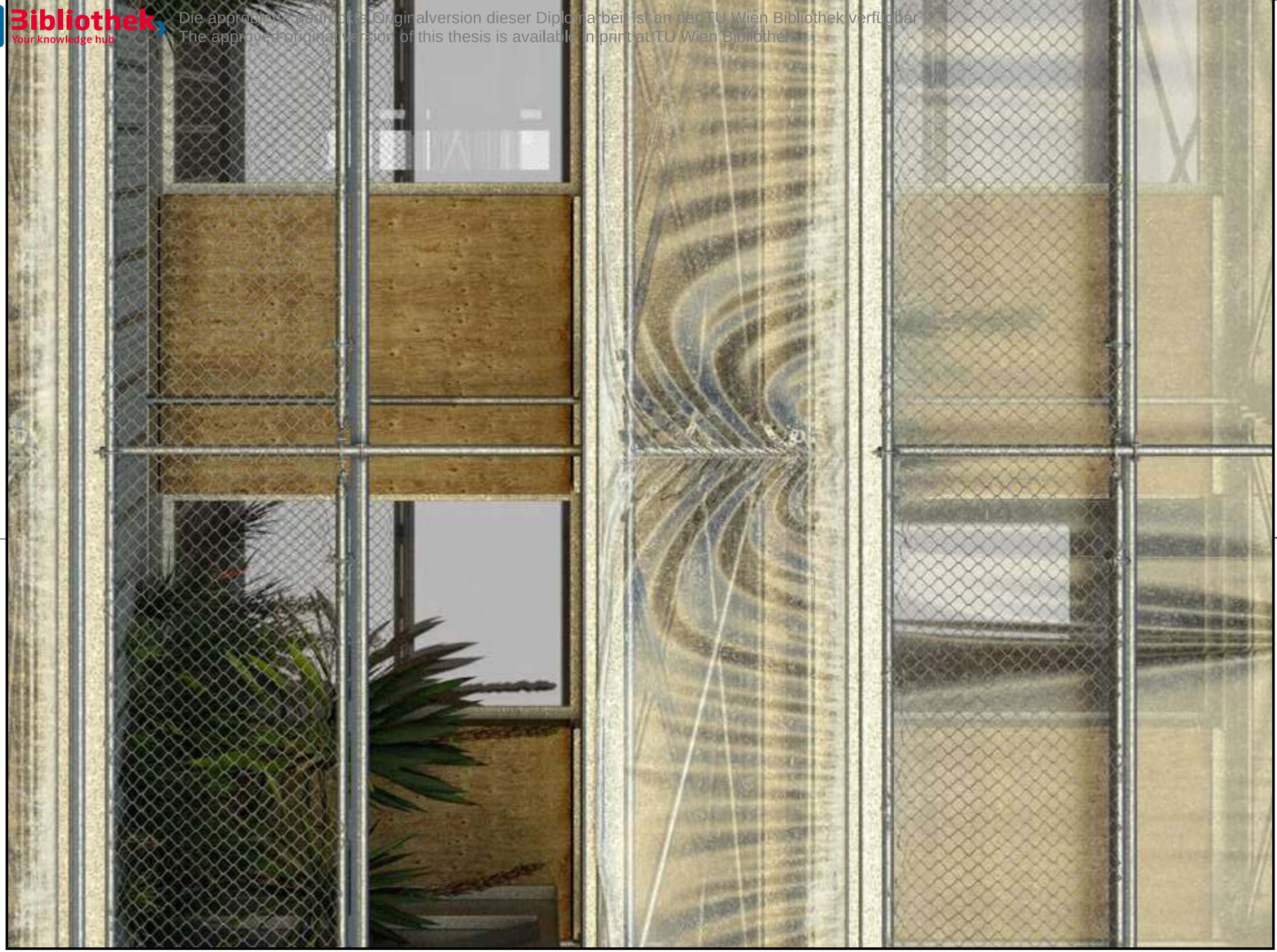


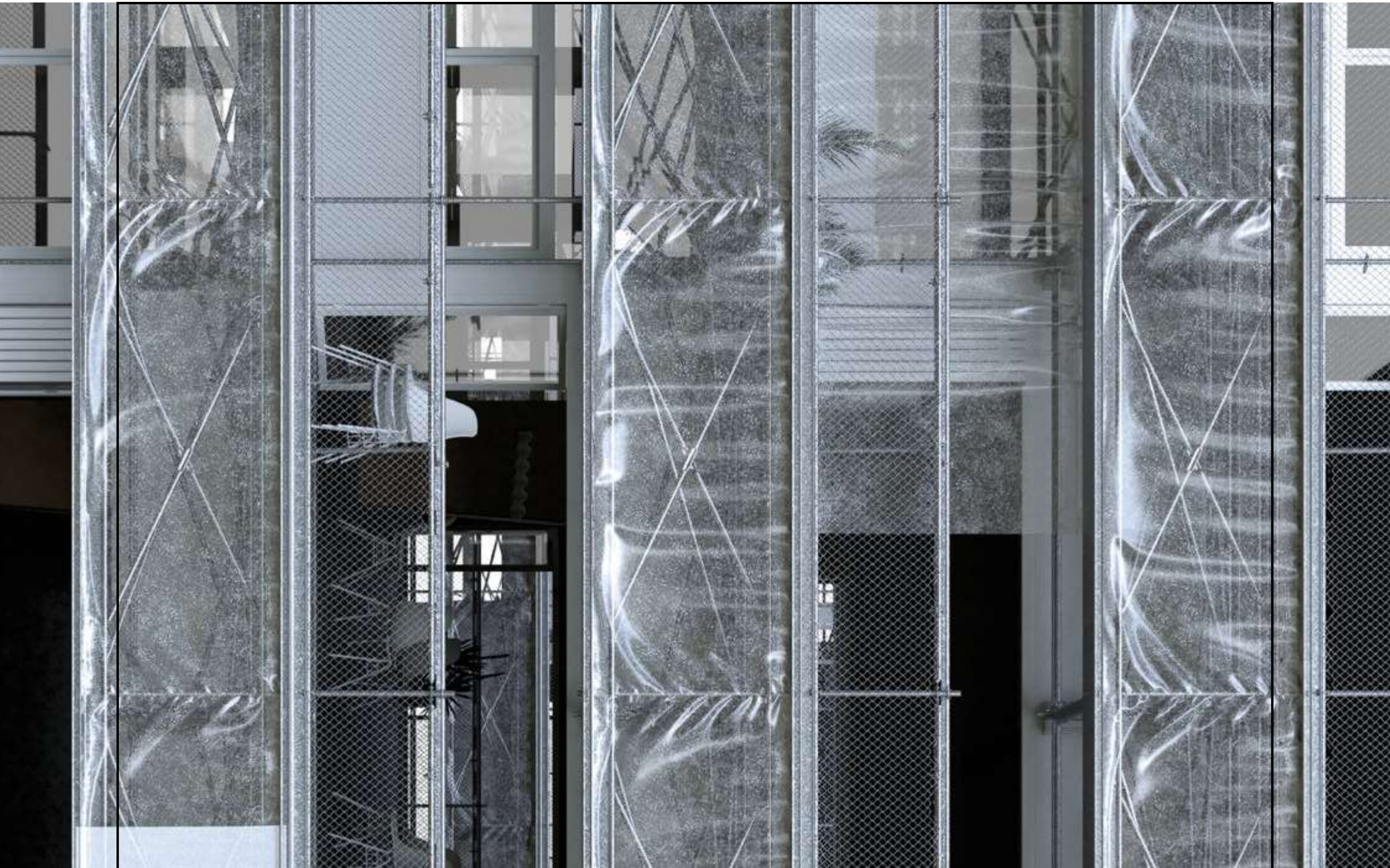






elevation, scale 1:50





7. Appendix

7.1 Bibliography

1.) Introduction

Municipal Department 20 — Energy Planning; Erker, Susanne; Cerveny, Michael; Lutter, Johannes. Wiener Klimafahrplan: Unser Weg zur klimagerechten Stadt. Vienna: Municipal Administration, March 2022. https://www.wien.gv.at/umwelt-klimaschutz/pdf/klima-fahrplan-lang.pdf.

Municipal Department 18 — Urban Development and Planning; Deistler, Julia; Homeier, Ina; Lengauer, Christina; Pangerl, Eva; Rücker, Lena; Lutter, Johannes; Cerveny, Michael; Bartik, Herbert; Hofinger, Johannes; Veigl, Andreas. Smart Klima City Strategie Wien: Der Weg zur Klimamusterstadt. Vienna: Municipal Administration, February 2022. https://smartcity.wien.gv.at/wp-content/uploads/sites/3/2022/03/scwr_klima_2022_web-neu.pdf.

2.) The Polycentric Region

Mitteregger, Mathias; Bruck, Emilia M.; Soteropoulos, Aggelos; Stickler, Andrea; Berger, Martin; Dangschat, Jens S.. AVENUE21. Automatisierter und vernetzter Verkehr: Entwicklungen des urbanen Europa. Berlin: Springer Vieweg, 2020.

Czech, Hermann; Alexander, Christopher; Ishikawa, Sara; Silverstein, Murray (Hg.). Eine Muster-Sprache: Städte, Gebäude, Konstruktion, 2th ed., trans. Hermann Czech. Wien: Löcker, 2011.

Scheuvens, Rudolf; Groh, Stefan; Söpper, Katharina; Mittringer, Kurt. “Wien: polyzentral: Forschungsstudie zur Zentrenentwicklung Wiens.” Werkstattbericht Paper No 158, Municipal Department 18 — Urban Development and Planning, Vienna, 2016. https://www.wien.gv.at/stadtentwicklung/studien/pdf/b008475.pdf.

Municipal Department 18 — Urban Development and Planning; Conrad, Katharina; Bartik, Herbert. “Mittelpunkte des städtischen Lebens: polyzentrales Wien.” Werkstattbericht Paper No 158, Municipal Department 18 — Urban Development and Planning, Vienna, 2019. https://www.wien.gv.at/stadtentwicklung/studien/pdf/b008563.pdf.

Forlati, Silvia; Mann, Andrea; Peer, Christian; Steeruwitz, Lina; Vlay, Bernd; Vögele, Heike; Zechner, Thekla. Mischung: Possible!: Wege zur zukunftsfähigen Nutzungsmischung, 2th ed.. Vienna: Institut für Architektur und Entwerfen, Abteilung Wohnbau und Entwerfen, Technische Universität Wien, 2017. https://smartcities.at/wp-content/uploads/sites/3/Mischung-Possible-Auflage-2-2-5.pdf.

3.) The Productive City

Christopher Alexander. “A City is not a Tree.” in Design After Modernism: Beyond the Object, ed. John Thackara. New York: Thames & Hudson Ltd, 1988.

Municipal Department 18 — Urban Development and Planning; Rosenberger, Michael; Braumann, Alfried; Emrich, Hans. “Produktive Stadt,” (Werkstattbericht Paper No 171, Municipal Department 18 — Urban Development and Planning, Vienna, 2017), 24, https://www.wien.gv.at/stadtentwicklung/studien/pdf/b008500a.pdf.

4.) The Adaptable Building

Stewart Brand. How Buildings Learn: What Happens After They’re Built. New York: Penguin Books, 1995. https://www.kobo.com/at/de/ebook/how-buildings-learn.

Andrea Benze, Julia Gill, and Saskia Hebert, Serieller Wohnungsbau: Standardisierung der Vielfalt. Berlin: Senatsverwaltung für Stadtentwicklung und Umwelt, 2013). https://www.stadtentwicklung.berlin.de/staedtebau/

baukultur/iba/download/studien/IBA-Studie_Serieller_Wohnungsbau.pdf.

Schmidt, Robert, III; Eguchi, Toru; Austin, Simon; Gibb, Alistair. "What Is The Meaning Of Adaptability In The Building Industry?" Conference paper, 16th International Conference on "Open and Sustainable Building": Proceedings of the international conference jointly organized by CIB W104 - Open Building Implementation and TECNALIA, Bilbao, May 2010. https://www.irbnet.de/daten/iconda/CIB17955.pdf.

Richard Roger-Bruno. ""IFD" Systems = Open Building "Plus"." Conference paper, 16th International Conference on "Open and Sustainable Building": Proceedings of the international conference jointly organized by CIB W104 - Open Building Implementation and TECNALIA, Bilbao, May 2010. https://www.irbnet.de/daten/iconda/CIB17955.pdf.

Drexler, Hans; Almeida, Filipa; Deilmann, Marie; Ehling, Frederik; Rehfinger, Anna; Rüdiger, Philip von. Holz: form- und kraftschlüssig: Entwicklung eines Voll-Holz-Bausystems mit form- und kraftschlüssigen geometrischen Verbindungen. Stuttgart: Fraunhofer IRB Verlag, 2020. https://dgj.eu/dgj/wp-content/uploads/2020/03/res022_BBSR_HFKS_Abschlussbericht-LAYOUT-_2020.02.13_pr_5.4.pdf.

online

3.) The Productive City

Eva Winroither. "Analyse: Warum Aspern Blau wählte," diepresse.com, October 13, 2015, https://www.diepresse.com/4842937/analyse-warum-aspern-blau-waehlte. Accessed: 13th May 2022.

4.) The Adaptable Building

"Manifesto Open Building," OpenBuilding.co, accessed May 10, 2022, https://www.openbuilding.co/. Accessed: 10th March 2022.

George Kafka. "Sustainable building, sustainable living: La Borda, Barcelona by Lacol," architectsjournal.co.uk, June 23, 2020, https://www.architectsjournal.co.uk/buildings/sustainable-building-sustainable-living-la-borda-barcelona-by-lacol. Accessed: 15th March 2022.

7.2 List of Figures

7.3 Source code

Listing 1: CreateLinearArrayGhc

```
1   using  Grasshopper.Kernel;
2  using Grasshopper.Kernel.Data;
3  using Grasshopper.Kernel.Types;
4  using Rhino.Geometry;
5
6  using System;
7  using System.Collections.Generic;
8
9
10 namespace FlexibleGrid.Array
11 {
12     public class CreateLinearArrayGhc : GH_Component
13     {
14         /// <summary>
15         /// Initializes a new instance of the MyComponent1 class.
16         /// </summary>
17         public CreateLinearArrayGhc()
18           : base("Array", "Array",
19               "This component provides an array by subdividing lines presented.",
20               "Flexible Grid", "Array")
21         {
22         }
23
24         /// <summary>
25         /// Registers all the input parameters for this component.
26         /// </summary>
27         protected override void RegisterInputParams(GH_Component.GH_InputParamManager
               pManager)
28         {
29             pManager.AddCurveParameter("Rail Curves", "R", "Segmented rail curves
                   which are used to generate an array. " +
30                 "Each linear rail curve has to have a length of its associated width x
                       2, or if shorter, " +
31                 "it has to be divisible by its associated width evenly. ",
                       GH_ParamAccess.list);
32             pManager.AddNumberParameter("Default Width", "D", "Default width of one
                   structural module.", GH_ParamAccess.item);
33             pManager.AddNumberParameter("Maximal Distance Crosswise", "C", "The
                   maximal distance crosswise of which splitting the linear base curve
                   affects adjacent segments.", GH_ParamAccess.item);
34             pManager.AddNumberParameter("Minimum Length", "M", "No linear rail curve
                   is subdivided if one of the resulting halves would be shorter than the
                    factor minimum length x associated width." +
35              "The factor minimum length has to be at least 2.", GH_ParamAccess.item);
36             pManager.AddGenericParameter("Variations Width", "D", "Variations of the
                   width regarding the default structural module.", GH_ParamAccess.list);
37         }
38
39         /// <summary>
40         /// Registers all the output parameters for this component.
41         /// </summary>
42         protected override void RegisterOutputParams(GH_Component.
               GH_OutputParamManager pManager)
43         {
44             pManager.AddCurveParameter("Rail Curves Subdivided", "R", "Rail curves are
                    subdivided according to a set of rules.", GH_ParamAccess.list);
45             pManager.AddPlaneParameter("Linear Array", "L", "Linear array presented as
                    an array of perpendicular frames.", GH_ParamAccess.tree);
46             pManager.AddPlaneParameter("Extracted Frames", "E", "Frames extracted.",
```

```
                GH_ParamAccess.tree);
47          }
48
49          /// <summary>
50          /// This is the method that actually does the work.
51          /// </summary>
52          /// <param name="DA">The DA object is used to retrieve from inputs and store
                in outputs.</param>
53          protected override void SolveInstance(IGH_DataAccess DA)
54          {
55              List<Curve> iLines = new List<Curve>();
56              double iDefaultWidth = double.NaN;
57              double iMaximalDistanceCrosswise = double.NaN;
58              double iMinimumLength = double.NaN;
59              List<GH_ObjectWrapper> iVariationsWidth = new List<GH_ObjectWrapper>();
60
61
62              if (!DA.GetDataList(0, iLines)) { return; }
63              if (!DA.GetData(1, ref iDefaultWidth)) { return; }
64              if (!DA.GetData(2, ref iMaximalDistanceCrosswise)) { return; }
65              if (!DA.GetData(3, ref iMinimumLength)) { return; }
66              DA.GetDataList(4, iVariationsWidth);
67
68
69              var linearArray = new FlexibleGrid.Array.Array
70                  (iLines, iDefaultWidth, iMaximalDistanceCrosswise, iMinimumLength,
                        iVariationsWidth);
71
72
73              // test if each provided segement is valid
74              foreach (Curve pathCrv in iLines)
75              {
76                  if (!pathCrv.IsValid)
77                  {
78                      AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
79                          "At least one segment is not valid.");
80                      return;
81                  }
82              }
83
84
85              // test if paths do intersect
86              if (iLines.Count >= 1)
87              {
88                  FlexibleGrid.Misc.Analyze intersect = new FlexibleGrid.Misc.Analyze();
89                  if (intersect.TestIfCrvsIntersect(iLines))
90                  {
91                      AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
92                          "Parts of rail curves do intersect.");
93                      return;
94                  }
95              }
96
97
98              // test if each line sufficient in its length
99              if (linearArray.Lines.Count >= 1)
100             {
101                 if (!linearArray.TestIfRailCurveIsTooShort())
102                 {
103                     AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
```

```
104                         "At least one linear rail curve is shorter than its associated
                                width x 2 and simultaneously not evenly divisible by its
                                associated width.");
105                     return;
106                 }
107             }
108
109
110             // group lines
111             if (linearArray.Lines.Count >= 1)
112             {
113                 if (!linearArray.GroupParallelLines())
114                 {
115                     AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
116                         "Lines could not be grouped.");
117                     return;
118                 }
119             }
120
121
122             // subdivide lines
123             if (linearArray.LinesParallel.PathCount >= 1)
124             {
125                 if (!linearArray.SubdivideLines())
126                 {
127                     AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
128                         "Lines could not be subdivided.");
129                     return;
130                 }
131             }
132
133
134             // get section points of linear paths
135             if (linearArray.LinesParallel.PathCount >= 1)
136             {
137                 if (!linearArray.GetLinearSectionPts())
138                 {
139                     AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
140                         "Section points could not be retrieved, because at least one
                                line subdivided is not evenly divisible by its associated
                                width.");
141                     return;
142                 }
143             }
144
145
146             // get perp frames
147             if (linearArray.SectionPts.Count >= 1)
148             {
149                 if (!linearArray.GetPerpFrames())
150                 {
151                     AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
152                         "Perp Frames could not be created.");
153                     return;
154                 }
155             }
156
157
158             // get reference points
159             if (linearArray.PerpFrames.PathCount >= 1)
```

```
160             {
161                 if (!linearArray.GetReferencePts())
162                 {
163                     AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
164                        "Exceptions are spotted. two possible reasons: first path
                            segment is too short to be subdivided and has not a
                            divisible length without remainder.");
165                     return;
166                 }
167             }
168 
169 
170             DA.SetDataList(0, linearArray.LinesSubdivided);
171             DA.SetDataTree(1, linearArray.PerpFrames);
172             DA.SetDataTree(2, linearArray.ExtractedFrames);
173 
174         }
175 
176         /// <summary>
177         /// Provides an Icon for the component.
178         /// </summary>
179         protected override System.Drawing.Bitmap Icon
180         {
181             get
182             {
183                 //You can add image files to your project resources and access them
                        like this:
184                 // return Resources.IconForThisComponent;
185                 return null;
186             }
187         }
188 
189         /// <summary>
190         /// Gets the unique ID for this component. Do not change this ID after release
                .
191         /// </summary>
192         public override Guid ComponentGuid
193         {
194             get { return new Guid("419fdf6c-96b0-462f-884b-fcb37619c778"); }
195         }
196     }
197 }
```

Listing 2: VaryWidthStructuralModuleGhc

```
using System;
using System.Collections.Generic;

using Grasshopper.Kernel;
using Rhino.Geometry;

namespace FlexibleGrid.Array
{
    public class VaryWidthStructuralModuleGhc : GH_Component
    {
        /// <summary>
        /// Initializes a new instance of the MyComponent1 class.
        /// </summary>
        public VaryWidthStructuralModuleGhc()
          : base("Varying Width", "Varying Width",
              "This component introduces a variation of a structural modules width.",
              "Flexible Grid", "Array")
        {
        }

        /// <summary>
        /// Registers all the input parameters for this component.
        /// </summary>
        protected override void RegisterInputParams(GH_Component.GH_InputParamManager
            pManager)
        {
            pManager.AddNumberParameter("Width", "W", "Width of one structural module.
                ", GH_ParamAccess.item);
            pManager.AddCurveParameter("Boundaries", "B", "Planar polyline as
                delimitation for a variation of the default width of one structural
                module.", GH_ParamAccess.list);
        }

        /// <summary>
        /// Registers all the output parameters for this component.
        /// </summary>
        protected override void RegisterOutputParams(GH_Component.
            GH_OutputParamManager pManager)
        {
            pManager.AddGenericParameter("Variation Width", "V", "Deviated width of
                the default width of one structural module.", GH_ParamAccess.item);
        }

        /// <summary>
        /// This is the method that actually does the work.
        /// </summary>
        /// <param name="DA">The DA object is used to retrieve from inputs and store
            in outputs.</param>
        protected override void SolveInstance(IGH_DataAccess DA)
        {
            var iWidth = double.NaN;
            List<Rhino.Geometry.Curve> iBoundaryCrvs = new List<Rhino.Geometry.Curve
                >();
            if (!DA.GetData(0, ref iWidth)) { return; }
            if (!DA.GetDataList(1, iBoundaryCrvs)) { return; }

            // test if each boundary curve is planar and closed
            foreach (Curve boundary in iBoundaryCrvs)
            {
```

```
52                  if (!boundary.IsClosed || !boundary.IsPlanar())
53                  {
54                      AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
55                              "At least one boundary curve is not closed or planar.");
56                      return;
57                  }
58
59              }
60
61              // define size array
62              int countCorners = 0;
63              foreach (Curve curve in iBoundaryCrvs)
64              {
65                  List<Point3d> cornerPts = new List<Point3d>();
66                  var kinks = new FlexibleGrid.Misc.Analyze();
67                  cornerPts = kinks.GetDiscontinuitys(curve);
68                  countCorners += cornerPts.Count;
69                  countCorners += 1;
70              }
71
72              /*_____________________________________
73               * COMPOSITION OF ABSTRACT DATA ARRAY
74               * _____________________________________
75               *  1.)             2.)             3.)
76               *  width       int.MaxValue    int.MaxValue
77               *  position x      position y      position z      => coordinates of
                     first corner point
78               *  position x      position y      position z      => coordinates of
                     second corner point
79               *  ...
80               *  int.MaxValue    int.MaxValue    int.MaxValue    => marking for the
                     start of a new boundary curve
81               *  position x      position y      position z      => coordinates of
                     first corner point
82               *  position x      position y      position z      => coordinates of
                     second corner point
83               *  ...
84               *  int.MaxValue    int.MaxValue    int.MaxValue    => end of array
85              */
86
87              // initialize array
88              var abstractData = new double[countCorners + 1, 3];
89              for (int i = 0; i < abstractData.GetLength(0); i++)
90              {
91                  for (int j = 0; j < abstractData.GetLength(1); j++)
92                  {
93                      abstractData[i, j] = int.MaxValue;
94                  }
95              }
96
97              // pass on span width
98              abstractData[0, 0] = iWidth;
99
100             // pass on corner points of boundary curve
101
102             int u = 0;
103             foreach (Curve curve in iBoundaryCrvs)
104             {
105                 List<Point3d> cornerPts = new List<Point3d>();
106                 var kinks = new FlexibleGrid.Misc.Analyze();
```

```
107                 cornerPts = kinks.GetDiscontinuitys(curve);
108
109                 for (int i = 0; i < cornerPts.Count; i++)
110                 {
111                     abstractData[u + 1, 0] = cornerPts[i].X;
112                     abstractData[u + 1, 1] = cornerPts[i].Y;
113                     abstractData[u + 1, 2] = cornerPts[i].Z;
114                     u++;
115                 }
116
117                 // empty line as marker for the begin of a new boundary curve
118                 u++;
119             }
120
121             // output abstract data array
122             var oAbstractData = new Grasshopper.Kernel.Types.GH_ObjectWrapper(
                    abstractData);
123             DA.SetData(0, oAbstractData);
124         }
125
126         /// <summary>
127         /// Provides an Icon for the component.
128         /// </summary>
129         protected override System.Drawing.Bitmap Icon
130         {
131             get
132             {
133                 //You can add image files to your project resources and access them
                        like this:
134                 // return Resources.IconForThisComponent;
135                 return null;
136             }
137         }
138
139         /// <summary>
140         /// Gets the unique ID for this component. Do not change this ID after release
                .
141         /// </summary>
142         public override Guid ComponentGuid
143         {
144             get { return new Guid("6886c783-03e0-4d7c-b370-05dd259a8f84"); }
145         }
146     }
147 }
```

Listing 3: Array

```
1  using  Grasshopper.Kernel;
2 using Grasshopper.Kernel.Data;
3 using Grasshopper.Kernel.Types;
4 using Rhino.Geometry;
5
6 using System;
7 using System.Collections.Generic;
8
9 namespace FlexibleGrid.Array
10 {
11     class Array
12     {
13         private double _tolerance = 0.00001;
14         public GH_Structure<GH_Plane> ExtractedFrames { get; private set;  }
15         public GH_Structure<GH_Curve> LinesParallel { get; private set; }
16         public List<Curve> LinesSubdivided { get; private set; }
17         public List<Curve> Lines { get; private set; }
18         public GH_Structure<GH_Plane> PerpFrames { get; private set; }
19         public List<Point3d> SectionPts { get; private set; }
20         public double DefaultWidth { get; private set; }
21         public List<double> VariationsWidth { get; private set; }
22         public List<Rhino.Geometry.PolylineCurve> VaryingWidthsBoundaryCrvs { get;
               private set; }
23         public double MaximalDistanceCrosswise { get; private set; }
24         public double MinimumLength { get; private set; }
25         FlexibleGrid.Array.ArrayMethods library = new FlexibleGrid.Array.ArrayMethods
               ();
26
27
28         public Array(List<Curve> lines, double defaultWidth, double
               maximalDistanceCrosswise, double minimumLength, List<GH_ObjectWrapper>
               variationsWidth)
29         {
30             this.DefaultWidth = defaultWidth;
31             ExtractedFrames = new GH_Structure<GH_Plane>();
32             this.Lines = lines;
33             LinesSubdivided = new List<Curve>();
34             LinesParallel = new GH_Structure<GH_Curve>();
35             PerpFrames = new GH_Structure<GH_Plane>();
36             SectionPts = new List<Point3d>();
37             VariationsWidth = new List<double>();
38             VaryingWidthsBoundaryCrvs = new List<PolylineCurve>();
39             this.MaximalDistanceCrosswise = maximalDistanceCrosswise;
40             this.MinimumLength = minimumLength;
41
42             // unwrap array
43             foreach (GH_ObjectWrapper variation in variationsWidth)
44             {
45                 var dataAbstract = variation.Value as double[,];
46
47                 List<Point3d> cornerPts = new List<Point3d>();
48                 for (int i = 1; i < dataAbstract.GetLength(0); i++)
49                 {
50                     if (dataAbstract[i, 0] != int.MaxValue && dataAbstract[i, 1] !=
                           int.MaxValue)
51                     {
52                         cornerPts.Add(new Point3d(dataAbstract[i, 0], dataAbstract[i,
                               1], dataAbstract[i, 2]));
53                     }
```

```
54
55                  else
56                  {
57                      VariationsWidth.Add(dataAbstract[0, 0]);
58                      VaryingWidthsBoundaryCrvs.Add(new Rhino.Geometry.PolylineCurve
                            (cornerPts));
59                      cornerPts.Clear();
60                  }
61              }
62          }
63      }
64
65
66      public bool TestIfRailCurveIsTooShort()
67      {
68          for (int i = 0; i < Lines.Count; i++)
69          {
70              double width = library.GetAssociatedWidth(Lines[i], DefaultWidth,
                    VaryingWidthsBoundaryCrvs, VariationsWidth);
71              double crvLength = Lines[i].GetLength();
72              double remainder = Math.Round(crvLength % width, 5);
73              if (crvLength < MinimumLength * width)
74              {
75                  if  (remainder != width && remainder != 0)
76                  {
77                      return false;
78                  }
79              }
80          }
81          return true;
82      }
83
84
85      public bool GroupParallelLines()
86      {
87          try
88          {
89              List<Vector3d> sharedTangents = new List<Vector3d>();
90              for (int i = 0; i < Lines.Count; i++)
91              {
92                  Vector3d tangent = Lines[i].TangentAtStart;
93                  tangent.Unitize();
94
95                  bool existing = false;
96                  for (int j = 0; j < sharedTangents.Count; j++)
97                  {
98                      Vector3d crossProduct = new Vector3d(sharedTangents[j].Y *
                            tangent.Z - sharedTangents[j].Z * tangent.Y,
99                          sharedTangents[j].Z * tangent.X - sharedTangents[j].X *
                                tangent.Z,
100                         sharedTangents[j].X * tangent.Y - sharedTangents[j].Y *
                                tangent.X);
101                     if (crossProduct.Length < _tolerance)
102                     {
103                         LinesParallel.Append(new GH_Curve(Lines[i]), new GH_Path(j
                                ));
104                         existing = true;
105                         break;
106                     }
107                 }
```

```
108
109                     if (!existing)
110                     {
111                         LinesParallel.Append(new GH_Curve(Lines[i]), new GH_Path(
                                sharedTangents.Count));
112                         sharedTangents.Add(tangent);
113                     }
114                 }
115                 return true;
116             }
117             catch
118             { return false; }
119         }
120
121
122         // subdivide lines
123
124         /*
125         Each iteration, a line is picked to begin the subdivision process with.
126         This line has the highest position in the z-direction and is the shortest
                amongst all lines sharing this position.
127         After splitting this line at its midpoint, each half is extended
128         by the remainder of its length divided by its associated width, and a
                perpendicular frame is generated.
129         Now each remaining line is subdivided if there is an intersection event with
                the perpendicular frame.
130         If so, the line is split at the point of intersection
131         and is extended by the same amount as the line picked to start the subdivision
                 process with.
132         Further rules will be introduced.
133         */
134         public bool SubdivideLines()
135         {
136             try
137             {
138                 for (int i = 0; i < LinesParallel.PathCount; i++)
139                 {
140                     List<Curve> linesToProcess = new List<Curve>();
141
142                     var list = LinesParallel.Branches[i];
143                     for (int j = 0; j < list.Count; j++)
144                     {
145                         linesToProcess.Add(list[j].Value);
146                     }
147
148                     while (linesToProcess.Count != 0)
149                     {
150                         Curve lineToStartSubdivisionProcessWith = null;
151                         Plane planeOfIntersection;
152
153                         // remove lines that have been processed
154                         for (int u = linesToProcess.Count - 1; u >= 0; u--)
155                         {
156                             double width = library.GetAssociatedWidth(linesToProcess[u
                                    ], DefaultWidth, VaryingWidthsBoundaryCrvs,
                                    VariationsWidth);
157
158                             if (Math.Round(linesToProcess[u].GetLength() % width, 5)
                                    == 0
159                                 || Math.Round(linesToProcess[u].GetLength() % width,
```

```
                5) == width)
160             {
161                 LinesSubdivided.Add(linesToProcess[u]);
162                 linesToProcess.RemoveAt(u);
163                 if (linesToProcess.Count == 0) goto Finish;
164             }
165         }
166
167         // sort rail curves in descending order by their position
                along the z-axis
168         FlexibleGrid.Misc.Sort sort = new FlexibleGrid.Misc.Sort();
169         List<double> zPos = new List<double>();
170         foreach (Curve line in linesToProcess)
171         {
172             zPos.Add(line.PointAtStart.Z);
173         }
174         sort.BubbleSort(ref zPos, ref linesToProcess);
175         linesToProcess.Reverse();
176         lineToStartSubdivisionProcessWith = linesToProcess[0];
177
178         // rail curve at index 0 is the rail curve to start the
                subdivision process with,
179         // except a shorter rail curve at the same position along the
                z-axis can be found
180         bool shorterLineToStartSubdivisionProcessWithFound = false;
181         int x = 0;
182         if (linesToProcess.Count > 1)
183         {
184             for (int u = 1; u < linesToProcess.Count; u++)
185             {
186                 if (linesToProcess[u].GetLength() < linesToProcess[0].
                        GetLength()
187                     && Math.Round(linesToProcess[u].PointAtStart.Z
                            , 5) == Math.Round(linesToProcess[0].
                            PointAtStart.Z, 5))
188                 {
189                     x = u;
190                     lineToStartSubdivisionProcessWith = linesToProcess
                            [u];
191                     shorterLineToStartSubdivisionProcessWithFound =
                            true;
192                 }
193             }
194         }
195         if (!shorterLineToStartSubdivisionProcessWithFound)
                linesToProcess.RemoveAt(0);
196         if (shorterLineToStartSubdivisionProcessWithFound)
                linesToProcess.RemoveAt(x);
197
198         // get plane of intersection at the midpoint of picked rail
                curve to start subdivision process with
199         double[] t = lineToStartSubdivisionProcessWith.DivideByCount
                (2, false);
200         Point3d midpoint = lineToStartSubdivisionProcessWith.PointAt(t
                [0]);
201         Vector3d tangent = new Vector3d(
                lineToStartSubdivisionProcessWith.TangentAt(t[0]));
202         planeOfIntersection = new Plane(midpoint, tangent);
203
204         // start subdivision process
```

```
205                         List<Curve> linesProcessed = new List<Curve>();
206                         List<Curve> linesSubdivided = new List<Curve>();
207
208                         linesProcessed = library.SubdivideLines(
                                lineToStartSubdivisionProcessWith, linesToProcess, ref
                                linesSubdivided, planeOfIntersection,
209                             MaximalDistanceCrosswise, MinimumLength, DefaultWidth,
                                    VaryingWidthsBoundaryCrvs, VariationsWidth);
210
211                         linesToProcess.Clear();
212                         foreach (Curve line in linesSubdivided) LinesSubdivided.Add(
                                line);
213                         foreach (Curve line in linesProcessed) linesToProcess.Add(line
                                );
214                     }
215                 }
216             Finish:
217                 return true;
218             }
219             catch { return false; }
220         }
221
222
223         public bool GetLinearSectionPts()
224         {
225             try
226             {
227                 for (int i = 0; i < LinesSubdivided.Count; i++)
228                 {
229                     // get width
230                     double width = library.GetAssociatedWidth(LinesSubdivided[i],
                            DefaultWidth, VaryingWidthsBoundaryCrvs, VariationsWidth);
231                     // test if curve length is evenly divisible by its associated
                            width
232                     if (LinesSubdivided[i].GetLength() % width < _tolerance
233                         || Math.Round(LinesSubdivided[i].GetLength() % width, 5) ==
                                width)
234                     {
235                         double[] t = LinesSubdivided[i].DivideByCount((int)Math.Round
                                ((LinesSubdivided[i].GetLength() / width)), true);
236                         foreach (double parameter in t) SectionPts.Add(LinesSubdivided
                                [i].PointAt(parameter));
237                     }
238                     else return false;
239                 }
240                 return true;
241             }
242             catch { return false; }
243         }
244
245
246         public bool GetPerpFrames()
247         {
248             try
249             {
250                 for (int i = 0; i < Lines.Count; i++)
251                 {
252                     // extract section points which are lying on the path curve
253                     List<Point3d> extractedSectionPts = new List<Point3d>();
254                     for (int j = 0; j < SectionPts.Count; j++)
```

```
255                     {
256                         double t;
257                         Lines[i].ClosestPoint(SectionPts[j], out t);
258                         if (SectionPts[j].DistanceTo(Lines[i].PointAt(t)) < _tolerance
                                ) extractedSectionPts.Add(SectionPts[j]);
259                     }
260
261                     if (extractedSectionPts.Count == 0) return false;
262
263                     // delete duplicates
264                     Restart:
265                     for (int u = 0; u < extractedSectionPts.Count - 1; u++)
266                     {
267                         for (int v = u + 1; v < extractedSectionPts.Count; v++)
268                         {
269                             if (extractedSectionPts[u].DistanceTo(extractedSectionPts[
                                    v]) < _tolerance)
270                             {
271                                 extractedSectionPts.RemoveAt(v);
272                                 goto Restart;
273                             }
274                         }
275                     }
276
277                     // sort points
278                     List<double> crvParameters = new List<double>();
279                     foreach (Point3d pt in extractedSectionPts)
280                     {
281                         double temp;
282                         Lines[i].ClosestPoint(pt, out temp);
283                         crvParameters.Add(temp);
284                     }
285
286                     FlexibleGrid.Misc.Sort sort = new FlexibleGrid.Misc.Sort();
287                     sort.BubbleSort(ref crvParameters);
288
289                     // get perp frames
290                     if (true)
291                     {
292                         GH_Path temp = new GH_Path(i);
293                         foreach (double t in crvParameters)
294                         {
295                             Point3d origin = new Point3d(Lines[i].PointAt(t));
296                             Vector3d tangent = new Vector3d(Lines[i].TangentAt(t));
297                             if (tangent.Rotate(90 * (Math.PI / 180), new Vector3d(0,
                                    0, 1)))
298                             {
299                                 PerpFrames.Append(new GH_Plane(new Plane(origin,
                                        tangent, new Vector3d(0, 0, 1))), temp);
300                             }
301                         }
302                     }
303                 }
304                 return true;
305             }
306             catch { return false; }
307         }
308
309
310         public bool GetReferencePts()
```

```
311         {
312             try
313             {
314                 for (int i = 0; i < PerpFrames.PathCount; i++)
315                 {
316                     List<Plane> extractedFrames = new List<Plane>();
317                     List<Plane> PlaneArray = new List<Plane>();
318                     var temp = PerpFrames.Branches[i];
319 
320                     foreach (GH_Plane plane in temp)
321                     {
322                         Plane rc_plane = new Plane();
323                         if (GH_Convert.ToPlane(plane, ref rc_plane, GH_Conversion.Both
                                ))
324                             PlaneArray.Add(rc_plane);
325                     }
326 
327                     // get for branch existing variations of width
328                     // get associated rail curve
329                     List<double> existingVariationsWidth = new List<double>();
330                     Curve railCurve = Lines[i];
331                     double[] t = railCurve.DivideByCount(500, true);
332                     for (int j = 0; j < t.GetLength(0); j++)
333                     {
334                         Point3d ptToTest = railCurve.PointAt(t[j]);
335                         double width = library.GetAssociatedWidth(ptToTest,
                                DefaultWidth, VaryingWidthsBoundaryCrvs, VariationsWidth);
336 
337                         // test if width already exists
338                         // if not add to list
339                         bool exists = false;
340                         foreach (double storedWidth in existingVariationsWidth)
341                         {
342                             if (width == storedWidth) exists = true;
343                         }
344                         if (!exists) existingVariationsWidth.Add(width);
345                     }
346 
347                     extractedFrames = library.GetReferencePts(PlaneArray,
                            existingVariationsWidth);
348 
349                     // reinsert output
350                     // convert
351                     GH_Path path = new GH_Path(i);
352                     List<GH_Plane> convertedFrames = new List<GH_Plane>();
353                     foreach (Plane plane in extractedFrames) convertedFrames.Add(new
                            GH_Plane(plane));
354                     ExtractedFrames.AppendRange(convertedFrames, path);
355                 }
356                 return true;
357             }
358             catch { return false; }
359         }
360 
361 
362     }
363 }
```

Listing 4: ArrayMethods

```
1  using  Grasshopper.Kernel;
2 using Grasshopper.Kernel.Data;
3 using Grasshopper.Kernel.Types;
4 using Rhino.Geometry;
5
6 using System;
7 using System.Collections.Generic;
8
9 namespace FlexibleGrid.Array
10 {
11     class ArrayMethods
12     {
13         private double _tolerance = 0.00001;
14
15
16         public double GetAssociatedWidth(Curve railCurveToTest, double defaultWidth,
               List<PolylineCurve> varyingWidthBoundaries, List<double> variationsWidth)
17         {
18             if (varyingWidthBoundaries.Count > 0)
19             {
20                 double[] t = railCurveToTest.DivideByCount(2, false);
21                 Point3d midPt = railCurveToTest.PointAt(t[0]);
22                 var groundPlane = new Plane(new Point3d(0, 0, 0), new Vector3d(0, 0,
                       1));
23                 for (int i = 0; i < varyingWidthBoundaries.Count; i++)
24                 {
25                     PointContainment isContained = varyingWidthBoundaries[i].Contains(
                           midPt, groundPlane, 0.01);
26                     if ((int)isContained == 1 && midPt.Z >= varyingWidthBoundaries[i].
                           PointAtStart.Z - _tolerance)
27                     {
28                         return variationsWidth[i];
29                     }
30                 }
31                 return defaultWidth;
32             }
33             else return defaultWidth;
34         }
35
36
37         public double GetAssociatedWidth(Point3d ptToTest, double defaultWidth, List<
               PolylineCurve> varyingWidthBoundaries, List<double> variationsWidth)
38         {
39             if (varyingWidthBoundaries.Count > 0)
40             {
41                 var groundPlane = new Plane(new Point3d(0, 0, 0), new Vector3d(0, 0,
                       1));
42                 for (int i = 0; i < varyingWidthBoundaries.Count; i++)
43                 {
44                     PointContainment isContained = varyingWidthBoundaries[i].Contains(
                           ptToTest, groundPlane, 0.01);
45                     if ((int)isContained == 1 && ptToTest.Z >= varyingWidthBoundaries[
                           i].PointAtStart.Z - _tolerance)
46                     {
47                         return variationsWidth[i];
48                     }
49                 }
50                 return defaultWidth;
51             }
```

```
52              else return defaultWidth;
53          }
54
55
56          public List<Plane> GetReferencePts(List<Plane> perpFrames, List<double>
                existingVariationsWidth)
57          {
58              // step 1:
59              // include all points which distance to one of the next three points in
                    line are matching to one of the span widths
60
61              List<Plane> extractedFrames = new List<Plane>();
62
63              for (int j = 0; j < perpFrames.Count - 1; j++)
64              {
65                  double dist = double.NaN;
66                  bool x = false;
67
68                  // test if the first point after point to test is matching the span
                        width
69                  dist = Math.Round(perpFrames[j].Origin.DistanceTo(perpFrames[j + 1].
                        Origin), 2);
70                  foreach (double spanWidth in existingVariationsWidth)
71                  {
72                      if (dist == spanWidth)
73                      {
74                          extractedFrames.Add(perpFrames[j]);
75                          x = true;
76                          break;
77                      }
78                  }
79
80                  // test if the second point after point to test is matching the span
                        width
81                  if (!x && j < perpFrames.Count - 2)
82                  {
83                      dist = Math.Round(perpFrames[j].Origin.DistanceTo(perpFrames[j +
                            2].Origin), 2);
84                      foreach (double spanWidth in existingVariationsWidth)
85                      {
86                          if (dist == spanWidth)
87                          {
88                              extractedFrames.Add(perpFrames[j]);
89                              x = true;
90                              break;
91                          }
92                      }
93                  }
94
95                  // test if the third point after point to test is matching the span
                        width
96                  if (!x && j < perpFrames.Count - 3)
97                  {
98                      dist = Math.Round(perpFrames[j].Origin.DistanceTo(perpFrames[j +
                            3].Origin), 2);
99                      foreach (double spanWidth in existingVariationsWidth)
100                     {
101                         if (dist == spanWidth)
102                         {
103                             extractedFrames.Add(perpFrames[j]);
```

```
104                             break;
105                         }
106                     }
107                 }
108             }
109
110             // add last point in array
111             extractedFrames.Add(perpFrames[perpFrames.Count - 1]);
112
113
114             // step 2:
115             //if the distance from one point to the point after the next one is
                    matching to one of the span widths, toss the middle point.
116
117             int u = 0;
118             while (u < extractedFrames.Count - 2)
119             {
120                 double dist = double.NaN;
121                 dist = Math.Round(extractedFrames[u].Origin.DistanceTo(extractedFrames
                        [u + 2].Origin), 2);
122
123                 foreach (double spanWidth in existingVariationsWidth)
124                 {
125                     if (dist == spanWidth)
126                     {
127                         extractedFrames.RemoveAt(u + 1);
128                         break;
129                     }
130                 }
131                 u += 1;
132             }
133             return extractedFrames;
134         }
135
136
137         public List<Curve> SubdivideLines(Curve lineToStartSubdivisionProcessWith,
                List<Curve> linesToProcess, ref List<Curve> linesSubdivided, Plane
                planeOfIntersection,
138             double maximalDistanceCrosswise, double minimumLength, double defaultWidth
                    , List<PolylineCurve> varyingWidthsBoundaries, List<double>
                    variationsWidth)
139         {
140             List<Curve> linesProcessed = new List<Curve>();
141             double width = GetAssociatedWidth(lineToStartSubdivisionProcessWith,
                    defaultWidth, varyingWidthsBoundaries, variationsWidth);
142             double extensionLength = width - ((lineToStartSubdivisionProcessWith.
                    GetLength() / 2) % width);
143
144             // split and extend line to start subdivision process with
145             if (true)
146             {
147                 Curve[] lineToStartWithSplitted;
148                 double[] t = lineToStartSubdivisionProcessWith.DivideByCount(2, false)
                        ;
149                 lineToStartWithSplitted = lineToStartSubdivisionProcessWith.Split(t
                        [0]);
150                 lineToStartWithSplitted[0] = lineToStartWithSplitted[0].Extend(
                        CurveEnd.End, extensionLength, CurveExtensionStyle.Line);
151
152                 if (lineToStartWithSplitted[1].Reverse())
```

```
153                 {
154                     lineToStartWithSplitted[1] = lineToStartWithSplitted[1].Extend(
                            CurveEnd.End, extensionLength, CurveExtensionStyle.Line);
155                     linesSubdivided.Add(lineToStartWithSplitted[0].Rebuild(2, 1, true)
                            );
156                     linesSubdivided.Add(lineToStartWithSplitted[1].Rebuild(2, 1, true)
                            );
157                 }
158             }
159 
160             // split lines to process at their intersection with the plane of
                    intersection provided it exists
161             for (int i = 0; i < linesToProcess.Count; i++)
162             {
163                 Point3d ptIntersection;
164                 var intersections = Rhino.Geometry.Intersect.Intersection.CurvePlane(
                        linesToProcess[i], planeOfIntersection, _tolerance);
165 
166                 if (intersections != null)
167                 {
168                     ptIntersection = intersections[0].PointA;
169                     if (ptIntersection.DistanceTo(linesToProcess[i].PointAtStart) >
                            _tolerance
170                         && ptIntersection.DistanceTo(linesToProcess[i].PointAtEnd) >
                                _tolerance)
171                     {
172                         double t;
173                         if (linesToProcess[i].ClosestPoint(ptIntersection, out t))
174                         {
175                             // split
176                             Curve[] lineToProcessSplitted = linesToProcess[i].Split(t)
                                    ;
177                             lineToProcessSplitted[0] = lineToProcessSplitted[0].Extend
                                    (CurveEnd.End, extensionLength, CurveExtensionStyle.
                                    Line);
178                             if (lineToProcessSplitted[1].Reverse())
179                             {
180                                 lineToProcessSplitted[1] = lineToProcessSplitted[1].
                                        Extend(CurveEnd.End, extensionLength,
                                        CurveExtensionStyle.Line);
181                             }
182 
183                             // get width of future splitted line to process
184                             double associatedWidthFirstPartOfLineToProcess =
                                    GetAssociatedWidth(lineToProcessSplitted[0],
                                    defaultWidth, varyingWidthsBoundaries, variationsWidth
                                    );
185                             double associatedWidthSecondPartOfLineToProcess =
                                    GetAssociatedWidth(lineToProcessSplitted[1],
                                    defaultWidth, varyingWidthsBoundaries, variationsWidth
                                    );
186 
187                             // if rail curve is out of reach or too short skip it
188                             double dist = planeOfIntersection.Origin.DistanceTo(new
                                    Point3d(ptIntersection.X, ptIntersection.Y,
                                    planeOfIntersection.OriginZ));
189                             if (dist <= maximalDistanceCrosswise
190                                 && lineToProcessSplitted[0].GetLength() >
                                        associatedWidthFirstPartOfLineToProcess *
                                        minimumLength + _tolerance
```

```
191                                 && lineToProcessSplitted[1].GetLength() >
                                        associatedWidthSecondPartOfLineToProcess *
                                        minimumLength + _tolerance)
192                                 {
193                                     linesProcessed.Add(lineToProcessSplitted[0].Rebuild(2,
                                            1, true));
194                                     linesProcessed.Add(lineToProcessSplitted[1].Rebuild(2,
                                            1, true));
195
196                                 }
197                                 else linesProcessed.Add(linesToProcess[i]);
198                             }
199                         }
200                         else linesProcessed.Add(linesToProcess[i]);
201                     }
202                     else linesProcessed.Add(linesToProcess[i]);
203                 }
204                 return linesProcessed;
205             }
206
207
208         }
209 }
```

Listing 5: CreateFlexibleGridGhc

```
1  using  Grasshopper.Kernel;
2 using Grasshopper.Kernel.Data;
3 using Grasshopper.Kernel.Types;
4 using Rhino.Geometry;
5
6 using System;
7 using System.Collections.Generic;
8
9 namespace FlexibleGrid.Grid
10 {
11     public class CreateFlexibleGridGhc : GH_Component
12     {
13         /// <summary>
14         /// Initializes a new instance of the MyComponent1 class.
15         /// </summary>
16         public CreateFlexibleGridGhc()
17           : base("Grid", "Grid",
18               "This component creates a flexible grid.",
19               "Flexible Grid", "Grid")
20         {
21         }
22
23         /// <summary>
24         /// Registers all the input parameters for this component.
25         /// </summary>
26         protected override void RegisterInputParams(GH_Component.GH_InputParamManager
               pManager)
27         {
28             pManager.AddPlaneParameter("Reference Frames", "R", "The needed frames to
                   generate the grid.", GH_ParamAccess.tree);
29             pManager.AddNumberParameter("Default Depth", "D", "The default depth of
                   one structural module.", GH_ParamAccess.item);
30             pManager.AddCurveParameter("Markings Height", "M", "The Heights of the
                   structural modules marked as lines.", GH_ParamAccess.list);
31             pManager.AddGenericParameter("Variations Depth", "V", "Introducing
                   variations of the depth of the structural modules.", GH_ParamAccess.
                   list);
32         }
33
34         /// <summary>
35         /// Registers all the output parameters for this component.
36         /// </summary>
37         protected override void RegisterOutputParams(GH_Component.
               GH_OutputParamManager pManager)
38         {
39             pManager.AddPlaneParameter("Support Bases", "B", "supporting bases",
                   GH_ParamAccess.tree);
40             pManager.AddPlaneParameter("Support Heads", "H", "supporting heads",
                   GH_ParamAccess.tree);
41             pManager.AddCurveParameter("Bridges", "B", "Bridges", GH_ParamAccess.list)
                   ;
42         }
43
44         /// <summary>
45         /// This is the method that actually does the work.
46         /// </summary>
47         /// <param name="DA">The DA object is used to retrieve from inputs and store
               in outputs.</param>
48         protected override void SolveInstance(IGH_DataAccess DA)
```

```
49          {
50              GH_Structure<GH_Plane> iReferenceFrames;
51              double iDefaultDepth = double.NaN;
52              List <Curve> iMarkingsHeight= new List<Curve>();
53              List<GH_ObjectWrapper> iVariationsDepth = new List<GH_ObjectWrapper>();
54
55              if (!DA.GetDataTree(0, out iReferenceFrames)) { return; }
56              if (!DA.GetData(1, ref iDefaultDepth)) { return; }
57              if (!DA.GetDataList(2, iMarkingsHeight)) { return; }
58              if (!DA.GetDataList(3, iVariationsDepth)) { return; }
59
60              FlexibleGrid.Grid.Grid spatialGrid = new FlexibleGrid.Grid.Grid(
                    iReferenceFrames, iDefaultDepth, iMarkingsHeight, iVariationsDepth);
61
62
63              // get supporting heads
64              if (spatialGrid.ReferenceFrames.PathCount > 0)
65              {
66                  if (!spatialGrid.GetSupportingHeads())
67                  {
68                      AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
69                          "A marking curve does not cover at least one reference point
                                for introducing height.");
70                      return;
71                  }
72              }
73
74
75              // level heights
76              if (spatialGrid.ReferenceFrames.PathCount > 0)
77              {
78                  if (!spatialGrid.LevelHeights())
79                  {
80                      AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
81                          "level heights.");
82                      return;
83                  }
84              }
85
86
87              // fit supporting bases and heads
88              if (spatialGrid.ReferenceFrames.PathCount > 0)
89              {
90                  if (!spatialGrid.FitSupportingBasesAndHeads())
91                  {
92                      AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
93                          "Reference frames and supporting tops could not be fitted.");
94                      return;
95                  }
96              }
97
98
99              // extract bridges
100             if (spatialGrid.SupportBasesFitted.PathCount > 0)
101             {
102                 if (!spatialGrid.GetBridges())
103                 {
104                     AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
105                         "Bridges could be not extracted.");
106                     return;
```

```
107                 }
108             }
109 
110             DA.SetDataTree(0, spatialGrid.SupportBasesFitted);
111             DA.SetDataTree(1, spatialGrid.SupportHeadsFitted);
112             DA.SetDataList(2, spatialGrid.Bridges);
113         }
114 
115         /// <summary>
116         /// Provides an Icon for the component.
117         /// </summary>
118         protected override System.Drawing.Bitmap Icon
119         {
120             get
121             {
122                 //You can add image files to your project resources and access them
                        like this:
123                 // return Resources.IconForThisComponent;
124                 return null;
125             }
126         }
127 
128         /// <summary>
129         /// Gets the unique ID for this component. Do not change this ID after release
                .
130         /// </summary>
131         public override Guid ComponentGuid
132         {
133             get { return new Guid("62c6b42c-3819-43f3-a04d-1ea69bbd073b"); }
134         }
135     }
136 }
```

Listing 6: VaryDepthStructuralModuleGhc

```
1  using  Grasshopper.Kernel;
2 using Rhino.Geometry;
3 using System;
4 using System.Collections.Generic;
5
6 namespace FlexibleGrid.Grid
7 {
8     public class VaryDepthStructuralModule : GH_Component
9     {
10        /// <summary>
11        /// Initializes a new instance of the MyComponent1 class.
12        /// </summary>
13        public VaryDepthStructuralModule()
14          : base("Varying Depth", "Varying Depth",
15              "This component introduces a variation of a structural modules depth.",
16              "Flexible Grid", "Grid")
17        {
18        }
19
20        /// <summary>
21        /// Registers all the input parameters for this component.
22        /// </summary>
23        protected override void RegisterInputParams(GH_Component.GH_InputParamManager
              pManager)
24        {
25            pManager.AddNumberParameter("Depth", "D", "Depth of a structural module",
                  GH_ParamAccess.item);
26            pManager.AddCurveParameter("Markings", "M", "Planar lines as marking for a
                   variation of the default depth.", GH_ParamAccess.list);
27        }
28
29        /// <summary>
30        /// Registers all the output parameters for this component.
31        /// </summary>
32        protected override void RegisterOutputParams(GH_Component.
              GH_OutputParamManager pManager)
33        {
34            pManager.AddGenericParameter("Variation Depth", "V", "Deviation of the
                  default depth.", GH_ParamAccess.item);
35        }
36
37        /// <summary>
38        /// This is the method that actually does the work.
39        /// </summary>
40        /// <param name="DA">The DA object is used to retrieve from inputs and store
              in outputs.</param>
41        protected override void SolveInstance(IGH_DataAccess DA)
42        {
43            var iDepth = double.NaN;
44            List<Rhino.Geometry.Curve> iMarkings = new List<Rhino.Geometry.Curve>();
45            if (!DA.GetData(0, ref iDepth)) { return; }
46            if (!DA.GetDataList(1, iMarkings)) { return; }
47
48            // test if each boundary curve is planar and closed
49            foreach (Curve marking in iMarkings)
50            {
51                if (!marking.IsPlanar())
52                {
53                    AddRuntimeMessage(GH_RuntimeMessageLevel.Error,
```

```
54                      "At least one marking curve is not planar.");
55                  return;
56              }
57
58          }
59
60          // define size array
61          int sizeArray = iMarkings.Count * 3 + 1;
62
63          /*_____________________________________
64          * COMPOSITION OF ABSTRACT DATA ARRAY
65          * _____________________________________
66          *  1.)             2.)             3.)
67          *  depth           int.MaxValue    int.MaxValue
68          *  position x      position y      position z       => coordinates of
                first corner point
69          *  position x      position y      position z
70          *  int.MaxValue    int.MaxValue    int.MaxValue     => mark for the start
                of a new boundary curve
71          *  position x      position y      position z       => coordinates of
                first corner point
72          *  position x      position y      position z
73          *  int.MaxValue    int.MaxValue    int.MaxValue     => mark for the start
                of a new boundary curve
74          *  position x      position y      position z       => coordinates of
                first corner point
75          *  position x      position y      position z
76          *  int.MaxValue    int.MaxValue    int.MaxValue     => end array
77          *   ...
78          */
79
80          // initialize array
81          var abstractData = new double[sizeArray, 3];
82          for (int i = 0; i < abstractData.GetLength(0); i++)
83          {
84              for (int j = 0; j < abstractData.GetLength(1); j++)
85              {
86                  abstractData[i, j] = int.MaxValue;
87              }
88          }
89
90          // pass on span width
91          abstractData[0, 0] = iDepth;
92
93          int u = 0;
94          foreach (Curve curve in iMarkings)
95          {
96              abstractData[u + 1, 0] = curve.PointAtStart.X;
97              abstractData[u + 1, 1] = curve.PointAtStart.Y;
98              abstractData[u + 1, 2] = curve.PointAtStart.Z;
99
100             abstractData[u + 2, 0] = curve.PointAtEnd.X;
101             abstractData[u + 2, 1] = curve.PointAtEnd.Y;
102             abstractData[u + 2, 2] = curve.PointAtEnd.Z;
103             // empty line as marker for the begin of a new boundary curve
104             u = u + 3;
105         }
106
107         // output abstract data array
108         var oAbstractData = new Grasshopper.Kernel.Types.GH_ObjectWrapper(
```

```
                abstractData);
109             DA.SetData(0, oAbstractData);
110         }
111
112         /// <summary>
113         /// Provides an Icon for the component.
114         /// </summary>
115         protected override System.Drawing.Bitmap Icon
116         {
117             get
118             {
119                 //You can add image files to your project resources and access them
                        like this:
120                 // return Resources.IconForThisComponent;
121                 return null;
122             }
123         }
124
125         /// <summary>
126         /// Gets the unique ID for this component. Do not change this ID after release
                .
127         /// </summary>
128         public override Guid ComponentGuid
129         {
130             get { return new Guid("17487d37-3544-45cf-9789-807972dd8e64"); }
131         }
132     }
133 }
```

Listing 7: Grid

```
1  using  Grasshopper.Kernel.Data;
2 using Grasshopper.Kernel.Types;
3 using Rhino.Geometry;
4
5 using System;
6 using System.Collections.Generic;
7 namespace FlexibleGrid.Grid
8 {
9     class Grid
10    {
11        private double _tolerance = 0.00001;
12        public List<PolylineCurve> Bridges;
13        public double DefaultDepth { get; private set; }
14        public List<Curve> MarkingsHeight { get; private set; }
15        public GH_Structure<GH_Plane> ReferenceFrames { get; private set; }
16        public GH_Structure<GH_Plane> SupportBasesFitted { get; private set; }
17        public GH_Structure<GH_Plane> SupportHeads { get; private set; }
18        public GH_Structure<GH_Plane> SupportHeadsFitted { get; private set;  }
19        public List<double> VariationsDepth { get; private set; }
20        public List<PolylineCurve> VaryingDepthMarkings { get; private set; }
21
22        // constructor for creating spatial grid
23        public Grid(GH_Structure<GH_Plane> referenceFrames, double defaultDepth, List<
              Curve> markingsHeight, List<GH_ObjectWrapper> variationsDepth)
24        {
25            this.Bridges = new List<PolylineCurve>();
26            this.DefaultDepth = defaultDepth;
27            this.MarkingsHeight = markingsHeight;
28            this.ReferenceFrames = referenceFrames;
29            this.SupportBasesFitted = new GH_Structure<GH_Plane>();
30            this.SupportHeads = new GH_Structure<GH_Plane>();
31            this.SupportHeadsFitted = new GH_Structure<GH_Plane>();
32            this.VariationsDepth = new List<double>();
33            this.VaryingDepthMarkings = new List<PolylineCurve>();
34
35            // unwrap array
36            foreach (GH_ObjectWrapper variation in variationsDepth)
37            {
38                var dataAbstract = variation.Value as double[,];
39
40                List<Point3d> startEndPts = new List<Point3d>();
41                for (int i = 1; i < dataAbstract.GetLength(0); i++)
42                {
43                    if (dataAbstract[i, 0] != int.MaxValue && dataAbstract[i, 1] !=
                          int.MaxValue)
44                    {
45                        startEndPts.Add(new Point3d(dataAbstract[i, 0], dataAbstract[i
                              , 1], dataAbstract[i, 2]));
46                    }
47
48                    else
49                    {
50                        VariationsDepth.Add(dataAbstract[0, 0]);
51                        VaryingDepthMarkings.Add(new Rhino.Geometry.PolylineCurve(
                              startEndPts));
52                        startEndPts.Clear();
53                    }
54                }
55            }
```

```
56          }
57
58
59          public bool GetSupportingHeads()
60          {
61              try
62              {
63                  for (int i = 0; i < ReferenceFrames.PathCount; i++)
64                  {
65                      List<Plane> referenceFrames = new List<Plane>();
66
67                      var list = ReferenceFrames.Branches[i];
68                      for (int j = 0; j < list.Count; j++)
69                      {
70                          referenceFrames.Add(new Plane(list[j].Value));
71                      }
72
73                      List<GH_Plane> supportHeads = new List<GH_Plane>();
74
75                      for (int j = 0; j < referenceFrames.Count; j++)
76                      {
77                          bool supportingTopFound = false;
78                          Point3d tempSupportTop = new Point3d();
79                          double dist = double.MaxValue;
80
81                          for (int k = 0; k < MarkingsHeight.Count; k++)
82                          {
83                              double t;
84
85                              if (MarkingsHeight[k].ClosestPoint(referenceFrames[j].
                                    Origin, out t))
86                              {
87                                  Point3d closestPt = MarkingsHeight[k].PointAt(t);
88                                  Vector3d direction = closestPt - referenceFrames[j].
                                        Origin;
89
90                                  Vector3d crossProduct = new Vector3d(Vector3d.
                                        CrossProduct(direction, new Vector3d(0, 0, 1)));
91
92                                  // Vector a and b are parallel if, and only if, a x b
                                        = 0;
93                                  if (crossProduct.Length < _tolerance)
94                                  {
95                                      double dotProduct = direction.X * 0 + direction.Y
                                            * 0 + direction.Z * 1;
96
97                                      // positive dot product -> vectors go in the same
                                            direction; negative dot product -> Vectors are
                                             going in the opposite direction
98                                      // A; in normal direction (positive world y axis)
99                                      if (dotProduct >= 0)
100                                     {
101                                         if (referenceFrames[j].Origin.DistanceTo(
                                                closestPt) < dist
102                                           && referenceFrames[j].Origin.DistanceTo(
                                                closestPt) > _tolerance)
103                                         {
104                                             supportingTopFound = true;
105                                             tempSupportTop = closestPt;
106                                             dist = referenceFrames[j].Origin.
```

```
                                                    DistanceTo(closestPt);
107                                             }
108                                         }
109                                     }
110                                 }
111                             }
112                             if (!supportingTopFound) return false;
113                             else
114                             {
115                                 Point3d transformedOrigin = new Point3d(referenceFrames[j
                                        ].Origin.X, referenceFrames[j].Origin.Y,
                                        tempSupportTop.Z);
116                                 supportHeads.Add(new GH_Plane(new Plane(transformedOrigin,
                                         referenceFrames[j].ZAxis)));
117                             }
118                         }
119                         GH_Path x = new GH_Path(i);
120                         SupportHeads.AppendRange(supportHeads, x);
121                     }
122                     return true;
123                 }
124                 catch { return false; }
125             }
126
127
128             public bool LevelHeights()
129             {
130                 try
131                 {
132                     for (int i = 0; i < SupportHeads.PathCount; i++)
133                     {
134                         List<Plane> supportHeads = new List<Plane>();
135
136                         var list = SupportHeads.Branches[i];
137                         for (int j = 0; j < list.Count; j++)
138                         {
139                             supportHeads.Add(new Plane(list[j].Value));
140                         }
141
142                         List<double> zPosReferencePts = new List<double>();
143                         List<int> groupSizesSameZPos = new List<int>();
144
145                         for (int j = 0; j < supportHeads.Count; j++)
146                         {
147                             bool exists = false;
148                             for (int k = 0; k < zPosReferencePts.Count; k++)
149                             {
150                                 if (Math.Round(supportHeads[j].Origin.Z, 5) ==
                                        zPosReferencePts[k])
151                                 {
152                                     exists = true;
153                                     int count = groupSizesSameZPos[k] + 1;
154                                     groupSizesSameZPos[k] = count;
155                                 }
156                             }
157                             if (!exists)
158                             {
159                                 zPosReferencePts.Add(Math.Round(supportHeads[j].Origin.Z,
                                        5));
160                                 groupSizesSameZPos.Add(1);
```

```
161                         }
162                     }
163
164                     // sort
165                     List<int> indexes = new List<int>();
166                     int x = 0;
167                     while (x < groupSizesSameZPos.Count)
168                     {
169                         indexes.Add(x);
170                         x++;
171                     }
172
173                     FlexibleGrid.Misc.Sort sort = new FlexibleGrid.Misc.Sort();
174                     sort.BubbleSort(ref groupSizesSameZPos, ref indexes);
175                     double heightGuideValue = zPosReferencePts[indexes[indexes.Count -
                            1]];
176
177                     // adjust heights to guide value
178                     for (int j = 0; j < supportHeads.Count; j++)
179                     {
180                         Point3d temp = supportHeads[j].Origin;
181                         temp.Z = heightGuideValue;
182                         Plane frameTransformed = new Plane(temp, supportHeads[j].ZAxis
                                );
183                         supportHeads[j] = frameTransformed;
184                     }
185
186                     List<GH_Plane> oSupportTops = new List<GH_Plane>();
187                     foreach (Plane plane in supportHeads) oSupportTops.Add(new
                         GH_Plane(plane));
188
189                     GH_Path y = new GH_Path(SupportHeads.Paths[i]);
190                     SupportHeads.RemovePath(new GH_Path(SupportHeads.Paths[i]));
191                     SupportHeads.AppendRange(oSupportTops, y);
192                 }
193                 return true;
194             }
195             catch { return false; }
196         }
197
198
199
200         //3.) fitting
201
202         public bool FitSupportingBasesAndHeads()
203         {
204             try
205             {
206                 // find associated depth by checking marks
207                 List<int> indexesAssociatedDepth = new List<int>();
208
209                 for (int i = 0; i < ReferenceFrames.PathCount; i++)
210                 {
211                     List<Plane> supportingBases = new List<Plane>();
212
213                     var list = ReferenceFrames.Branches[i];
214                     for (int j = 0; j < list.Count; j++)
215                     {
216                         supportingBases.Add(new Plane(list[j].Value));
217                     }
```

```
218
219                 indexesAssociatedDepth.Add(int.MaxValue);
220
221
222                 for (int j = 0; j < VaryingDepthMarkings.Count; j++)
223                 {
224                     for (int k = 0; k < supportingBases.Count; k++)
225                     {
226                         double t;
227                         if (VaryingDepthMarkings[j].ClosestPoint(supportingBases[k
                               ].Origin, out t))
228                         {
229                             if (VaryingDepthMarkings[j].PointAt(t).DistanceTo(
                                   supportingBases[k].Origin) > _tolerance)
230                             {
231                                 break;
232                             }
233                             if (k == supportingBases.Count - 1)
234                             {
235                                 indexesAssociatedDepth[i] = j;
236                                 j = VaryingDepthMarkings.Count;
237                             }
238                         }
239                     }
240                 }
241             }
242
243
244             GH_Structure<GH_Plane> supportHeadsFitted = new GH_Structure<GH_Plane
                    >();
245             GH_Structure<GH_Plane> supportBasesFitted = new GH_Structure<GH_Plane
                    >();
246
247             for (int i = 0; i < ReferenceFrames.PathCount; i++)
248             {
249                 List<Plane> supportingBases = new List<Plane>();
250                 List<Plane> supportingHeads = new List<Plane>();
251
252                 var list = ReferenceFrames.Branches[i];
253                 for (int j = 0; j < list.Count; j++)
254                 {
255                     supportingBases.Add(new Plane(list[j].Value));
256                 }
257
258                 list.Clear();
259                 list = SupportHeads.Branches[i];
260                 for (int j = 0; j < list.Count; j++)
261                 {
262                     supportingHeads.Add(new Plane(list[j].Value));
263                 }
264
265                 // get depth
266                 double depth;
267                 if (indexesAssociatedDepth[i] == int.MaxValue) depth =
                        DefaultDepth;
268                 else depth = VariationsDepth[indexesAssociatedDepth[i]];
269
270                 for (int j = 0; j < supportingBases.Count; j++)
271                 {
272                     // move reference points
```

```
273                     Vector3d normalA = new Vector3d(supportingBases[j].ZAxis);
274                     Vector3d a = new Vector3d();
275                     Vector3d b = new Vector3d();
276                     if (normalA.Rotate(Math.PI * 0.5, new Vector3d(0, 0, 1)))
277                     {
278                         Vector3d origin = new Vector3d(supportingBases[j].Origin);
279                         a = new Vector3d(origin + (normalA * depth / 2));
280                     }
281                     if (normalA.Rotate(Math.PI, new Vector3d(0, 0, 1)))
282                     {
283                         Vector3d origin = new Vector3d(supportingBases[j].Origin);
284                         b = new Vector3d(origin + (normalA * depth / 2));
285                     }
286                     // append to list
287                     List<GH_Plane> tempA = new List<GH_Plane>() { new GH_Plane(new
                          Plane(new Point3d(a), supportingBases[j].ZAxis)), new
                         GH_Plane(new Plane(new Point3d(b), supportingBases[j].
                         ZAxis)) };
288                     GH_Path pathA = new GH_Path(i, j);
289                     supportBasesFitted.AppendRange(tempA, pathA);
290
291                     // move support tops
292                     Vector3d normalB = new Vector3d(supportingHeads[j].ZAxis);
293                     Vector3d d = new Vector3d();
294                     Vector3d c = new Vector3d();
295                     if (normalB.Rotate(Math.PI * 0.5, new Vector3d(0, 0, 1)))
296                     {
297                         Vector3d origin = new Vector3d(supportingHeads[j].Origin);
298                         d = new Vector3d(origin + (normalB * depth / 2));
299                     }
300                     if (normalB.Rotate(Math.PI, new Vector3d(0, 0, 1)))
301                     {
302                         Vector3d origin = new Vector3d(supportingHeads[j].Origin);
303                         c = new Vector3d(origin + (normalB * depth / 2));
304                         // append to list
305                         List<GH_Plane> tempB = new List<GH_Plane>() { new GH_Plane
                              (new Plane(new Point3d(c), supportingHeads[j].ZAxis)),
                               new GH_Plane(new Plane(new Point3d(d),
                              supportingHeads[j].ZAxis)) };
306                         GH_Path pathB = new GH_Path(i, j);
307                         supportHeadsFitted.AppendRange(tempB, pathB);
308                     }
309                 }
310             }
311
312             SupportBasesFitted = supportBasesFitted;
313             SupportHeadsFitted = supportHeadsFitted;
314             return true;
315         }
316         catch { return false; }
317     }
318
319
320     public bool GetBridges()
321     {
322         try
323         {
324             FlexibleGrid.Misc.Sort sort = new FlexibleGrid.Misc.Sort();
325
326             // flatten tree and delete duplicates
```

```
327                 List<Plane> supportBases = new List<Plane>();
328                 for (int i = 0; i < SupportBasesFitted.PathCount; i++)
329                 {
330                     var list = SupportBasesFitted.Branches[i];
331
332                     //check if reference point lies on a marking curve for height
333                     double x = (list[0].Value.Origin.X + list[1].Value.Origin.X) / 2;
334                     double y = (list[0].Value.Origin.Y + list[1].Value.Origin.Y) / 2;
335                     double z = list[0].Value.Origin.Z;
336                     Point3d avgPt = new Point3d(x, y, z);
337
338                     bool covered = false;
339                     for (int u = 0; u < MarkingsHeight.Count; u++)
340                     {
341                         double t;
342                         if (MarkingsHeight[u].ClosestPoint(avgPt, out t))
343                         {
344                             if (avgPt.DistanceTo(MarkingsHeight[u].PointAt(t)) <
                                    _tolerance)
345                             {
346                                 covered = true;
347                                 break;
348                             }
349                         }
350                     }
351
352                     if (covered)
353                     {
354                         for (int j = 0; j < list.Count; j++)
355                         {
356                             supportBases.Add(list[j].Value);
357                         }
358                     }
359                 }
360                 sort.DeleteDuplicates(ref supportBases);
361
362                 List<Plane> supportHeads = new List<Plane>();
363                 for (int i = 0; i < SupportHeadsFitted.PathCount; i++)
364                 {
365                     var list = SupportHeadsFitted.Branches[i];
366
367                     for (int j = 0; j < list.Count; j++)
368                     {
369                         supportHeads.Add(list[j].Value);
370                     }
371                 }
372                 sort.DeleteDuplicates(ref supportHeads);
373
374                 // generate bridges
375                 for (int i = 0; i < supportBases.Count; i++)
376                 {
377                     Point3d supportBase = new Point3d(supportBases[i].Origin);
378                     Vector3d normal = supportBases[i].ZAxis;
379
380
381                     // test if support base is already supported
382                     bool supported = false;
383                     for (int j = 0; j < supportHeads.Count; j++)
384                     {
385                         if (supportBase.DistanceTo(supportHeads[j].Origin) <
```

```
                    _tolerance)
386                 {
387                     supported = true;
388                     break;
389                 }
390             }
391             if (supported) continue;
392
393
394             // if not supported generate bridge
395             int indexClosestSupportHeadA = int.MaxValue;
396             int indexClosestSupportHeadB = int.MaxValue;
397             int indexClosestSupportBaseA = int.MaxValue;
398             int indexClosestSupportBaseB = int.MaxValue;
399
400             double distBaseHeadA = double.MaxValue;
401             double distBaseHeadB = double.MaxValue;
402             double distBaseBaseA = double.MaxValue;
403             double distBaseBaseB = double.MaxValue;
404
405
406             // find closest support heads
407             for (int j = 0; j < supportHeads.Count; j++)
408             {
409                 Vector3d bridge = new Vector3d(supportHeads[j].Origin -
                        supportBase);
410                 Vector3d crossProduct = new Vector3d(Vector3d.CrossProduct(
                        normal, bridge));
411
412                 // Vector a and b are parallel if, and only if, a x b = 0;
413                 if (crossProduct.Length < _tolerance)
414                 {
415                     double dotProduct = bridge.X * normal.X + bridge.Y *
                            normal.Y + bridge.Z * normal.Z;
416
417                     // positive dot product -> vectors go in the same
                            direction; negative dot product -> Vectors are going
                            in the opposite direction
418                     // A; in normal direction (positive world y axis)
419                     if (dotProduct < 0)
420                     {
421                         if (bridge.Length < distBaseHeadA)
422                         {
423                             indexClosestSupportHeadA = j;
424                             distBaseHeadA = Math.Round(bridge.Length, 5);
425                         }
426                     }
427
428                     // B; in opposite direction of normal(negative world y
                            axis)
429                     if (dotProduct >= 0)
430                     {
431                         if (bridge.Length < distBaseHeadB)
432                         {
433                             indexClosestSupportHeadB = j;
434                             distBaseHeadB = Math.Round(bridge.Length, 5);
435                         }
436                     }
437                 }
438             }
```

```
439
440
441                 // find closest support bases
442                 for (int j = 0; j < supportBases.Count; j++)
443                 {
444
445                     Vector3d bridge = new Vector3d(supportBases[j].Origin -
                            supportBase);
446                     Vector3d crossProduct = new Vector3d(Vector3d.CrossProduct(
                            normal, bridge));
447
448                     // Vector a and b are parallel if, and only if, axb= 0;
449                     if (crossProduct.Length < _tolerance)
450                     {
451                         double dotProduct = bridge.X * normal.X + bridge.Y *
                                normal.Y + bridge.Z * normal.Z;
452
453                         // positive dot product -> vectors go in the same
                                direction; negative dot product -> Vectors are going
                                in the opposite direction
454                         // A; in normal direction (positive world y axis)
455                         if (dotProduct < 0)
456                         {
457                             if (bridge.Length < distBaseBaseA && bridge.Length >
                                    _tolerance)
458                             {
459                                 indexClosestSupportBaseA = j;
460                                 distBaseBaseA = Math.Round(bridge.Length, 5);
461                             }
462                         }
463
464                         // B; in opposite direction of normal(negative world y
                                axis)
465                         if (dotProduct >= 0)
466                         {
467                             if (bridge.Length < distBaseBaseB && bridge.Length >
                                    _tolerance)
468                             {
469                                 indexClosestSupportBaseB = j;
470                                 distBaseBaseB = Math.Round(bridge.Length, 5);
471                             }
472                         }
473                     }
474                 }
475
476
477                 // check if head or base is closer and assign
478                 Point3d bearingA = new Point3d();
479                 Point3d bearingB = new Point3d();
480
481                 if (distBaseHeadA != double.MaxValue || distBaseBaseA != double.
                        MaxValue)
482                 {
483                     if (distBaseHeadA <= distBaseBaseA)
484                     {
485                         bearingA = supportHeads[indexClosestSupportHeadA].Origin;
486                     }
487                     else bearingA = supportBases[indexClosestSupportBaseA].Origin;
488                 }
489                 else continue;
```

```
490
491                     if (distBaseHeadB != double.MaxValue || distBaseBaseB != double.
                            MaxValue)
492                     {
493                         if (distBaseHeadB <= distBaseBaseB)
494                         {
495                             bearingB = supportHeads[indexClosestSupportHeadB].Origin;
496                         }
497                         else bearingB = supportBases[indexClosestSupportBaseB].Origin;
498                     }
499                     else continue;
500
501                     //generate ouput
502                     List<Point3d> tempA = new List<Point3d>() { bearingA, supportBase
                            };
503                     List<Point3d> tempB = new List<Point3d>() { supportBase, bearingB
                            };
504                     Bridges.Add(new PolylineCurve(tempA));
505                     Bridges.Add(new PolylineCurve(tempB));
506                 }
507                 // delete duplicates
508                 for (int j = Bridges.Count - 1; j >= 0; j--)
509                 {
510                     double[] tA = Bridges[j].DivideByCount(2, false);
511                     Point3d midPtA = Bridges[j].PointAt(tA[0]);
512                     for (int k = j - 1; k >= 0; k--)
513                     {
514                         double[] tB = Bridges[k].DivideByCount(2, false);
515                         Point3d midPtB = Bridges[k].PointAt(tB[0]);
516
517                         if (midPtA.DistanceTo(midPtB) < _tolerance)
518                         {
519                             Bridges.RemoveAt(j);
520                             break;
521                         }
522                     }
523                 }
524                 return true;
525             }
526             catch { return false; }
527         }
528
529
530     }
531 }
```

Listing 8: Analyze

```
1  using  System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using Grasshopper.Kernel;
7 using Rhino.Geometry;
8
9 namespace FlexibleGrid.Misc
10 {
11     class Analyze
12     {
13         public List<Point3d> GetDiscontinuitys(Rhino.Geometry.Curve curveToTest)
14         {
15             List<Point3d> crvPts = new List<Point3d>();
16             double t0 = curveToTest.Domain.Min;
17             double t1 = curveToTest.Domain.Max;
18             double t;
19             bool search = true;
20
21             while (search)
22             {
23                 if (curveToTest.GetNextDiscontinuity(Continuity.C1_locus_continuous,
                       t0, t1, out t))
24                 {
25                     Point3d pt = curveToTest.PointAt(t);
26                     crvPts.Add(pt);
27                     if (t1 != t) t0 = t; else search = false;
28                 }
29                 else search = false;
30             }
31             crvPts.Insert(0, curveToTest.PointAtStart);
32             return crvPts;
33         }
34
35
36         public List<Point3d> GetDiscontinuitys(Rhino.Geometry.PolylineCurve
                   curveToTest)
37         {
38             List<Point3d> crvPts = new List<Point3d>();
39             double t0 = curveToTest.Domain.Min;
40             double t1 = curveToTest.Domain.Max;
41             double t;
42             bool search = true;
43
44             while (search)
45             {
46                 if (curveToTest.GetNextDiscontinuity(Continuity.C1_locus_continuous,
                       t0, t1, out t))
47                 {
48                     Point3d pt = curveToTest.PointAt(t);
49                     crvPts.Add(pt);
50                     if (t1 != t) t0 = t; else search = false;
51                 }
52                 else search = false;
53             }
54             crvPts.Insert(0, curveToTest.PointAtStart);
55             return crvPts;
56         }
```

```
57
58
59          public bool TestIfCrvsIntersect(List<Curve> segments)
60          {
61              // Test if input curves do intersect.
62              // The test is negative if two segments are connected.
63              // The test is positive if a third crv points to a knot. (T joint)
64              // The test is positive if one or more curve are intersected
65              // inbetween their end points and start points.
66              bool intersect = false;
67
68              // test for T joints
69              List<Point3d> startAndEndPts = new List<Point3d>();
70              for (int i = 0; i < segments.Count; i++)
71              {
72                  startAndEndPts.Add(segments[i].PointAtStart);
73                  startAndEndPts.Add(segments[i].PointAtEnd);
74              }
75
76              int counterDuplicates = 0;
77              for (int i = 0; i < startAndEndPts.Count; i++)
78              {
79                  for (int j = 0; j < startAndEndPts.Count; j++)
80                  {
81                      if (j == i) continue;
82                      else
83                      {
84                          if (startAndEndPts[i].DistanceTo(startAndEndPts[j]) < 0.01)
                                counterDuplicates++;
85                      }
86                  }
87                  if (counterDuplicates >= 2)
88                  {
89                      intersect = true;
90                      return intersect;
91                  }
92                  counterDuplicates = 0;
93              }
94
95              // test for intersections
96              for (int i = 0; i < segments.Count; i++)
97              {
98                  for (int j = i + 1; j < segments.Count; j++)
99                  {
100                     // Calculate the intersection
101                     const double intersection_tolerance = 0.001;
102                     const double overlap_tolerance = 0.0;
103                     var events = Rhino.Geometry.Intersect.Intersection.CurveCurve(
                            segments[i], segments[j],
104                         intersection_tolerance, overlap_tolerance);
105
106                     // Process the results
107                     if (events.Count != 0)
108                     {
109                         intersect = true;
110                         List<Point3d> tempStartAndEndPt = new List<Point3d>();
111                         tempStartAndEndPt.Add(segments[i].PointAtStart);
112                         tempStartAndEndPt.Add(segments[i].PointAtEnd);
113                         tempStartAndEndPt.Add(segments[j].PointAtStart);
114                         tempStartAndEndPt.Add(segments[j].PointAtEnd);
```

```
115                         for (int k = 0; k < tempStartAndEndPt.Count - 1; k++)
116                         {
117                             for (int l = k + 1; l < tempStartAndEndPt.Count; l++)
118                             {
119
120                                 if (tempStartAndEndPt[k].DistanceTo(tempStartAndEndPt[
                                        l]) < 0.01)
121                                 {
122                                     intersect = false;
123                                 }
124                             }
125                         }
126                         if (intersect)
127                         {
128                             return intersect;
129                         }
130                     }
131                 }
132             }
133             return intersect;
134         }
135
136
137     }
138 }
```

Listing 9: Sort

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Grasshopper.Kernel;
using Rhino.Geometry;


namespace FlexibleGrid.Misc
{
    class Sort
    {
        private double _tolerance = 0.00001;

        public void BubbleSort(ref List<double> valuesToSort)
        {
            bool swapped = true;

            while (swapped)
            {
                swapped = false;
                for (int i = 0; i < valuesToSort.Count - 1; i++)
                {
                    if (valuesToSort[i] > valuesToSort[i + 1])
                    {
                        double tempKey;
                        tempKey = valuesToSort[i];
                        valuesToSort[i] = valuesToSort[i + 1];
                        valuesToSort[i + 1] = tempKey;

                        swapped = true;
                    }
                }
            }
        }


        public void BubbleSort(ref List<double> keys, ref List<Curve> crvsToSort)
        {
            bool swapped = true;

            while (swapped)
            {
                swapped = false;
                for (int i = 0; i < keys.Count - 1; i++)
                {
                    if (keys[i] > keys[i + 1])
                    {
                        double tempKey;
                        tempKey = keys[i];
                        keys[i] = keys[i + 1];
                        keys[i + 1] = tempKey;

                        List<Curve> tempValues = new List<Curve>();
                        tempValues.Add(crvsToSort[i]);
                        crvsToSort[i] = crvsToSort[i + 1];
                        crvsToSort[i + 1] = tempValues[0];

```

```
60                          swapped = true;
61                      }
62                  }
63              }
64          }
65
66
67          public void BubbleSort(ref List<int> keys, ref List<int> integersToSort)
68          {
69              bool swapped = true;
70
71              while (swapped)
72              {
73                  swapped = false;
74                  for (int i = 0; i < keys.Count - 1; i++)
75                  {
76                      if (keys[i] > keys[i + 1])
77                      {
78                          int tempKey;
79                          tempKey = keys[i];
80                          keys[i] = keys[i + 1];
81                          keys[i + 1] = tempKey;
82
83                          List<int> tempIntegersToSort = new List<int>();
84                          tempIntegersToSort.Add(integersToSort[i]);
85                          integersToSort[i] = integersToSort[i + 1];
86                          integersToSort[i + 1] = tempIntegersToSort[0];
87
88                          swapped = true;
89                      }
90                  }
91              }
92          }
93
94
95          public void DeleteDuplicates(ref List<Plane> planesToTest)
96          {
97              for (int i = 0; i < planesToTest.Count - 1; i++)
98              {
99                  Next:
100                 for (int j = i + 1; j < planesToTest.Count; j++)
101                 {
102                     if (planesToTest[i].Origin.DistanceTo(planesToTest[j].Origin) <
                            _tolerance)
103                     {
104                         planesToTest.RemoveAt(j);
105                         goto Next;
106                     }
107                 }
108             }
109         }
110
111
112     }
113 }
```