TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Prefabrication Topologies

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

Manfred Berthold
Prof Arch DI Dr

E253
Institut für Architektur und Entwerfen

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

Matthias Danzmayr
BSc
0625312

Wien, am 31. Mai 2015

Abstract

Diese Arbeit beschäftigt sich mit den Möglichkeiten zur Automatisierung des Architekturschaffens. Nach Betrachtung von Vorbildern aus der Geschichte, aus anderen Bereichen menschlichen Schaffens und Produzierens, sowie der Analyse derzeitiger Vorfertigungsverfahren wird der Schluss gezogen, dass ein großes theoretisches Potential zur Optimierung der angewandten Prozesse vorliegt.

Im weiteren wird ein prototypischer Modellprozess entwickelt mit dem Ziel ein architektonisches Artefakt zu produzieren. Zur Verwendung für die automatisierte Produktion von Architektur wird zunächst eine dreiachsige CNC Fräse gebaut. Die Möglichkeiten dieser Maschine stellen gleichzeitig die Grenzen für die weitere Entwicklung des Modellprozesses dar. Dieser wird nun vorangetrieben durch die Erstellung von physischen Prototypen. Auf der Basis von Erkenntnissen die aus dem Bau Selbiger gezogen werden, wird schließlich ein Algorithmus erstellt. Mit dessen Hilfe wird zum Abschluss dieser Erkundung ein architektonisches Artefakt zur Veranschaulichung des entwickelten Architekturschaffensprozesses gebaut.

Abstract

This thesis engages opportunities for the automisation of architectural creation. After an investigation into historic archetypes and scrutiny of other areas of human efffort and creation as well as an analysis of current prefabrication processes the conclusion is drawn that there looms a great theoretical potential for the optimization of the currently applied processes.

In the following a prototypical model process is developed aiming at the production of an architectural artefact. Initially a three axis CNC router is built as a tool for the automated fabrication of architecture within this work. This machine's abilities' limitations represent the constraints imposed on the following development of the model process which is now advanced. The creation of physical prototypes is used to close in on the realization of potentials for improvement of the current processes. Finally, on the basis of this newly gained knowledge an algorithm is created. This algorithm is used for the creation of an architectural artefact which serves as a demonstration of the developed process for architectural creation and marks the end of this endeavor.

P r e f a b r i c a t i o n T o p o l o g i e s

"This is the real news of our century. It is highly feasible to take care of all of humanity at a higher standard of living than anybody has ever experienced or dreamt of. To do so without having anybody profit at the expense of another so that everybody can enjoy the whole earth. And it can all be done by 1985."

Richard Buckminster-Fuller, *The World Game*, 1971

Tabel of Contents

1 Introduction

In April of 2012 *The Economist* printed a special report on what has become to be called *The Third Industrial Revlution*. After the *Neolithic Revolution* that began to take place approximately 10 000 BC this first of the industrial revolutions which happened around 1800 is deemed the most significant change in human history. It was heralded by the first broad use of industrial machines to accomplish tasks that had previously been extraordinarily labour intensive. It was followed by the *Second Industrial Revolution* or *Technological Revolution* that lasted for a period of approximately half a century up until the beginning of *World War I*. It collimated in the uprise of mass manufacture and the invention and application of the assembly line with its most famous and inspiring initial product the Ford Model T. These two revolutions evened the way for today's consumption based societal system and a change in social structure that resulted in the forming of a middle class. Wealth was generated at a rate never seen before and society became increasingly urban, a development whose far reaching beneficial effects we are only beginning to understand today.

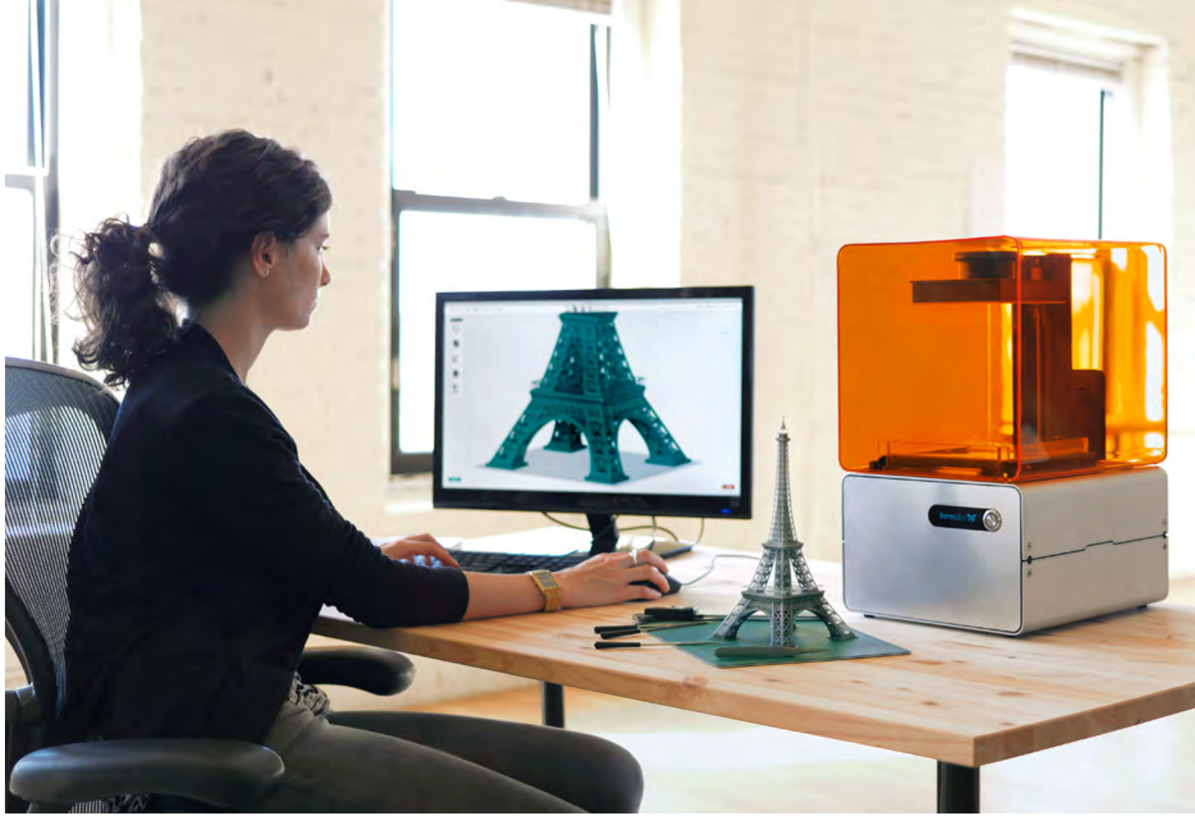
The most recently proclaimed third revolution of industry is a digital one. Again the way humans create

Figure 1: Cover of *The Economist*, Issue April 21st - 27th 2012.

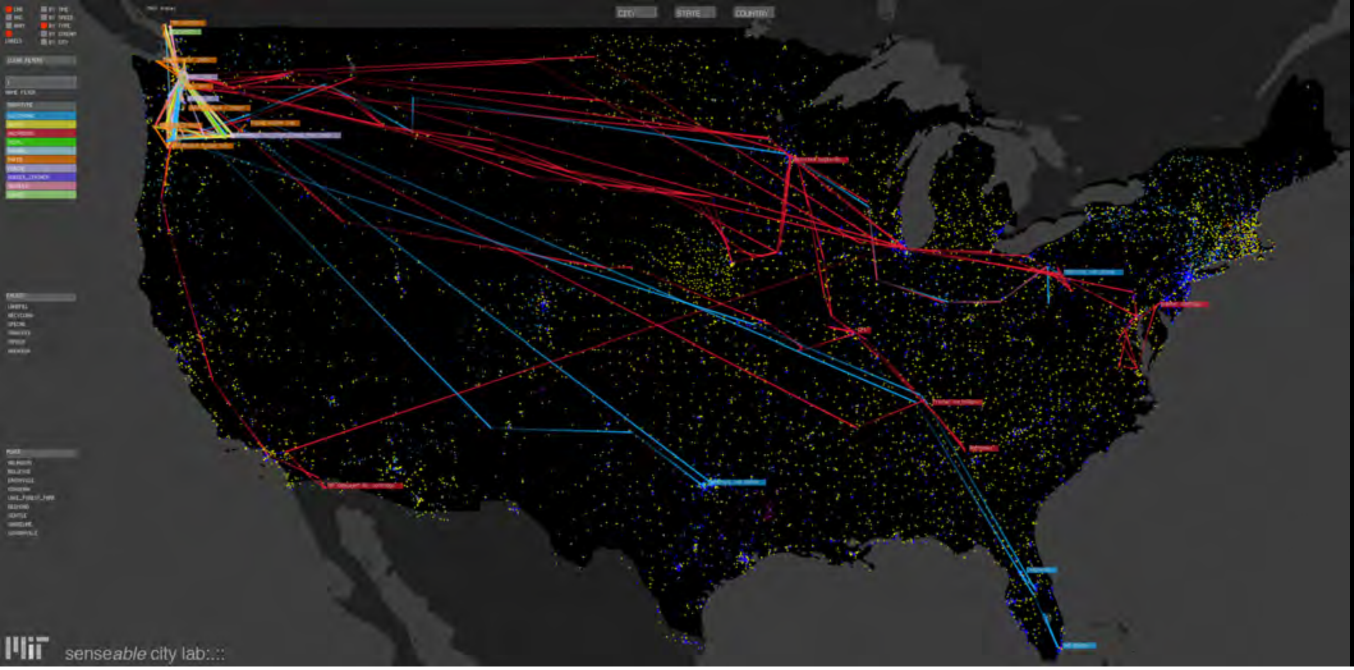
Opposite page:
Figure 2: The process of 3D printing from screen to printer to physical object. Pictured here is the Formlabs 1 3D printer.

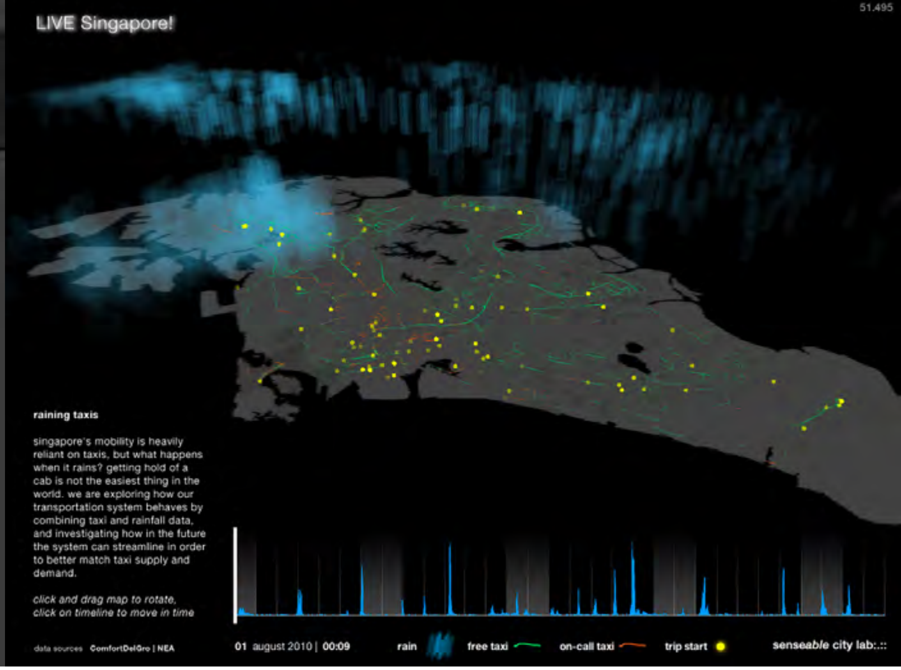
artefacts is fundamentally changing. The advent of 3D printing inspires great hopes for social reform and unimaginable technological leaps. While so forth the mass production paradigm dictated centralisation as the only economically viable means for the creation of goods, it is believed that the latest technologies will allow distributed production. As so much of human structuring of the world is dependent on the strategies used to cater to people's consumption be it for reasons of life support or leisure, the high hopes lying in a change of the nature of the networks of production become conceivable. Some economists' optimism goes so far as to expect a change in how people work together so significant that it will drastically alter the capitalist nature of our current economic system.

3D printing is the most conveyable of the myriad of technologies potentially allowing this change in artefact creation. It is used representative of all the manufaturing strategies that allow for economic non mass production. There are several processes all including some form of cumputer numerically controlled production machines that can achieve this. The underlying principle is always the use of a machine that can produce any product that fits into its margins of ability with more or less the same effort and with no or only marginal involvement of human labour. Either through subtractive or additive processes an artefact is created from a material universal to a certain number of applications by the machine. Parts produced by several machines could also be automatically combined by robots within such a process.



The foundation on which this revolution rises are the digital technoligies that have been ever more rapidly developing during the last decades and that have already recreated so many networks of human activity such as the media, commerce and communication. They have given rise to new forms of cooperation. Wikipedia, OpenStreetMap, Linux, Apache, Drupal and other collaborative software developments didn't only challenge but oftentimes dwarfed much of their proprietory and corporate competition. Education from the most renowned institution is given away for free online and crowdsourced platforms such as reddit and twitter revolutionize the way we access news. If CNC technol-





raining taxis

singapore's mobility is heavily reliant on taxis, but what happens when it rains? getting hold of a cab is not the easiest thing in the world. we are exploring how our transportation system behaves by combining taxi and rainfall data, and investigating how in the future the system can streamline in order to better match taxi supply and demand.

click and drag map to rotate, click on timeline to move in time

data sources ComfortDelGro | NEA

01 august 2010 | 00:09

ogies build a bridge across the gap from the digital and virtual to the physical world all these developments can cross over it as well, one could wish.

Computing is becoming increasingly pervasive in our physical world already. The number of artefacts that can connect to each other to exchange information is exploding and the notion of an *Internet of Things* is gaining momentum rapidly. The data these intelligence enabled objects collect and relay back to us can teach us much about ourselves and the systems we have put in place to serve us. In Singapore sensor networks and even social media feeds are used to look for weaknesses in the design of the city's transportation system, a project by MIT Senseable City Lab conducted in Seattle tracked garbage to learn about gross inefficiencies in the recycling system and analysis of telephone records quantified how much our social behaviour is influenced by political boarders. If individual people can learn to improve themselves through confrontation with the quantification of their behaviour, a trend sometimes called self hacking, it has to be assumed that societies are capable of the same.

Manufacturing has not stopped developing after the introduction of the first assembly line. Modern automobile factories are higly automated through the use of robots of all sorts. Everything moves by itself and only few people are present. The large spaces where all this happens are kept clean meticulously and storage on site is kept to the bare minimum. Preassembled components are brought in just in time when they are needed. Parts and subassemblies are slightly different

for each car that is to be built but can be easily identified with barcodes and RFID Tags.

We are putting bricks on top of each other. We will be putting bricks on top of each other forever. In light of the current situation, however, these bricks will soon be connected to the internet to tell us about how much load they convey and if they operate under acceptable parameters. They might differ from today's bricks in size and function and they will be produced by digitally controlled machines fed with data generated by algorithms that analyze our designs to determine the unique properties that each of these bricks has to exhibit for it to work. This thesis is seeking a new understanding of the notion of a brick and the process to create this new most basic module of building.

Opposite page, from left to right:
Figure 3: Visualization of the project *Trash|Track* of the MIT Senseable City Lab. Dots on the map represent different kinds of trash processing plants and landfills. Lines depict movements of trash items after they have been discarded in Seattle.

Figure 4: One of many visualizations that where made for the MIT Senseable CIty Lab's LIVE! Singapore project. Here the dataset of the percipitation information and the dataset of taxi trips are combined to show how they relate to each other.

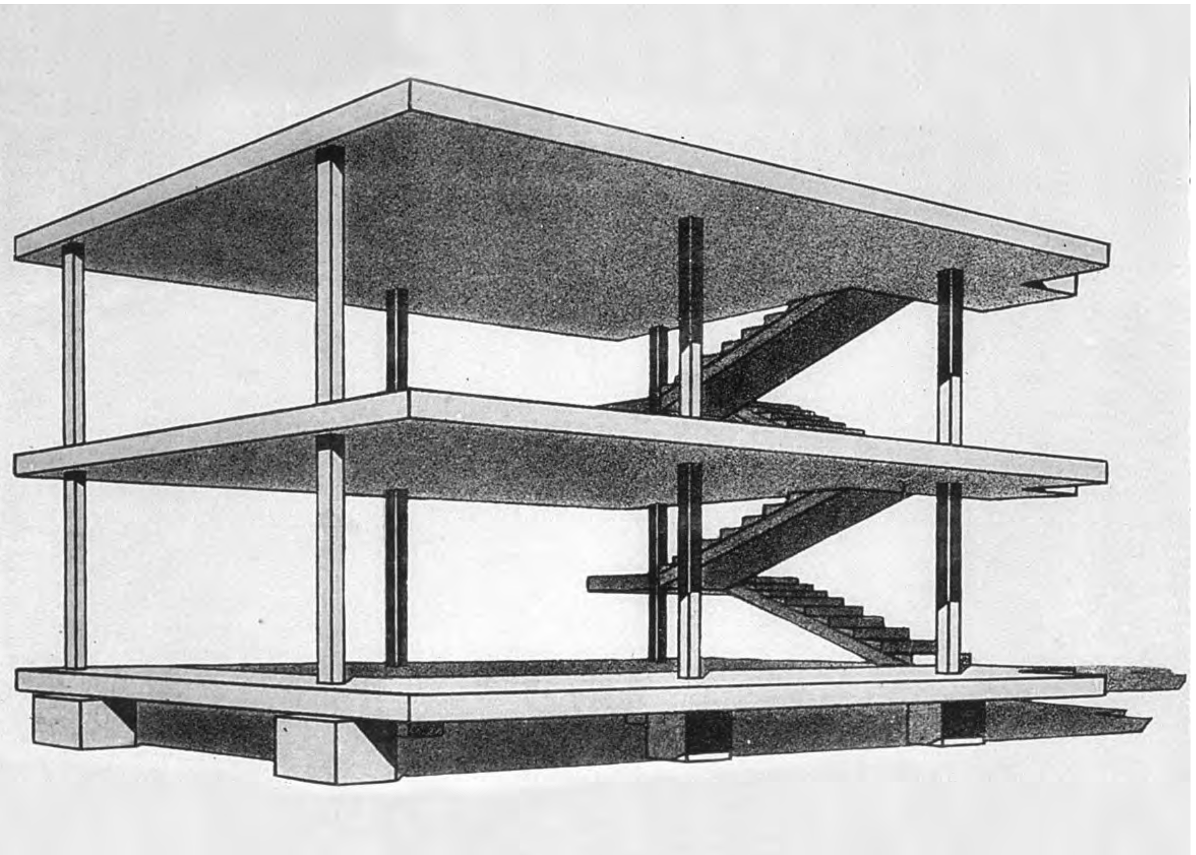
This page:
Figure 5: Tesla Model S factory floor, Fremont, California.

2 Theory

1 *Le Corbusier*

"We must create the mass-production spirit.
The spirit of constructing mass-production houses.
The spirit of living in mass-production houses.
The spirit of conceiving mass-production houses."

Le Corbusier, *Towards a New Architecture*, trans. Frederick Etchells, 1923



Le Corbusier wrote his book *Towards a New Architecture* to build a strong case for the adoption of rational processes in construction whatever they might yield. In it he observes many opportunities to adapt manufacturing processes of other fields to architecture. Ultimatley humans should arrive at a house-machine whose purpose is to efficiently provide an environment controlled habitat for us. He points out our way of thinking as the biggest obstacle in our way to get there because his idea and the onset of mass production for so many seem tantamount to "*the abandonment of true art, of good workmanship and of dignity*".

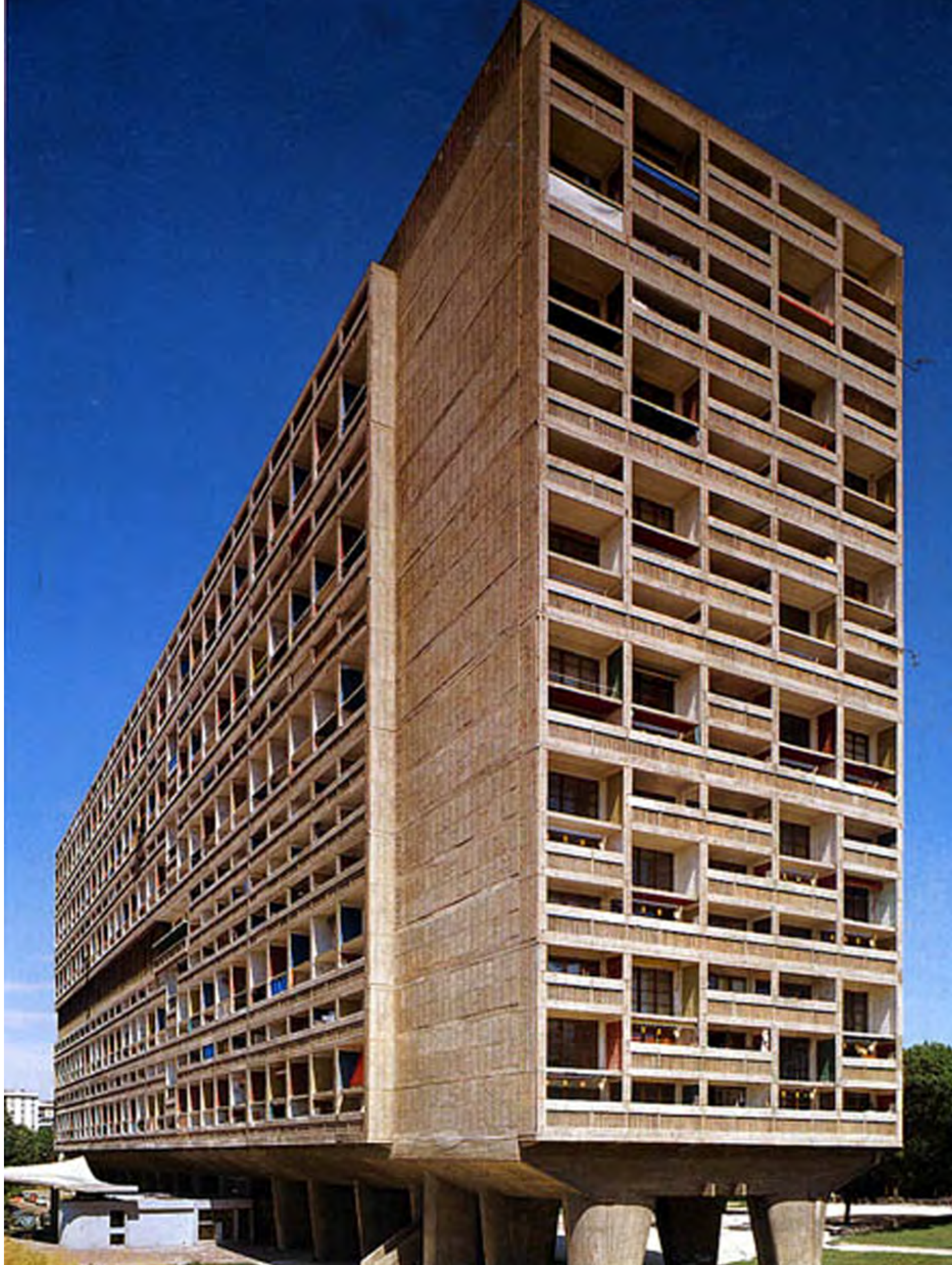
For Le Corbusier's notion of the perfect 'House Machine' to work he insists on the necessity of standards. The standard is established through a process of rational evaluation and practical testing on sure bases. Then the standard can encourage fierce competition between producers of parts, modules or whole houses. Through this, every minute detail of the design will be under intensive scrutiny ultimately leading to perfection of the product. The standard gives order and direction to human effort, he states.

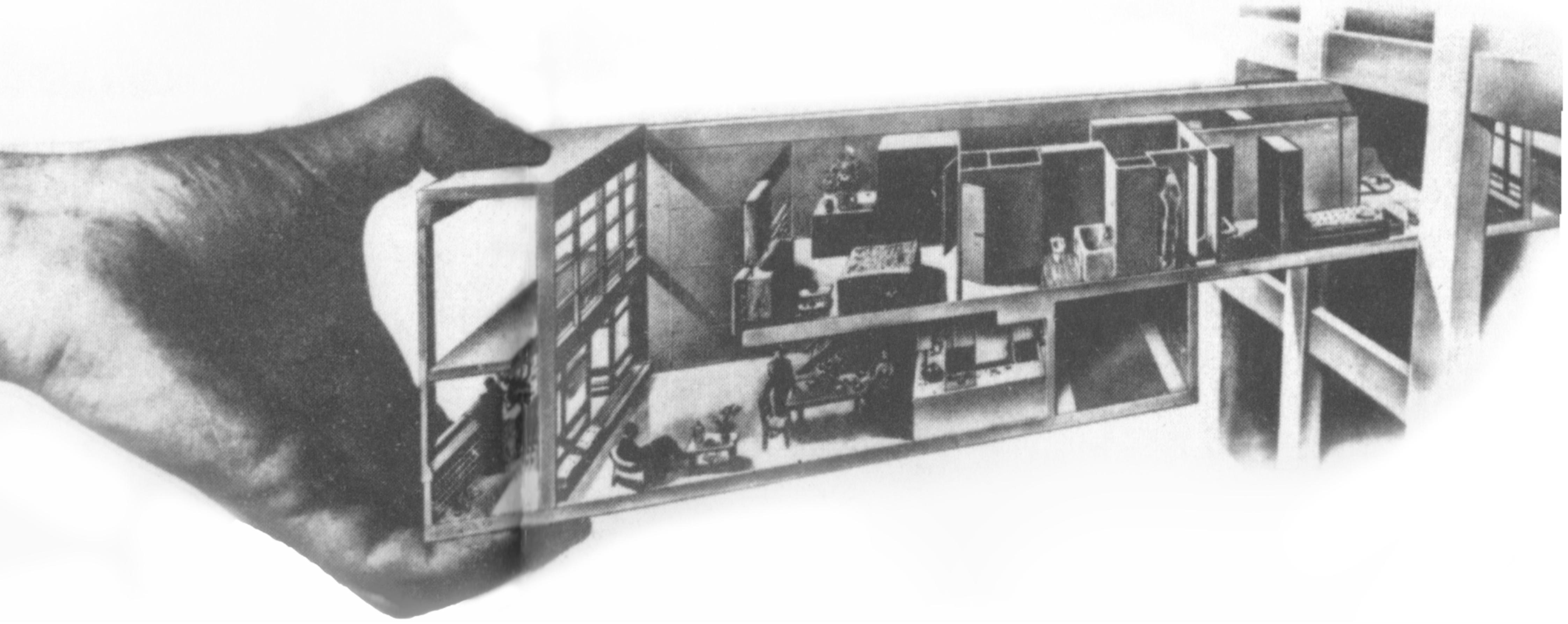
The 'House Machine', which is based on these rules, is a result of logic. This is why "dead concepts of houses" have to be removed from our minds before evaluating it, as the result of the proposed process might look alien on first glance. Le Corbusier compares it to the invention of the first plane. It was not built to resemble a flying animal, but was merely the solution to "the problem of sustaining solid bodies in the air". Analogous to that, the desired dwelling shall not be a copy of a house, but the tool to enable human life.

Together with the engineer Max Du Bois, Le Corbusier developed the 'Maison Dom-ino' as a mass produceable building out of reinforced concrete. The name 'Dom-ino' is a combination of the Latin word 'domus' (house) and 'dominos'--to reference the pieces of the game, which the parts of the building resemble. The construction of the house aimed at allowing the great-

Opposite Page:
Figure 6: La Maison Domino 1914-1915

Right:
Figure 7: Le Corbusier's Unite D'Habitation in Marseille





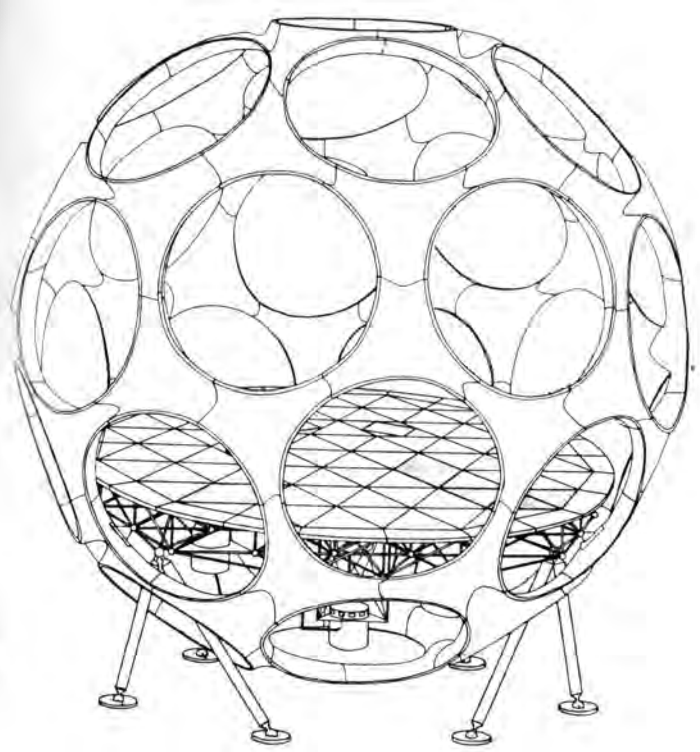
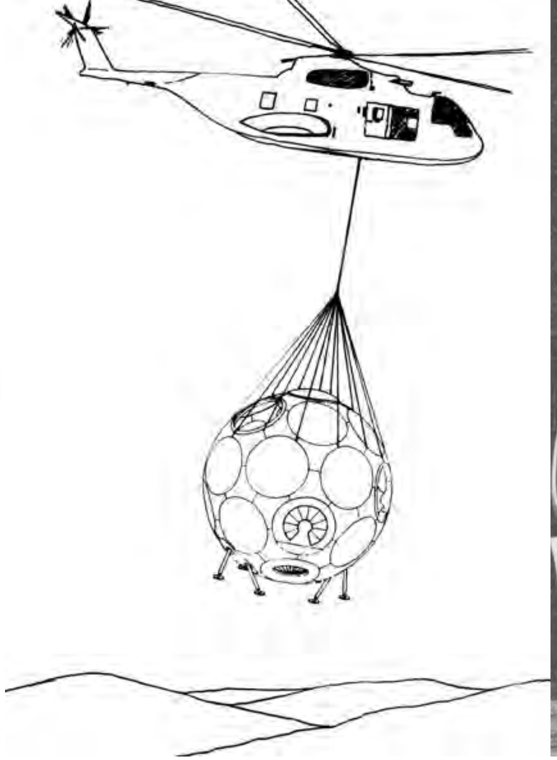
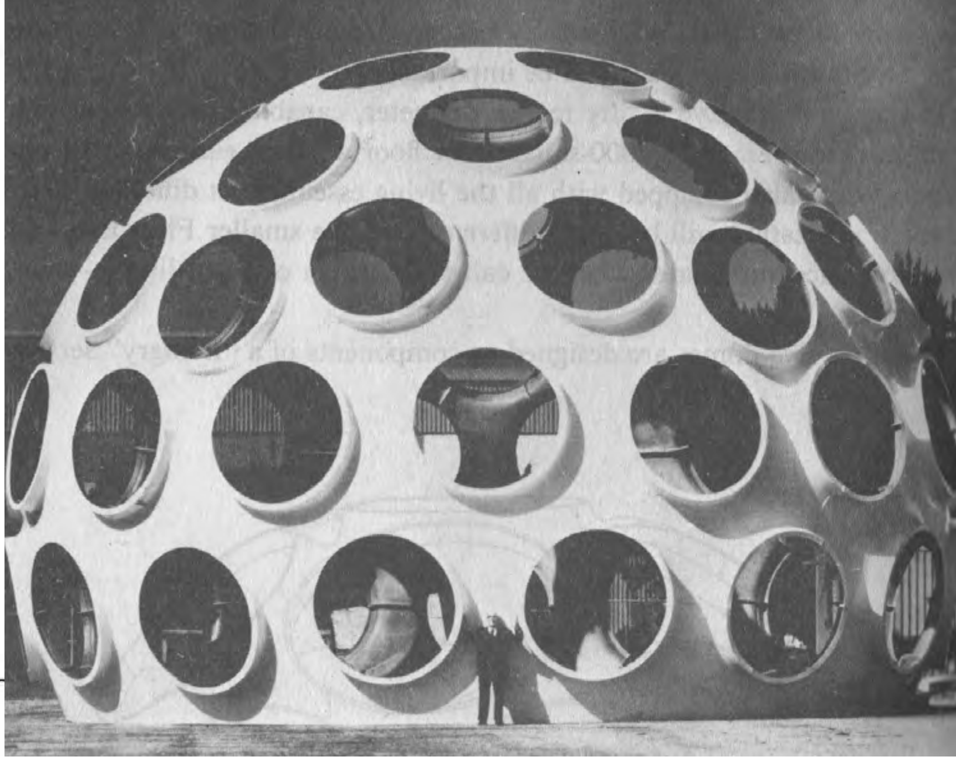
est degree of freedom for the layout of the floor plan. Structural walls as well as beams were unnecessary in its design. With *la Maison Dom-ino* Le Corbusier made a proto-architectural statement about building economics and how we create architecture that built the stilistic foundation for his later work and forestalled his later formulation of the architect's *Five Points* in his seminal work *Towards A New Architecture*: 1 pilotis; 2 roof garden; 3 free facade; 4 free plan; 5 horizontal windows.

Figure 8: Rendering of the Unite D'Habitation depicting Le Corbusier's wine rack concept for the stacking of individual habitation units.

Opposite page, from left to right:
Figure 9: Buckminster Fuller's Fly's Eye Dome in a rendering.

Figure 10: Illustration of the Fly's Eye Dome's deployment at the agency of a helicopter.

Figure 11: Fly's Eye Dome Prototype with a 50 foot diameter.

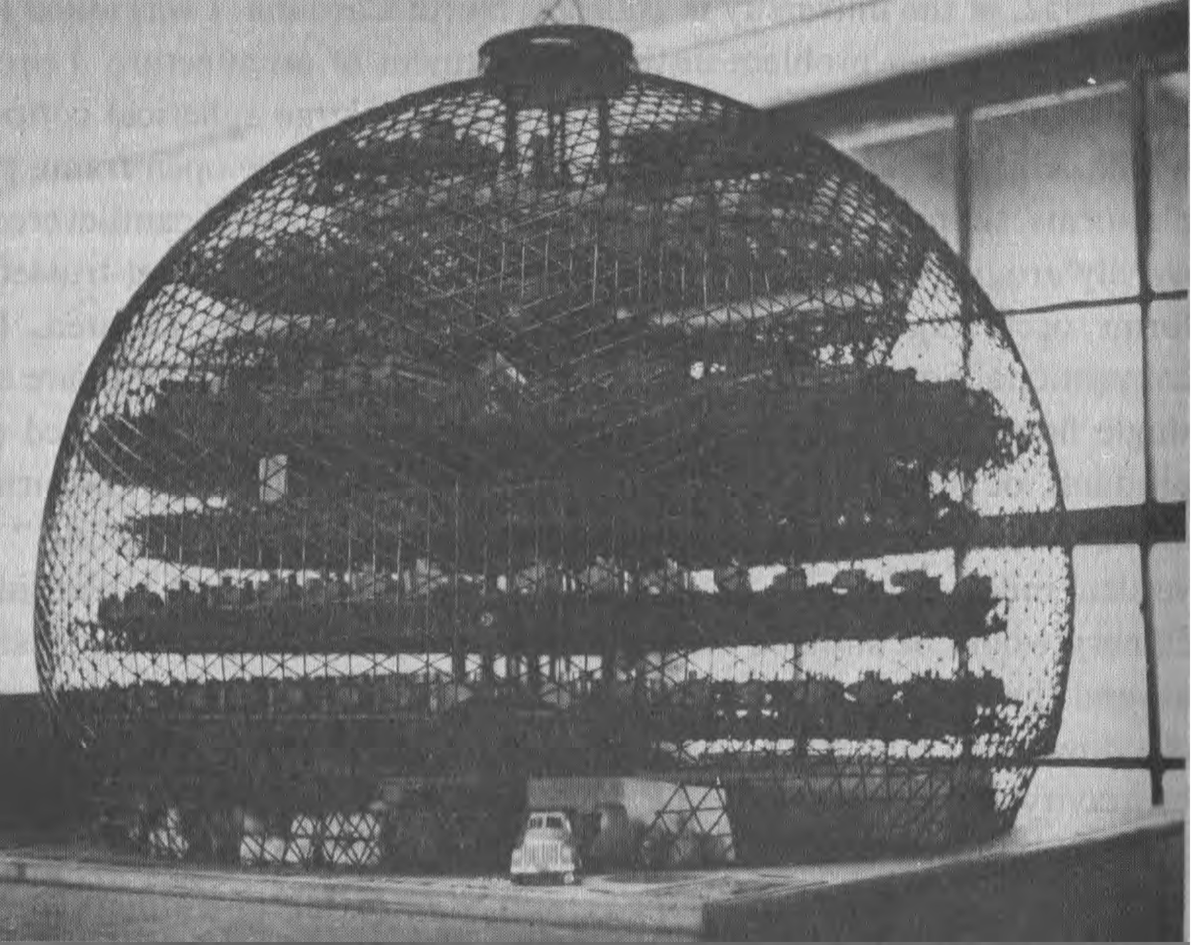




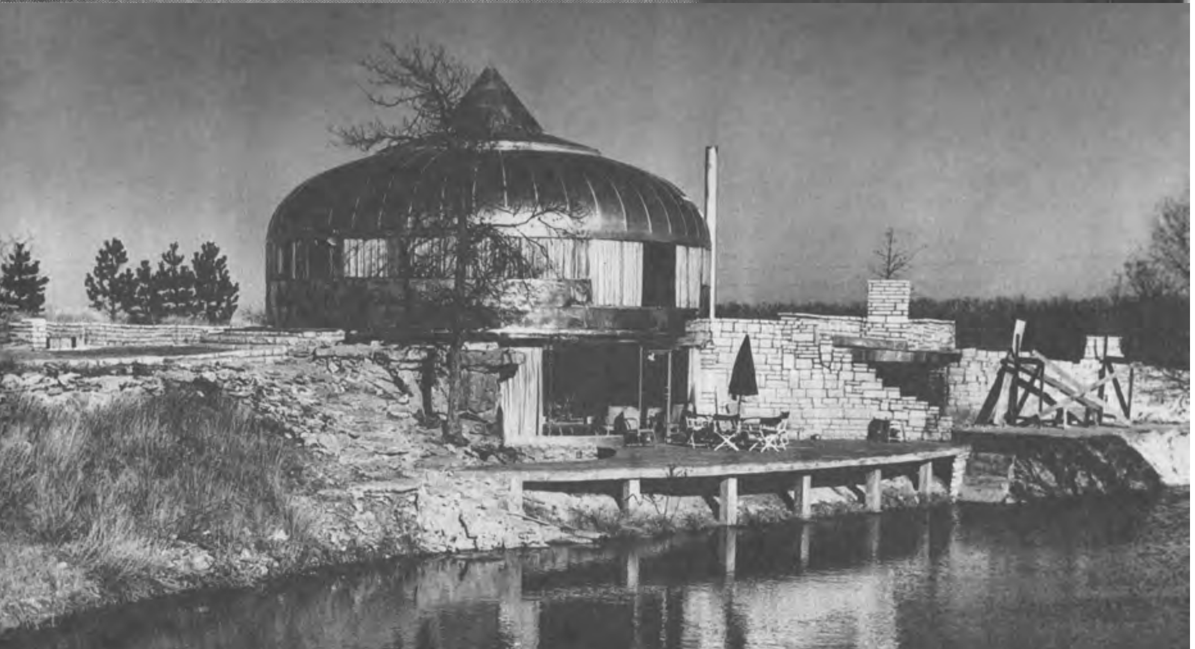
The Unité D'Habitation in Marseilles is the translation of the same thought process to the housing block. Le Corbusier designed a simple three-dimensional structural grid and filled it with living units and other functions of capsuled design following the notion of a wine rack being laden with bottles. While these modules were not fabricated individually the idea to do so is latent in the building.

2 Richard Buckminster Fuller

At the age of 32 in 1927 and after realizing his failure in the business world Buckminster Fuller decided to utilise his efforts to work towards the improvement of the situation of all humanity. Through the creation of artefacts and knowledge on the sole basis of facts he was determined to increase the planets life support capabilities. He thought of our world, planet Earth, as a limited pool of resources. that was drifting through space. This way he came to coin the term 'Spaceship Earth''. The available assets our spaceship is carrying ought to be used only in the most efficient of ways to forge artefacts for our use. Hence his tight association with the notion of doing more with less.

Buckminster Fuller conceived of many artefacts for inhabitation during his long search for efficiency in human effort. The Fly's Eye Dome (as seen in Figure 9, Figure 10 and Figure 11) is designed for mass production and light weight which is synonym with sparse use





of resources. The circular openings are meant to serve as doors, windows or mounting space for appliances such as solar cells. Its light weight permits that it is delivered by air. The fully equipped unit with a diameter of 26 feet compares to a car in price and weight. Buckminster Fuller even went so far as to invent artefacts and processes to elliminate the need for plumbing in his abodes. A steam gun was meant to serve as a device for personal hygiene and excrement was to be deprived of water and then collected for energy production in a caloric facility.

The Wichita House (Figure 13) was developed on the basis of Fuller's initial living machine called the Dymaxion house. It was planned to go into production by the airplane manufacturer Beech in Wichita, Kansas.However, in 1947 it was decided that people were not ready for buying into a concept so futuristic. The house was meant to be sold at a price that would reflect the current value of $75 000. Just as the Dymaxion House the Witchita House was envisioned by Fuller in a broader context of systemization of production and the lifestyle of the inhabitants and hence conceptualized sustainability long before the homonymous movement.

Figure 12, top: A design for an automated cotton mill in Raleigh, North Carolina, USA by Buckminster Fuller.

Figure 13, bottom: View of the Wichita House, 1948.

Richard Buckminster Fuller invented many more concepts of buildings and factories like the automated cotton mill for Raleig, California (Figure 12). Many of these structures had a geodesic dome as their hull. Fuller was convinced that humanity should use shapes that are inherently efficient structures. One of his most ambitious proposals even was to cover a part of Manhattan with a glass dome, arguing that the cost for such an undertaking would be dwarfed by the following savings through increases of efficiency in city management and environmental control.

Current Trends

1 *Aeroplane Manufacturing*

Airplanes are manufactured on giant assembly lines in huge buildings. The ability to move very large and heavy objects allows for the assembly process to be broken down into more manageable chunks. After the completion of these huge modules they are transported thousands of kilometers to the point of their final assembly. The sections are completely equipped and have to be able to stand on their own to allow for their transportation. Sections for the Airbus A380 are transported by land and sea from all over Europe to Toulouse, France. Sections of smaller planes are even transported by air. Airbus runs a fleet of A300-600ST Beluga airlifters exclusively for that purpose.

What makes this practice economically sound is that the reduction of parts joined at any one point decreases complexity to a more manageable level and allows a clearer look at the subassemblies' performances and optimization opportunities. Modules are manufactured by contractors that are choosen by their ability to reliably deliver a certain quality at the best price. The Boeing 777 is produced by more than 40 contractors spread over more than twelve countries.

2 *Car Manufacturing*

Although cars are considerably smaller artefacts than planes the gains in efficiency and quality of modularized manufacturing are just as high. There are different types of suppliers. Such that only produce very basic elements like fasteners but also several categories of subassembly producers. Some of them operate their facilities around the world from where the final assembly will be completed, but it has become a trend to have others that supply very high level subassemblies right on the premisses of the car manufacturer. Cars are becoming more complex and so the processes utilized to construct them have to become smarter as well. From

design to polishing the paint every part of the equation that results in the finished product benefits from a high degree of modularisation as it drastically reduces the individual complexity of each subassembly. It is axiomatically easier to manage the assembly of a hundred parts ten times than of a thousand parts once. The capsuled complexity within a part does not factor into that consideration.

The appearance of the assembly lines behind the processes have changed according to the new paradigm of joining as well. As many tasks as possible are delegated to robot arms and large parts are moved around autonomously by computerized trolleys that can sense their environment and avoid obstacles. The floors are spotless and all operations requiring humans have been engineered ergonomically so they can be performed with ease.

3 Prefabricated Housing

The current status of the prefabrication industry already includes a myriad of digital technologies. Manufacturing principles from other industries are being introduced and prefabrication assembly lines are continually evolving. In most cases flat elements are outfitted with almost all necessary features before they are transported to the site of construction. There are also a number of usually isolated examples such as IKEA's

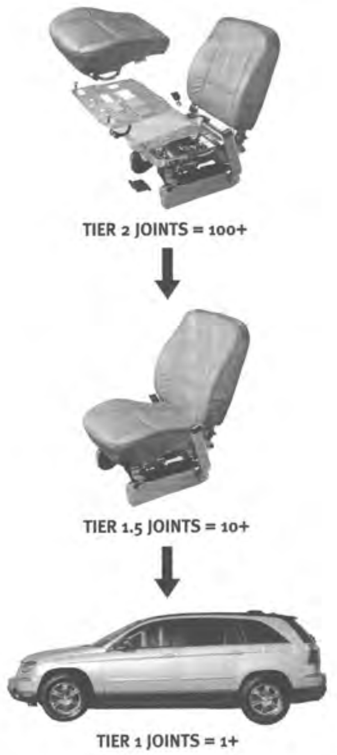
Opposite page:
Figure 14: Modules of the Airbus A380 in France.

Bottom, left to right:
Figure 15: Assembly line for the Boeing 747-8I in Everett, Washington.

Figure 16: Tesla Factory Auto Bot Fremont, California.

Figure 17: Tesla Factory Fremont, California.

Right:
Figure 18: Hierarchy of subassemblies reducing number of joints.



Boklok prefabrication houses where three dimensional modules are agregated but they are the exception.

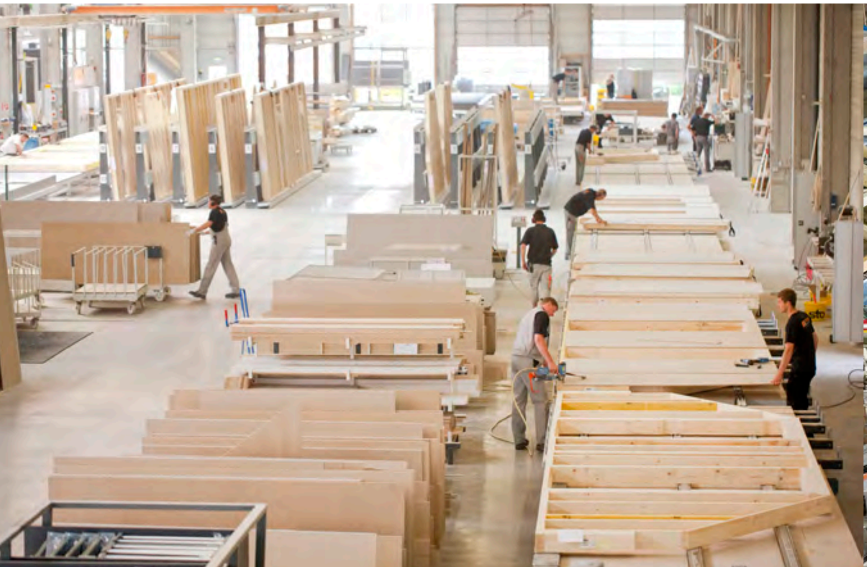
Both instances utilize revolutionary processes borrowed from industries whose products have evolved substantially even during the shaping of these processes. However, they apply them to the creation of a product that they do not allow to evolve for diverse reasons and which exhibits no sign of the underlying revolution. This gives rise to the assumption that there must be a huge untapped potential for increases in prefabrication's efficiency.

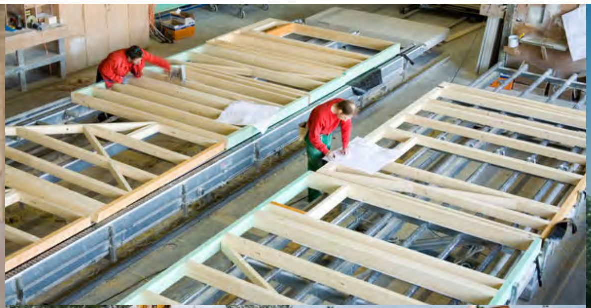
From left to right:
Figure 19: Production facility of FingerHaus GmbH in Frankenber, Germany

Figure 20: ELK Fertighaus Gmbh factory, Schrems, Austria.

Figure 21: ELK Fertighaus Gmbh construction site.

Figure 22: Example for a house by ELK Fertighaus Gmbh





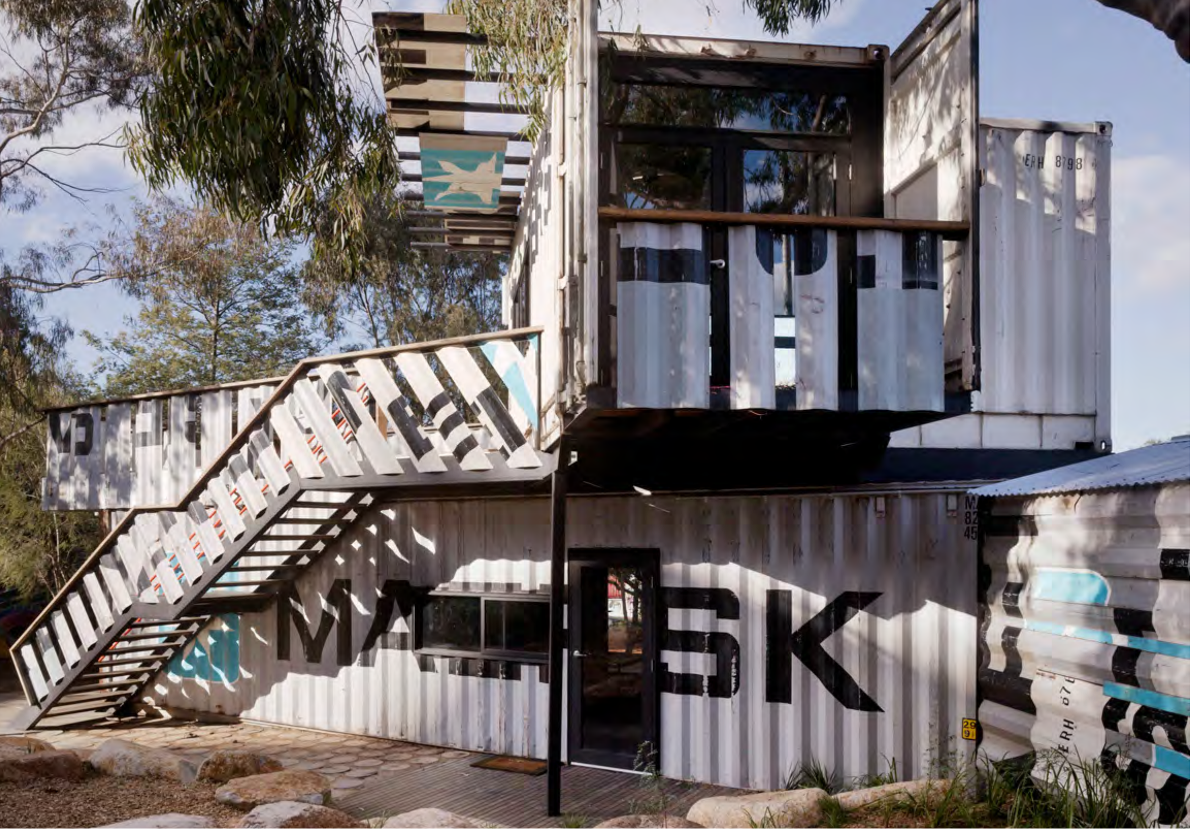
Proposal

1 Main Idea

The main focus of this thesis is to experimentally explore the opportunities of full automation and mass customization in architecture, utilizing currently available CNC-technologies. Architectural creation consists of a form finding process and all the processes that lead to the existence of the envisioned architectural artifact. While the first is a process of human creativity the latter are predictable processes based on physical laws and regulations. It is the opinion of the author that at the maximum degree of architectural automation only the form finding process is achieved by a human while all other processes will be taken over by algorithms and CNC controlled tools.

One can argue that the quality of architectural experience relies strongly on the detailing of the architectures finish. Today an architect designs details through examples leaving certain ones to best practice solutions. In the process outlined above she defines rules along which all details are automatically solved. This gives the architect much greater power to imprint a building with her idea of material and detail in every corner. While it seems like the process of automation removes the designer further from the resulting architecture it actually yields the opportunity to get much closer to the

Figure 23: Freitag Flagship Store 2006



This page:
Figure 24: Children's Activity Center by Phooey Architects 2007

Opposite page, top to bottom:
Figure 25: Habitat 67, completed 1967 in Monteal, Canada by Israel born architect Moshe Safdie as part of the 1967 EXPO. The project was the subject of the architect's Master's Thesis at McGill University.

Figure 26: Nagakin Capsule Tower, completed 1972 in Shimbashi, Tokyo, Japan by architect Kisho Kurokawa during the Japanese Metabolism

processes of creating it and that without the need for an elaborate drawing staff.

As code becomes the new language to define architecture existing open-source paradigms for software can be utilised to improve it. The new process could be thought of like conceiving of a shape and then using preexisting shared sets of code to skin it. As this would also imply the materials used in construction, local solutions could be implemented simply by building on code that has been successfully applied in the area previously and that harnesses regionally available resources and human skills.

2 *Constraints*

A number of guidelines have to be set in place to give the process meaning. As there are infinite possibilities to solve the problem that is the conversion of a shape to all the parts necesary to construct it, constraints have to be imposed. They ensure a meaningful outcome of the experiments of this thesis.

The goal is the modelling of a possibly complete cycle from design to finished construction. This consists of the modelling of a shape, the CNC fabrication of all the parts necessary to build the shape, the creation of an algorithm that converts the shape to all the parts necessary to build it and lastly the assembly of the prototype. One additional step that also has to be mentioned is the building of a suitable and capable enough CNC-machine.

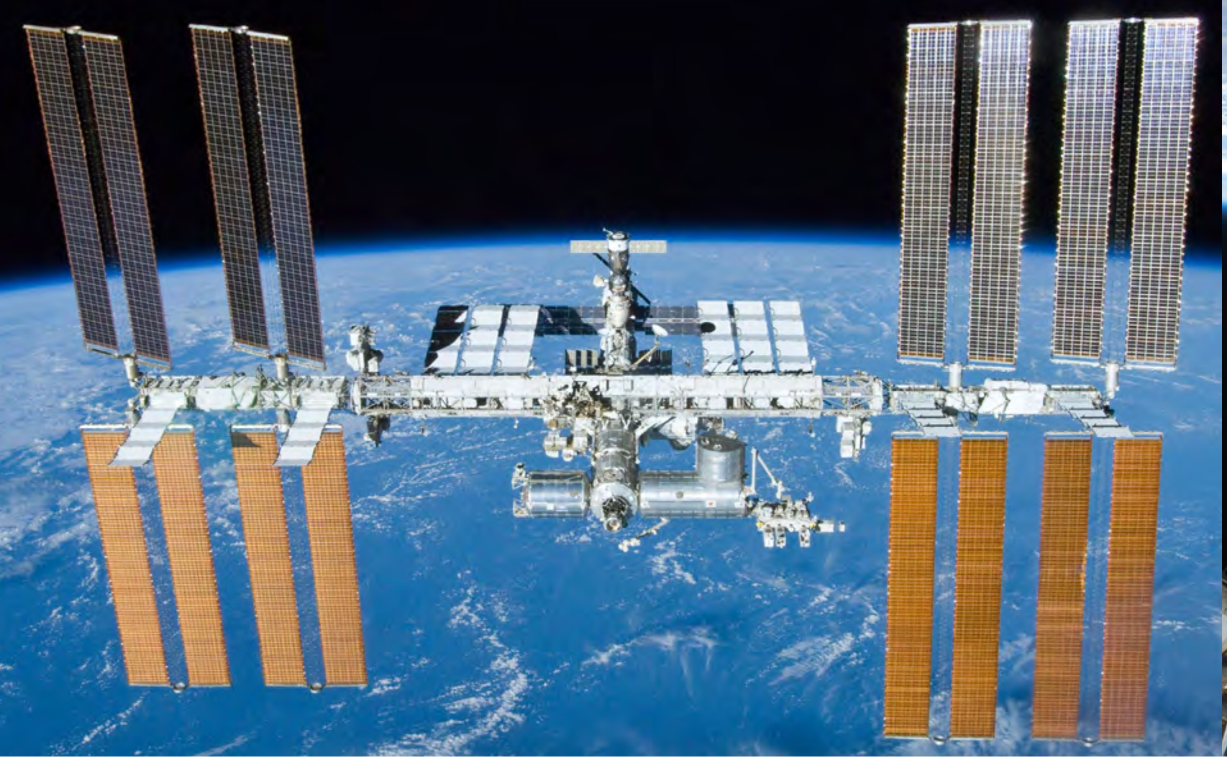
To keep complexity to the necessary minimum the main constraint is the limitation of the complexity of parts used to build the shape. The algorithm will only create two dimensional parts. While there are numerous CNC machines capable of producing highly complex 3D geometries such as a myriad of different 3D printers and 5+ axes CNC mills this limitation allows the use of one of the simplest machines. A three axis mill is able to produce all the necessary parts under these preconditions.

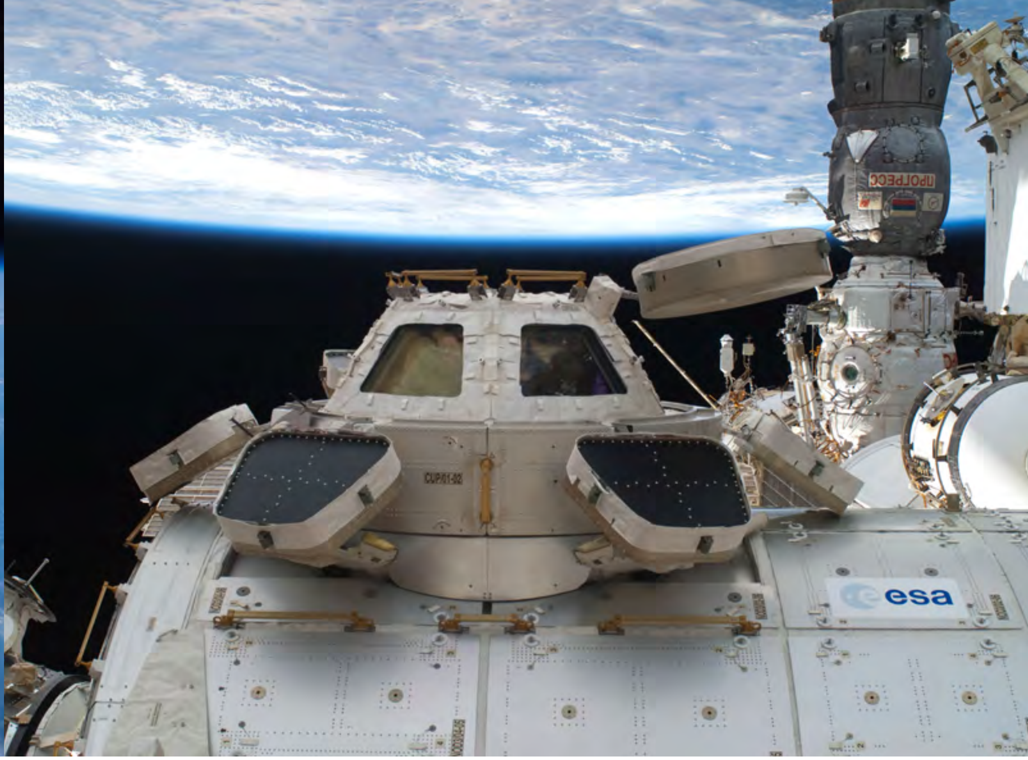
All connections between parts have to be of a non-permanent nature like screw joints or key and slot type connections. While this offers more convenience in construction it also gives the resulting artefact a high degree of adaptibility and modularity. The latter will be described in more detail in the following section on interfaces and modules.

3 Interfaces and a fresh perspective of modules

Modular architecture has always had the spectre of monotony floating above it—in some cases it managed to be exciting none the less, but that is not the point this text wants to make. In most cases modules have been interpreted as boxes that fit together much like standard LEGO® bricks to form complete buildings.

This paradigm can be observed in shipping container architecture where intermodal freight containers are recycled as building blocks. Sometimes they are positioned for temporary uses such as offices at construction sites, but in many cases they have been refitted for



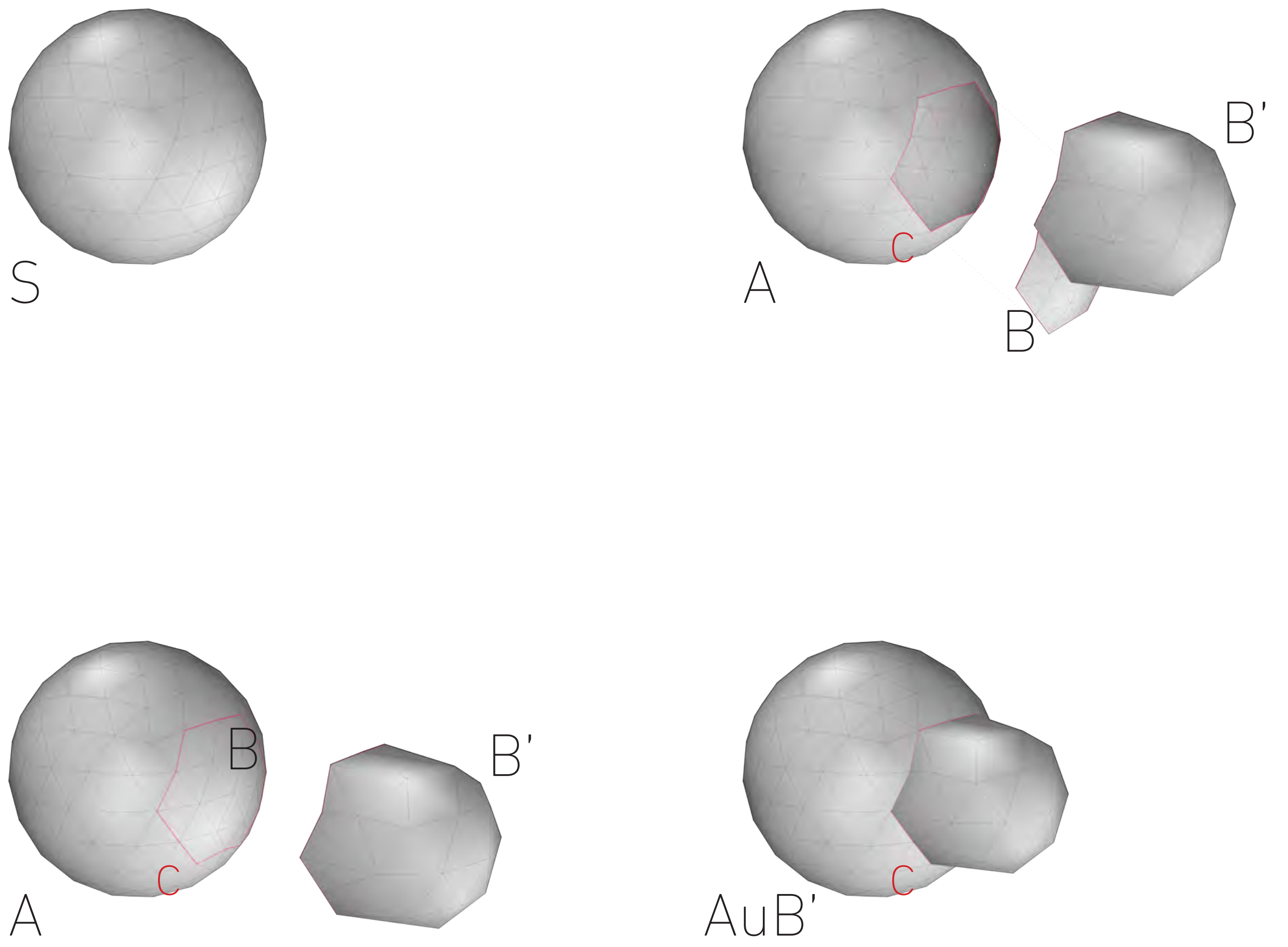


permanent constructions of varying purposes (Figure 23 and Figure 2).

Alltogether Architects have looked at modularisation very much as a strict extrapolation of architecture's original module, the brick. Hence architectural modules are very stable in their shape and size. Mass production processes are exploited to create the same or roughly the same thing over and over again cheaply, efficiently and to a high quality standard. Then the products of this process are transported to a site where they are finally stacked to a building. While this process is certainly rational it only occasionally creates interesting aesthetics that profit from it. Two such examples can be found in Montreal, Canada (Figure 4.3) and Tokyo, Japan (Figure 26).

This thesis wants to propose a different understanding of the term module that builds on concepts derived from space engineering and software development.

First another term used in software development has to be introduced—interface. An interface sets the rules for communication between different parts of a programm. It basically defines what the communicating parties are to expect of each other. That includes information they have to provide as well as what they have to be ready to receive. This way the interface can be seen as a preset contract that entities can commit


S
A
B'
B
C
A
B
B'
C
AuB'
C

to, to allow themselves to pursue a certain goal. The concept also allows a high degree of adaptability of the program as a whole as it permits anyone unfamiliar with the software's details to replace parts only by knowing about the interfaces they use. This also means that new parts can have completely different behaviour than their predecessors as long as they oblige to the rules of interfaces that are used to communicate with them by older parts of the software.

Secondly space architecture. The International Space Station ISS is a modular space station. This means that it is composed of modules docked to eachother. The modules can be replaced or extended by others which allows easier changes of mission. The modules vary in size, shape, objective and also place of design and manufacturing. Despite these complications they all fit together forming one airthight connected structure in outer space. And while that alone is a big achievment they also manage to form networks for the distribution of power and all other necessary utilities as well as transfer physical loads. The reason for this is that while shape and size of the modules have been left to their designers who mold them to their objectives there are strict rules for the modalities of connecting to other parts of the station—in software development terms, all modules use the same interface.

While this physical representation of an interface still seems banal it already shows that it enables modules to differ in every other aspect. The interface is defined physically as well as in terms of properties that describe what it allows to interchange between modules. In the case of the ISS that includes air, electricity, data, people, objects up to a certain size, etc. Its physical definition is fixed. All interfaces have the same physical appearance.

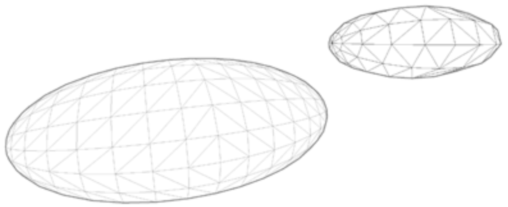
This thesis proposes an additional level of abstraction which will be the parameterization of the interface's physical properties. If a building is the combination of its non-permanently connected parts that all are interfacing each other, each continuous subset B of this set S of parts is interfacing the subset A=S-B along a distinguished interface c. In this sense set is synonymous to module. hence the above is a floating definition of module. Further it can be said that module B can be replaced by an infinite number of other modules B' as long as they are compatible to the interface c.

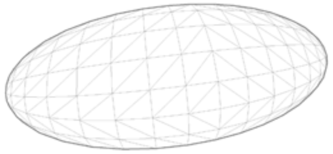
Page 26:
Figure 27: The International Space Station in orbit around the earth.

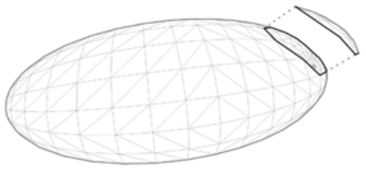
Figure 28: The ISS's Cupola is an observation window for ISS personal. It showcases the modularity of the ISS. The Cupola is attached to the same kind of interface used to connect all of the ISS's modules.

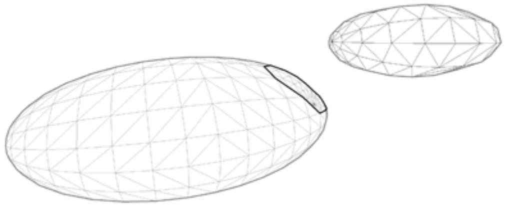
Page 27:
Figure 29:A universal definition of the module.

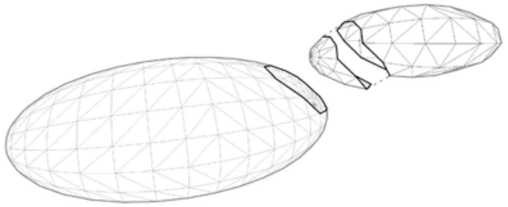
Opposite Page:
Figure 30: The steps of interfacing two volumes by the principle of universal modularization.

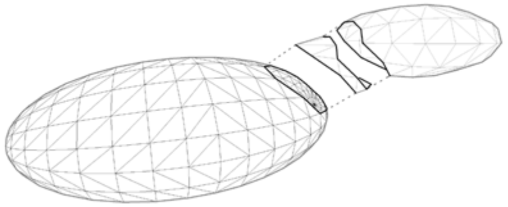

1

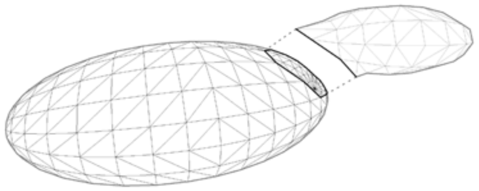

2

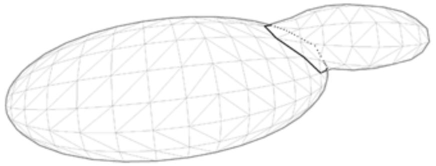

3


4


5


6


7


8

Steps

1 Creation of a CNC tool that has the capacity to produce a prototype of the system that is to be developed.

A three axis CNC router was chosen as the most economically viable option concerning acquisition and operation for the given purpose. It is mostly constrained to two dimensional operations and in consequence helps to keep the project within realistic bounds.

2 Experimental approach to a building strategy universal enough to be capable of creating any given geometry.

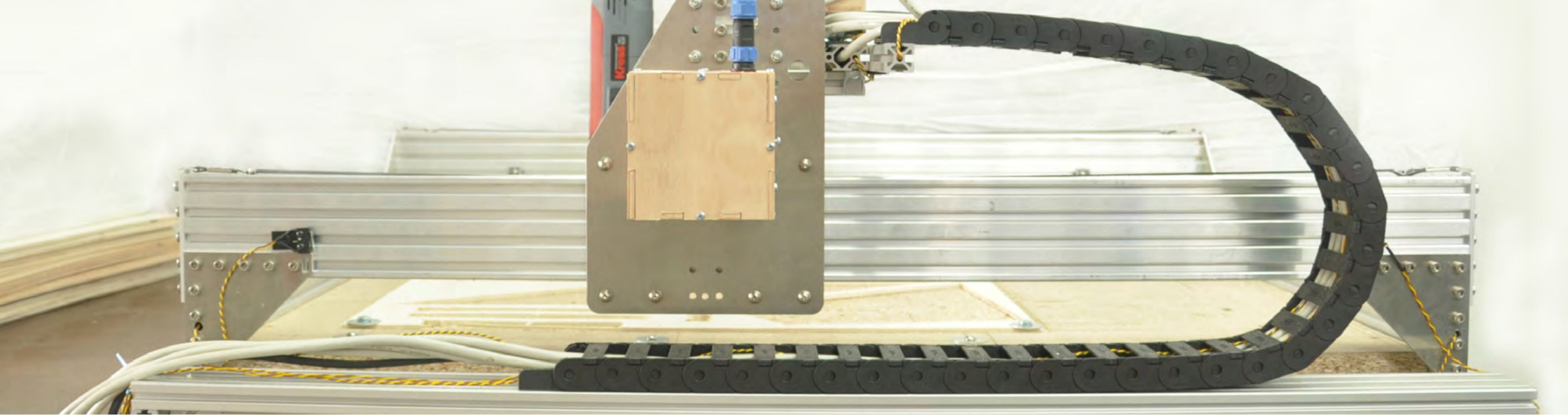
Prototypes were created to close in on a feasible solution to the stated problem.

3 Programming of an algorithm to automate the creation of necessary building parts for the assembly of said geometries.

4 Creation of a representative complete prototype to further illustrate the concept.

5 Exemplary extrapolations to show possible next steps in the developing of the underlying idea of automation in architecture.

3 Building the Tool



Components

1 Structural Frame

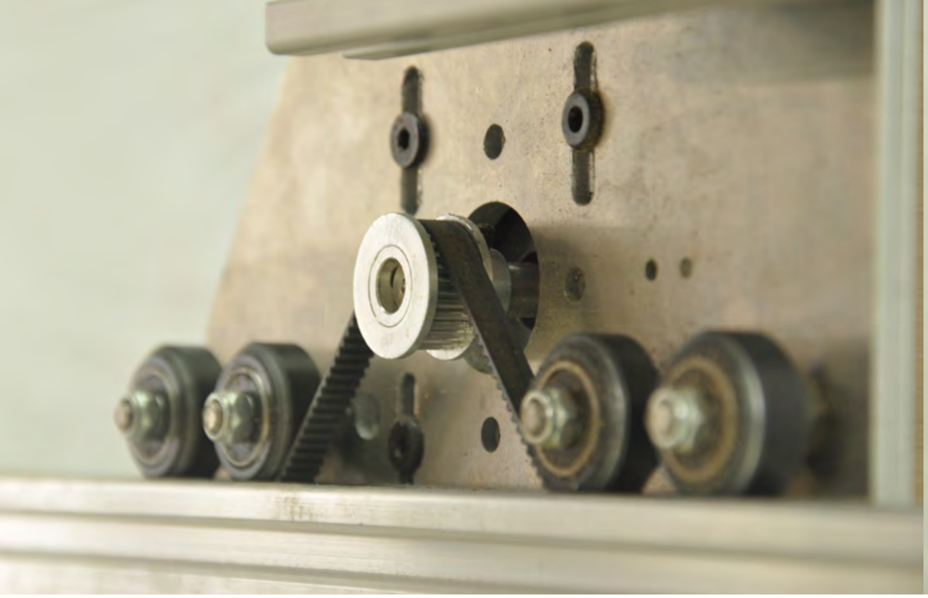
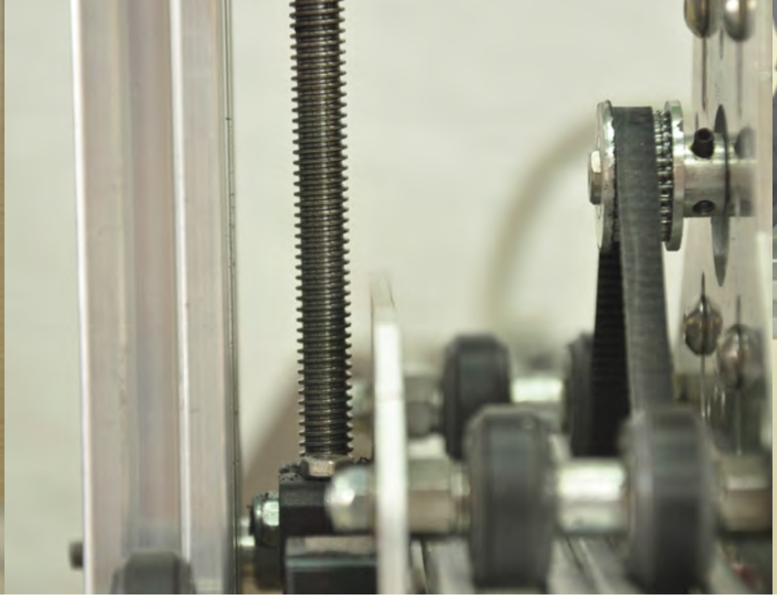
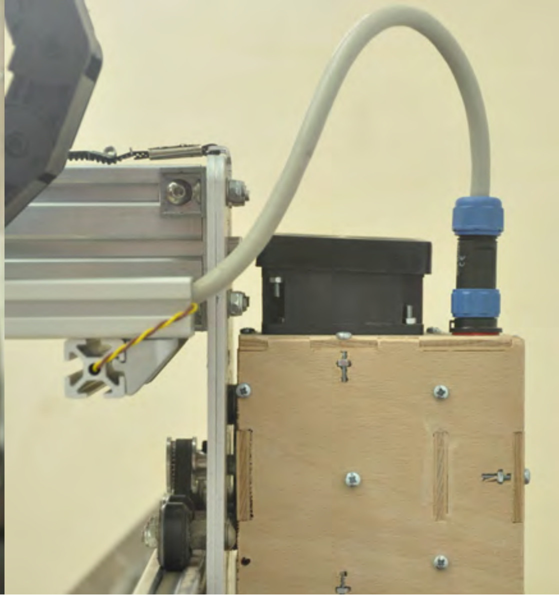
The main structure of the CNC router consists of special extruded aluminum profiles that are joined mostly by screws. Special nuts are used that slide into grooves on the sides of the extrusions to give hold to the numerous screws and bolts that hold the frame together. Perforated plates and angle brackets give it additional rigidity.

There is a rectangular base frame that serves as the main source of bracing, substructure for mounting the wasteboard and railing for the gantry that carries the spindle. It is made up of four high profiles that form the rectangle and several additional extrusions that serve as bracings. This makes it the biggest connected part and the only part of the router that does not move.

As mentioned before, on top of the main frame sits the Y-Axis gantry. It can slide back and forth along the two main base profiles pointing in X-direction that it grips from the sides with wheels fitting into these profiles' top and bottom grooves.

2 Motors and Motion System

The router is set in motion by a total of four high powered stepper-motors. They are classified as NEMA 23 and produce a torque of 4.5 Nm. NEMA stands for National Electric Manufacturers Association which is an organizational entity dedicated to the creation of

technical standards for the electroindustry.[1] The standardization document concerning the motors used for the router is called ‘ICS 16-2001’.[2] The number in the designation indicates the size of the motors casing. [3]

The gantry that moves along the X-axis is propelled by two inversely synchronously operating stepper motors mounted to either side of it. The motors turn pulleys to drag the gantry along timing belts that run along the base’s profiles in X-direction. The belts are held at constant tension by tensioning springs at each of their ends to avoid any backlash in the gantry’s movement.

The gantry carries another guided sled that moves along the machine’s Y-axis which is the gantry’s profile’s direction. Very simliarly to how the gantry grips the base frame’s rails at each of its sides, the Y-axis sled locks into the gantry’s extrusion from top and bottom

1 National Electrical Manufacturers Association, “About the National Electrical Manufacturers Association,” http://www.nema.org/About/pages/default.aspx (accessed May 15 2015)

2 National Electrical Manufacturers Association, “All Standards (One Page),” https://www.nema.org/Standards/Pages/All-Standards-One-Page.aspx (accessed May 15 2015)

3 National Electrical Manufacturers Association, “NEMA Standards Publication ICS 16,” Rosslyn 2001

Figure 31, opposite:
Thr router’s main frame.

Figure 32, top, left:
One leg of the gantry sliding along atop the main frame. The belt is fixed at either side fo the frame

Figure 33, top, middle:
Y-axis and Z-axis motion system

Figure 34, top, right:
Motor in custom-made housing with 8-pin connector and active cooling.

with the same type of ball bearing wheels. A third motor is mounted to the sled and moves it along a timing belt much like the two previous ones create movement along the X-axis.

Movement along the machine's third and final axis is created differently. Instead of a belt-driven system a threaded rod is used to drive the spindle up and down. The rod is mounted vertically on the sled and is connected to the fourth and final stepper motor at its top with a special coupling. By rotating it the motor drives it up and down in a block with a threaded hole that is mounted to the sled at a fixed position.

All four motors rotate their shaft by 1.8 degrees with each step which gives them a resolution of 200 steps per rotation. Through a process called microstepping this number can be drastically increased giving the motors up to the 200 fold of their original resolution. This allows control of the machine movement at a scale of one hundreth of a milimeter and lower.

3 Spindle

There are many different spindles for many different purposes ranging from high powered water cooled spindles for metal working to lower grade spindles with a few hundred watts of power and the size of a handheld tool like a Dremel® Multitool. However there are several other factors next to power that are significant for the selection of a spindle.

First of all it is most important that a spindle can be set to appropriate RPMs. They are one of the factors at play resulting in the chip-load value that is determinant for the quality of a cut or milling process. The wrong setting can result in frayed or uneven cuts. It also greatly influences the amount of heat generated by the milling cutter and in consequence its wear and possibly additional effects on the workpiece. In the worst case too much heat generated by the cutter can result in fires.

More complex spindles' RPM can be controlled by a microcontroller in accordance to factors like the current feed-rate—i.e. the speed at which the tool moves relative to the workpiece. This feature can be the enabling factor to achieve a higher degree of automation in manufacturing parts with a CNC Router as it allows the operator to post several different tool operations that require different RPM settings to the machine at once without the need for pauses to reconfigure the spindle.

Higher quality spindles also boast the ability to stabilize their RPM independent of the loads they have to overcome. While lower quality models' RPM might decrease and increase significantly due to differences in feed-rate, cut-depth or material irregularities during the same operation those that can vary the power output according to fluctuating resistance ensure a homogenous milling quality.

Allthough stepper motors are used to propel the router's axes which allows for discret actuations without attaching sensors to them, it is necessary nonetheless to get signals about certain states of the machine. Some of the sensors are necessary for convenience in operation and better automation and others deliver readings critical for safety during the mill's operation.

The actuators of the router's axes exert a substantial amount of torque, very probably sufficient to destroy the machine by moving an axis further than it is supposed to move. The effects would be catastrophic to the device as well as potentially harmful to its operator and surroundings considering that powered parts like the running spindle could get loose and be sent flying through the air. In consequence the most important sensors to start with are those that give feedback about an axis having reached its maximum position. They are named limit switches and should be placed at all points where a motor could move an axis beyond the bounds of the router. The effect of one of these switches being triggered is usually tantamount to pressing the machine's emergency stop button.

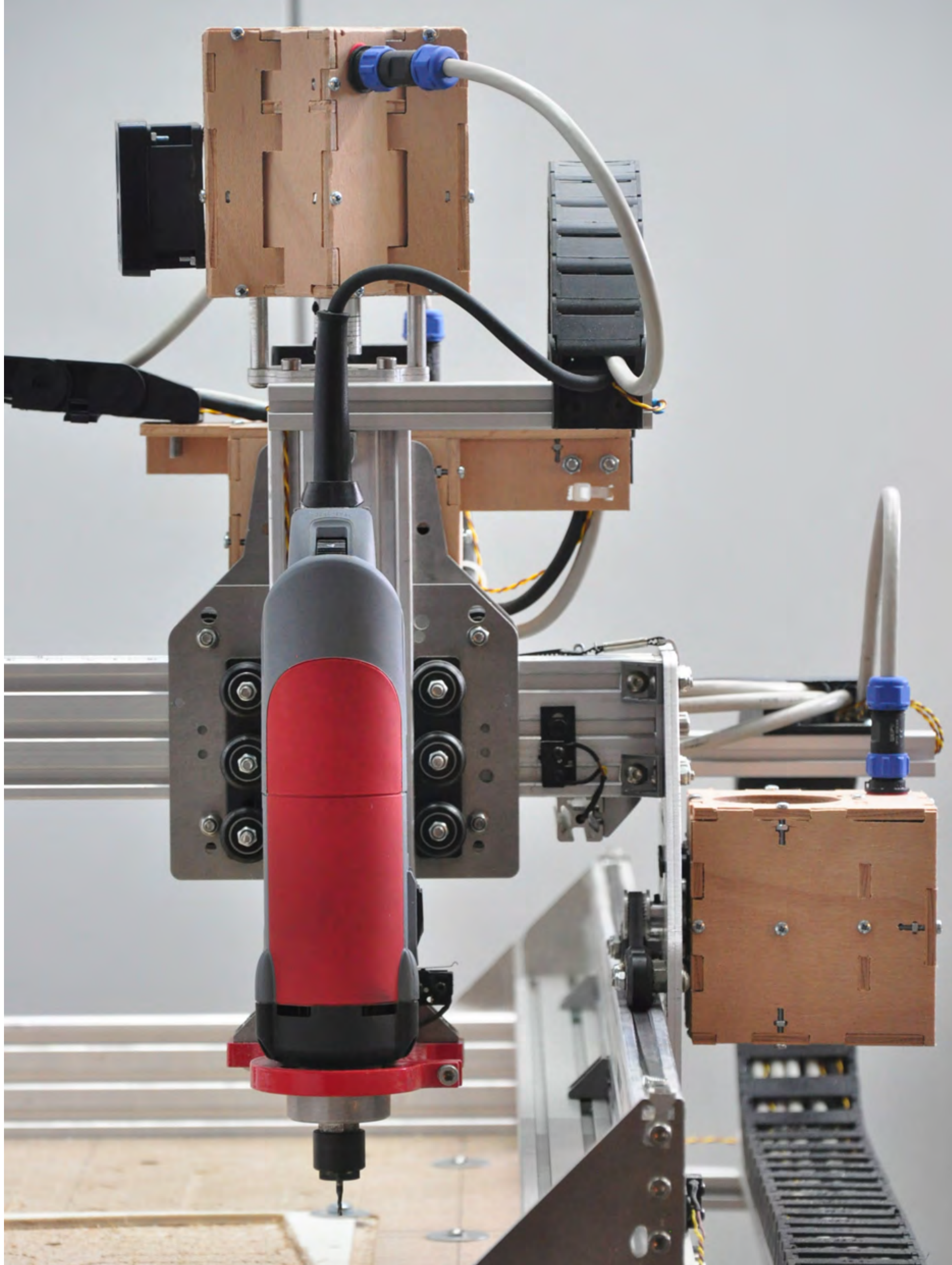
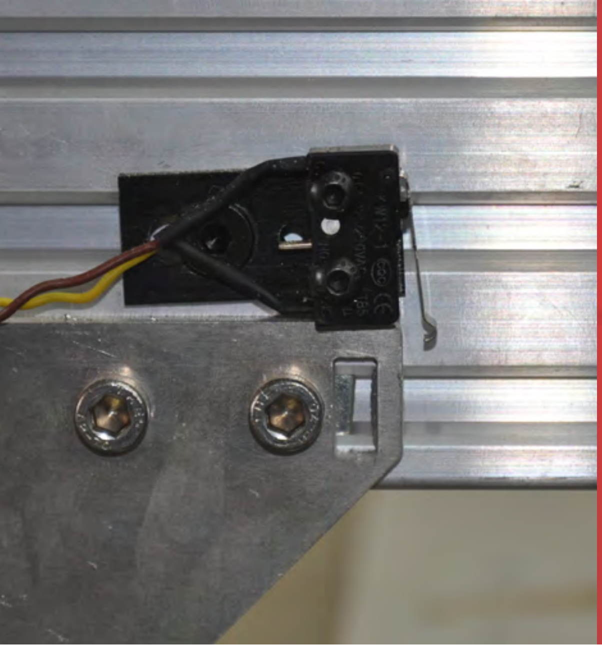
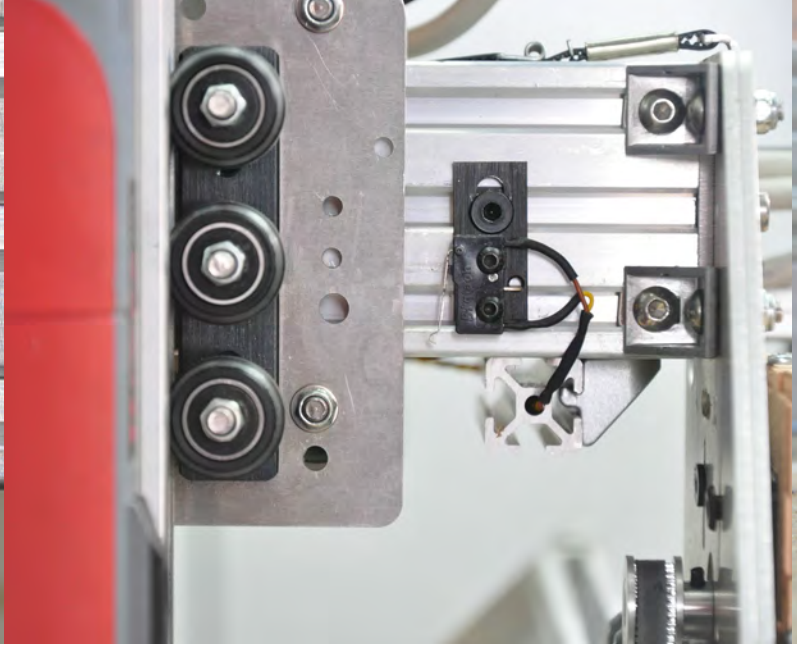
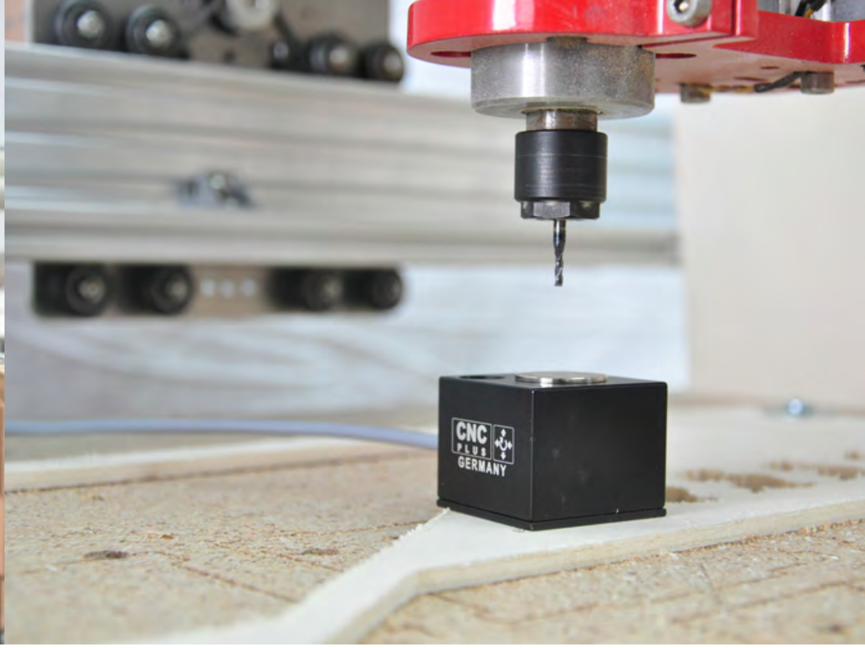


Figure 35:
A KRESS FME 1050-1 spindle mounted to the router ready for operation. Its power chord is routed through the same cable carrying tracks as the wires going to sensors, stepper-motors and other devices on the structure. Proper cable management is of paramount importance for smooth operation of the router.

Another set of sensors that is necessary for smooth operation is allowing the mill to find its zero position on every axis. For this purpose very high precision sensors should be used to enable the correct repositioning of the tool after any reboot or mishap during operation. There is usually at least one indicator for every axis that can simultaniously serve as a limit switch. On a gantry router that has two motors moving one of its axes there have to be two sensors dedicated to that particular axis as these motors are capable of independent movement and hence could cause the axis to move into a state not perpendicular to the other axes.

The last sensor that will be discussed here is the touch plate. Its purpose is to determine the cutters z-position in reference to the workpiece before starting an operation. In its simplest implementation it consists of a grounded plate of conductive metal and a metal clamp

Figure 36, top, left:
A limit and homing switch on the router's X-axis. It is a simple switch with a prolonged lever.

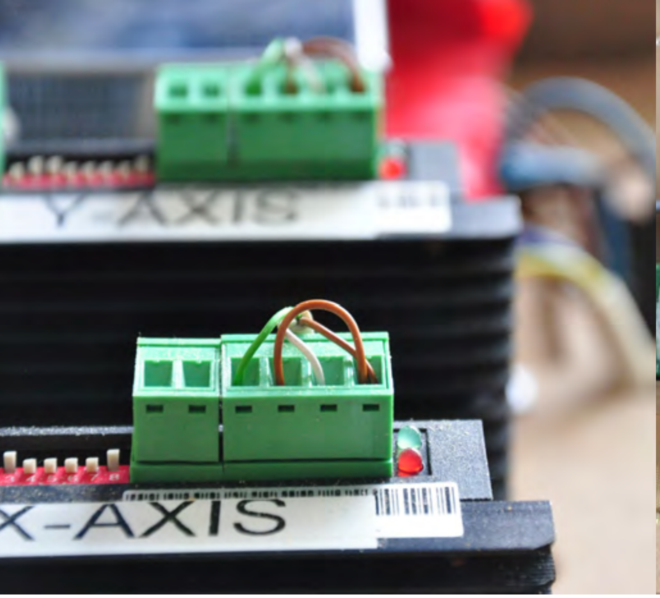
Figure 37, top, middle:
The sliding carriage moving along the Y-axis moving close to its homing switch.

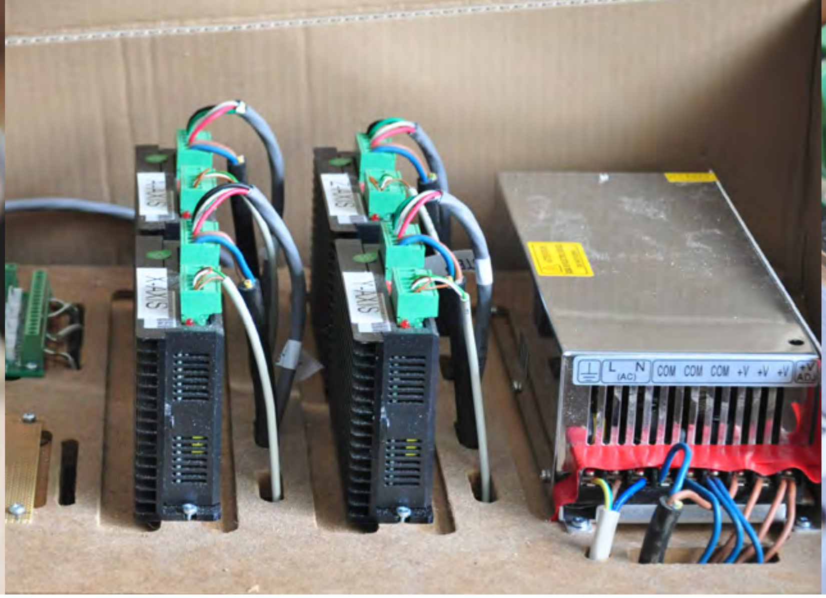
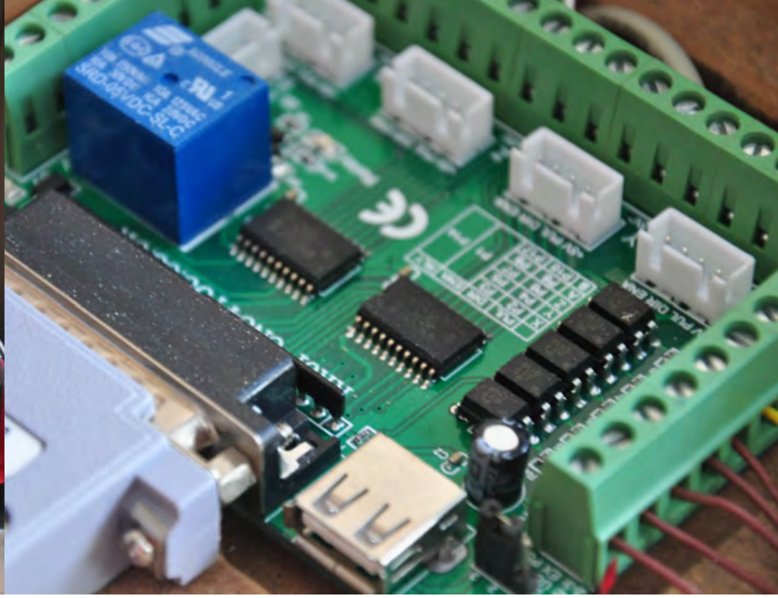
Figure 38, top, right:
A deployable switch used similarly to a touch-plate sensor for determining the correct Z-offset for the workpiece underneath it. The cutter is lowered to trigger the switch.

Figure 39, opposite, left:
Stepper motor drivers.

Figure 40, opposite, middle:
Drivers and power supply on custom milled electronics tray

Figure 41, opposite, right:
CNC-controller circuit-board



connected to one of a controller's input pins. The clamp is hooked up to the tool in the spindle which is then lowered until it touches the plate which was previously placed beneath it on top of the workpiece. It can be detected when physical contact is established by the controller which at that point stops the downward movement and uses the current z-position offset by the touch plate's thickness as the new zero relative to the workpiece.

5 Controllers and Drivers

It takes several circuit boards to control all the router's motors and read the inputs of all sensors that give feedback about the machine's state of operation. They serve as translators between the attached controlling computer and the actuators, motors and active components of the router as well as sensors and switchboards that can give additional control of the machine.

The main device used is the circuit board that is directly connected to the computer via serial or USB connection. It receives commands from the software package used by the computer to execute toolpaths. This circuit board that is usually called CNC controller also receives signals from sensors and switches which it relays to the computer. The controller is also translating the computers commands to move the mill's actuators into signals for the drivers of each of the machine's stepper motors which it is connected to with three signal wires each.

The drivers that run on DC power are catered to by one or more power supplies. They interpret the controllers signals and transform them into movements of their respective stepper. There are as many drivers as there are stepper motors, each driver can only control a single motor.

Stability

One of the most important factors influencing the ultimate precision of a CNC router is the stability of the construction. Any play of the spindle will translate directly into imprecisions of the workpiece. If the machine is at a stand still it should not or only barely be possible to manually move the tip of the tool in any direction. The more of a jiggle one can induce on it, the greater the inaccuracies that have to be expected.

Stability does not only affect precision but also the speed of operation of the router. The more rigid the construction the faster the tool can be moved through the material without sacrificing precision. That is because the higher the velocity of the tip of the cutter is, the greater the forces in opposition to its movement get. As the tool rotates at a certain rate the size of the chips of material that are removed with each rotation increases with the tools speed relative to the workpiece. This is what is called the chip-load. With higher chip-loads the force necessary to cut the chips gets larger synchronously and acts as a deflecting power against the tool. If the chip-load is too high these forces can exceed the constructions threshold and result in countless phenomena apart from mere inaccuracies including violent vibrations and unanticipated tool movements.

It is very hard to judge at what point the threshold of a system in terms of feed-rate and chip-load is reached as first signs which usually include slight inaccuracies are subtle and can have a multitude of causes such as the grain of the material or a dulling tool. It takes a period of experimentation to find the optimal values of operation for each material and tool.

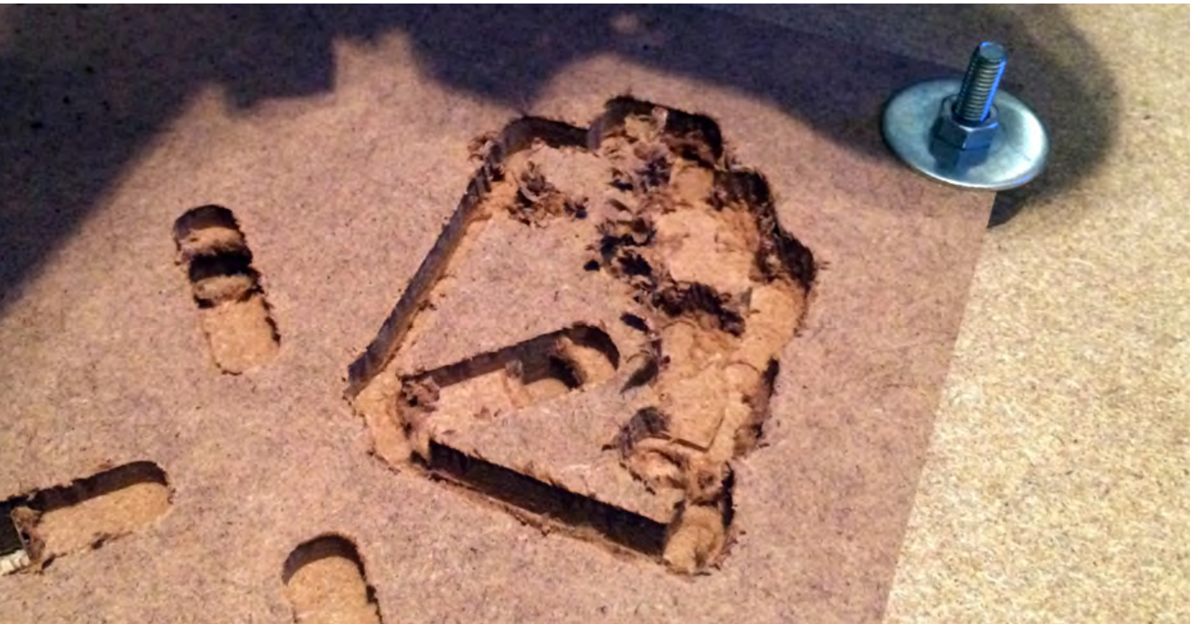


Figure 42: Slight vibration can rapidly increase and make the tool move around uncontrollably leading to potentially hazardous situations as cutters can break and be sent flying across the workshop. The consequences can be seen on this photo showing a failed milling of a part from meadium density fibre board.

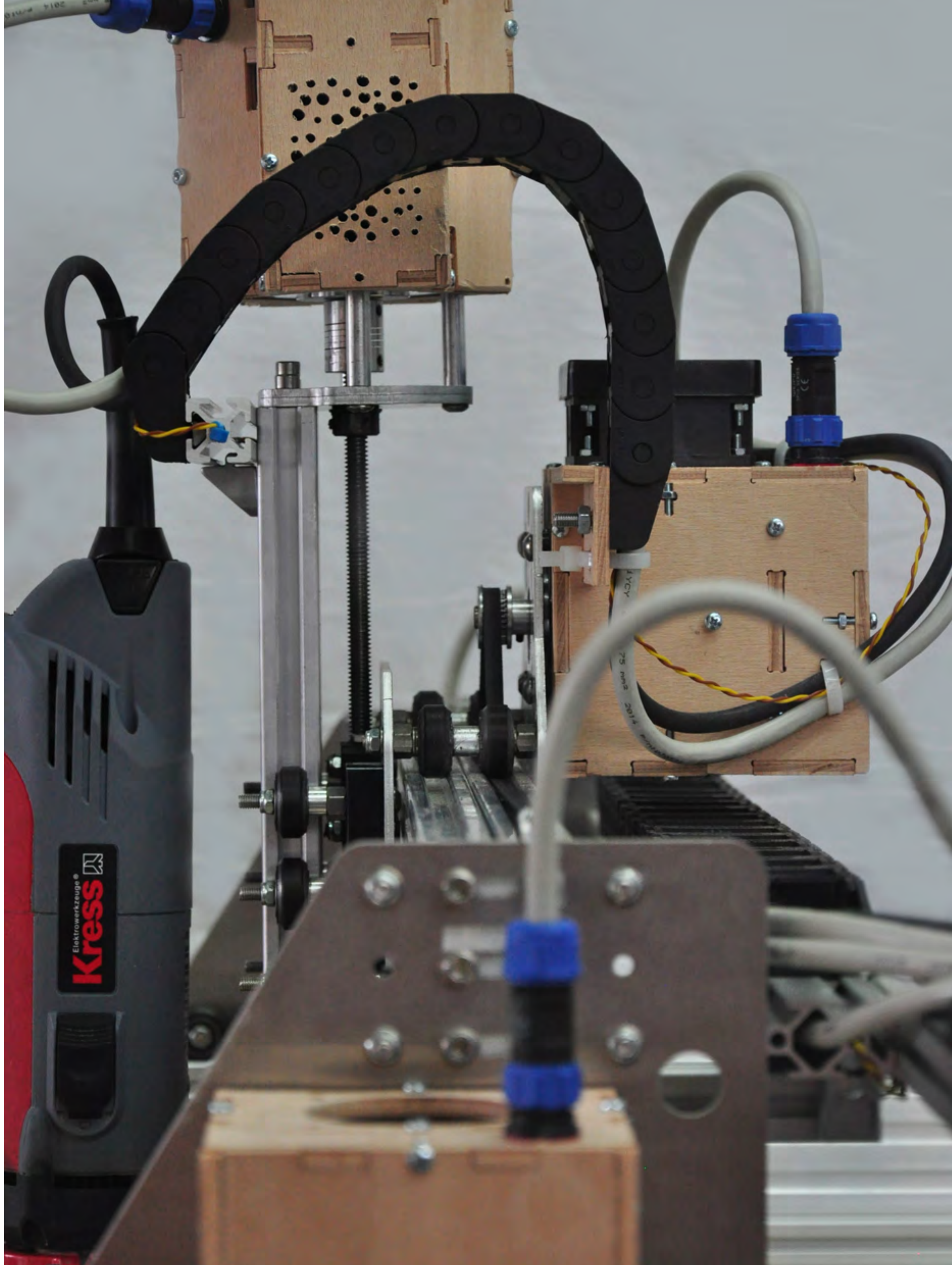
Figure 43, opposite page: View along the router's gantry providing a glimpse at the Y-axis's timing belt and pulley as well as the vertical threaded rod that is used to position the spindle along the Z-axis.. It is connected to the stepper motor in the wooden housing at the top with the metal coupling visible just below it.

Points in Space

For the process of CNC milling to work the machine has to have precise control over its cutter's location. It can easily establish a relative position by cummulatively adding up its movements during operation, but his already implies that it needs an absolute position to start with that can be used as a reference. If that is a given as well the machine can smoothly maneuver to all the points in space of the toolpath it shall follow.

Whenever the machine comes to an unplanned halt or something goes wrong that forcefully changes the cutter's position the machine can no longer know its acutal position. It can only follow willful movements. If the cutter hits an obstacle too hard to cut through or plunges into the material too deeply it can happen that the forces on the belts of the motion system become too great. That can make the belts slip over the pulley's teeth. The machine has no means of sensing this irregularity and assumes that everything is moving as intended. It does not correct the toolpath according to the faulty execution and tries to continue as if nothing happened. At this point the only way to regain precision is to stop the machine and home it.

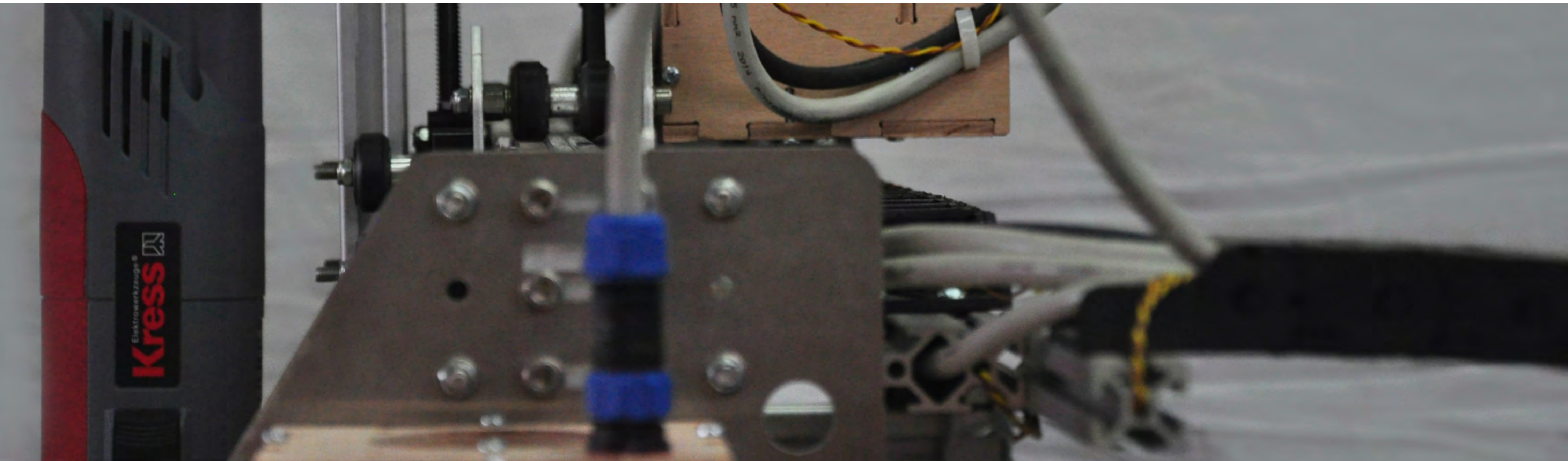
The process of homing is best accomplished by the use of highly precise sensor-switches. The control software runs a homing protocol that lets it detect each axis's home or zero position at a time by slowly moving it in a predetermined direction until it hits the homing switch. It is very important for these switches to



be mounted firmly on the router's frame to get reliable readings from them. Depending on the application of the machine it has to be possible to home it to within a certain fraction of a milimeter in space which should never be larger than one tenth of a millimeter.

To enable the machine's software to follow its deliberate movements it needs several data about its motors. The usually used stepper motors are controlled by commanding them to rotate in a given direction by a certain number of steps. Stepper motors usually rotate by 1.8 degrees per step. This native resolution is then artificially increased by a factor of up to 200 or more through a process called microstepping. If the software is aware of the parameters involved in this process and the motor's original resolution it can determine by how much it rotates the motors shaft with every step. To calculate the axial movement resulting from this rotation two more parameters are necessary. Firstly the number of teeth of the pulley attached to the motor and secondly the used belt's pitch. If the motor is attached to a threaded rod as a means of axial movement rather than a pulley and timing belt system the necessary dimension to determine the axis's movement is of course the rod's thread-pitch.

The described system works very precisely as steps are hardly lost along the way from the software to the actual motor. Also a stepper motor can not be rotated by agency of any mechanical forces on its shaft as long as power is on, allowing it to stay perfectly in place while the machine is operating. However, as soon as the motion system is powered off it becomes possible to manipulate the actuators' positions. This effects that a router has to be homed again every time it is powered on.



Maintainability

It was a great challenge to get this router which was built from a kit that is available online to a point of smooth operation. Several measures had to be taken to make the mill function in a stable manner and automated the processes involved to a degree that was satisfactory for the demands of the intended purpose. The difficulty lay in the extensive periods of experimentation that were necessary to identify the problems at the basis of unbeneficial vibrations, inaccurate movements and imperfect cut surfaces.

To allow for interuptions in the execution of toolpaths it was necessary to install homing sensors of higher quality than the ones that came with the kit. The method of attaching them to the structure of the router also had to be improved as they have to sit firmly in place even if the gantry bumps into them. Before this is accomplished any situation that forces the execution of a toolpath to stop might very well result in the loss of all milling that happened on the workpiece previously as most times it is not possible to manually rehome the machine with sufficient accuracy. After elaborate research a source for apropriate sensors was found. The replacement switches came with a metal base that could be fixed onto the frame's extruded profiles with a special piece of hardware resembling a nut that can be slid into the profile's grooves.

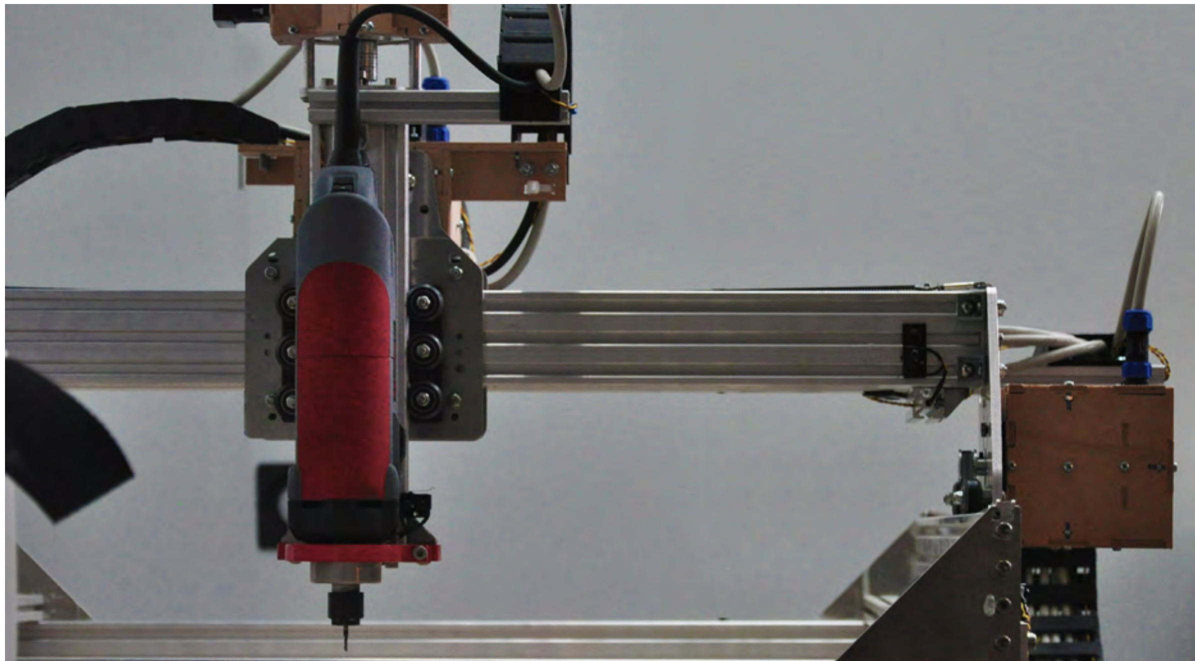
First experiments with the machine also led to initially unexplainable vibrations that could get intense enough to send the tool of course moving around violently leading to catastrophic damage rendering workpieces

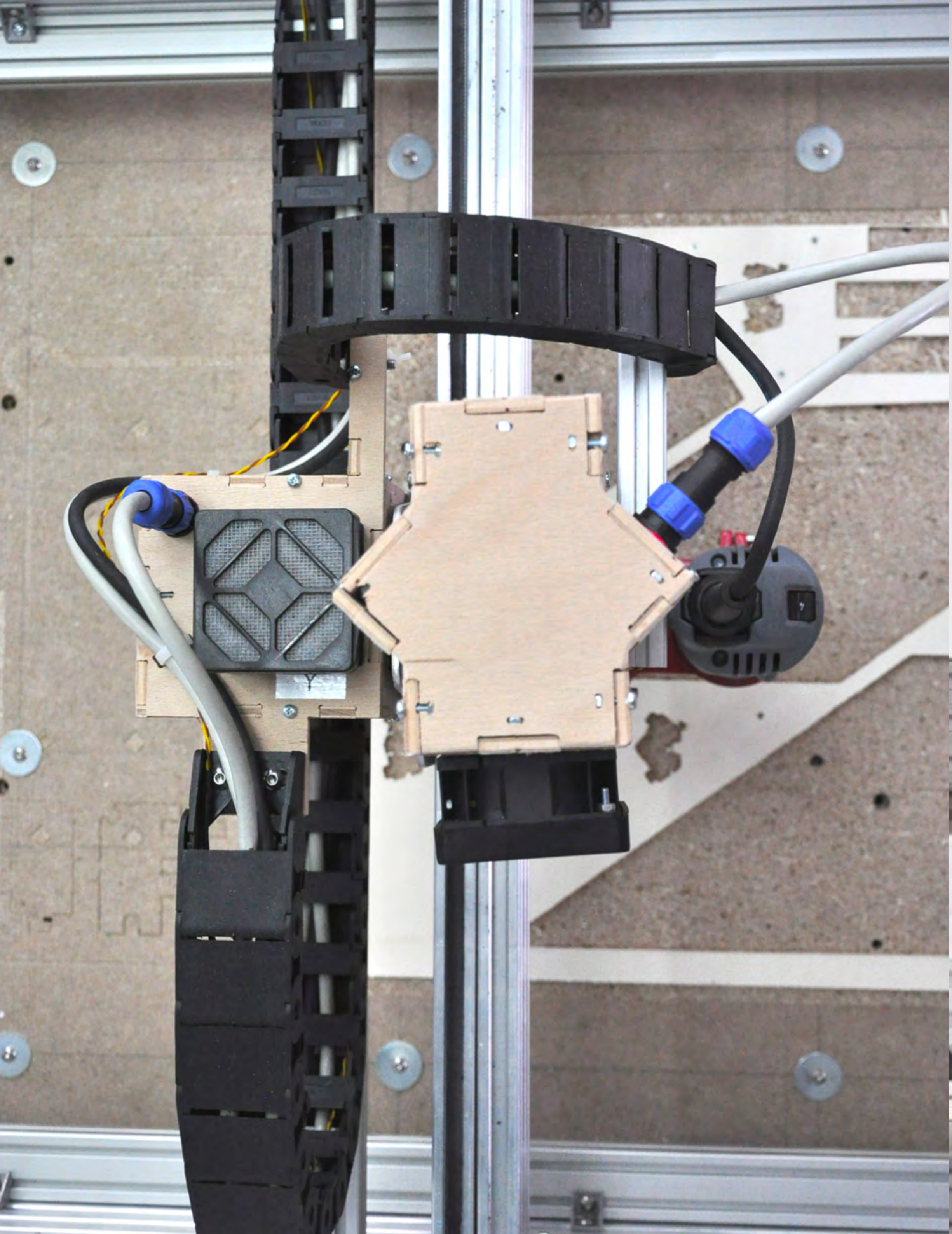
Figure 44, opposite:
Wiring node on the gantry next to the X-axis motor.

Figure 45, left:
The gantry carrying the spindle's slide was initially suspected to be one of the main weakpoints in the construction permitting disruptive vibrations that caused several experiments to fail.

Figure 46, next page, left:
Cable tracks on the gantry supplying Y and Z-Axes

Figure 47, next page, right:
Major mistakes were made in the design of this sled carrying the milling motor and its Z-axis motion system. The incorrigible instability can only be dealt with by lowering the routers feedrate.



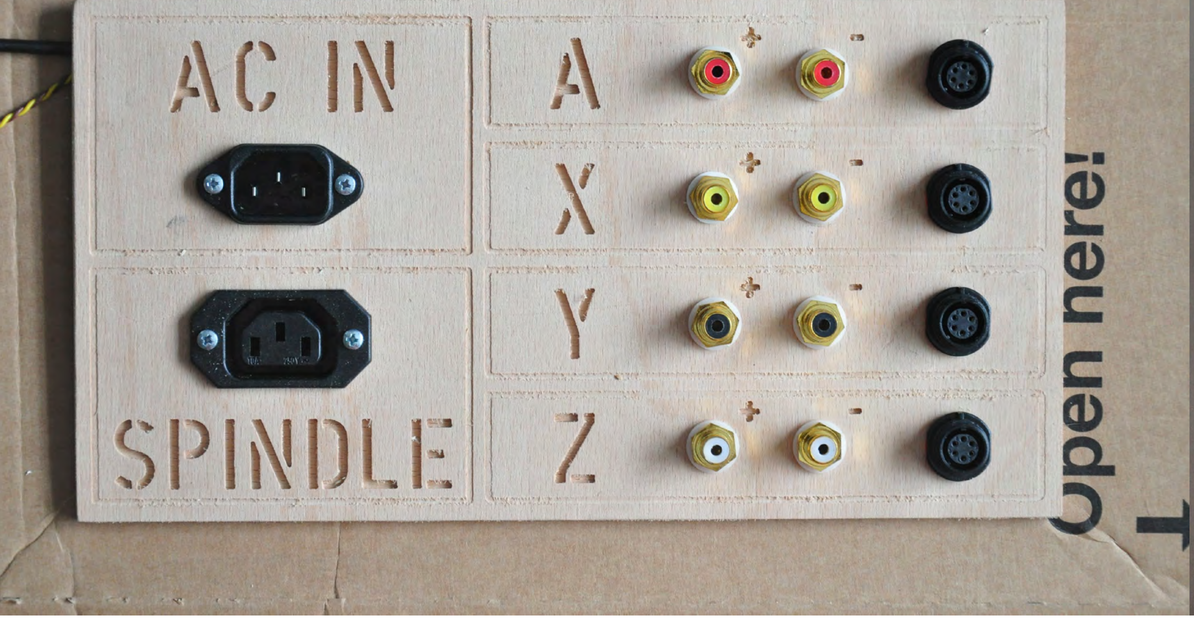

Y

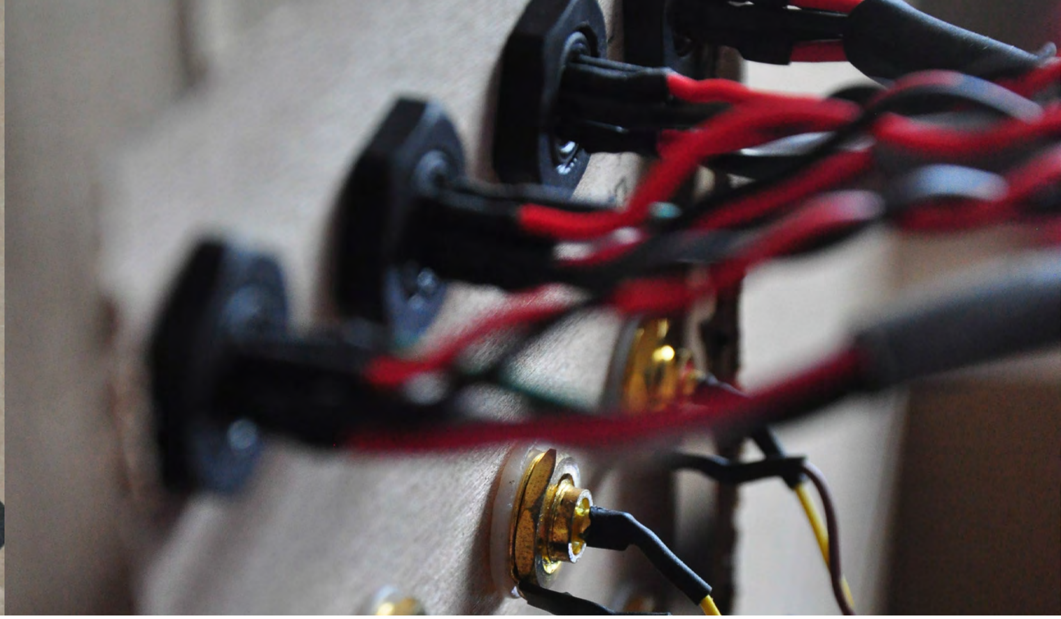
Kress

useless and causing the timing belts to slip on their pulleys. (see Figure 42, page 38) It took many failures until the chain of causation could be traced all the way to its last link. The earliest suspicions saw the fault for this erratic behaviour of the machine in such unrecoverable circumstances as an insufficiency of strength on the side of the motors, a general lack of rigidity of the construction and a misfit in the dimensioning of the timing belts. To the author's great relieve, but only after many frustrating failures a pattern in the occurence of described misbehaviour could be identified leading to the discovery of the first actual error. The likelihood of erratic behaviour was rising with the depth of the groove in which the cutting bit was operating. Two explanations fit this explanation; firstly chip build-up in the groove deflecting the cutter and secondly the cutter touching the wall of the groove causing it to bounce from side to side with accumulating vigor within the groove. Using a multipass strategy for cutting thick material greatly reduced the occurences of the problem. This approach has the cutter go along the contour two or more times for each step into the material with different offsets from the actual contour leading to a wider cutting groove—wide enough to avoid bouncing of the tool. Nevertheless the problem persisted if only with drastically decreased frequency. Many more failed experiments were necessary to find the second source of vibration. The spoilboard itself was not rigidly enough fixed to the substructure and in consequence stimulated vibrations when certain operations were performed. After the installation of additional aluminum bracing beneath the wasteboard the problem was almost entirely solved. The tightening of the timing belts ultimately helped to make the issue disappear for good.

Another great nuasance with interuptions and also starting of new operation cycles was the accurate zeroing of the tool along the Z-axis to the top of the material. A widely used technique to solve this problem is the following: a sheet of thin paper is put on the surface of the material to be cut. While the cutter is gradually lowered one constantly moves the paper underneath it. Ultimately the tip of the tool touches the paper preventing further movement. The operator stops the process of lowering and sets the thereby determined Z-position to zero. While this method works well for most applications it is slow and tedious as well as not as precise as it could and in many cases should be. A touchplate or zeroing switch as described in the previous section *Components* allows to increase precision and to automate the process to the push of a button. (see Figure 37, page 36)

Most other issues with quality and general operation that the author experienced with the machine were solveable only by changing faulty milling strategies. The combination of spindle-RPM and feedrate have to be figured out for every material. A factor playing into this is blacklash of axes and especially instability of the Z-axis which can only be countered by drastic reduction in feedrate. The machine at hand is of a construction with severe shortcomings of Z-axis stability resulting in a play of at least a milimeter at the tip of the tool. While its motors and spindle would be capable of very high feedrates the milling strategies have to be



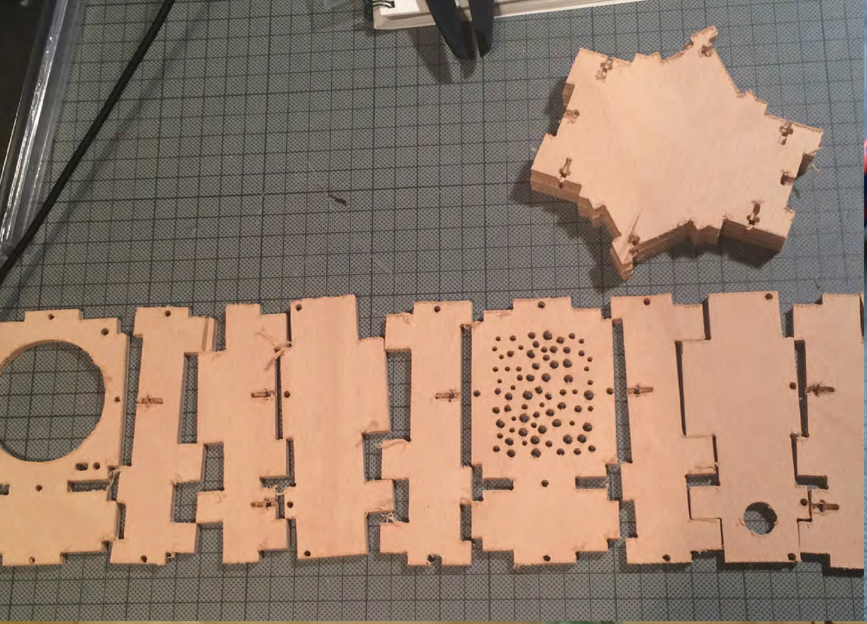
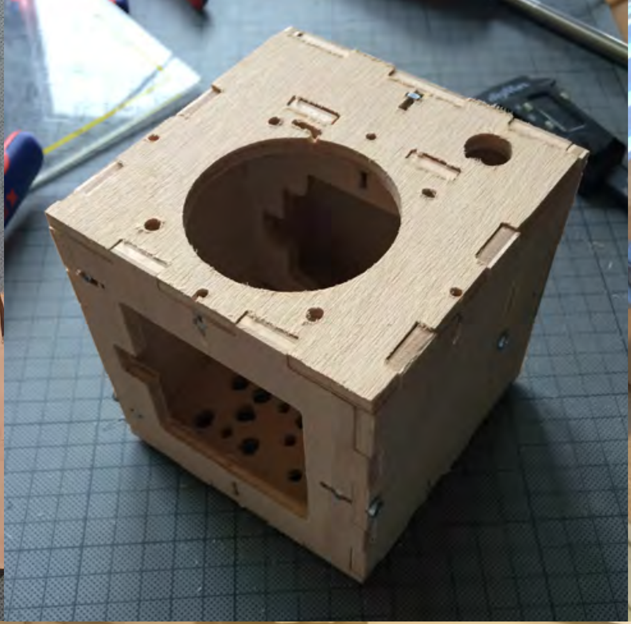


adapted to this mistake in design which is impossible to correct without reconstructing the whole gantry.

Additionaly several improvements were made that allowed for more convenient maintainance and operation. Custom housings were created for all motors. They feature a fan for active cooling of the stepper motor during operation. They also boast a socket connection for the signal wires and fan power which are combined into a single cable per motor for a more orderly electrical system. The original design of the router did not feature any type of wire management so it was necessary to add cable-tracks and all necessary substructures to keep signal wires and power cords out of harms way. Lastly an electrical boxed was conceived including a connector panel that allowed for quick reconnection of all components to the controlling ciruit boards and a tray to serve as a base for orderly mounting all electrical components involved in the driving of the router. (see Figure 40, page 36)

Figure 48, top, left:
The electrical box's connector panel is the interface to all eceltrical components of the router and connection to the power network.

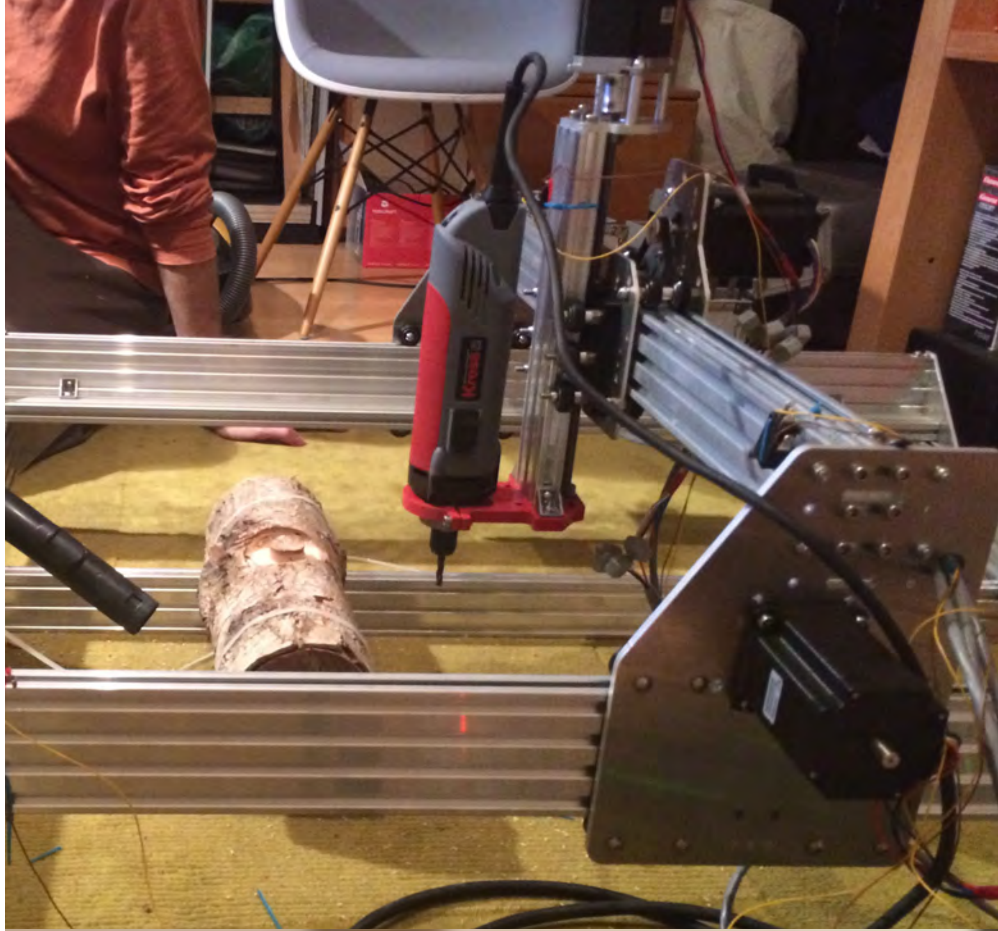
Figure 49, top, right:
The back of the connector panel. 54 wires soldered to the sockets connect them to the respective controllers, motor drivers and sensor inputs.

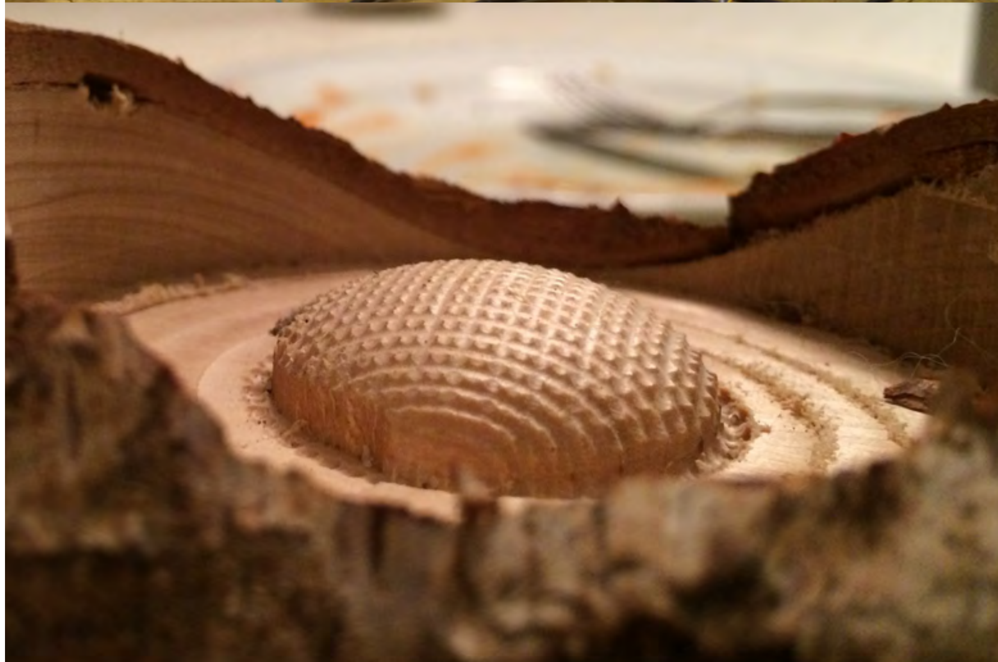



4 Experiments

First Artefacts

These are the first steps I took using the router. In contrast to using a laser cutter the operation of a CNC router has a long list of parameters that changes with the specific application. For every particular result a strategy has to be developed by which to approach the problem of removing he unwanted material which becomes especially complex if the desired artifact has three dimensional features that have to be milled. The experiments depicted here both demanded three dimensional milling and led to the conclusion that it was necessary to stick to 2d operations in the development of the algorithm to make the manufacturing of prototypes possible in a reasonable time frame.





from left to right, column-wise

Figure 50, top:
The first milling experiment after completion of the router was the execution of a three dimensional toolpath on a log to check the routers functionality.

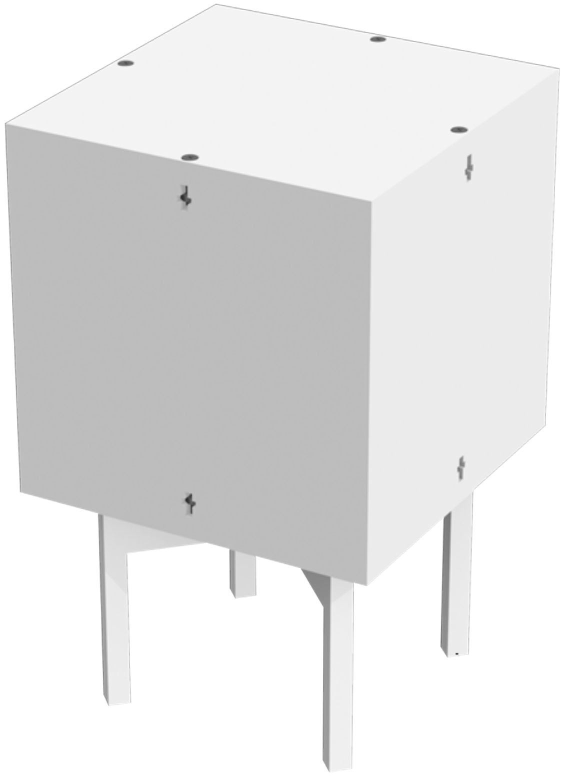
Figure 51, bottom:
The result of the first experiment.

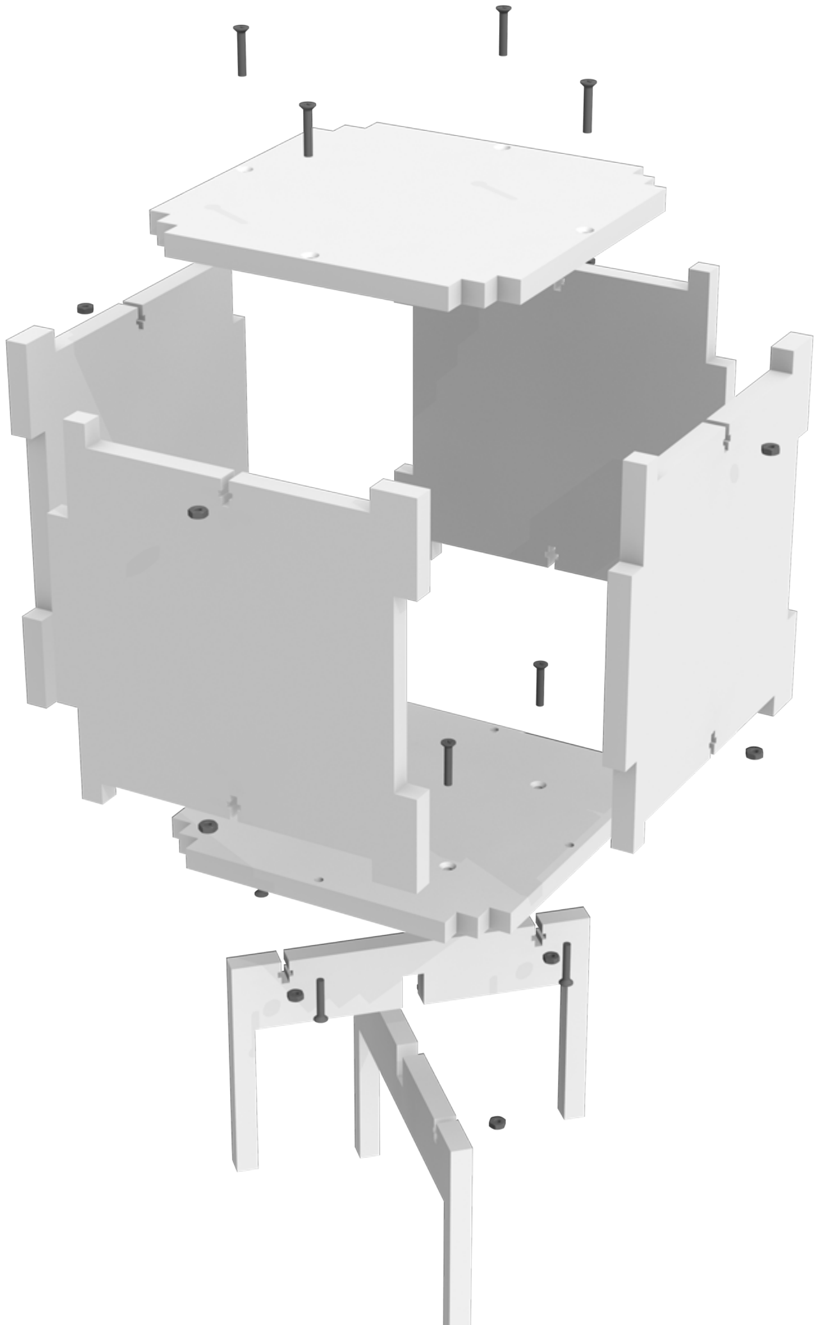
Figure 52, top:
The first of several finishing passes for a chain link. This experiment aimed at the milling of a more complex three dimensional artefact. A parametric chain was designed in Grasshopper.

Figure 53, bottom:
A finished chain link.

Figure 54, top:
The result of a roughing pass for the links of the parametric chain.

Figure 55, bottom:
All finished chain links connected to each other.





Prototyping

1 *Prototype I*

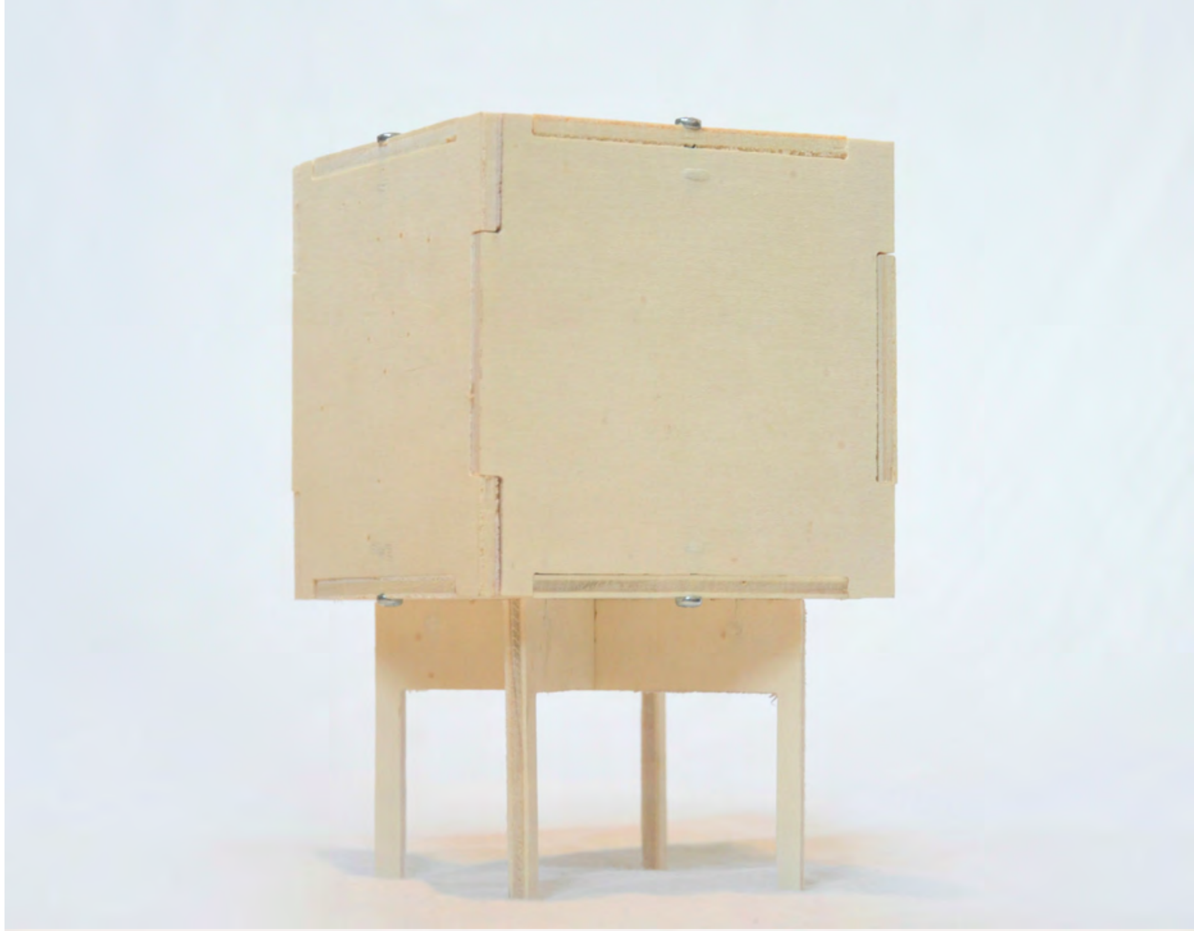
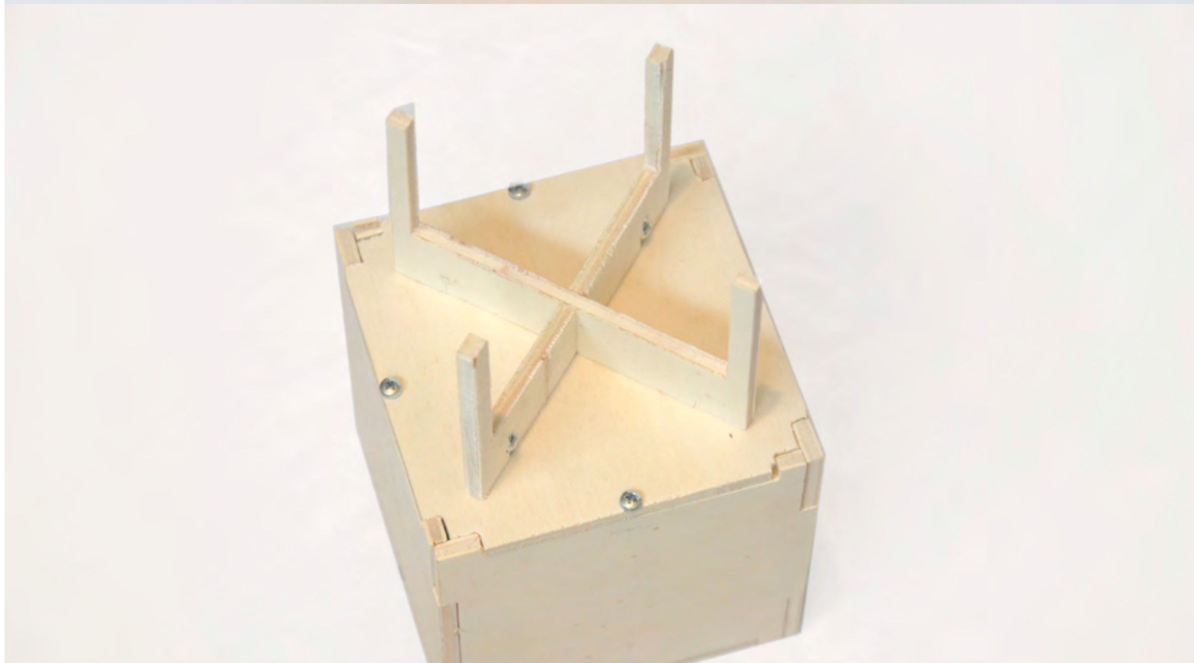
This first actual prototype in the development of a universal system for physically creating geometries is a simple cube. It is elevated by pillars paying respect to the first of Le Corbusier's Five Points of architecture. The connections between the parts that were cut from poplar plywood are accomplished through the use of screws and nuts. The latter were held in receptacles milled into the wood at all necessary places. The resulting enclosure consisted of eight milled parts and ten nuts and screws respectively.

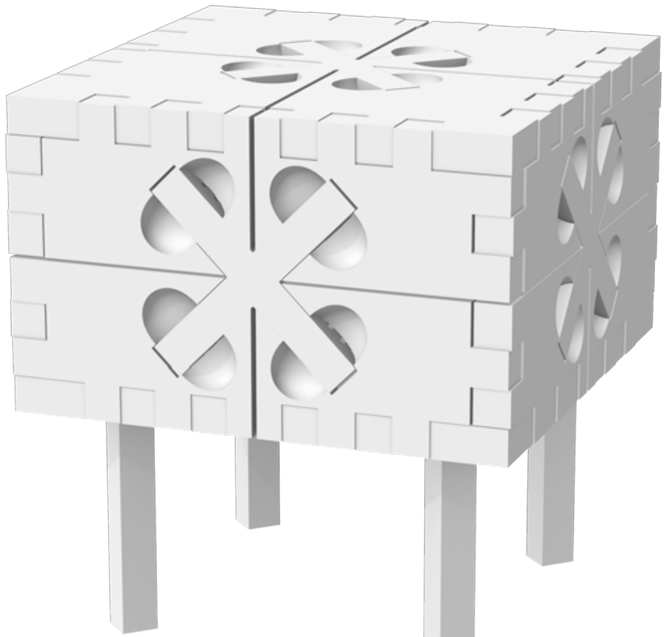
The mode of fastening was successful. However, it needed some adaptation to be a viable option for use with arbitrary angles between parts as the implementation at hand was only capable of connecting boards at right angles. While the paradigm of pillars for relaying mechanical loads into the ground continued to seem the most rational option for the envisioned way of

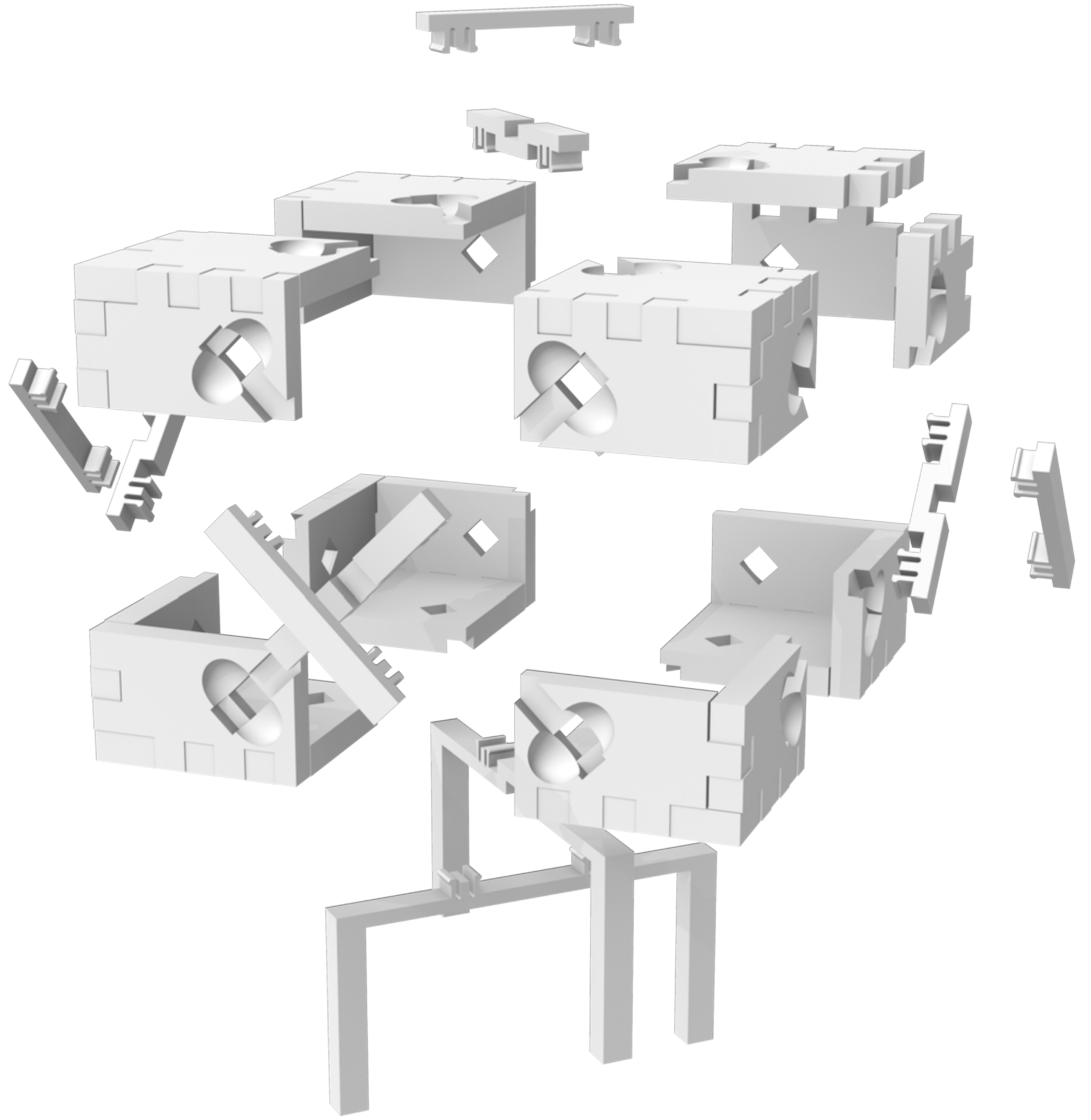
Opposite Page:
Figure 56: Renderings assembled and in an explosion view.

This page, from top to bottom:
Figure 57: The completed cube made from poplar plywood.

Figure 58: View of the bottom of the object showing the screws used for connection.





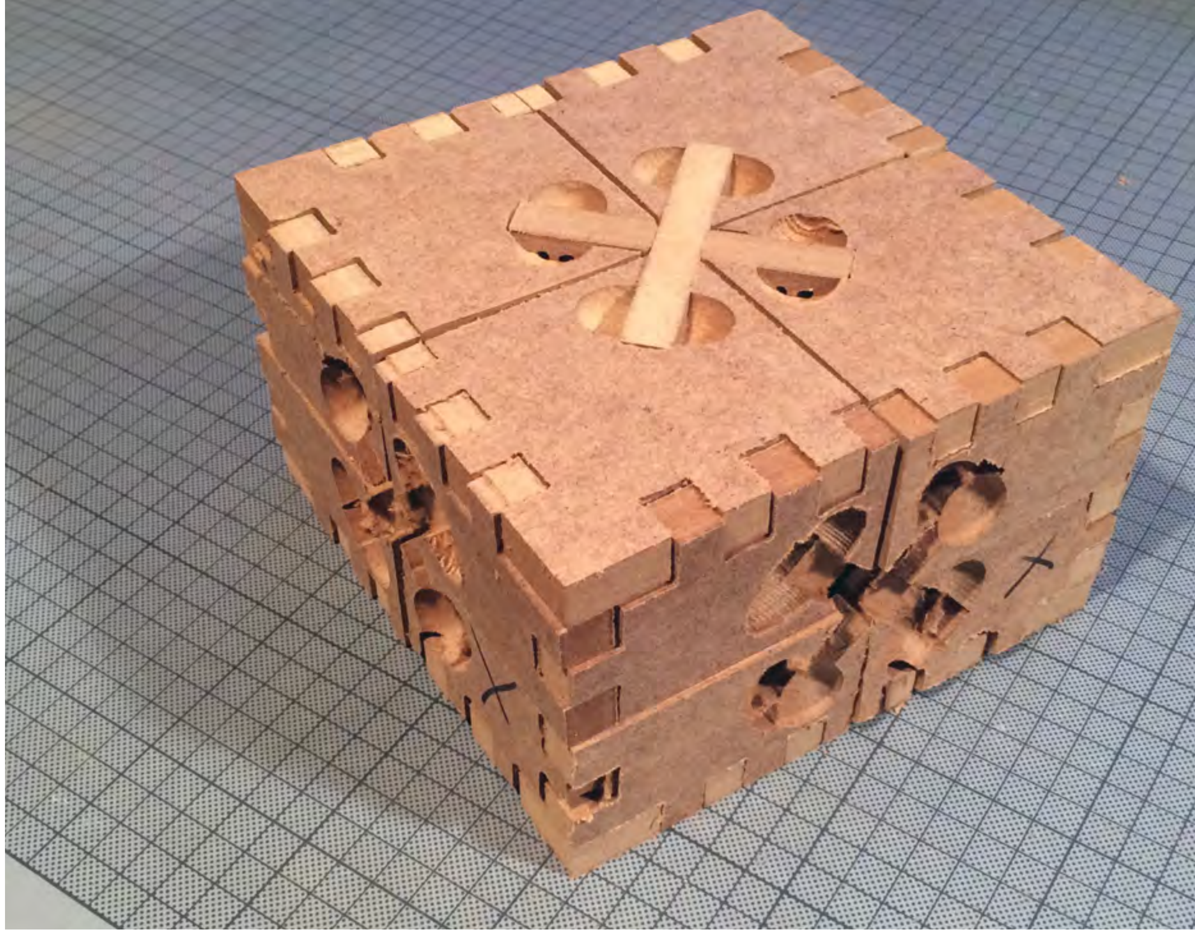
building it was decided to abandon its implementation for further iterations of the prototyping as this yielded no direct insights.

2 Application of the Joint-Redutction paradigm

For the second prototype a principal of complexity management in manufacturing described in a previous chapter was applied. The realisation was made that it is unfavorable to carry out the most complicated joints on the construction site which are the points where elements meet at an angle producing complicated details. In current prefabrication elements are walls and ceilings. For the second protoype this was changed. Subassemblies that would arrive at the site of construction would already encompas the most difficult parts of the construction leaving joints in unproblematic areas to be done on site. This paradigm of corner-subassemblies was further pursued in the following developments.

This iteration also boasts an experiment with a screwless clip connection and a degree of three dimensional milling to create even surfaces by sinking the connecting clips half way into the material of the other parts. While this approach seemed interesting at the scale and complexity of the prototype at hand it seemed restrictive to scaleability. Additionally the use of three dimen-

Opposite Page:
Figure 59: Juxtaposition of renderings of an iteration on the basic cube assembled and in an explosion view.



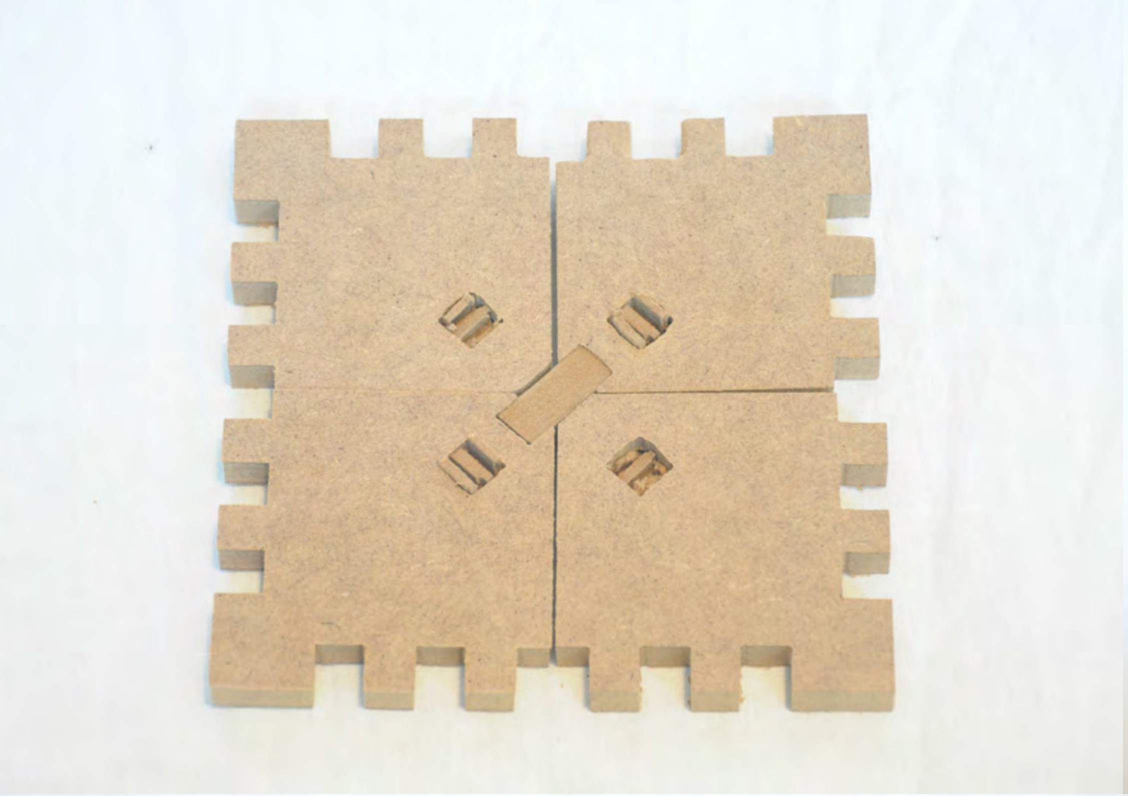
This page:
Figure 60: Prototype using an iterated system for connections.

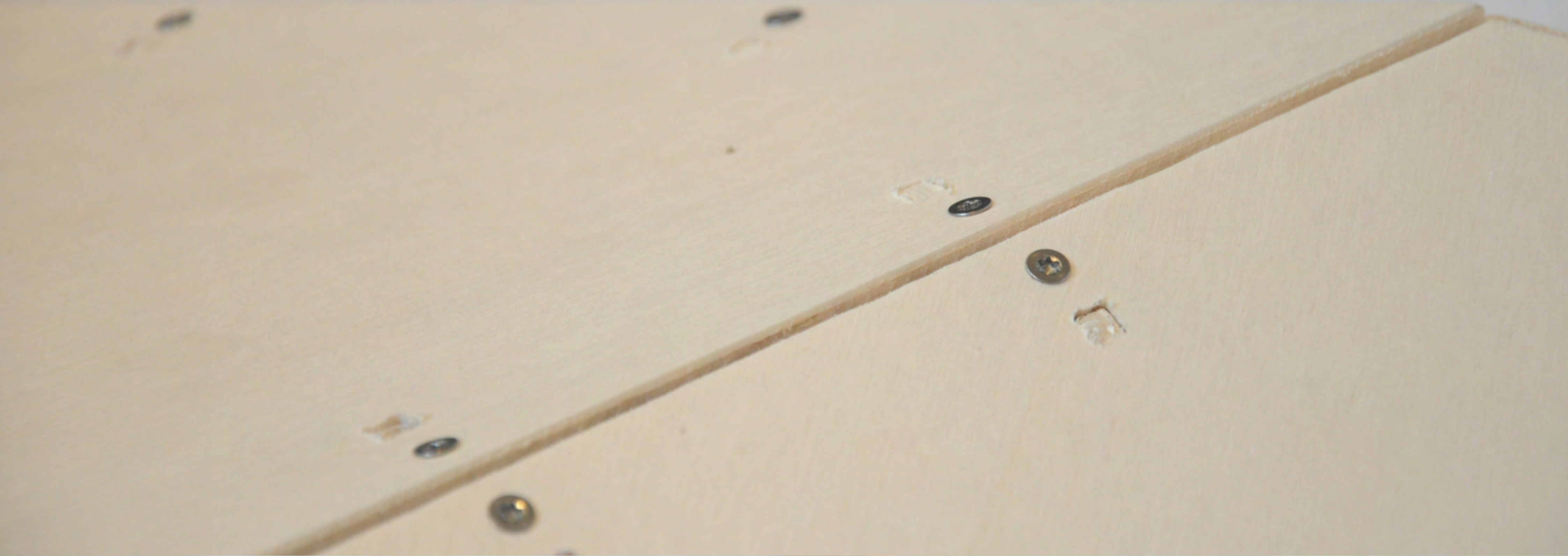
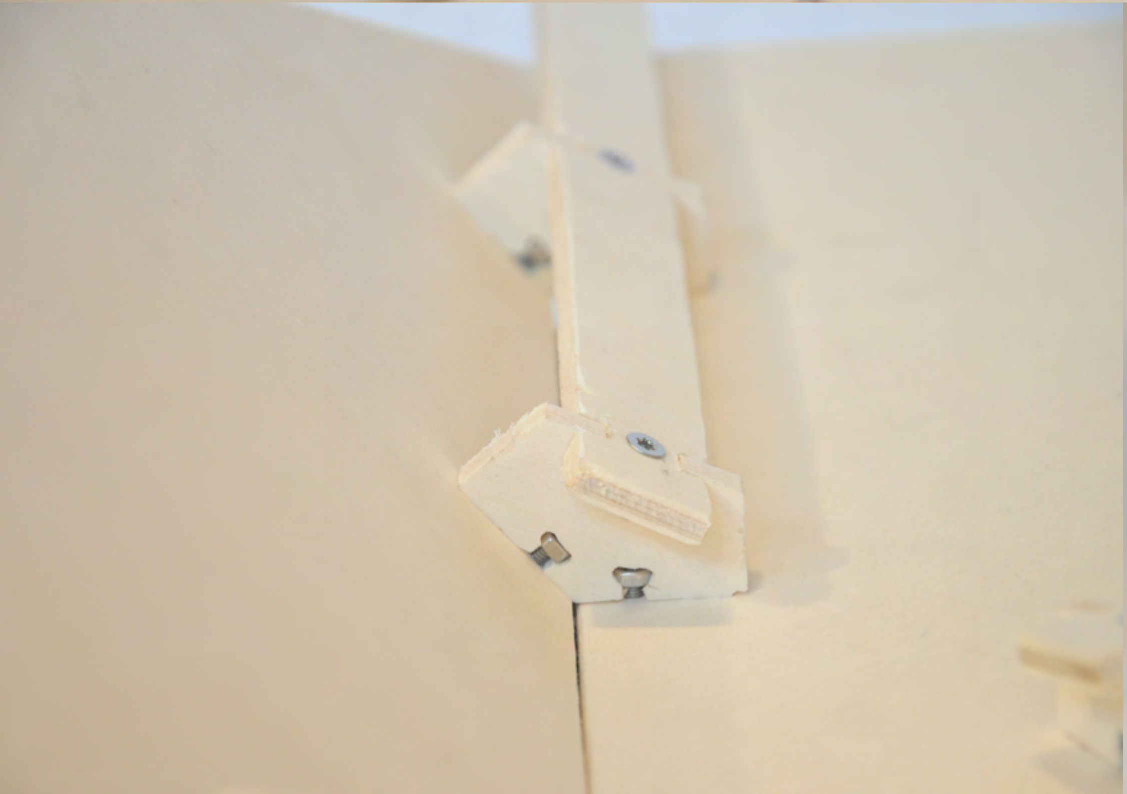
Next Page, from left to right, column-wise:
Figure 61: Detail of the used connection, interior view.

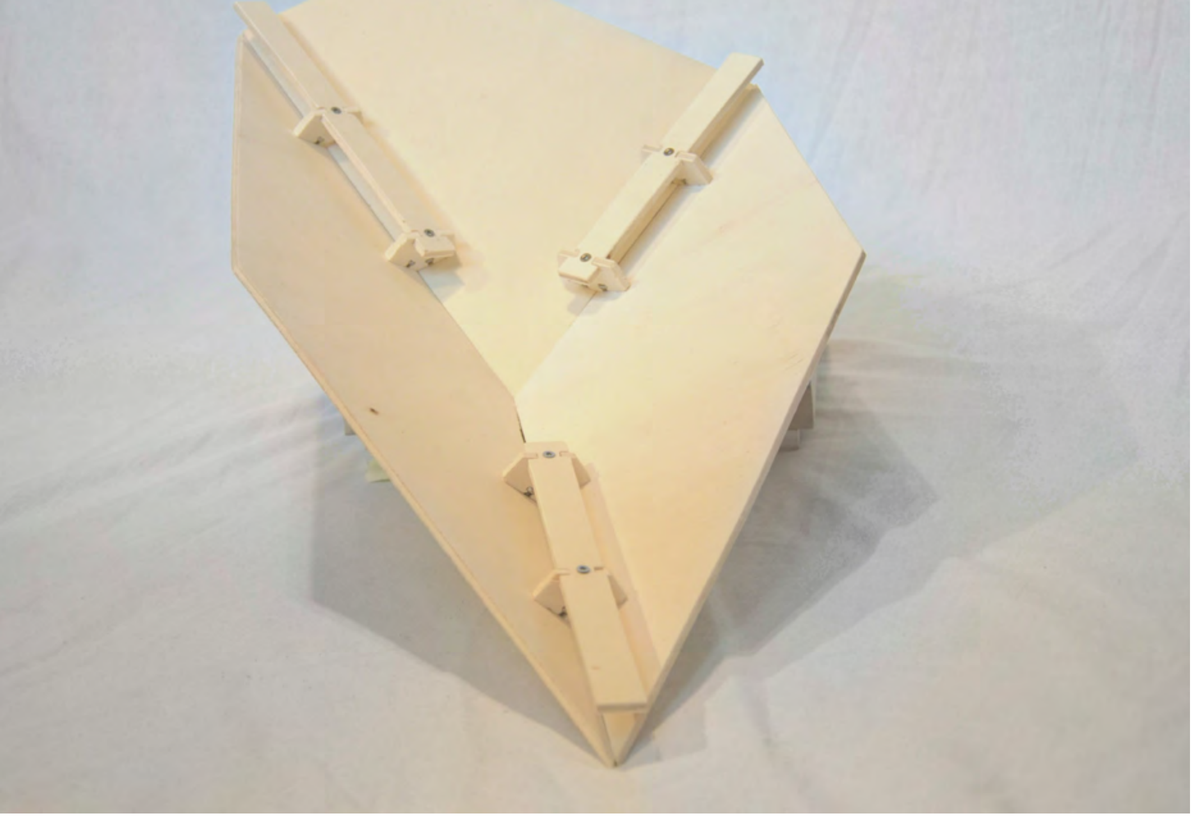
Figure 62: Detail of the used connection, exterior view.

Figure 63: View of one of eight subassemblies.

Figure 64: View of one of eight subassemblies.





Previous Page, clockwise from top:
Figure 65: Edge connection detail of Arbitrary Corner Prototype

Figure 66: Detail view.

Figure 67: The connection between the individual elements achieved satisfactory precision.

This Page:
Figure 68: Inside view of Arbitrary Corner Prototype

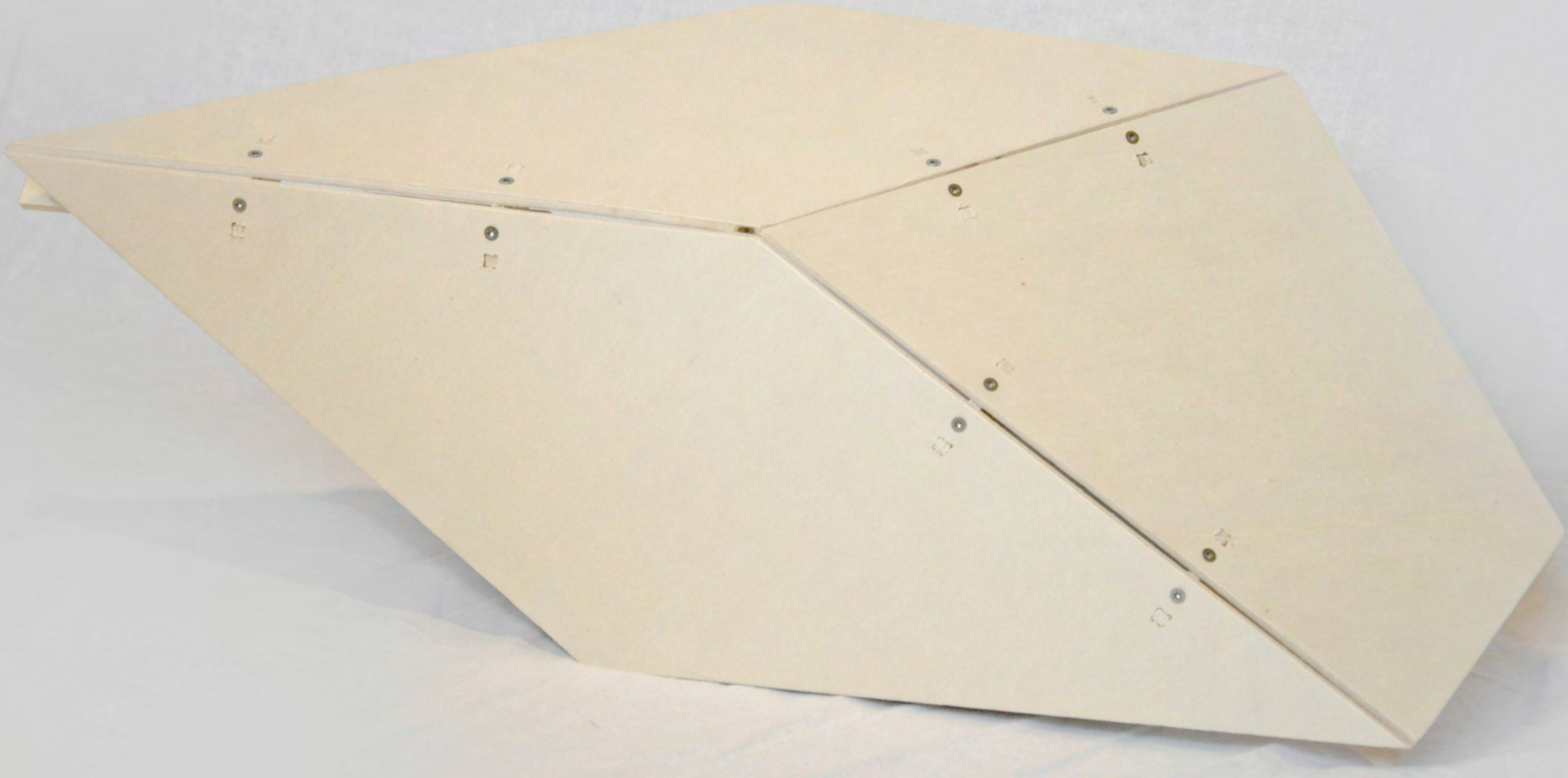
Figure 69: The Arbitrary Corner Prototype is the subassembly of all shards meeting at a corner and all parts involved.

sional milling increased fabrication time significantly while also reducing the degree of possible automation, because it demanded an additional tool change. Hence both features were abandoned.

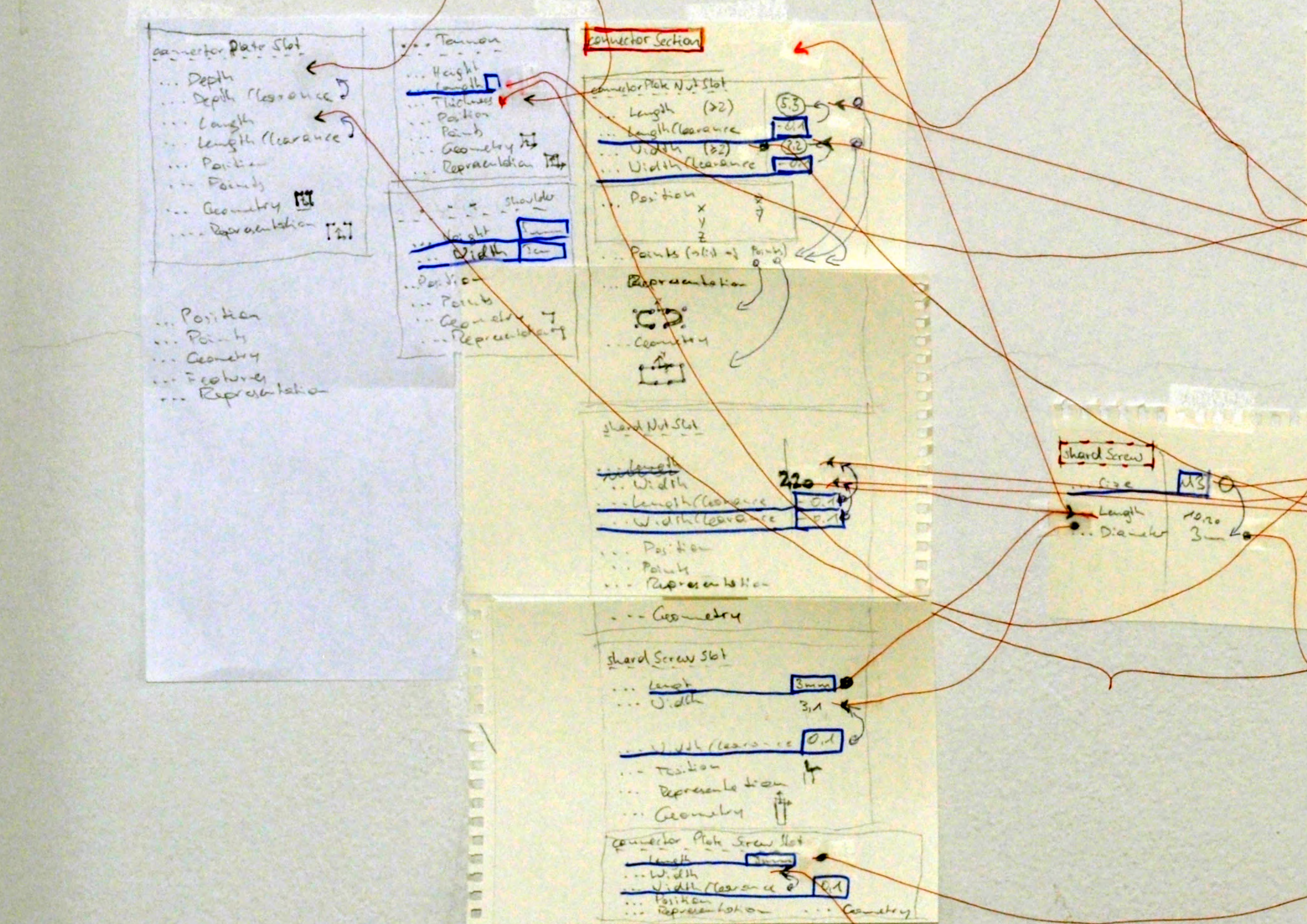
3 Arbitrary Corner Prototype

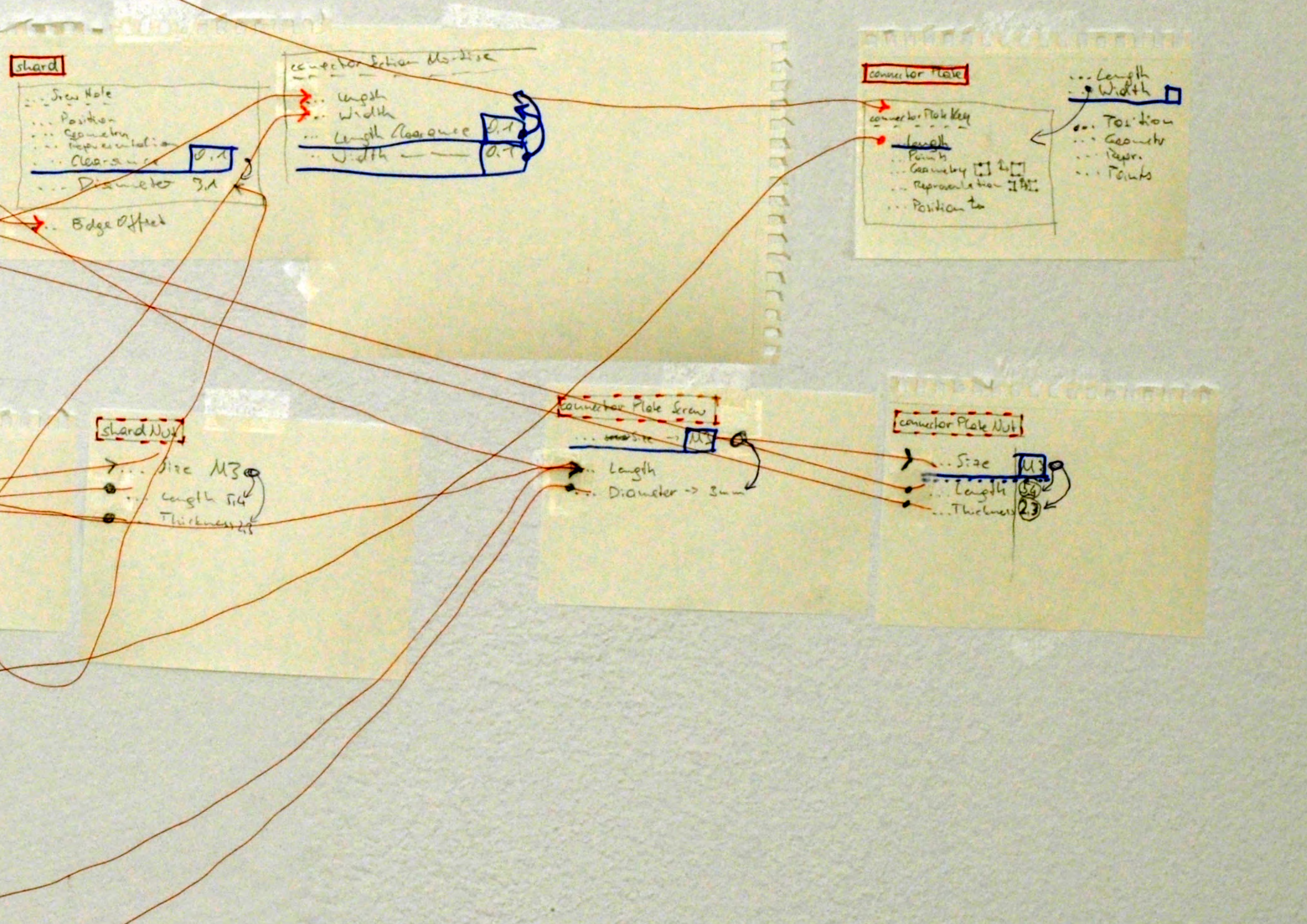
One more step had to be made to build a foundation for the creation of a capable algorithm. This was accomplished with this prototype that implemented a strategy to create corner-subassemblies spanning arbitrary corners with any number of adjacent edges and faces. It used a variation of the screw and nut connections first utilized in *Prototype I*.

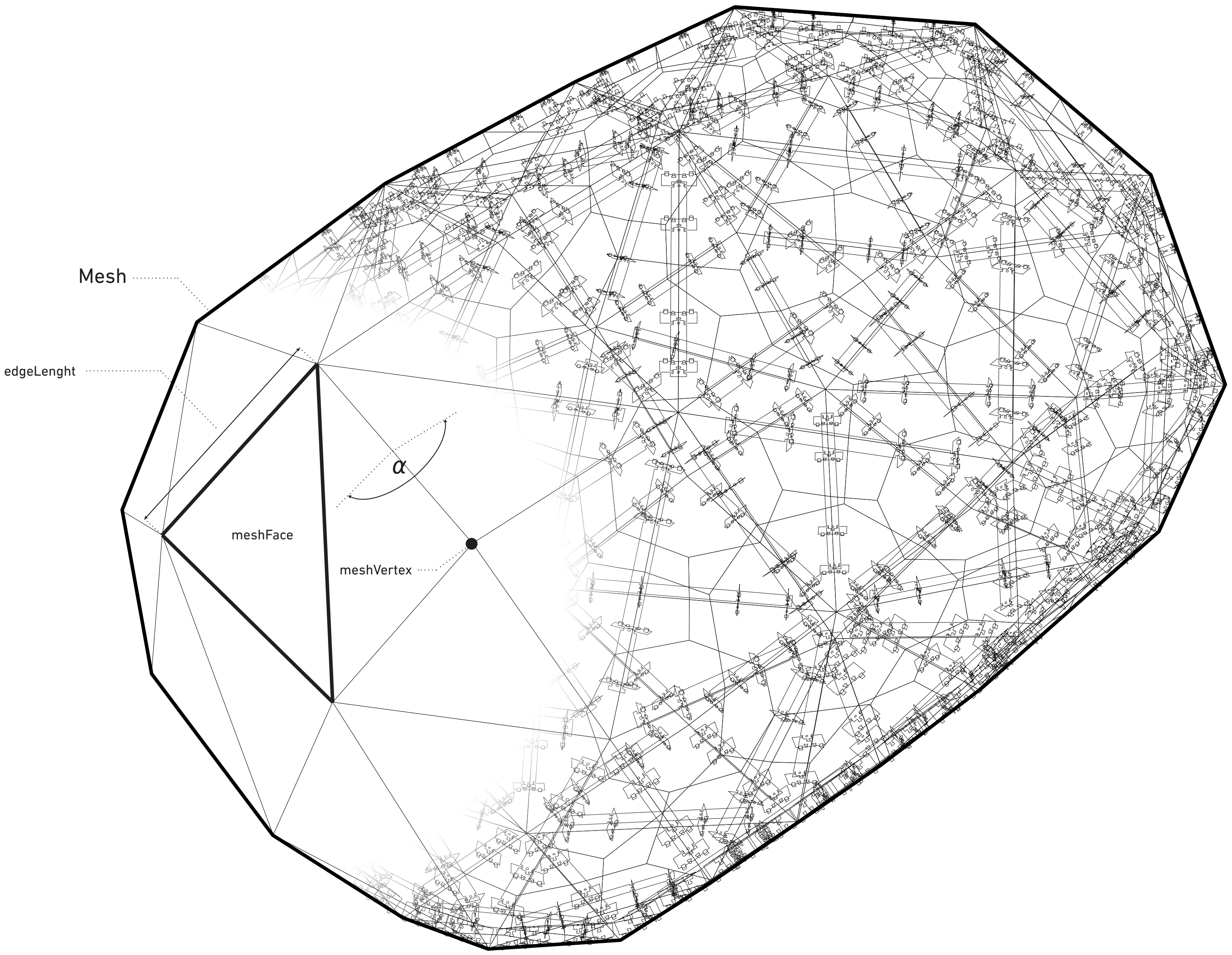
The resulting corner-subassembly consisted of three types of parts. *Shards* were the pieces of the faces. *Connector Sections* were used to connect them firmly at the correct angles on their inside and finally Connector Plates would connect to these sections reaching between subassemblies to interconnect them. On this scheme of segmentation the algorithm was built.



5 The Algorithm


connector Plate Slot
... Depth
... Depth Clearance
... Length
... Length Clearance
... Position
... Points
... Geometry
... Representation
... Position
... Points
... Geometry
... Features
... Representation
... Tenon
... Height
... Length
... Thickness
... Position
... Points
... Geometry
... Representation
shoulder
... Height
... Width
... Position
... Points
... Geometry
... Representation
connector Section
connectorPlate Nut Slot
... Length (≥2)
... Length Clearance
... Width (≥2)
... Width Clearance
5.3
0.1
2.2
0.1
... Position
x
y
z
... Points (a list of Points)
... Representation
... Geometry
shared Nut Slot
... Length
... Width
... Length Clearance
... Width Clearance
2.20
0.1
0.1
... Position
... Points
... Representation
... Geometry
shared Screw Slot
... Length
3mm
... Width
3,1
... Width Clearance
0.1
... Position
... Representation
... Geometry
connector Plate Screw Slot
... Length
... Width
... Width Clearance
0.1
... Position
... Representation
... Geometry
shared Screw
... Size
M3
... Length
... Diameter
10.20
3


shard
... Screw Hole
... Position
... Geometry
... Representation
... Clearance
0.1
... Diameter 3.1
... Edge Offset
connector Section Max-Size
... Length
... Width
... Length Clearance
0.1
... Width
0.1
connector Plate
... Length
... Width
... Position
... Geometry
... Repr.
... Points
connector Plate Key
... Length
... Points
... Geometry
... Representation
... Position
shard Nut
... Size M3
... Length 5.4
... Thickness 2.3
connector Plate Screw
... Size -> M3
... Length
... Diameter -> 3mm
connector Plate Nut
... Size M3
... Length 5.4
... Thickness 2.3


Mesh
edgeLenght
α
meshFace
meshVertex

Process

The python algorithm that was programmed to perform the automation from mesh to parts and assemblies is written in python and utilizes rhinoscript and the rhino geometry engine for most geometrical operations. These are parts of the geometry software package Rhinoceros 3D.

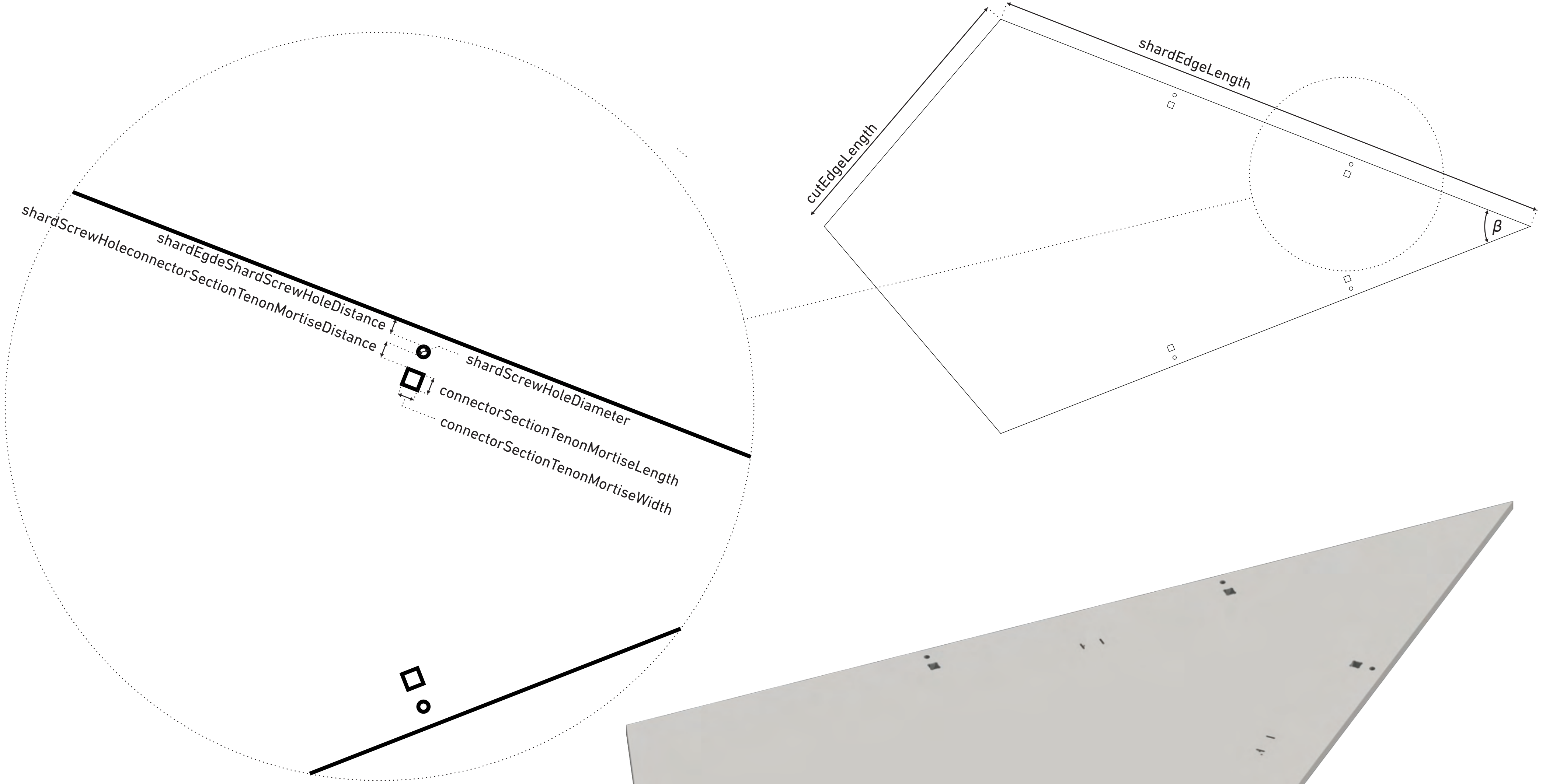
The algorithm is designed in such a way that the sole necessary input is a closed mesh. This mesh is analyzed and to calculate all the parts necessary for its construction. To allow this to work the mesh has to be free of any degenerate faces or holes. It is also beneficial if its faces are of even size and balanced ratio to allow for enough space for the substructure that holds the faces together. If all these requirements are satisfied the algorithm will be able to produce several outputs which are a wireframe or optional polysurface preview of the resulting parts and contours of all the parts on the same plane. The latter feature several details necessary for the correct creation of toolpaths like additional opsitions for drill holes and the seperation of complex parts into simpler elements for higher precision results with lower grade CNC routers.

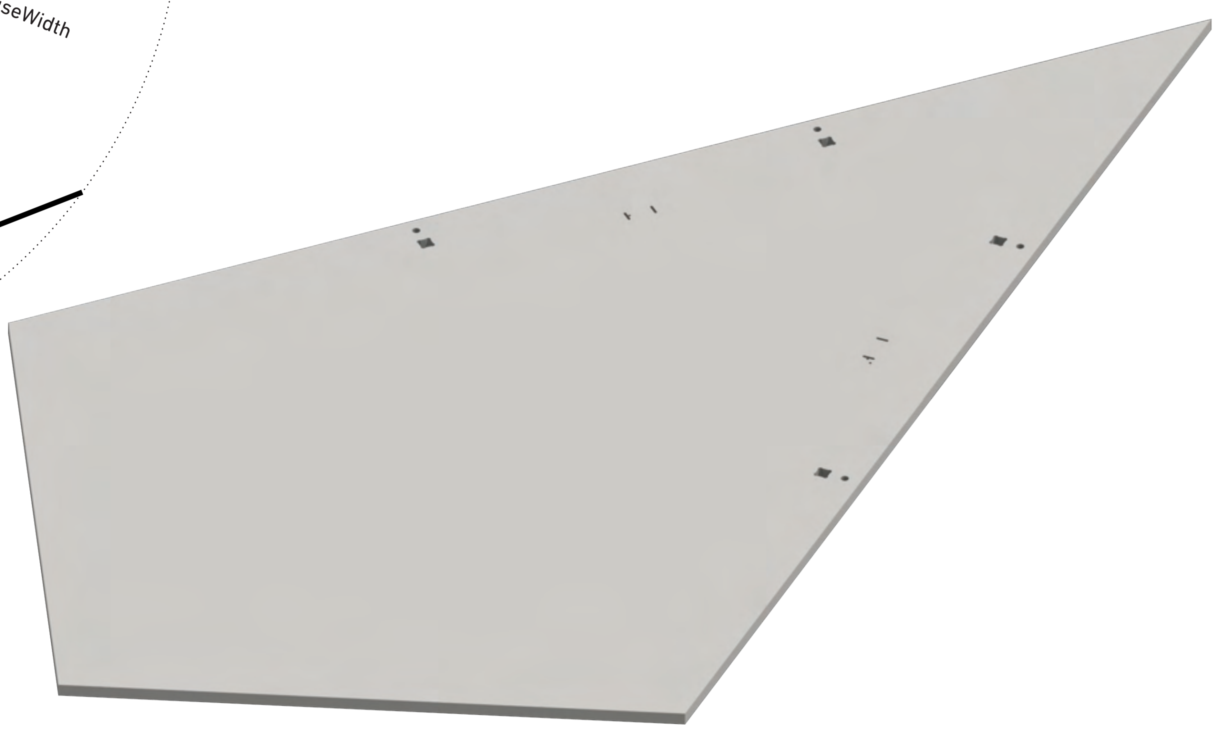
The algorithm starts by disecting the mesh to obtain all necessary values. The positions of vertices and their topology are used to determine all initial parameters such as angles between faces, the lengths of edges and a long list of values that can be derived from them. All these variables are stored in objects that then communicate with each other to calculate further data that are dependent on more complex interrelations.

By the time all calculations are completed the algorithm has produced a data-reflection of the mesh and the parts that will be created. The next steps it takes transform this data back into geometry and display it with the help of Rhinoceros 3D.

Previous page:
Figure 70: String diagram to manage the complexity of interdependencies of parameters necessary for the different elements created by the algorithm.

Opposite page:
Figure 71: This picture shows a bare mesh transitioning to the product of the algorithm viewed as a wireframe. All of the mesh's parameters necessary for its conversion to the building parts are labeled


shardEdgeLength
cutEdgeLength
β
shardScrewHoleconnectorSectionTenonMortiseDistance
shardEgdeShardScrewHoleDistance
shardScrewHoleDiameter
connectorSectionTenonMortiseLength
connectorSectionTenonMortiseWidth



Elements

In the following the three different types of building blocks will be introduced. The opposing and the two following pages show illustrations of examples for each of these part families indicating and naming the most important parameters that go into their creation.

1 Shards

Shards are parts of the original mesh faces and build the geometries shell which makes them the room creating elements. They tipically span two edges of a face between their common vertex and their respective midpoints and extend to the center of the face making them quadrilaterals. Their edges are colinear with convex edges of the mesh and offsets of mesh edges that are concave to avoid intersecting each other. They are interconnected with their neighboring shards by two *Connector Sections* each. A screw hole and a mortise build the interface for each of these connecting elements.

2 Connector Sections

These parts are the most complex of the three elements. They build the interface between *Shards* and *Connector Plates*. To accomplish this they boast three receptacles for nuts and additional connecting features to ensure a higher degree of stability. At the same time they are also the smallest of the three base elements.

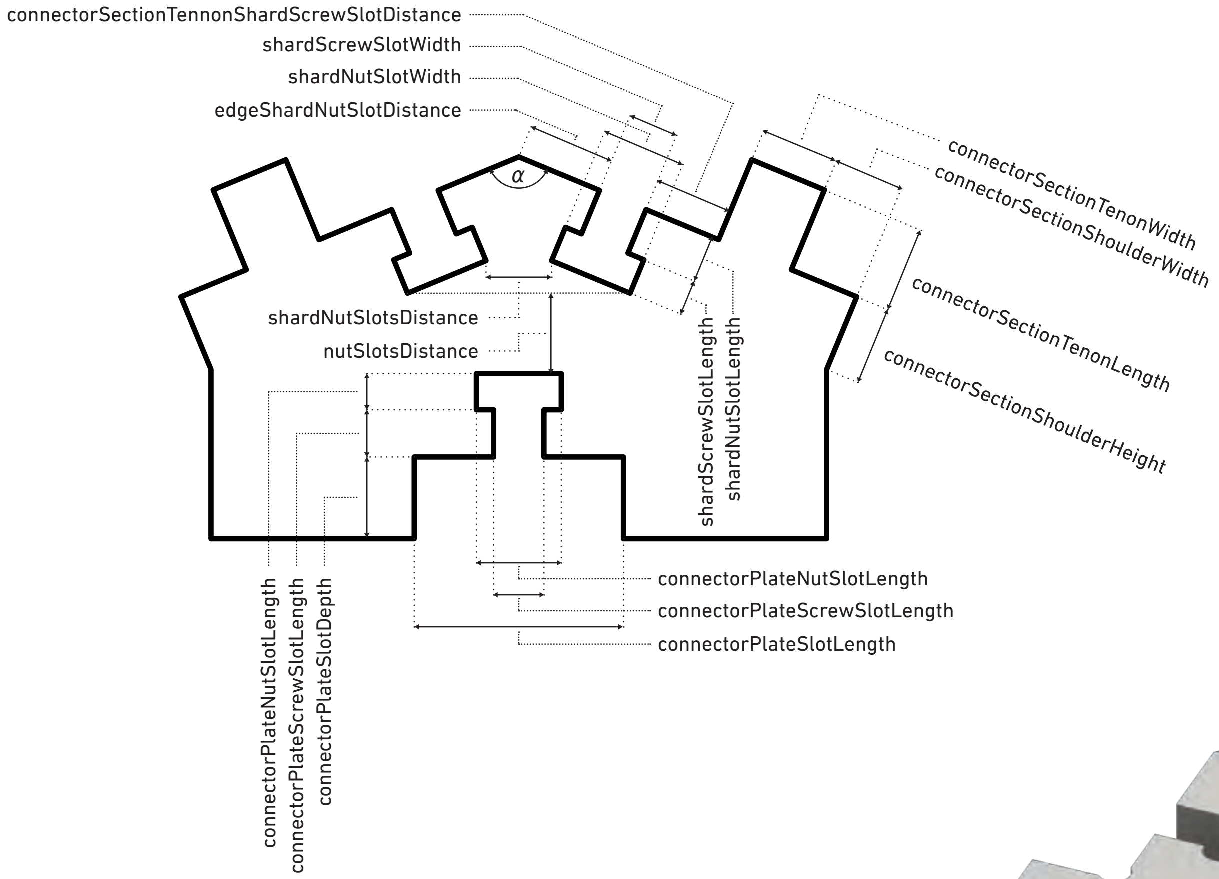
3 Connector Plates

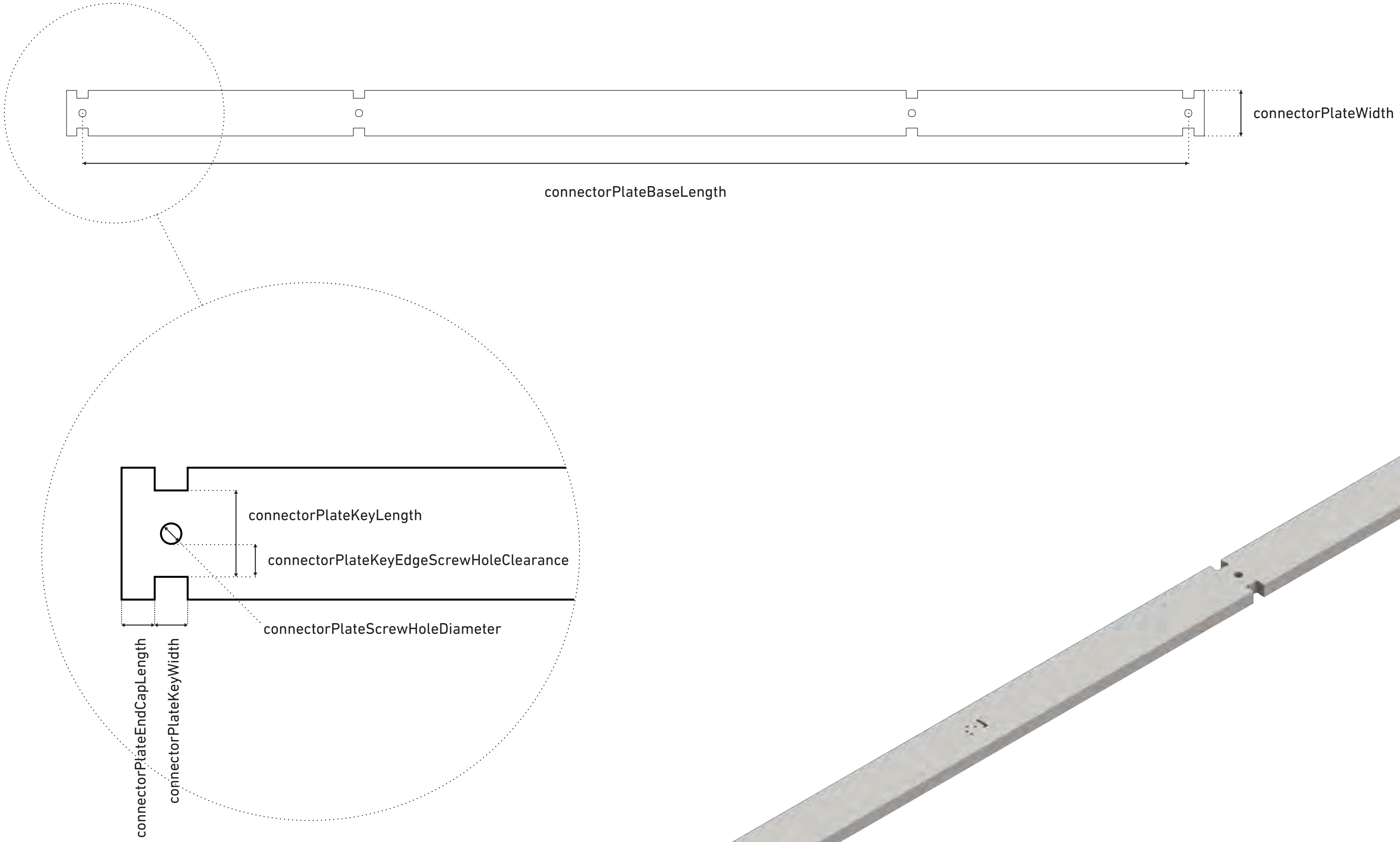
The job of *Connector Plates* is the interconnection of corner subassemblies which typically consist of three or more *Shards* and twice that number of *Connector Sections*. They are connected to the *Connector Sections* with a key and slot system and additionally fastened with a screw. There is one *Connector Plate* for each of a meshes Edges.

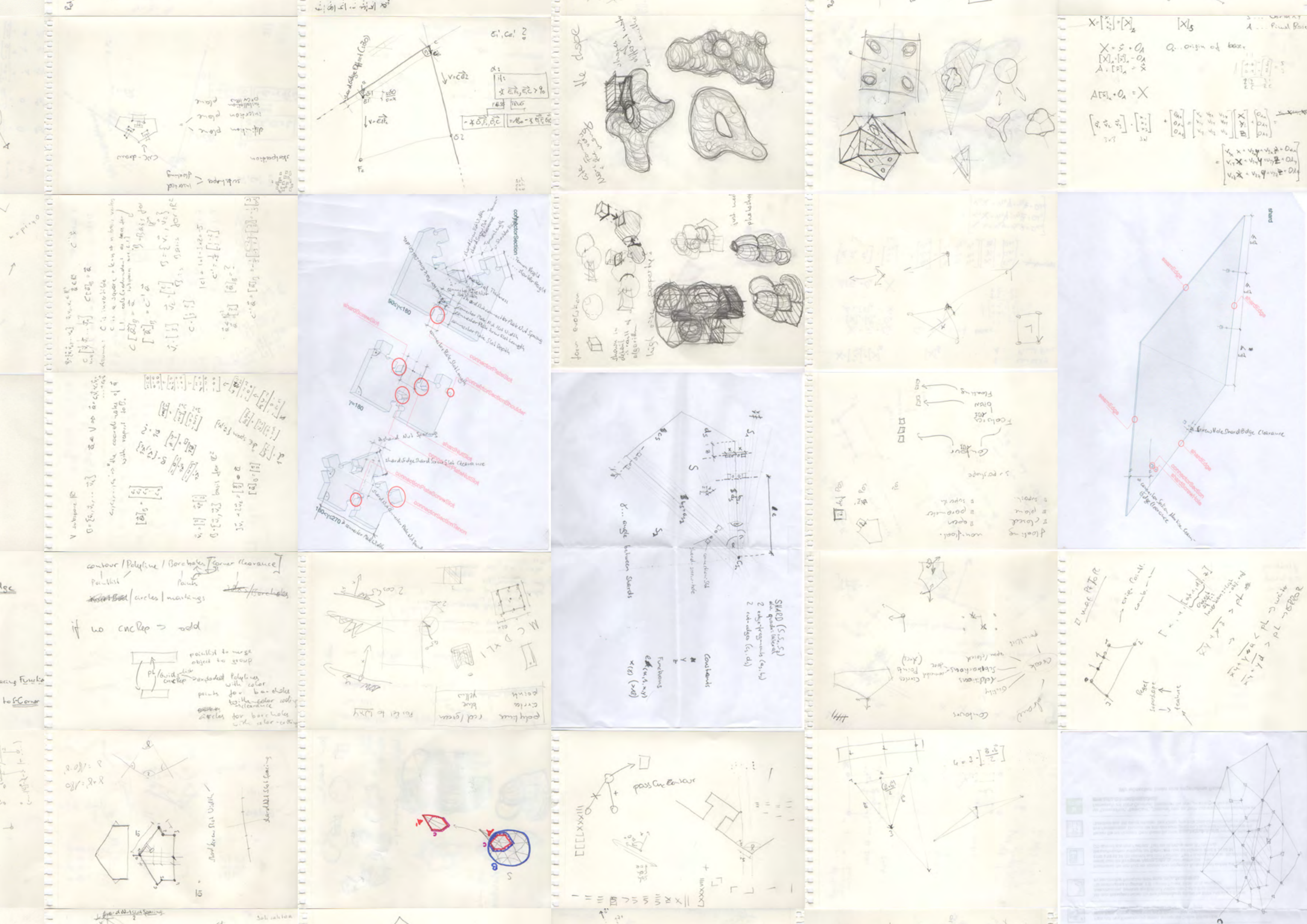
Opposite page:
Figure 72: Shard element details

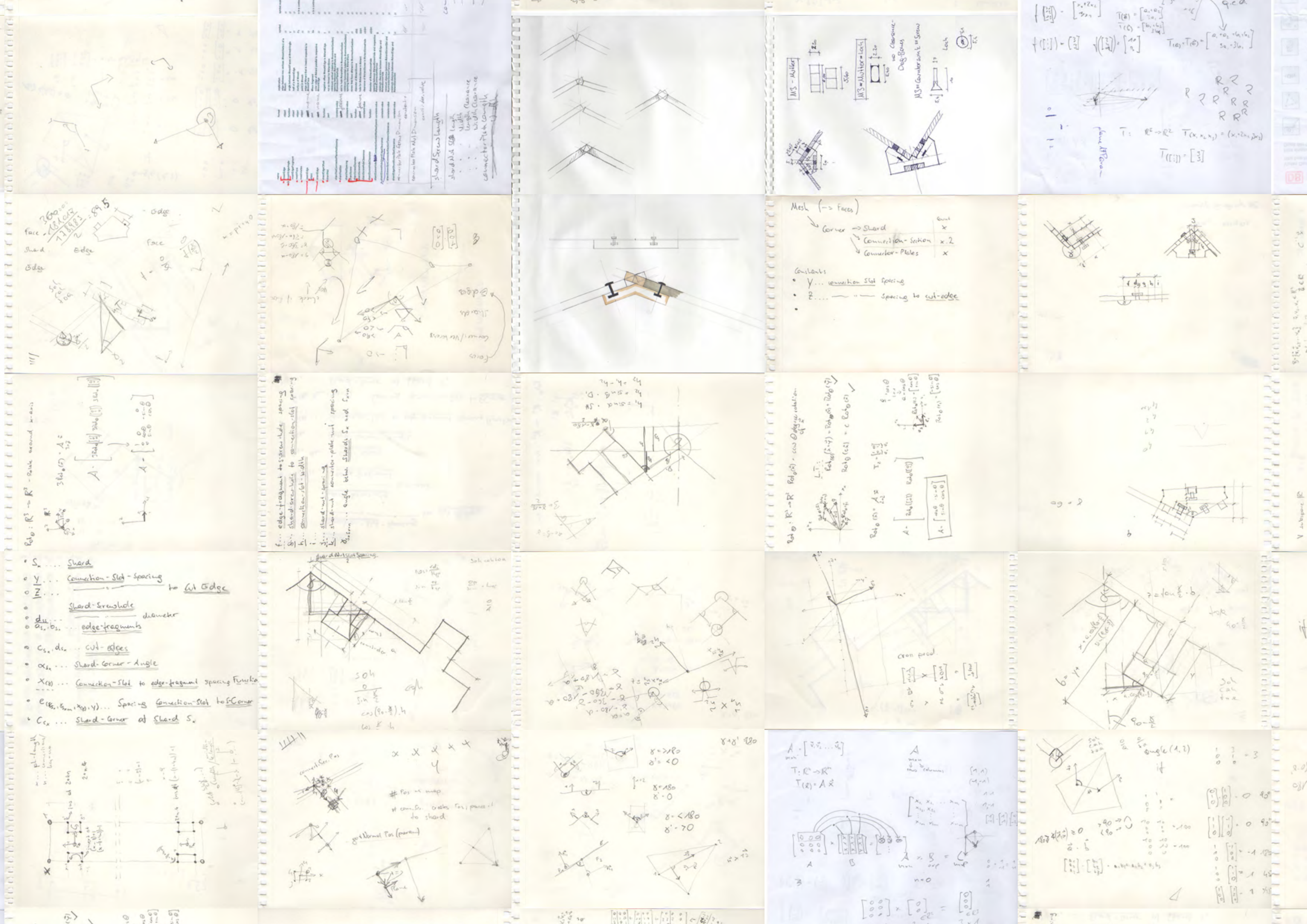
Next spread, from left to right:
Figure 73: Connector Section element details

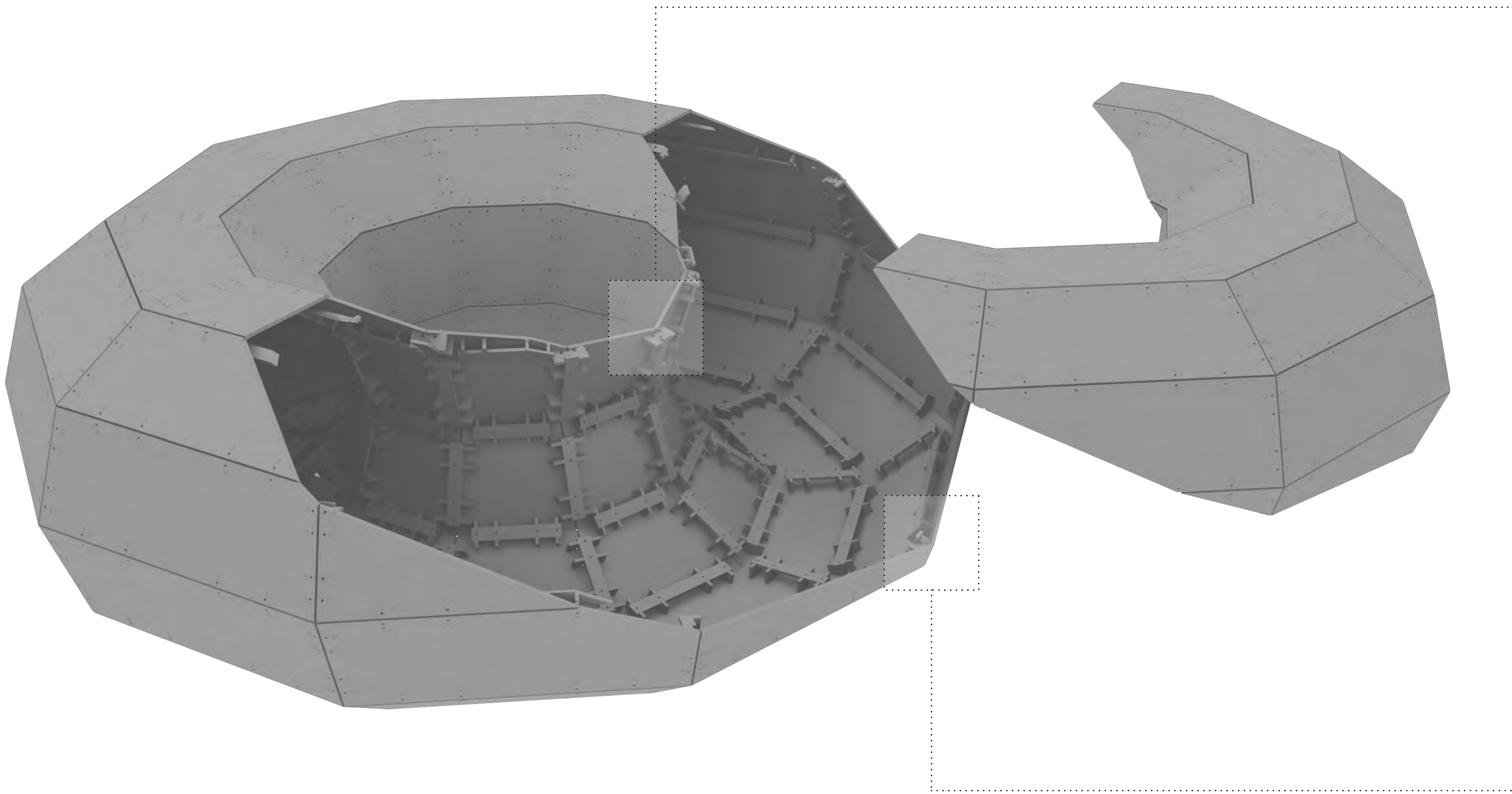
Figure 74: Connector Plate element details

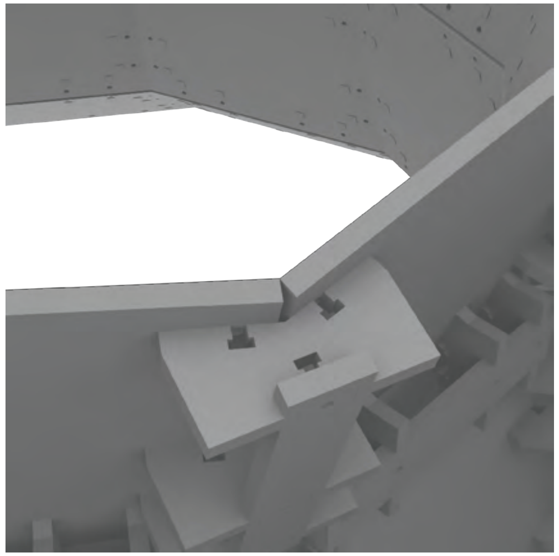

connectorSectionTennonShardScrewSlotDistance
shardScrewSlotWidth
shardNutSlotWidth
edgeShardNutSlotDistance
α
shardNutSlotsDistance
nutSlotsDistance
shardScrewSlotLength
shardNutSlotLength
connectorSectionTenonWidth
connectorSectionShoulderWidth
connectorSectionTenonLength
connectorSectionShoulderHeight
connectorPlateNutSlotLength
connectorPlateScrewSlotLength
connectorPlateSlotLength
connectorPlateNutSlotLength
connectorPlateScrewSlotLength
connectorPlateSlotDepth

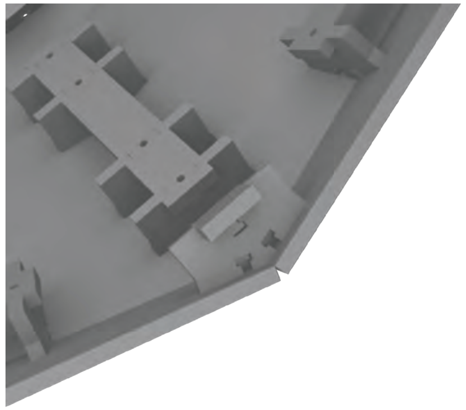

connectorPlateWidth
connectorPlateBaseLength
connectorPlateKeyLength
connectorPlateKeyEdgeScrewHoleClearance
connectorPlateScrewHoleDiameter
connectorPlateEndCapLength
connectorPlateKeyWidth


contour / Polyline / Borcholes [Corner Clearance]
circles / markings
cncRep
connectorSection
seamEdge
shardEdge
shard
SHARD (S1, S2, S3)
Constants
Functions
α ... angle between Shards
passCncContour
form evolution
Q ... origin of base


Mesh (-> Faces)
Corner -> Shard
Connection-Section x.2
Connector-Plates
Constants
Y ... connection slot spacing
Z ... Spacing to cut-edge
S_x ... Shard
Y ... Connection-Slot-Spacing
Z ... to Cut Edge
Shard-Screwhole
d_sc ... diameter
a_s, b_s ... edge-fragments
c_s, d_s ... cut-edges
α_s ... Shard-Corner-Angle
X(a) ... Connection-Slot to edge-fragment spacing Function
e ... Spacing Connection-Slot to S-Corner
C_s ... Shard-Corner at Shard S_x
M5-Mutter
M3 = Mutter + Loch
Loch
T: R^2 -> R^2
T(x) = A x
cross prod.
angle(1,2)
Face
Shard
Edge







Complications

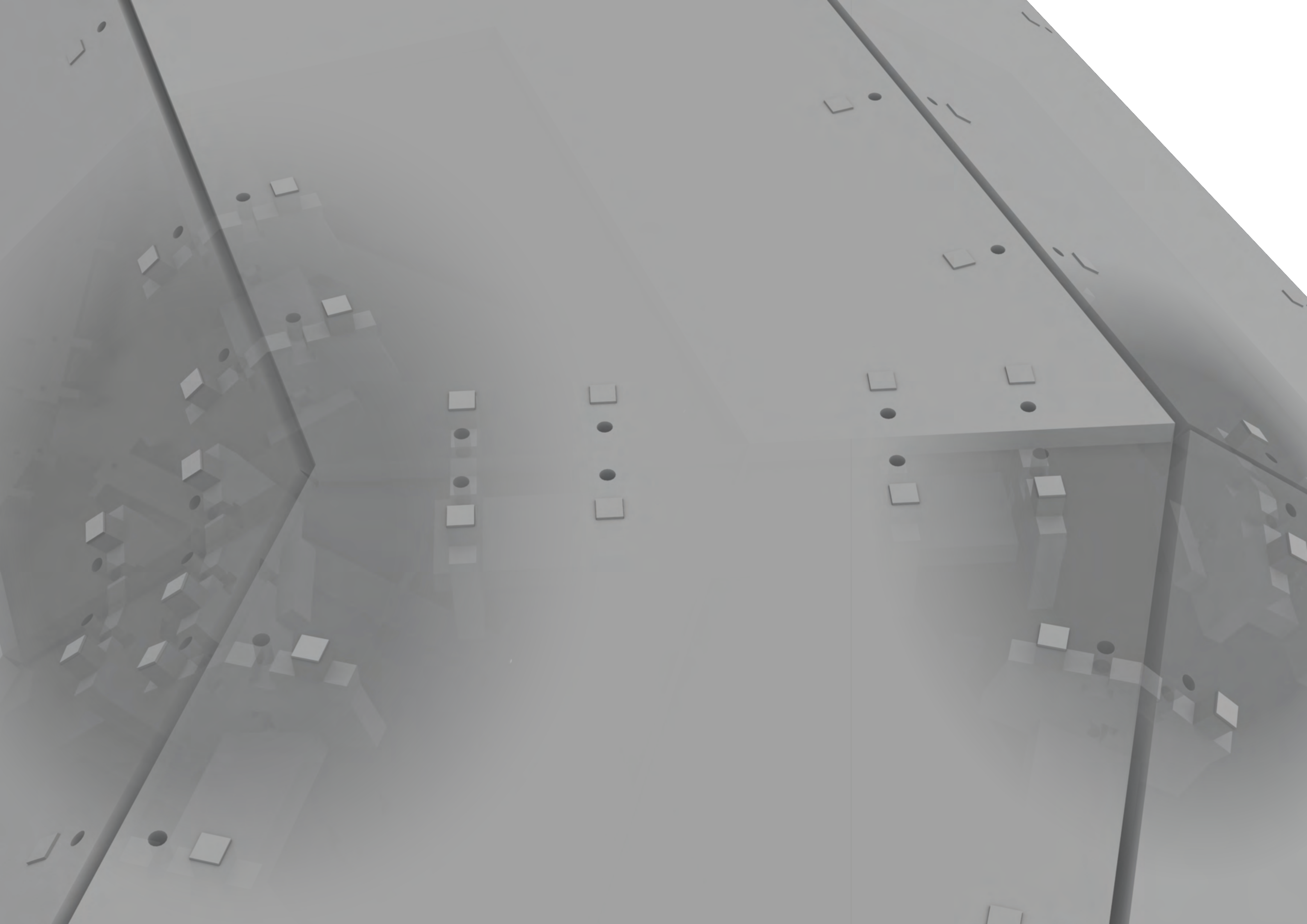
1 Convex and concave edges

Faces of the mesh that enclose an angle beyond 180 degrees constitute two additional challenges in the process of automating the creation of a physical structure resembling their geometry. The first complication lies in the correct determination of their inner angle. To accomplish this it is not enough to simply calculate the angle between their respective normal vectors because this will always yield an angle below 180 degrees. It is necessary to evaluate the normals' relation to a third vector that connects the faces' centers.

After this is burden is overcome the elements building the shell of the geometry have to be altered to avoid reciprocal intersection along the edge. This is achieved by offsetting the respective edges of the shell elements, which in this case are the *Shards* towards the inside of the concerned face by a value that can be trigonomet-

Previous page:
Figure 75: Collage of notes taken during the conception of the algorithm

Left:
Figure 76: A sliced torus after the execution of the algorithm revealing the inner structure of the final artefact. Details on the right depict the difference between the solutions to concave (top) and convex (bottom) edges at a level of the final elements.



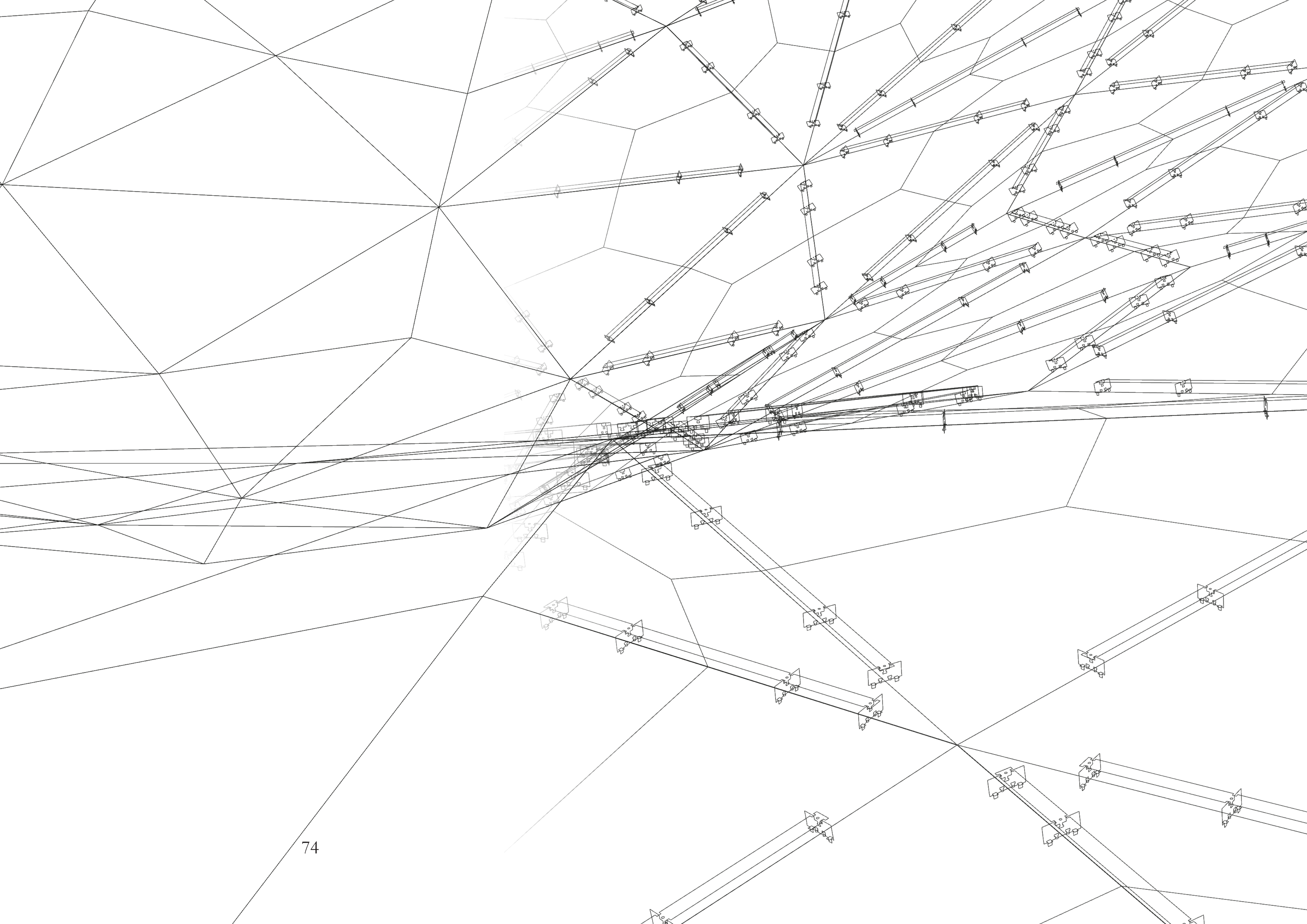
rically determined as a function of the angle and the elements' thickness.

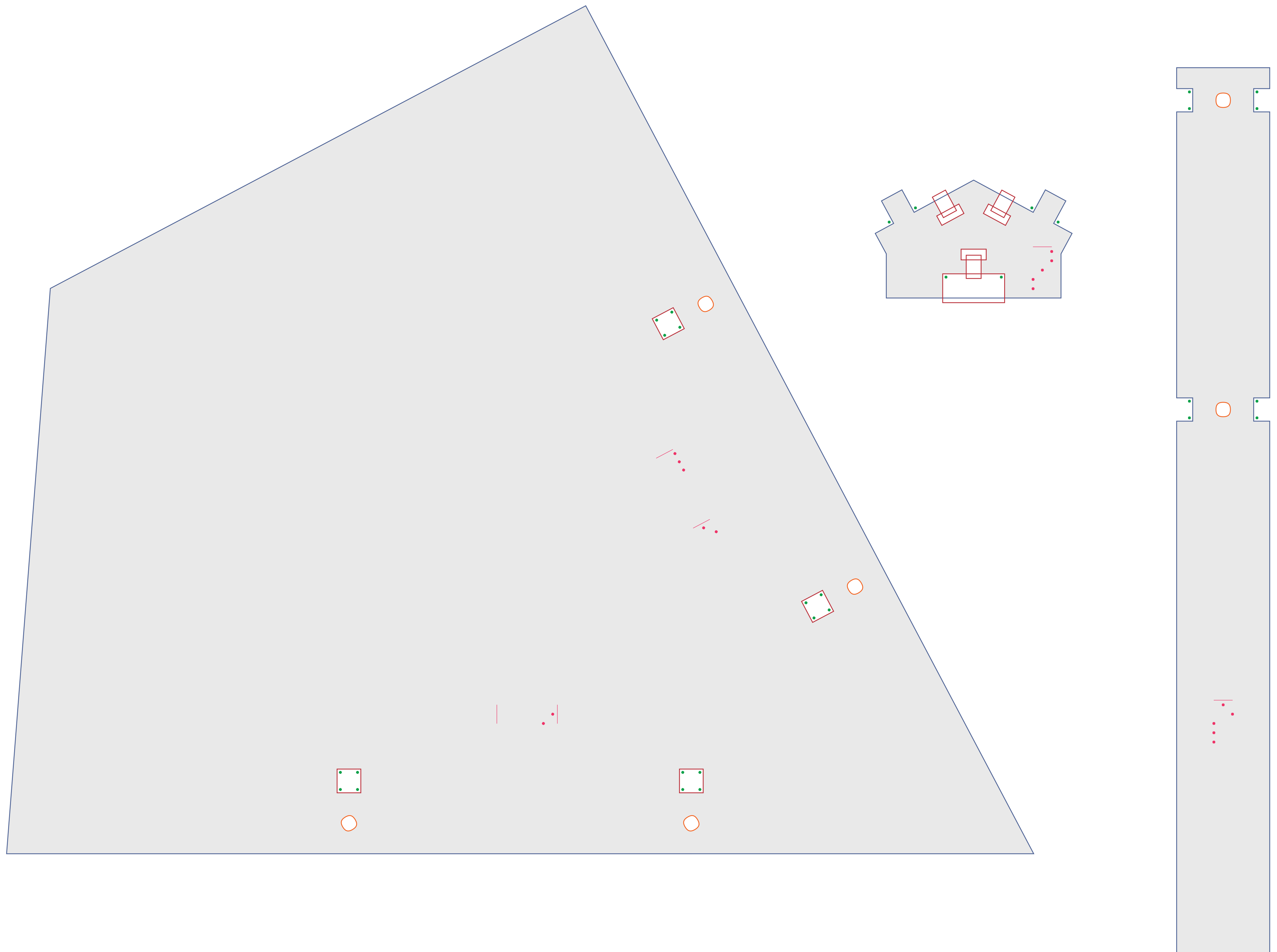
2 Multi dependencies

Mostly spacings between features of elements like slots can be dependant on more than one minimum clearance. It then depends on an additional indirectly connected value which one of the aforementioned parameters takes effect. One of these cases is also connected to concave and cenvex edges. Their status determines the position of the screw slots and tenons of the *Connector Sections*. While a convex edge will result in the slots moving together too far a concave edge will have them move apart from each other. The value with the most dependencies of all is certainly the *Connector Sections* overall height which at the same time is of great importance to guarantee the part's structural performance.

Figure 77: Transparent shards give a view of the underlying structure. Clearances of screwholes and mortises as well as positions of tenons and screwslots and positions of shard edges have to be calculated differently depending on the concave or convex nature of the respective edge.

Next spread:
Figure 78: The progression from mesh (left) to 3d preview (right)





CNC Output

1 Contours

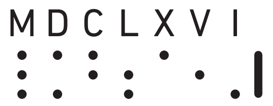
Contours are the outer boundaries of parts but also inner recesses. These have to be distinguishable easily as they have to be cut in a specific order starting with inner cuts. Additionally complex contour’s like the ones of *Connector Sections* are split up into sub-contours to improve the precision of the milling process wile allowing for greater feedrates. This feature of the algorithm while complicated to achieve was especially crucial as there are thousands of parts for every slightly more complex geometry and in consequence the necessity for slight decreases in cutting speed result in tremendous increases of the overall fabrication time.

2 Clearance drillings

Clearance drillings as indicated by green dots in the illustration are necessary because a round cutter as is used in CNC milling can not achieve to cut sharp inner corners. Hence bore holes are drilled at each of the corners which have to be fully cleared.

3 Bore holes

Bore holes for screws are cut with a different type of toolpath which makes it necessary that they too are rendered seperately.



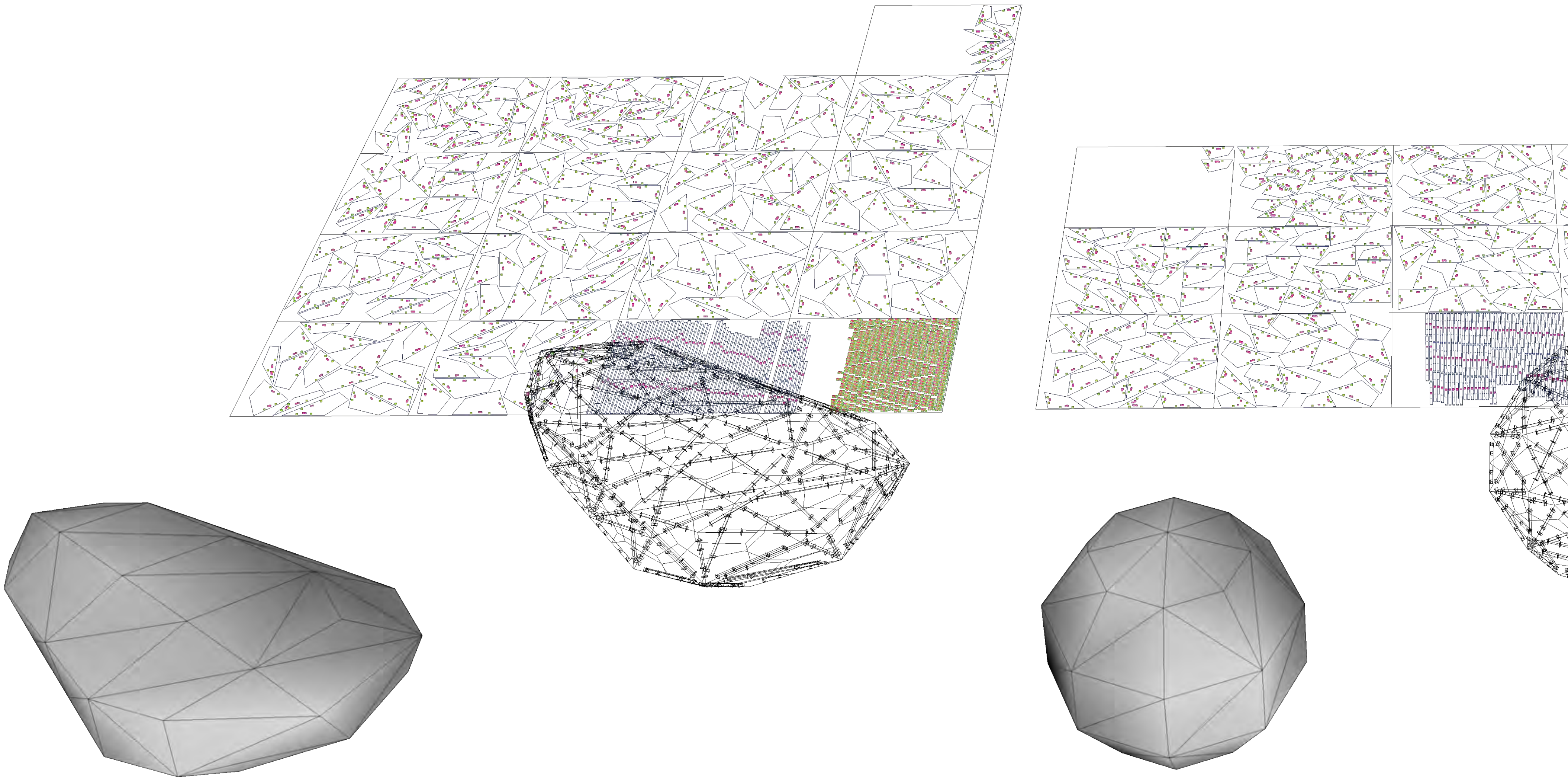
4 ID codes

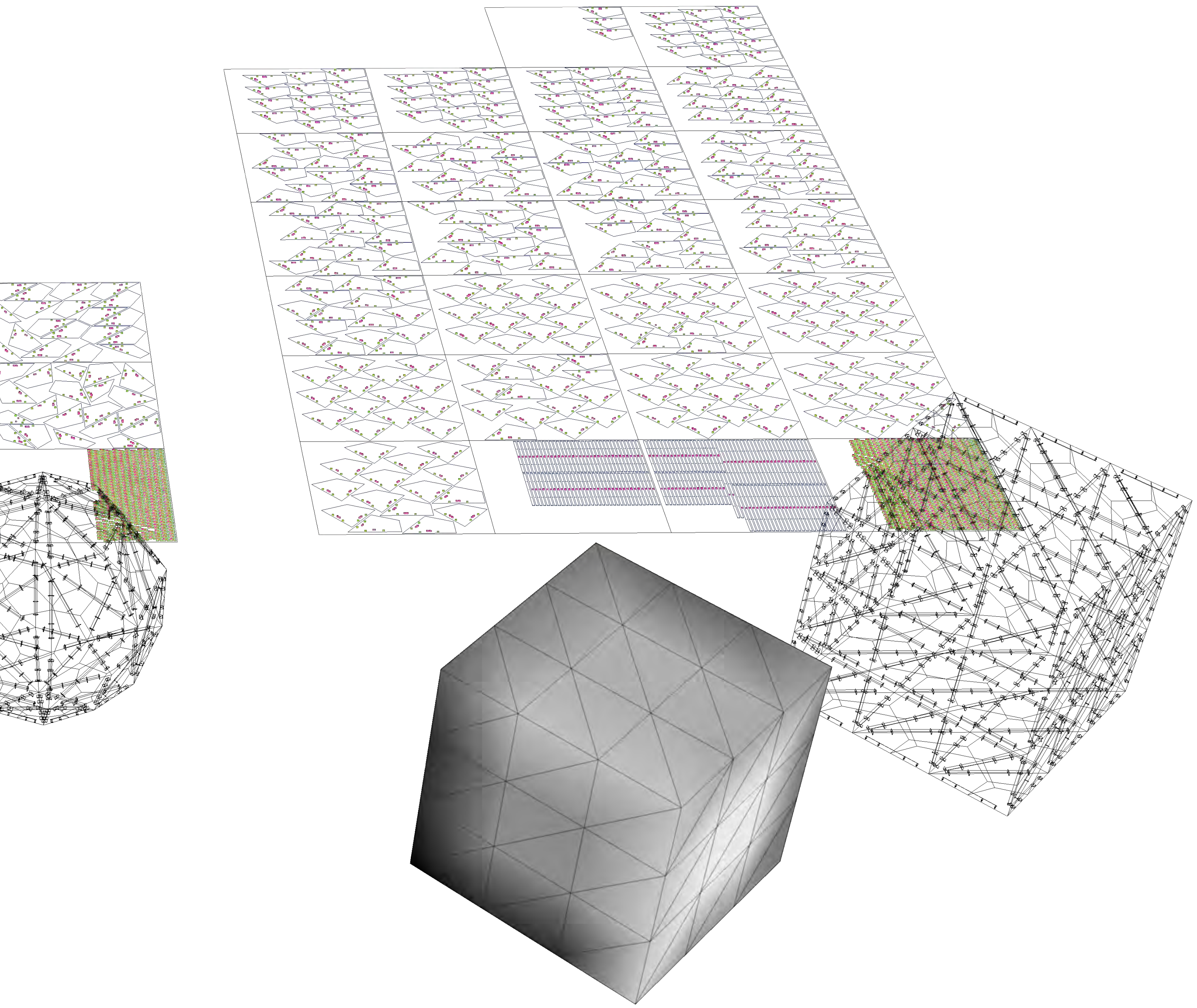
As a set of parts can easily exceed a thousand elements it becomes imperative to implement a system that allows their quick identification. For this purpose a code based on roman numerals was developed and optimised for the application with a CNC mill. The letters were changed into patterns of drill holes and a straight line signals the end of a code also indicating the direction of reading.

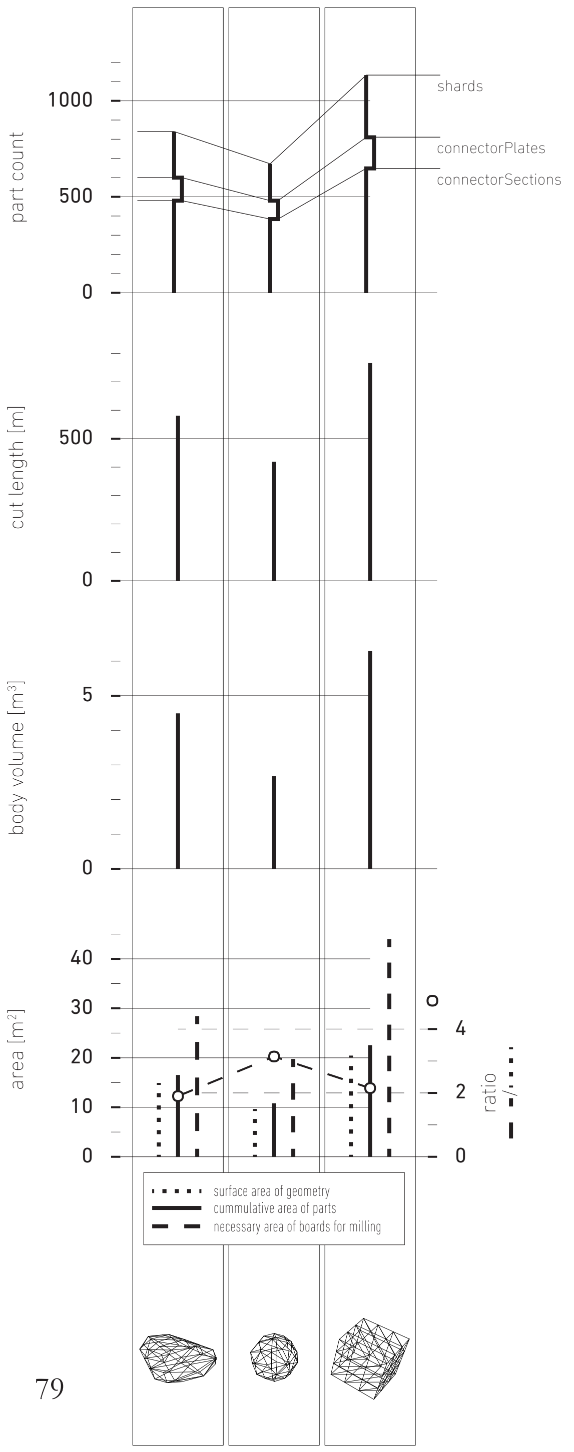
Opposite page:
Figure 79: The algorithm’s output for processing by a CNC toolpath generation software. The different entities are color coded for convenient processing.

Above:
Figure 80: The key to the code used to ID the parts with encoded roman numerals.

Next spread:
Figure 81: Algorithm outputs for different types of geometries with analysis of their characteristic effects on several key values as follows: number of necessary parts; accumulated length of contours that have to be cut along; their enclosed volumes; a juxtaposition of the area of the initial geometries’ surfaces, the cummulative surface of all parts and the combined gross area of the necessary boards of plywood.

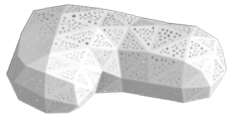


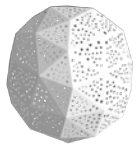


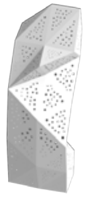


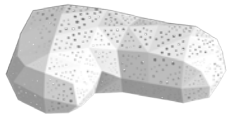
part count
1000
500
0
shards
connectorPlates
connectorSections
cut length [m]
500
0
body volume [m^3]
5
0
area [m^2]
40
30
20
10
0
ratio
4
2
0
surface area of geometry
cummulative area of parts
necessary area of boards for milling

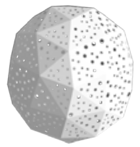
6 Exemplary Extrapolations

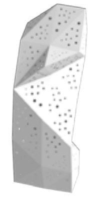

0.0185
1350
72

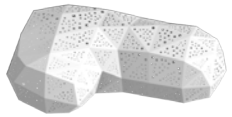

0.0474
1502
80

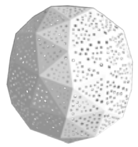

0.0302
2718
144

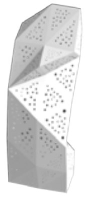

0.0134
640
72

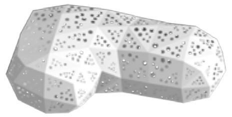

0.0388
712
80

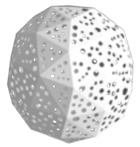

0.0265
1288
144

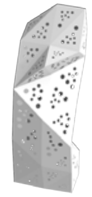

0.0183
1208
72

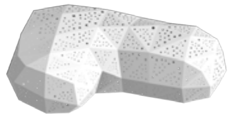

0.0480
1344
80

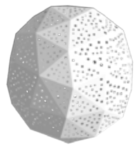

0.0302
2432
144

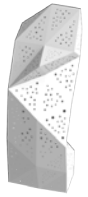

0.0375
498
72

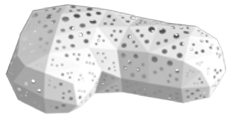

0.0833
554
80

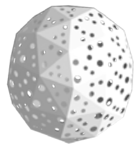

0.0567
1002
144

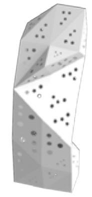

0.0145
1066
72

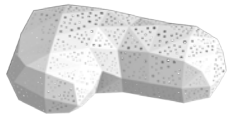

0.0373
1186
80

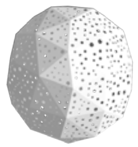

0.0250
2146
144

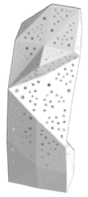

0.0232
214
72

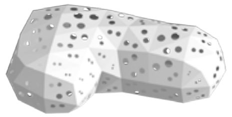

0.0779
238
80

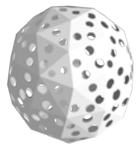

0.0496
430
144

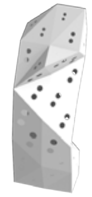

0.0168
782
72

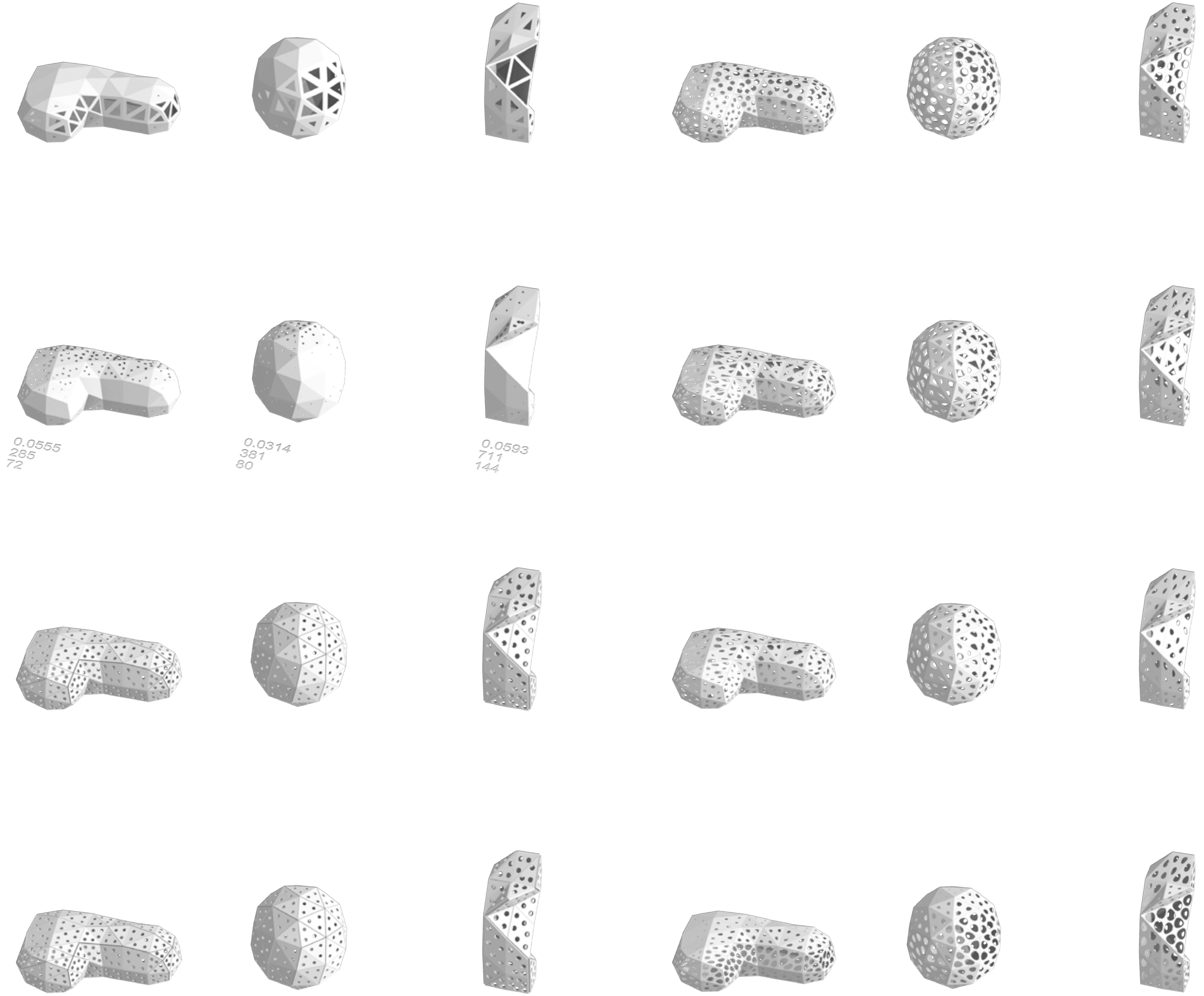

0.0437
870
80


0.0280
1574
144


0.0186
72
72


0.0960
80
80


0.0468
144
144


0.0555
285
72
0.0314
381
80
0.0593
711
144

1 Apertures

The fully parametric definition of the building with an algorithm offers a convenient way to create openings in the hull as a function of further parameters such as the path of the sun and climate to state the obvious, but also more complex circumstances as functions of adjacent spaces. The previous pages show a number algorithmic studies performed on different types of geometries along such parameters as exposure to sunlight. The object oriented structure of the algorithm allows for convenient adaptation to such concepts.

Next to parametric aesthetics another interesting opportunity lies ahead if one considers reapplying the same restrictions to apertures that have been imposed on the creation of the structure. Those include the elimination of permanent connections and for the current prototypical and conceptual phase do not allow any other machines than cnc routers for production of all used parts. The result would have to be a very different reinterpretation of what we imagine to be a window, door or gate.

2 Load Bearing System

OMA's CCTV Building in Beijing, China exhibits a spectacular exoskeleton that on first inspection might seem like a purely formal gesture. Contrary to that it is the result of a complex parametric process regulating the structures mesh size according to structural demands. Arup, the engineering contractor for the building states that the finer the grid is in any area of the buildings surface the higher the loads that occure at that point. This essentially renders the buildings hull a gigantic data visualization at full scale.Especially because of its parametric nature this concept appears as an interesting archetype. It achieves different performances by varying the density of deployment of identical elements.

Translating it to this works algorithm and its paradigms results in beautifully organic results. The structure produced by the code is already naturally stronger wherever the faces of the underlying mesh are smaller as this increases the ratio of sturctural elements in the given are. Using structural analysis algorithms meshes could be prepared to exhibit characteristics that when translated into buildings become efficient structures.

Previous spread, from left to right:
Figure 82: Surface to aperture ratio studies with different parameters of aperture size and spacing.

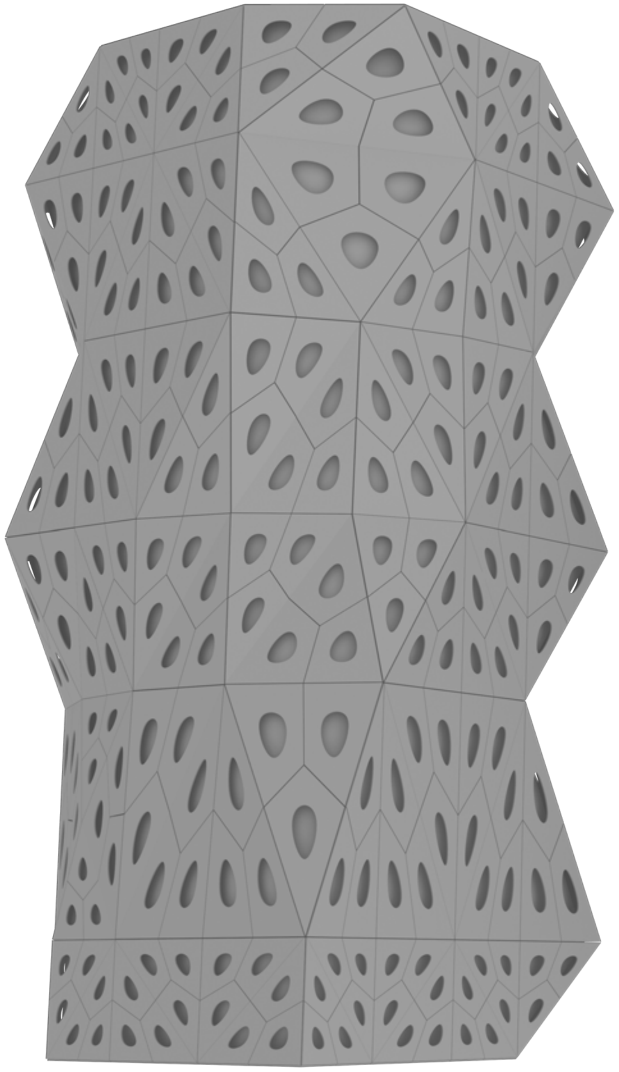
Figure 83: Studies of different shapes of openings; several studies looking at the consideration of the sun's path to maximize and minimize exposure.

Opposite page: clockwise from top:
Figure 84: OMA's CCTV Building, Beijing.

Figure 85: The exoskeleton changes in density to accomodate different loads.

Figure 86: The building's outer structure allows it to withstand earthquakes.

Next page:
Figure 87: An experiment to apply the CCTV building's structural concept to the mesh concept of this work's algorithm.



7 The Artefact

Key Facts

Surface area of Artefact: approx. 11m^2
Volume of Artefact: approx.

Custom parts: 2875
 shards: 700
 connector sections: 1400
 connector plates: 350
Screws and nuts: 4200 each

Plywood used (600mm by 490mm sheets): 21m^2 (72)
Holes drilled by CNC Router: 37 871
Total length cut by router: 565m
Total length cut by lasercutter: 314m
Total length etched by lasercutter: 37m

Previous spread, from left to right:
Figure 88: Milling and adjacent simultaneous sorting.

Figure 89: Close-up of a shard being cut.

Next Spread:
Figure 90: Final Assembly

Following Spreads:
Finished Artefact and Details
Figure 91
Figure 92
Figure 93
Figure 94
Figure 95
Figure 96
Figure 97
Figure 98

Production Processes

The following processes were necessary to manufacture the ertefact.

1. The geometry had to be created virtually.

2. All parts are calculated by the algorithm.

3. The wooden parts are CNC milled.

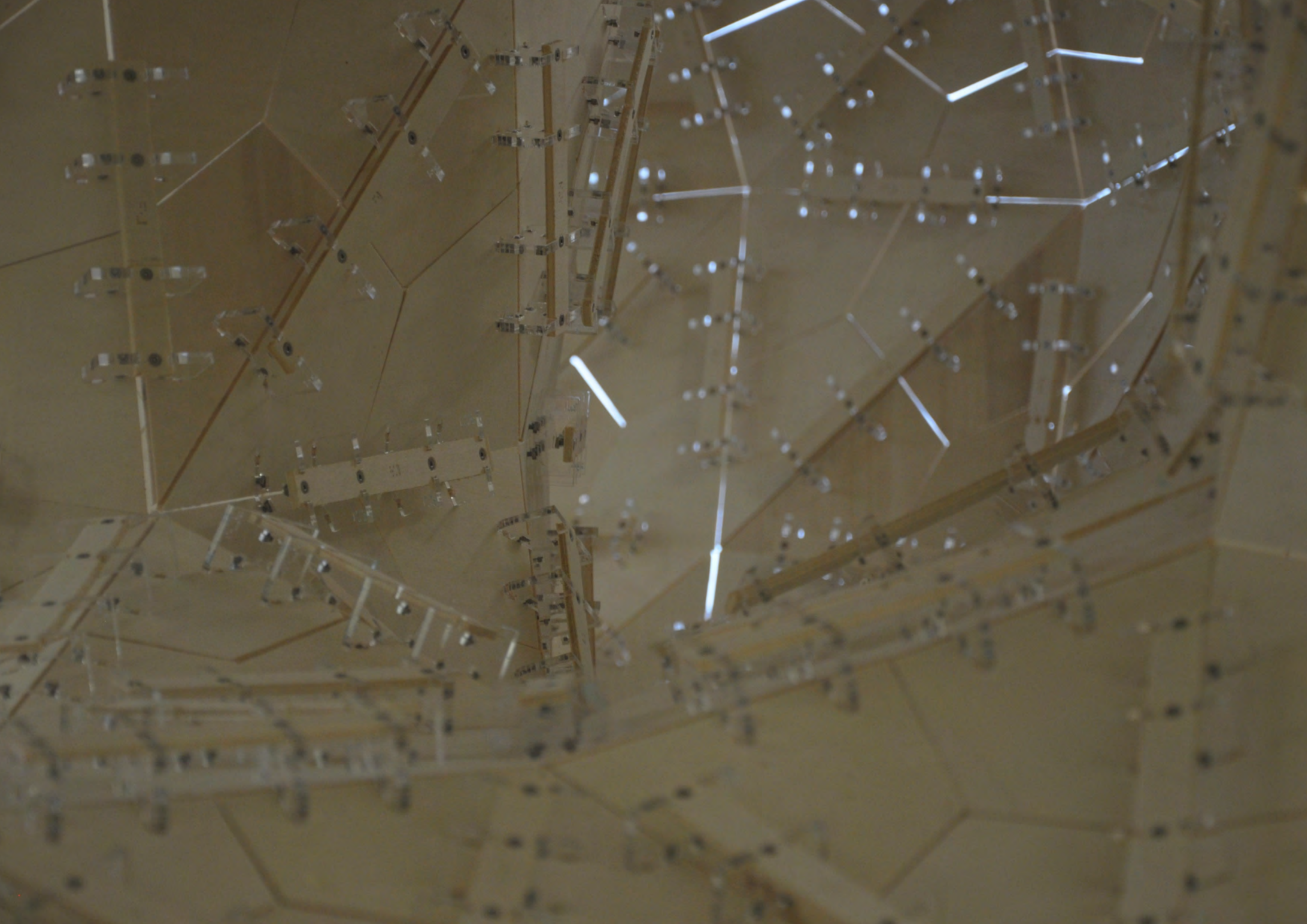
4. Certain parts are laser cut from acrylic because of their more detailed complex shape. This was outsourced after the insufficiency of the cnc milled parts was discovered.

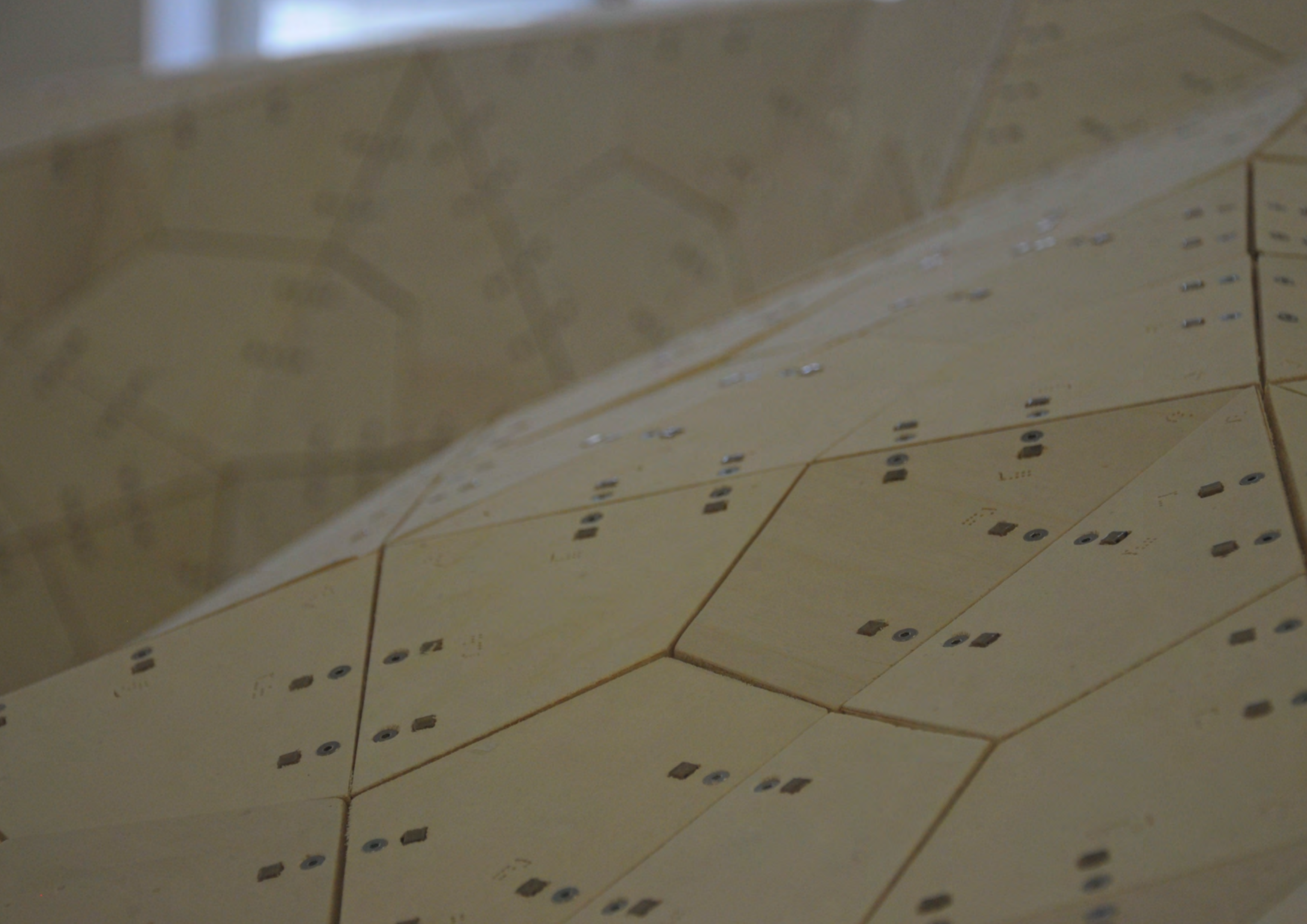
5. Several sorting operations have to be performed on all part categories. Most of the parts are grouped to later be combined for the creation of the lowest orders of subassemblies. Sorting operations took a very significant amount of manpower to complete and hence are the biggest weak point in the process.

6. Connector Sections have to be processed to hold three nuts each which is accomplished with the help of adhesive tape that holds the nuts in place until they are used in subassemblies and final assembly. Only then the adhesive tape is removed from all Connector Sections.

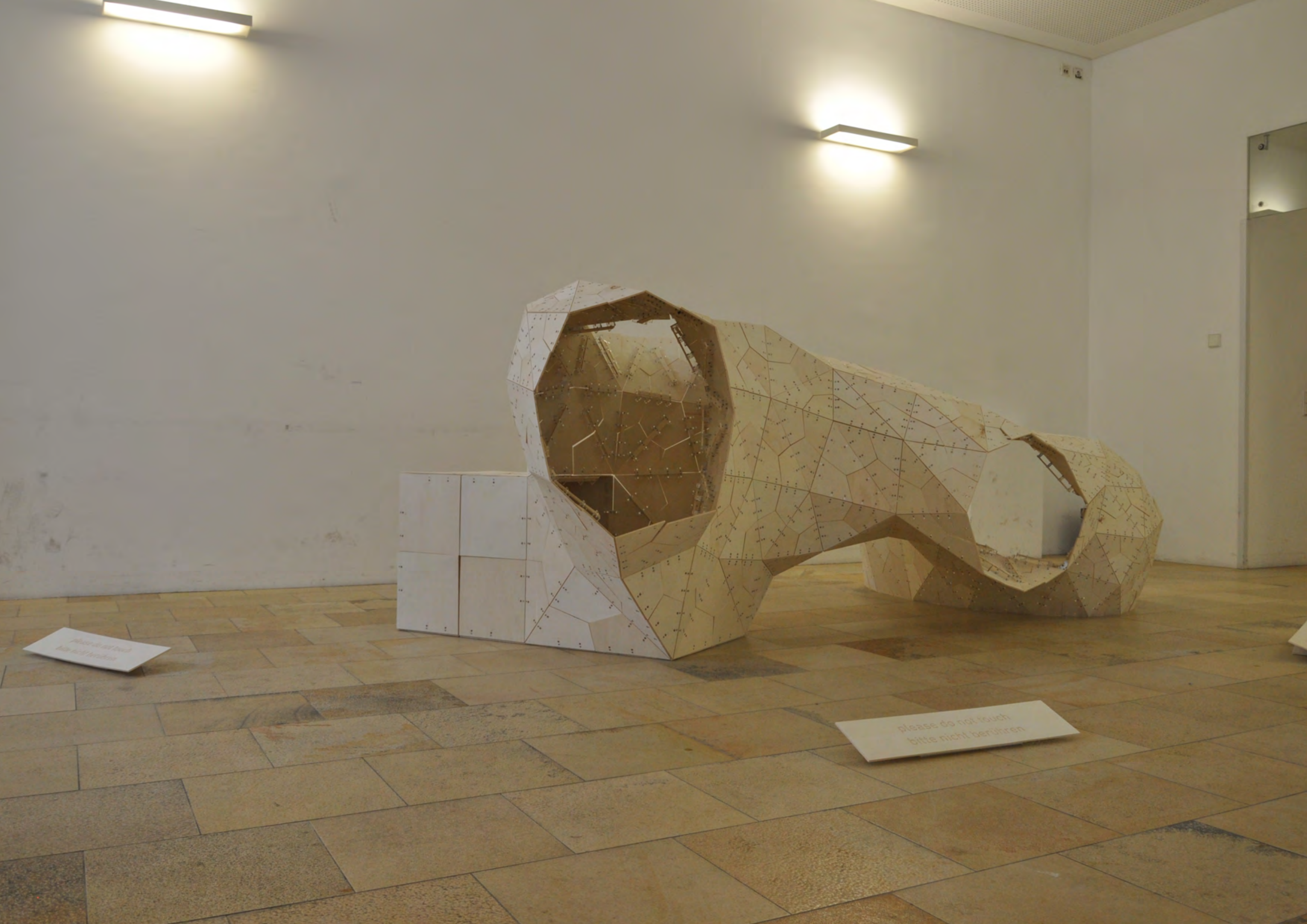
7. Corner Subassemblies are formed from adjacent Shards and associated Connectorsections.

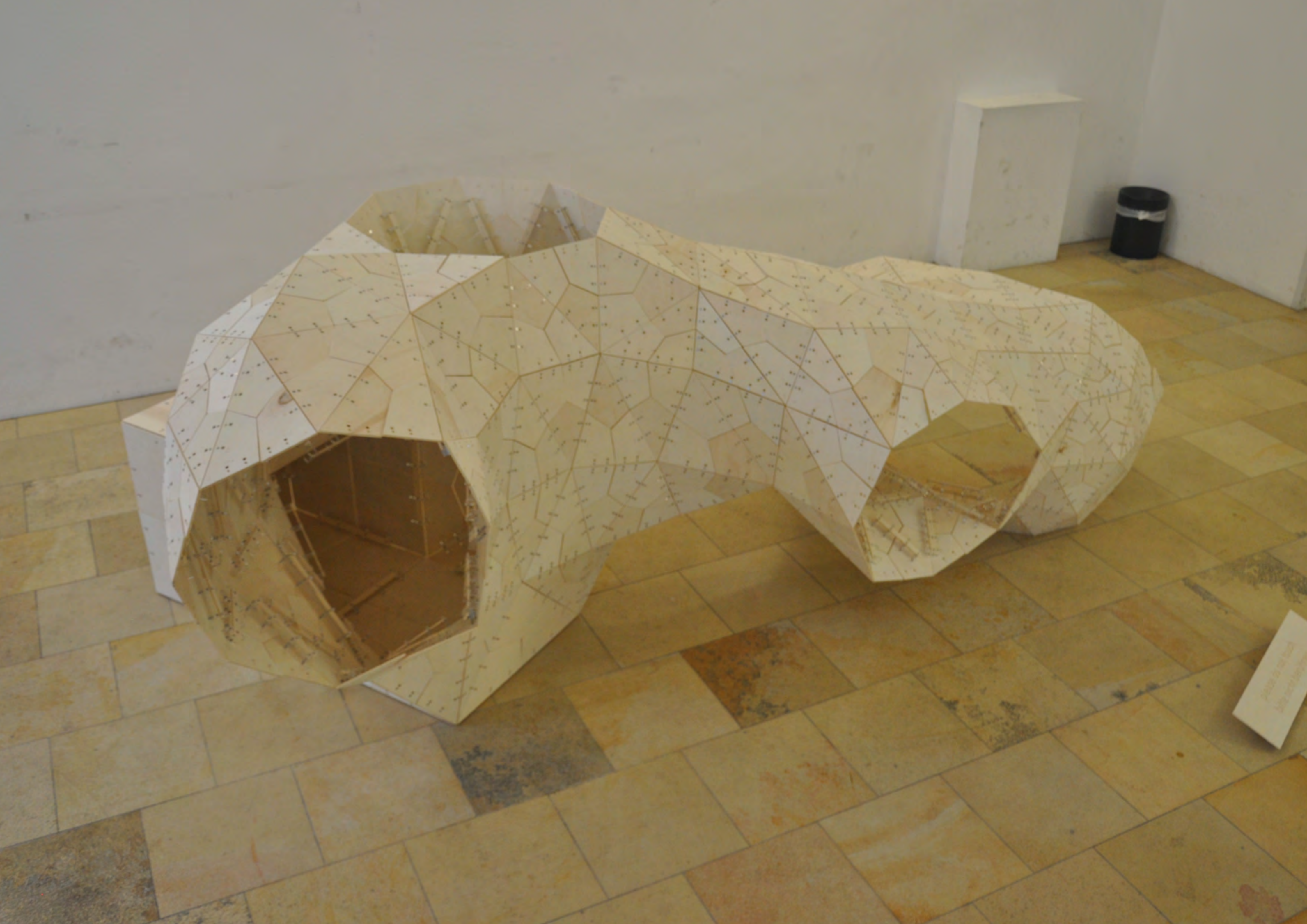
8. Subassemblies are interconnected to larger junks or added to the final whole structure depending on convenieance





please do not touch
bitte nicht berühren
please do not touch
bitte nicht berühren


please do not touch
bitte nicht berühren
please do not touch
bitte nicht berühren



Supplemental

The Code

1 *Geometry Classes*

```
import rhinoscriptsyntax as rs
import Rhino
import math
import copy
from Contour import ConnectorSection, ConnectorPlate, Shard
from miscFunctions import *

class Mesh(object):

    def __init__(self, mesh):
        self.mesh = mesh
        self.faceNormals = rs.MeshFaceNormals(self.mesh)
        self.faceVertices = rs.MeshFaceVertices(self.mesh)
        self.vertices = rs.MeshVertices(self.mesh)
        self.faces = []
        self.edges = []
        self.corners = []
        self.shards = []
        self.createFaces()
        self.createEdges()
        self.createCorners()
        self.createShards()

    def createShards(self):
        self.shards = [[Shard(mesh = self, cornerID = vertexID, face = face) for
vertexID in face.getPointIDs()] for face in self.faces]


    def printStats(self):
        for edge in self.edges: edge.printStats()
        for face in self.faces: face.printStats()

    #getNormal is used to get the face-normal of a face specified via its index in the
mesh’s face-list
    #Preconditions:
        #faceID... the index of the face in mesh.faces whose normal is needed
    #Postconditions:
        #RETURN is the normal of the specified mesh-face as Rhino.Geometry.Vector3d
    def getNormal(self, faceID):
        return self.faceNormals[faceID]

    #Postconditions:
        #RETURNs self.faces
    def getFaces(self):
        return self.faces

    #Postconditions:
        #RETURNs self.edges
    def getEdges(self):
        return self.edges

    def getCorners(self):
        return self.corners

    #creates Face-objects for all faces of mesh

    def createFaces(self):
        #print self.faceVertices
        for face, normal, center in zip(self.faceVertices, self.faceNormals,
rs.MeshFaceCenters(self.mesh)): self.faces.append(Face(face, normal, center, self))
        return
    #creates Edge-objects for all faces of mesh

    def createEdges(self):
        edgeVertices = []

        #obtain edge-vertices from all faces and sort each edge's vertices
        for faceObject in self.faces:
            face = faceObject.getPointIDs()
            zipped = zip(face, face[1:] + face[:1])
            faceEdges = [edge if edge[0]<edge[1] else (edge[1], edge[0]) for edge in
zipped]
            edgeVertices.extend(faceEdges)
        #print 'edgeVertices all'
        #print edgeVertices
        #filter duplicates
        edgeVertices = list(set(edgeVertices))

        #create Edge-objects and append to self.edges
        for edge, idx in zip(edgeVertices,range(len(edgeVertices))): self.edges.
append(Edge(edge, self, idx))


        return

    #create a Corner-Object for every Vertex

    def createCorners(self):
        self.corners = []
        for cornerID in range(len(self.vertices)):
            self.corners.append(Corner(cornerID, self))

    #return actual Point3d for Vertex-ID
    def getPoint(self, ID):
        return self.vertices[ID]
```

```
class Edge(object):

    def __init__(self, pointIDs, parentMesh, idx ):
        self.pointIDs = pointIDs
        self.edgeID = idx
        self.faces = []
        self.connectorSections = []
        self.connectorPlate = None
        self.mesh = parentMesh
        self.midpoint = self.calcMidpoint()
        self.findFaces()
        self.angle = self.calcAngle()
        self.shardEdgeOffset = self.calcShardEdgeOffset()
        self.shardScrewPosition = self.calcShardScrewPosition()
        self.connectorSectionTenonPosition = self.calcConnectionSectionTenonPosition()
        self.spawnConnectorSections()
        self.spawnConnectorPlate()
        #print ‘EDGE....................................’
        #print ‘Angle:’
        #print self.angle
        #print ‘shardEgeOffset:’
        #print self.shardEdgeOffset
        #print ‘shardScrewPosition:’
        #print self.shardScrewPosition
        #print ‘connectorSectionTenonPosition:’
        #print self.connectorSectionTenonPosition

    def getEdgeId(self):
        return self.edgeID

    #returns the Edges Vector
    def getEdgeVector(self):
        return rs.VectorCreate(self.mesh.getPoint(self.pointIDs[1]), self.mesh.
getPoint(self.pointIDs[0]))


    #Postconditions:
        #RETURNs the connectionSectionTennonPosition along the edge via distance to the
shardScrewPostion
    def getConnectorSectionTenonPosition(self):
        return self.connectorSectionTenonPosition


    #Postconditions:
        #RETURNs the shardScrewPosition along the edge via distance to the shardEdge
    def getShardScrewPosition(self):
        return self.shardScrewPosition


    def spawnConnectorPlate(self):
        #get positions for plates from connectorSections sorted in list
        cPPos = [conSec.getConnectorPlatePosition() for conSec in self.
connectorSections]
        cPPosSortedOrigins = rs.SortPoints([plane.Origin for plane in cPPos])
        cPPosSorted = []
        for point in cPPosSortedOrigins:
            cPPosSorted.append(next(plane for plane in cPPos if plane.Origin == point))
        #use them as parameter for ConnectorPlate-Creation
        self.connectorPlate = ConnectorPlate(cPPos = cPPosSorted, mesh = self.mesh,
edge = self)


    def findFaces(self):
        allFaces = self.mesh.getFaces()

        self.faces = [face for face in allFaces if face.hasEdge(self)]


    def spawnConnectorSections(self):
        #print ‘ids:’
        #print ‘’
        cSSpacing = [1,1,2,1,1]
        edgePoints = [self.mesh.getPoint(ID) for ID in self.pointIDs]
        #spawnPoint = midpoint(edgePoints[0], edgePoints[1])
        edgeVector = rs.VectorCreate(edgePoints[1], edgePoints[0])
        edgeLength = rs.VectorLength(edgeVector)
        edgeUVector = rs.VectorUnitize(edgeVector)
        parts = sum(cSSpacing)
        lengthPart = edgeLength/parts
        spawnPoints = [rs.PointAdd(edgePoints[0], rs.VectorScale(edgeUVector, lengthPart
* sum(cSSpacing[0:x]))) for x in range(1,len(cSSpacing))]



        faceNormals = [face.getNormal() for face in self.faces]
        spawnYAxis = rs.VectorReverse(midpoint(faceNormals[0], faceNormals[1]))
        spawnZAxis = rs.VectorCreate(edgePoints[0], edgePoints[1])
        positions = [rs.PlaneFromNormal(spawnPoint, spawnZAxis, rs.VectorCrossProduct(s
pawnZAxis,spawnYAxis)) for spawnPoint in spawnPoints ]


        self.connectorSections.extend(ConnectorSection(mesh = self.mesh, finalBasis =
position ,edge = self) for position in positions)


    def printStats(self):
        print ‘###EDGE’
        print ‘Angle:’
        print self.angle
        print ‘Adjacent Faces:’
        print [face.getPointIDs() for face in self.faces]

    #Postconditions:
        #RETURN Edge’s pointIDs
    def getPointIDs(self):
        return self.pointIDs
    #calcAngle calculates the Angle between the two faces of the parentMesh meeting in
the Edge
    #Postconditions:
        #RETURN is the angle in degrees
    def calcAngle(self):
        if len(self.faces) == 2:
            normal1 = self.faces[0].getNormal()
            normal2 = self.faces[1].getNormal()
            faceCenter1 = self.faces[0].getCenter()
            faceCenter2 = self.faces[1].getCenter()
            connection = rs.VectorCreate(faceCenter2, faceCenter1)
            dotN1C = rs.VectorDotProduct(normal1, connection)

            if dotN1C > 0:
                angle = rs.VectorAngle(normal1, normal2) +180
            elif dotN1C < 0:
                angle = 180-rs.VectorAngle(normal1, normal2)
            elif dotN1C == 0:
                angle = 180
            return angle
        else: return 0

    def calcShardEdgeOffset(self):
        if self.angle <= 180 : return 0
        else: return materialThickness/math.tan( math.radians( ( 360-self.angle ) / 2 )
)


    #Postconditions:
        #RETURNs the shardScrewPostion along the edge via distance to the shardEdge
    def calcShardScrewPosition(self):

        screwHoleDiameter = screwDiameter[shardScrewDimension] + shardScrewHoleClearance
        options = []

        ##option1
        ##connectorSectionTip to shardScrewSlot(Position)
        ###segment1
        segment1 = (shardNutSlotSpacing/2)/math.sin( math.radians( self.angle/2 ) )
        ###segment2
        segment2 = (shardScrewSlotLength + shardNutSlotWidth) / math.tan( math.radians( 
self.angle/2 ) )
        ###segment3
        segment3 = shardNutSlotLength/2
        ###sum up to get shardEdgeShardScrewSlotDistance
        options.append(segment1 + segment2 + segment3)

        #option1 = math.tan( radians( self.angle/2 ) ) * ( ( shardScrewSlotLength + 
shardNutSlotWidth ) + ( ( ( shardNutSlotSpacing / 2 ) + ( ( shardNutSlotWidth / 2 ) * 
(math.cos( radians( 90 - self.angle / 2 ) )) ) ) / ( math.sin( radians( 90 - self.angle 
/ 2 ) ) ) ) )


        ##option2
        ##shardEdge to shardScrewHole(Position)
        options.append(self.shardEdgeOffset + shardEgdeshardScrewHoleClearance + 0.5 * 
screwHoleDiameter)

        return max(options)

    #Postconditions:
        #RETURNs the connectionSectionTennonPosition along the edge via distance to the 
shardScrewPostion
    def calcConnectionSectionTenonPosition(self):
        options = []
        ##option1
        options.append(shardScrewHoleConnectorSectionMortiseClearance + 
shardScrewHoleDiameter/2 + connectorSectionMortiseLength/2)
        ##option2
        options.append(shardScrewSlotConnectorSectionTenonClearance + 
shardScrewSlotWidth/2 + connectorSectionTenonLength/2)


        return max(options)

    #Postconditions:
        #RETURN is the Edge’s angle
    def getAngle(self):
        return self.angle

    #check if PointID is part of Edge and Return the respective Truth-Value
    def hasPointID(self, pointID):
        if pointID in self.pointIDs: return True
        else: return False

    #check if face is part of Edge and Return the respective Truth-Value
    def hasFace(self, face):
        if face in self.faces: return True
        else: return False

    #calculates and returns Edge midpoint
    def calcMidpoint(self):
        return midpoint(self.mesh.getPoint(self.pointIDs[0]),self.mesh.getPoint(self.
pointIDs[1]))


    #return midpoint
    def getMidpoint(self):
        return self.midpoint

    def getShardEdgeOffset(self):
        return self.shardEdgeOffset

    #return associated ConnectorSection-Objects
    def getConnectorSections(self):
        return self.connectorSections

class Corner(object):

    def __init__(self, vertexID, mesh):
        self.vertexID = vertexID
        self.mesh = mesh
        self.edges = []
        self.shards = []
        self.faces = []
        self.findFaces()
        self.findEdges()

    #check if shard is
    def hasShard(self, shard):
        if self.vertexID == shard.getCornerID():
            self.shards.append(shard)
            return True
        else: return False

    def findFaces(self):
        self.faces = []
        self.faces.extend( face for face in self.mesh.getFaces() if face.
hasPointID(self.vertexID) )

    def findEdges(self):
        self.edges = []
        self.edges.extend( edge for edge in self.mesh.getEdges() if edge.
hasPointID(self.vertexID) )
```

```
    def getCornerID(self):
        return self.vertexID

    def getCornerPoint(self):
        return self.mesh.getPoint(self.vertexID)

class Face(object):

    #instantiation of Face-object
    #Preconditions:
        #facePointIDs... ordered list of point-indices of the face's points
        #parentMesh... Mesh-object that Face is a part of
    def __init__(self, facePointIDs, normal, center, parentMesh):
        self.facePointIDs = facePointIDs
        self.cleanFacePointIDs()
        self.normal = normal
        self.center = center
        self.mesh = parentMesh
        self.edges = []

    def cleanFacePointIDs(self):
        if self.facePointIDs[-1] == self.facePointIDs[-2]:
            self.facePointIDs = self.facePointIDs[0:-1]

    def printStats(self):
        print '###FACE'
        print 'Points:'
        print self.facePointIDs
        print 'Edges:'
        print [edge.getPointIDs() for edge in self.edges]
        print 'Normal:'
        print self.normal
        print ‘Center:’
        print self.center

    #check if Edge is part of Face
    #Preconditions:
        #edge... object of Type Edge
    #Postconditions:
        #if Edge is part of Face:
            #Face appends Edge to its self.edges and
            #RETURN True
        #if Edge is NOT part of Face:
            #RETURN False
    def hasEdge(self, edge):
        edgePointIDs = edge.getPointIDs()
        if all(self.hasPointID(x) for x in edgePointIDs):
            if not edge in self.edges: self.edges.append(edge)
            return True
        else: return False

    #Postconditions:
        #RETURN is a Point3d of the face’s center
    def getCenter(self):
        return self.center

    #Postconditions:
        #RETURN is a Vector3d of the face’s normal
    def getNormal(self):
        return self.normal

    #check if PointID is part of Face and Return the respective Truth-Value
    def hasPointID(self, pointID):
        if pointID in self.facePointIDs: return True
        else: return False
    #Postconditions:
        #RETURN is a list of the face’s vertex-indices
    def getPointIDs(self):
        return self.facePointIDs
```

```
import rhinoscriptsyntax as rs
import Rhino
import math
import copy
from globalVariables import *
from miscFunctions import *
#from GeometryClasses import *
from abc import ABCMeta, abstractmethod
from io import *
from CncRep import *

class Contour(object):
    """Class for closed Contour-Objects
    """
    __metaclass__ = ABCMeta
    defBasis_default = rs.WorldXYPlane()    #default plane of geometry creation
    registry = []                           #registry for contours that have to be 
registered
    materialThickness = materialThickness

    def __init__(self, mesh, finalBasis, defBasis = defBasis_default):
        """Creates instance of Contour

        Parameters:
            *mesh:
                parent mesh of Contour-Object
                type: GeometryClasses.Mesh Object
            *defPlane:
                basis of the Contour's geometry creation
                type: Rhino.Geometry.Plane
        """

        self.mesh = mesh                        #Contour's parent mesh
        self.defBasis = defBasis                #basis of the Contour's geometry
creation
        self.pointlist = []                     #ordered list of points and Feature-
Objects
        self.registered = self.isRegistered()   #boolean value in this field determines
if Object is going to be registered in Contour-Class-registry
        self.__register()
        self.finalBasis = finalBasis
        self.floaters = False
        self.extrusionDirection = False
        self.create()


    def __register(self):
        """Add Object to Contour-Class-Registry depending if self.registered

        Preconditions:
            self has to have field self.registered

        """
        if self.registered: Contour.registry.append(self)

    def addSubCncRep(self, subCncRep):
        self.cncRep.addSub(subCncRep)

    @staticmethod
    def isRegistered():
        return True

    def isCncSlave(self):
        return False

    def previewRegistry(self):
        for obj in self.registry: obj.drawPreview()

    def drawPreview(self, in3d = False): #, solid = False):
        """Draw Contour at actual position in mesh
        Parameters:
            *solid (not yet implemented): will determine if contour is extruded to a
solid
                type: boolean
        """
        return self.drawOnBasis(self.finalBasis, extruded = in3d)

    def cncContour(self):
        """
        Return:
            cnc-ready contour defined relative to WorldXY or supershape
            type: Rhino.GUID of Curve/Exploded Polyline/Group
        """

        self.drawOnBasis(rs.WorldXYPlane())

        return


    def passCncContour(self):
        return self.pointsOnBasis(self.cncBasis(), cnc = True)

    def cncBasis(self):
        try: superBasis = self.supershape.cncBasis()
        except:
            cncBasis = rs.WorldXYPlane()
        else:
            if self.getInsBasis() == self.getFinalBasis():
                cncBasis = self.changeBasisBasis(self.getInsBasis(), self.supershape.
getFinalBasis(), superBasis)
            else:
                cncBasis = self.changeBasisBasis(self.getInsBasis(), self.supershape.
getDefBasis(), superBasis)
        return cncBasis

    def extrudeContour(self, curveID, drawBasis):
        extrusionPath = rs.AddLine(drawBasis.Origin, rs.PointAdd(drawBasis.Origin,
self.extrusionDirection))
        extrusion = rs.ExtrudeCurve(curveID, extrusionPath)
        rs.DeleteObject(extrusionPath)
        rs.DeleteObject(curveID)
        return extrusion

    def drawOnBasis(self, basis, extruded = False ):
        """Draw Contour on basis
        Parameters:
            *basis:
                basis for construction of geometry
                type: Rhino.Geometry.Plane
        """
        tempPoints = self.pointsOnBasis(basis)
        #draw Polyline at actual position in space
        geoID = rs.AddPolyline(tempPoints)
        if extruded:
            self.extrusionDirection = rs.VectorScale(basis.ZAxis, materialThickness)
            extrusion = self.extrudeContour(geoID, basis)
            rs.CapPlanarHoles(extrusion)

        if not self.floaters == False:
            geoList = []
            geoList.append(geoID)
            for floater in self.floaters: geoList.append(floater.drawFloating(basis))
            if extruded:
                extrusionsToSubtract = []
                for floater in geoList[1:]: extrusionsToSubtract.append(self.
extrudeContour(floater, basis))
                for extr in extrusionsToSubtract: rs.CapPlanarHoles(extr)
                geoID = rs.BooleanDifference([extrusion], extrusionsToSubtract, True)
            else:
                groupID =  rs.AddGroup()
                rs.AddObjectsToGroup(geoList, groupID)
                return groupID

        return geoID

    def pointsOnBasis(self, basis, cnc = False):
        transMat = rs.XformChangeBasis(basis, self.defBasis)
        tempList = []
        for obj in self.pointlist:
            #try: addition = obj.pointsOnBasis( self.changeBasisBasis(obj.getInsBasis(),
self.defBasis, basis) )
            #except: tempList.append(rs.PointTransform(obj, transMat))
            #else: tempList.extend(addition)
            if isinstance(obj, Feature):
                if cnc:
                    if obj.isCncSlave(): addition = obj.passCncContour()
                    else: continue
                else: addition = obj.pointsOnBasis( self.changeBasisBasis(obj.
getInsBasis(), self.defBasis, basis) )
                tempList.extend(addition)
            else:
                tempList.append(rs.PointTransform(obj, transMat))
        return tempList

    def addFloat(self, obj):
        if not self.floaters:
            self.floaters = []
        self.floaters.append(obj)

    @abstractmethod
    def create(self):
        return

    def getFinalBasis(self):
        """Returns Objects finalBasis
        """
        return self.finalBasis

    def getDefBasis(self):
        return self.defBasis

    def getPointlist(self):
        return self.pointlist

    def planeAtParameter(self, parameter):
        #determine Point at parameter
        #determine direction at Parameter -> x-axis
        temp = self.contourPoint(parameter)
        origin = temp[0]
        #rs.AddPoint(origin)
        xAxis = temp[1]
        #determine y-axis
        yAxis = rs.VectorRotate(xAxis, 90, rs.WorldXYPlane().ZAxis)
        #create Plane
        plane = rs.PlaneFromFrame(origin, xAxis, yAxis)
        #return Plane
        return plane
        #calculate point and vector at parameter

    @staticmethod
    def changeBasisBasis(basis, basisBasisOld, basisBasisNew):
        transMat = rs.XformChangeBasis(basisBasisNew, basisBasisOld)
        return rs.PlaneTransform(basis, transMat)

    def contourPoint(self, parameter):
        """
        pointDistances = []

        lastPt = False
        for pt in self.pointlist:
            if lastPt:
                pointDistances.append(rs.VectorLength(rs.VectorCreate(pt, lastPt)))
            lastPt = pt
        """
        paramLength = self.lengthAtParam(parameter)

        ptsAroParam = self.pointsAroundParameter(parameter)

        unitVector = rs.VectorUnitize(rs.VectorCreate(ptsAroParam[1], ptsAroParam[0]))
        point = rs.VectorAdd(ptsAroParam[0], rs.VectorScale(unitVector, (paramLength -
ptsAroParam[3])))

        return (point, unitVector)

    def insertFeature(self, feature):
        #find out first index of insertion
        insIdx = self.pointsAroundParameter(feature.getInsParam())[2]+1
        #insert points into pointlist
        self.pointlist.insert(insIdx, feature)

    def pointsAroundParameter(self, param):
        #returns array of [point before parameter, point after, point before index in
self.pointlist)
        paramLength = self.lengthAtParam(param)
        cList = self.getCleanList()
        pointDistances = [ rs.Distance(x,y) for x,y in zip(cList, cList[1:] +
[cList[0]]) ]
        lengthAddUp = 0
        ptCount = 0
        while lengthAddUp + pointDistances[ptCount] <= paramLength:
            lengthAddUp += pointDistances[ptCount]
            if ptCount + 1 < len(pointDistances):
                ptCount +=1
            else: break

        pointBefore = cList[ ptCount ]
        if ptCount == len(cList)-1:
            pointAfter = cList[ 0 ]
        else:
            pointAfter = cList[ ptCount + 1 ]

        return [pointBefore, pointAfter, ptCount, lengthAddUp]

    def getCleanList(self):
        """returns a clean pointlist where features are replaced with the origins of
their bases
        """
        cList=[]
        for obj in self.pointlist:
            try: cList.append(obj.getInsBasis().Origin)
            except: cList.append(obj)
        return cList

    def unspooled(self):
        #length of contour unspooled

        cList = self.getCleanList   ()
        pL = rs.AddPolyline(cList)
        pLL = rs.CurveLength(pL)
        rs.DeleteObject(pL)
        return pLL

    #returns parameter for length

    def paramAtLength(self, length):
        unspoo = self.unspooled()
        return length/unspoo

    #returns length for parameter

    def lengthAtParam(self, param):
        return (self.unspooled()*param)

class Feature(Contour):
    __metaclass__ = ABCMeta

    def __init__(self, mesh, insParameter, supershape, finalBasis = False, defBasis = 
Contour.defBasis_default, insBasis = False):
        self.supershape = supershape
        self.cncRep = CncRep(self)
        self.insParameter = insParameter
        if insBasis == False : self.insBasis = self.calcInsBasis(self.supershape, self.
insParameter)
```

```
        else: self.insBasis = insBasis
        super(Feature, self).__init__(mesh = mesh, finalBasis = self.
calcFinalBasis(finalBasis), defBasis = defBasis )
        self.insertInSupershape()
        self.cncRep.configure()

    @staticmethod
    def isRegistered():
        return False

    def getSupershape(self):
        return self.supershape

    def calcInsBasis(self, supershape, insParameter):
        return supershape.planeAtParameter(insParameter)

    def calcFinalBasis(self, finalBasis):
        """Calculates final Basis-Plane of geometry
        Return:
            Final Basis-Plane
            type: Rhino.Geometry.Plane
        """
        try:
            if not finalBasis == False: return finalBasis
            else: raise Exception ("Final not defined")
        except:
            sdBasis = self.supershape.getDefBasis()
            iBasis = self.insBasis
            sBasis = self.supershape.getFinalBasis()
            finalBasis = self.changeBasisBasis(iBasis, sdBasis, sBasis)
            return finalBasis
    def insertInSupershape(self):
        self.supershape.insertFeature(self)

    def getInsParam(self):
        return self.insParameter

    def getInsBasis(self):
        return self.insBasis

class RectConnectedFeature(Feature):
    __metaclass__ = ABCMeta
    def __init__(self, mesh, insParameter, supershape, width, length, finalBasis =
False, defBasis = Contour.defBasis_default):
        #width and length are such that the length is the side parallel to the features
main axis
        try: self.width = width + self.widthClearance
        except: self.width = width
        try: self.length = length + self.lengthClearance
        except: self.length = length
        super(RectConnectedFeature, self).__init__(mesh = mesh, insParameter =
insParameter, supershape = supershape, finalBasis = finalBasis, defBasis = defBasis )
        #self, mesh, insParameter, supershape, finalBasis = False, defBasis = Contour.
defBasis_default


    def create(self):
        self.pointlist = []
        self.pointlist.append(createPoint(self.width*-0.5, 0, 0 ))
        self.pointlist.append(createPoint(self.width*-0.5, -1 * self.length, 0 ))
        self.pointlist.append(createPoint(self.width*0.5, -1 * self.length, 0 ))
        self.pointlist.append(createPoint(self.width*0.5, 0, 0 ))

class FloatingFeature(Feature):
    __metaclass__ = ABCMeta
    def __init__(self, mesh, supershape, insBasis, finalBasis , defBasis = Contour.
defBasis_default):
        super(FloatingFeature, self).__init__(mesh, insParameter = False, supershape =
supershape, finalBasis = finalBasis, defBasis = defBasis, insBasis = insBasis)



    def insertInSupershape(self):
        self.supershape.addFloat(self)

    @abstractmethod
    def drawFloating(self, customBasisBasis = False):
        return

class ConnectorSection(Contour):

    def __init__(self, mesh, finalBasis, edge):
        """
        Preconditions:
            finalBasis... the intended position of ConnectorSection
            edge... the ConnectorSection's parent-Edge
        Postconditions:
            ConnectorSection-Object is created
        """
        self.cncRep = CncRep(self)
        self.edge = edge
        self.gamma = self.edge.getAngle()
        self.shardEdgeShardScrewSlotClearance = shardEdgeShardScrewSlotClearance
        self.shardNutSlotSpacing = shardNutSlotSpacing
        self.shoulderHeight = connectorSectionShoulderHeight
        self.shoulderWidth = connectorSectionShoulderWidth
        self.height = self.calculateHeight()
        super(ConnectorSection, self).__init__(mesh = mesh, finalBasis = finalBasis)
        self.insertTenons()
        self.insertCPInterf()
        self.insertSScrewInterf()
        self.cncRep.configure()

    def configureCncRep(self):
        self.cncRep.addOutCut(self.passCncContour())
        self.createConnectorSectionLabel()

    def extrudeContour(self, curveID, drawBasis):
        transVec = rs.VectorScale(self.extrusionDirection,-0.5)
        movedBasis = rs.PointAdd(drawBasis.Origin, transVec)
        extrusionPath = rs.AddLine(movedBasis, rs.PointAdd(movedBasis, self.
extrusionDirection))
        rs.MoveObject(curveID, transVec)
        extrusion = rs.ExtrudeCurve(curveID, extrusionPath)
        rs.DeleteObject(extrusionPath)
        rs.DeleteObject(curveID)
        return extrusion

    def createConnectorSectionLabel(self):

        labelSize = 3
        #basis
        #origin:
        cL = self.originallist
        origin = rs.PointAdd( cL[3], (-2,2,0))
        #x axis
        xAxis = rs.VectorCreate(cL[2], cL[3])
        #y axis
        yAxis = rs.VectorCreate(cL[4], cL[3])
        basis = rs.PlaneFromFrame(origin, xAxis, yAxis)
        label = self.edge.getEdgeId()
        self.cncRep.addLabel(basis, label, labelSize)


    def getPosition(self):
        return self.finalBasis

    def insertCPInterf(self):
        """create and insert the parts interface with a ConnectorPlate
        :postcondition: ConnectorSection has an interface to its ConnectorPlate
        """
        #create CPKSlot
        cPKSlot = KeySlot(mesh = self.mesh, insParameter = 0.5, supershape = self, width
= connectorPlateKeyLength)
        #create ScrewSlot
        sSlot = ScrewSlot(mesh = self.mesh, insParameter = 0.5, supershape = cPKSlot,
screwDimension = connectorPlateScrewDimension, length = connectorPlateScrewSlotLength)
        #create NutSlot        self, mesh, insParameter, supershape, nutDimension
        NutSlot(mesh = self.mesh, insParameter = 0.5, supershape = sSlot, nutDimension
= connectorPlateNutDimension)


    def insertSScrewInterf(self):
        sSDis = self.edge.getShardScrewPosition() + self.edge.getShardEdgeOffset()

        #create ScrewSlot1
        sSParam1 = self.paramAtLength(sSDis)
        sSlot1 = ScrewSlot(mesh = self.mesh, insParameter = sSParam1, supershape = self,
screwDimension = shardScrewDimension, length = shardScrewSlotLength)
        #create NutSlot1
        NutSlot(mesh = self.mesh, insParameter = 0.5, supershape = sSlot1, nutDimension
= shardNutDimension)


        #create ScrewSlot2
        sSParam2 = self.paramAtLength(self.unspooled() - sSDis)
        sSlot2 = ScrewSlot(mesh = self.mesh, insParameter = sSParam2, supershape = self,
screwDimension = shardScrewDimension, length = shardScrewSlotLength)

        #create NutSlot1

        NutSlot(mesh = self.mesh, insParameter = 0.5, supershape = sSlot2, nutDimension
= shardNutDimension)


    def insertTenons(self):

        length1 = self.edge.getConnectorSectionTenonPosition() + self.edge.
getShardScrewPosition() + self.edge.getShardEdgeOffset()

        param1 = self.paramAtLength(length1)

        Tenon(mesh = self.mesh, insParameter = param1, supershape = self, width =
connectorSectionTenonLength, length = connectorSectionTenonHeight)

        length2 = self.unspooled() - length1

        param2 = self.paramAtLength(length2)

        Tenon(mesh = self.mesh, insParameter = param2, supershape = self, width =
connectorSectionTenonLength, length = connectorSectionTenonHeight)


    def getConnectorPlatePosition(self):

        #calc and return the relative Position of respective connectorPlate

        unitY = rs.VectorUnitize(self.finalBasis.YAxis)

        transVec = rs.VectorScale(unitY, -1 * self.height)

        cPPos = copy.copy(self.finalBasis)

        cPPos.Translate(transVec)

        #rotate PosPlane around xAxis -90deg

        cPPos = rs.RotatePlane(cPPos, -90, cPPos.XAxis)

        return cPPos


    def calculateHeight(self):

        if self.gamma <= 180: segment1 = ( math.cos(math.radians(self.gamma/2)) * (self.
edge.getShardScrewPosition() + shardNutSlotLength/2) ) + math.cos( math.radians( 90 -
self.gamma/2 ) ) * (shardNutSlotWidth + shardScrewSlotLength)

        else:
            segment1 = ( math.sin(math.radians( 90 - (self.gamma - 180) / 2 )) * (
shardScrewSlotLength + shardNutSlotWidth )  ) - ( ( math.sin( math.radians( ( self.
gamma-180 ) / 2 ) ) ) * ( self.edge.getShardScrewPosition() - shardNutSlotLength/2 ) )

        segment2 = shardNutSlotConnectorPlateNutSlotSpacing
        segment3 = connectorPlateNutSlotWidth + connectorPlateScrewSlotLength +
connectorPlateSlotHeight
        return segment1+segment2+segment3


    def create(self):


        point1Distance = self.edge.getShardEdgeOffset() + self.edge.
getShardScrewPosition() + self.edge.getConnectorSectionTenonPosition() +
connectorSectionTenonLength/2 + connectorSectionShoulderWidth

        point0 = createPoint(0,0,0)
        point1 = rs.Polar(point0, -90+self.gamma/2, point1Distance)
        point2 = rs.Polar(point1, -90+self.gamma/2-90, self.shoulderHeight)
        point3 = rs.Polar(point2, -90, self.height+(point2.Y))
        point4 = rs.PointAdd(point3, (-2*point3.X,0,0))
        point5 = rs.PointAdd(point2, (-2*point2.X,0,0))
        point6 = rs.PointAdd(point1, (-2*point1.X,0,0))


        self.originallist = [point0, point1, point2, point3, point4, point5, point6,
point0]
        self.pointlist = [point0, point1, point2, point3, point4, point5, point6,
point0]


class Shard(Contour):


    def __init__(self, mesh, cornerID, face):
        self.cncRep = CncRep(self)
        self.edgeLabelInfo = []
        tempFinalBasis = rs.PlaneFromNormal(face.getCenter(), face.getNormal())
        self.cornerID = cornerID
        self.face = face
        self.edges = []
        self.corner = False
        self.mesh = mesh
        self.findEdges()
        self.findCorner()
        super(Shard, self).__init__(mesh = mesh, finalBasis = tempFinalBasis, defBasis
= tempFinalBasis)
        self.cncRep.configure()

    def configureCncRep(self):
        self.cncRep.addOutCut(self.passCncContour())
        self.createShardEdgeLabel()

    def createShardEdgeLabel(self):
        labelSize = 3
        for edge,edgeUNormal in self.edgeLabelInfo:
            #create basis
            origin = midpoint(edge.getMidpoint(), self.corner.getCornerPoint() )
            normal = rs.VectorUnitize(self.face.getNormal())
            xAxis = edgeUNormal
            edgeDistance = edge.getShardEdgeOffset() + 25 + labelSize
            basis = rs.PlaneFromNormal(rs.VectorAdd(origin,rs.VectorScale(xAxis,
edgeDistance)), normal, xAxis)
            basis = rs.RotatePlane(basis, -90, normal)

            #find out if edge point ids are ordered in same direction as xAxis and
correct if necessary
            edgePointIDs = edge.getPointIDs()
            if rs.VectorAngle(basis.XAxis, edge.getEdgeVector()) > 90:
                edgePointIDs = (edgePointIDs[1],edgePointIDs[0])

            #changeBasis to cnc position
            basis = self.changeBasisBasis(basis, self.getFinalBasis(), self.cncBasis())
            self.cncRep.addShardLabel(basis, edgePointIDs, labelSize)

    #identify and store edges along shard
    def findEdges(self):
        for edge in self.mesh.getEdges():
            if all([edge.hasPointID(self.cornerID), edge.hasFace(self.face)]): #possible
performance enhancement by first making list of edges with cornerID
                self.edges.append(edge)

    #find the Corner the shard is at
    def findCorner(self):
        for corner in self.mesh.getCorners():
            if corner.hasShard(self):
                self.corner = corner
                return

    def createDetails(self):

        cornerPt = self.mesh.getPoint(self.corner.getCornerID())

        for edge in self.edges:
            #edgeVector
            eVec = edge.getEdgeVector()
            ePt = edge.getMidpoint()
            halfEdge = rs.Distance(cornerPt, ePt)
            #edgeNormal in Face-Plane
            pointOnFace = self.face.getCenter()
            normalPlane = rs.PlaneFromNormal(pointOnFace, eVec)
            pointOnEdge = rs.PlaneClosestPoint(normalPlane, ePt)
            normalVec = rs.VectorCreate(pointOnFace, pointOnEdge)
            normalVecU = rs.VectorUnitize(normalVec)
            self.edgeLabelInfo.append((edge,normalVecU))
            #centerPlane for later detail-positions
            #connectorSection-Positions
            conSecPos = [conSec.getPosition().Origin for conSec in edge.
getConnectorSections() if rs.Distance(conSec.getPosition().Origin, cornerPt) < halfEdge]
            #shardScrewHole-Positions
            sSHDist = edge.getShardEdgeOffset() + edge.getShardScrewPosition()
            normalVecSSH = rs.VectorScale(normalVecU, sSHDist)
            sSHPos = [ rs.PlaneFromFrame( rs.PointAdd( pos, normalVecSSH ), eVec,
normalVec ) for pos in conSecPos ]
            #createScrewHoles for all sSHPos
            for pos in sSHPos: ScrewHole(mesh = self.mesh, supershape = self,
screwDimension = shardScrewDimension, insBasis = pos, finalBasis = pos)
            #conn.Sec.Mortise-Positions
            cSMDist = sSHDist + edge.getConnectorSectionTenonPosition()
            normalVecSSH = rs.VectorScale(normalVecU, cSMDist)
            cSMPos = [ rs.PlaneFromFrame( rs.PointAdd( pos, normalVecSSH ), eVec,
normalVec ) for pos in conSecPos ]
            #create conn.Sec.Mortises for all cSMPos
            for pos in cSMPos: Mortise( mesh = self.mesh, supershape = self, tenonLength
= connectorSectionTenonLength, insBasis = pos, finalBasis = pos)


    def create(self):
        """creates Points for contour
        :Postcondition: the corner-point has to be at index 0 of pointlist
        """
        #cornerPoint
        cornerPoint = self.mesh.getPoint(self.cornerID)
        #edge1 midpoint
        e1Mid = self.edges[0].getMidpoint()
        #face-Center
        faceCenter = self.face.getCenter()
        #edge2 midpoint
        e2Mid = self.edges[1].getMidpoint()
        self.pointlist = [cornerPoint, e1Mid, faceCenter, e2Mid]
        self.createDetails()
        self.edgeOffsets()

    def edgeOffsets(self):
        for edge in self.edges: self.offsetEdge(edge)
        self.pointlist.append(self.pointlist[0])
        return

    def offsetEdge(self, edge):
        """Offsets the shard-edge parallel to the passed Edge-Object
        :Postcondition: self.pointlist is changed
        """
        #check if edge has to be offset
        if not edge.getShardEdgeOffset() == 0 :
            #Vector along Edge
            eMid = edge.getMidpoint()
            cPoint = self.mesh.getPoint(self.cornerID)
            eVec = rs.VectorCreate(cPoint, eMid)
            #Vector to FCenter
            fCenter = self.face.getCenter()
            fVec = rs.VectorCreate(fCenter, eMid)
            #Vector other Edge
            oEdge = self.edges[0]
            if oEdge == edge: oEdge = self.edges[1]
            oMid = oEdge.getMidpoint()
            oVec = rs.VectorCreate(oMid, cPoint)
            #Offset Distance
            offDist = edge.getShardEdgeOffset()
            #Angle1 for Calc
            angle1 = rs.VectorAngle(eVec, fVec)
            if angle1 > 90: angle1 = 180 - angle1
            #translation distance 1 (for ePoint)
            transDist1 = offDist / math.sin(math.radians(angle1))
            #TransVector1 (for ePoint)
            transVec1 = rs.VectorScale(rs.VectorUnitize(fVec), transDist1)
            #Angle2 for Calc
            angle2 = rs.VectorAngle(eVec, oVec)
            if angle2 > 90: angle2 = 180 - angle2
            #translation distance 2 (for ePoint)
            transDist2 = offDist / math.sin(math.radians(angle2))
            #TransVector2 (for cPoint)
            transVec2 = rs.VectorScale(rs.VectorUnitize(oVec), transDist2)
            #translate Points:
            if eMid in self.pointlist:
                idx1 = self.pointlist.index(eMid)
                self.pointlist[idx1] = eMid.Add(eMid, transVec1)
                idx2 = 0
                self.pointlist[idx2] = self.pointlist[idx2].Add(self.pointlist[idx2],
transVec2)
        return

    #get the shards cornerID

    def getCornerID(self):
        return self.cornerID



class ConnectorPlate(Contour):

    def __init__(self, cPPos, mesh, edge):
        self.edge = edge
        self.cncRep = CncRep(self)
        self.cPPos = cPPos
        self.correctCPPosDirection()
        self.width = connectorPlateWidth
        self.connectorPlateEndCapLength = connectorPlateEndCapLength
        self.connectorPlateKeyWidth = connectorPlateKeyWidth
        self.connectorPlateKeyLength = connectorPlateKeyLength
        self.baseLength = rs.Distance(self.cPPos[0].Origin, self.cPPos[-1].Origin)
        super(ConnectorPlate, self).__init__(mesh = mesh, finalBasis = self.cPPos[0])
        for pos in cPPos:
            ScrewHole(mesh = self.mesh, supershape = self, screwDimension = 
connectorPlateScrewDimension, insBasis = copy.copy(pos), finalBasis = copy.copy(pos))
        self.forgeThyKeys()
        self.cncRep.configure()



    def configureCncRep(self):
        self.cncRep.addOutCut(self.passCncContour())
        self.createConnectorPlateLabel()



    def createConnectorPlateLabel(self):

        labelSize = 3
        #basis
        #origin:
        vec1 = rs.VectorCreate(self.cPPos[2].Origin, self.cPPos[1].Origin)
        #y axis
        xAxis = self.cPPos[1].YAxis
        yAxis = -1 * self.cPPos[1].XAxis
        origin = rs.PointAdd( rs.PointAdd(self.cPPos[1].Origin, rs.VectorScale(vec1, 
0.5)), rs.VectorScale(rs.VectorUnitize(yAxis), labelSize/-2))
        basisF = rs.PlaneFromFrame(origin, xAxis, yAxis)
        basis = self.changeBasisBasis(basisF, self.finalBasis, self.cncBasis())
        label = self.edge.getEdgeId()
        self.cncRep.addLabel(basis, label, labelSize, center = True)
    def correctCPPosDirection(self):
        vector1 = rs.VectorCreate(self.cPPos[-1].Origin,self.cPPos[0].Origin)
        vector2 = self.cPPos[0].YAxis #change
        if rs.VectorAngle(vector1, vector2) < 1: self.cPPos = self.cPPos[::-1]

    def forgeThyKeys(self):
        #make generic key pointlist1
        gKPL1 = [
                  createPoint( self.width / 2,                        self.
connectorPlateKeyWidth / 2, 0),
                  createPoint( self.connectorPlateKeyLength / 2 ,    self.
connectorPlateKeyWidth / 2, 0),
                  createPoint( self.connectorPlateKeyLength / 2,     self.
connectorPlateKeyWidth / -2, 0),
                  createPoint( self.width / 2,                        self.
connectorPlateKeyWidth / -2, 0)
                  ]
        gKDP1 = [
                  createPoint(
                              self.connectorPlateKeyLength / 2  + math.cos( math.radians(
45 ) ) * drillDiameter/2,
                              self.connectorPlateKeyWidth / 2 - math.cos( math.radians(
45 ) ) * drillDiameter/2,
                              0
                              ),
                  createPoint(
                              self.connectorPlateKeyLength / 2  + math.cos( math.radians(
45 ) ) * drillDiameter/2,
                              self.connectorPlateKeyWidth / -2 + math.cos( math.radians(
45 ) ) * drillDiameter/2,
                              0
```

```
                                )
                        ]
        #make generic key pointlist2
        gKPL2 = [
                        createPoint( self.width / -2,                       self.
connectorPlateKeyWidth / -2, 0),

                        createPoint( self.connectorPlateKeyLength / -2,     self.
connectorPlateKeyWidth / -2, 0),

                        createPoint( self.connectorPlateKeyLength / -2 ,    self.
connectorPlateKeyWidth / 2, 0),

                        createPoint( self.width / -2,                       self.
connectorPlateKeyWidth / 2, 0)

                        ]
        gKDP2 = [
                        createPoint(  self.connectorPlateKeyLength / -2  - math.cos( math.
radians( 45 ) ) * drillDiameter/2, self.connectorPlateKeyWidth / -2 + math.cos( math.
radians( 45 ) ) * drillDiameter/2, 0 ),

                        createPoint(  self.connectorPlateKeyLength / -2  - math.cos( math.
radians( 45 ) ) * drillDiameter/2, self.connectorPlateKeyWidth / 2 - math.cos( math.
radians( 45 ) ) * drillDiameter/2, 0 )

                        ]


        for pos, index in zip(self.cPPos, range(len(self.cPPos))):
            transVecY = -1 * rs.Distance(pos.Origin, self.cPPos[0].Origin)
            #print "##########################################"
            #print index
            #print transVecY
            transVec = createPoint(0,transVecY+0.5,0)

            #make pointlist1 for key
            kPL1 = translatePoints(gKPL1, transVec)
            kDP1 = translatePoints(gKDP1, transVec)
            #calc insertIdx
            insertIdx1 = 1 + index * 4
            #insert pointlist1
            insertList(self.pointlist, kPL1, insertIdx1)

            #make pointlist2 for key
            kPL2 = translatePoints(gKPL2, transVec)
            kDP2 = translatePoints(gKDP2, transVec)
            #calc insertIdx
            insertIdx2 = -1 - index * 4
            #insert pointlist2
            insertList(self.pointlist, kPL2, insertIdx2)
            self.cncRep.addDrillPt(kDP1+kDP2,True)

        self.pointlist.append(self.pointlist[0])

    def create(self):
        self.pointlist = []
        self.pointlist.append( createPoint( 1 * self.width /
2,         1 * self.connectorPlateKeyWidth / 2 + self.connectorPlateEndCapLength,
0 ))

        self.pointlist.append( createPoint( 1 * self.width / 2,         -1 * ( self.
baseLength + self.connectorPlateKeyWidth / 2 + self.connectorPlateEndCapLength ),
0 ))

        self.pointlist.append( createPoint( -1 * (self.width / 2),      -1 * ( self.
baseLength + self.connectorPlateKeyWidth / 2 + self.connectorPlateEndCapLength ),
0 ))

        self.pointlist.append( createPoint( -1 * (self.width /
2),      1 * self.connectorPlateKeyWidth / 2 + self.connectorPlateEndCapLength,
0 ))



class Tenon(Feature):
    def __init__(self, mesh, insParameter, supershape, width, length, finalBasis = 
False, defBasis = Contour.defBasis_default):

        #width and length are such that the length is the side parallel to the features 
main axis

        self.width = width
        self.length = length
        super(Tenon, self).__init__(mesh = mesh, insParameter = insParameter, supershape 
= supershape, finalBasis = finalBasis, defBasis = defBasis )

        #self, mesh, insParameter, supershape, finalBasis = False, defBasis = Contour.
defBasis_defaults



    def configureCncRep(self):
        cncTransMat = rs.XformChangeBasis(self.cncBasis(), self.defBasis)
        self.cncRep.addDrillPt(rs.PointTransform(createPoint(  -1 * (self.width * 0.5 
+ math.cos( math.radians( 45 ) ) * drillDiameter/2), math.cos( math.radians( 45 ) ) * 
drillDiameter/2, 0 ), cncTransMat))

        self.cncRep.addDrillPt(rs.PointTransform(createPoint(   1 * (self.width * 0.5 
+ math.cos( math.radians( 45 ) ) * drillDiameter/2), math.cos( math.radians( 45 ) ) * 
drillDiameter/2, 0 ), cncTransMat))



    def isCncSlave(self):
        return True

    def create(self):
        self.pointlist = []
        self.pointlist.append(createPoint(self.width*-0.5, 0, 0 ))
        self.pointlist.append(createPoint(self.width*-0.5, 1 * self.length, 0 ))
        self.pointlist.append(createPoint(self.width*0.5, 1 * self.length, 0 ))
        self.pointlist.append(createPoint(self.width*0.5, 0, 0 ))

class KeySlot(RectConnectedFeature):
    widthClearance = 0.1
    lengthClearance = 0.1
    lengthCncOverlap = 1
    materialThickness = materialThickness

    def __init__(self, mesh, insParameter, supershape, width):
        super(KeySlot, self).__init__(mesh = mesh, insParameter = insParameter,
supershape = supershape, width = width + self.widthClearance, length = self.
materialThickness + self.lengthClearance)


    def configureCncRep(self):
        cncTransMat = rs.XformChangeBasis(self.cncBasis(), self.defBasis)
        cncCut = self.cncContour()
        self.cncRep.addInCut(cncCut)
        self.cncRep.addDrillPt(
                               rs.PointTransform(
                                                 createPoint(
                                                             -0.5 * self.width + math.
cos( math.radians( 45 ) )  * drillDiameter/2,
                                                             -1 * self.length  + math.
cos( math.radians( 45 ) ) * drillDiameter/2,
                                                             0
                                                             ) ,
                                                 cncTransMat
                                                 )
                               )
        self.cncRep.addDrillPt(
                               rs.PointTransform(
                                                  createPoint(
                                                              self.width*0.5  - math.
cos( math.radians( 45 ) ) * drillDiameter/2,
                                                              -1 * self.length  + math.
cos( math.radians( 45 ) ) * drillDiameter/2,
                                                                                       0
                                                                                       ) ,
                                                                           cncTransMat
                                                                           )
                                                         )

    def cncContour(self):
        cncTransMat = rs.XformChangeBasis(self.cncBasis(), self.defBasis)
        cncContour = []
        #one-sided cnc length overlap
        cncContour.append(createPoint(self.width*-0.5, self.lengthCncOverlap, 0 ))
        cncContour.append(createPoint(self.width*-0.5, -1 * self.length, 0 ))
        cncContour.append(createPoint(self.width*0.5, -1 *  self.length, 0 ))
        cncContour.append(createPoint(self.width*0.5, self.lengthCncOverlap, 0 ))
        cncContour.append(cncContour[0])
        return rs.PointArrayTransform(cncContour, cncTransMat)

class ScrewSlot(RectConnectedFeature):
    widthClearance = 0.1
    lengthCncOverlap = 1

    def __init__(self, mesh, insParameter, supershape, screwDimension, length):
        self.screwDimension = screwDimension
        width = self.widthClearance + screwDiameter[self.screwDimension]
        super(ScrewSlot, self).__init__(mesh = mesh, insParameter = insParameter,
supershape = supershape, width = width, length = length)


    def configureCncRep(self):
        cncCut = self.cncContour()
        self.cncRep.addInCut(cncCut)

    def cncContour(self):
        cncTransMat = rs.XformChangeBasis(self.cncBasis(), self.defBasis)
        cncContour = []
        #one-sided cnc length overlap
        cncContour.append(createPoint(self.width*-0.5, self.lengthCncOverlap, 0 ))
        cncContour.append(createPoint(self.width*-0.5, -1 * self.length - self.
lengthCncOverlap, 0 ))
        cncContour.append(createPoint(self.width*0.5, -1 *  self.length - self.
lengthCncOverlap, 0 ))
        cncContour.append(createPoint(self.width*0.5, self.lengthCncOverlap, 0 ))
        cncContour.append(cncContour[0])
        return rs.PointArrayTransform(cncContour, cncTransMat)

class NutSlot(RectConnectedFeature):
    widthClearance = 0
    lengthClearance = 0

    def __init__(self, mesh, insParameter, supershape, nutDimension):
        self.nutDimension = nutDimension
        width = self.widthClearance + nutWidth[self.nutDimension]
        length = self.lengthClearance + nutThickness[self.nutDimension]
        super(NutSlot, self).__init__(mesh = mesh, insParameter = insParameter,
supershape = supershape, width = width, length = length)


    def configureCncRep(self):
        cncCut = self.cncContour()
        self.cncRep.addInCut(cncCut)

    def cncContour(self):
        cncTransMat = rs.XformChangeBasis(self.cncBasis(), self.defBasis)
        return rs.PointArrayTransform(self.pointlist+[self.pointlist[0]], cncTransMat)

class ScrewHole(FloatingFeature):
    screwClearance = shardScrewHoleClearance
    def __init__(self, mesh, supershape, screwDimension, insBasis, finalBasis):
        self.diameter = self.screwClearance + screwDiameter[screwDimension]
        super(ScrewHole, self).__init__(mesh = mesh, supershape = supershape, insBasis
= finalBasis, finalBasis = finalBasis, defBasis = 'irrelevant')


    def create(self):
        return

    def configureCncRep(self):
        self.cncRep.addBoreCircle(self.cncBasis(), self.diameter/2)



    def drawFloating(self, customBasisBasis = False):

        if not customBasisBasis:
            geoID = rs.AddCircle(self.finalBasis, self.diameter/2)
        else:
            if self.finalBasis == self.insBasis:
                newPlane = self.changeBasisBasis(self.insBasis, self.supershape.
getFinalBasis(), customBasisBasis )
                geoID = rs.AddCircle(newPlane, self.diameter/2)
            else:
                newPlane = self.changeBasisBasis(self.insBasis, self.supershape.
getDefBasis(), customBasisBasis )
                geoID = rs.AddCircle(newPlane, self.diameter/2)
        return geoID

class Mortise(FloatingFeature):
    widthClearance = 0.1
    lengthClearance = 0.1

    def __init__(self, mesh, supershape, tenonLength, insBasis, finalBasis = False,
defBasis = Contour.defBasis_default):
        self.length = tenonLength + self.lengthClearance
        self.width = materialThickness + self.widthClearance
        super(Mortise, self).__init__(mesh = mesh, supershape = supershape, insBasis =
insBasis, finalBasis = finalBasis, defBasis = defBasis)



    def drawFloating(self, customBasisBasis = False):

        if not customBasisBasis: geoID = self.drawOnBasis(self.finalBasis)
        else:
            if self.finalBasis == self.insBasis:
                geoID = self.drawOnBasis(self.changeBasisBasis(self.insBasis, self.
supershape.getFinalBasis(), customBasisBasis))
            else:
                geoID = self.drawOnBasis(self.changeBasisBasis(self.insBasis, self.
supershape.getDefBasis(), customBasisBasis))
        return geoID

    def create(self):
        self.pointlist = []
        self.pointlist.append(createPoint(self.width*0.5, self.length * 0.5, 0 ))
        self.pointlist.append(createPoint(self.width*0.5, self.length * -0.5, 0 ))
        self.pointlist.append(createPoint(self.width*-0.5, self.length * -0.5, 0 ))
        self.pointlist.append(createPoint(self.width*-0.5, self.length * 0.5, 0 ))
        self.pointlist.append(self.pointlist[0])



    def configureCncRep(self):
        cncCut = self.cncContour()
        self.cncRep.addInCut(cncCut)
        cncTransMat = rs.XformChangeBasis(self.cncBasis(), self.defBasis)
        self.cncRep.addDrillPt( rs.PointTransform( createPoint(
                                                              self.width * 0.5 - math.
cos( math.radians( 45 ) ),
                                                              self.length * 0.5 - math.
cos( math.radians( 45 ) ),
                                                              0
                                                              ) , cncTransMat ))
        self.cncRep.addDrillPt( rs.PointTransform( createPoint(
                                                              self.width * 0.5 - math.
cos( math.radians( 45 ) ),
                                                              self.length * -0.5 + math.
cos( math.radians( 45 ) ),
                                                              0
                                                              ) , cncTransMat ))
        self.cncRep.addDrillPt( rs.PointTransform( createPoint(
                                                              self.width * -0.5 + math.
cos( math.radians( 45 ) ),
                                                              self.length * -0.5 + math.
cos( math.radians( 45 ) ),
                                                              0
                                                              ) , cncTransMat ))
        self.cncRep.addDrillPt( rs.PointTransform( createPoint(
                                                              self.width * -0.5 + math.
cos( math.radians( 45 ) ),
                                                              self.length * 0.5 - math.
cos( math.radians( 45 ) ),
                                                              0
                                                              ) , cncTransMat ))

    def cncContour(self):
        cncTransMat = rs.XformChangeBasis(self.cncBasis(), self.defBasis)
        return rs.PointArrayTransform(self.pointlist, cncTransMat)
```

```
import Rhino
import math
import copy
from globalVariables import *
from Contour import *


#Insert a list into another list element by element, optionally at an index
def insertList(targetList, sourceList, index):
    i = index
    for x in sourceList:
        targetList.insert(i, x)
        if not i<0: i += 1
    return targetList

#create a Rhino.Geometry.Point3d from 3 Coordiantes
def createPoint(x,y,z):
    point = Rhino.Geometry.Point3d(x,y,z)
    return point

def translatePoints(points, vector):
    points = [point.Add(point, vector) for point in points]
    return points

#Map any geometry from a Source-Reference-Plane to a Target-Reference-Plane
#Postconditions:
    #RETURN is mapped geometry
def mapToPlane(sourcePlane, targetPlane, objects):

    if sourcePlane == targetPlane:
        return objects
    #drawPlane(targetPlane)

    sourceOrigin = sourcePlane.Origin
    targetOrigin = targetPlane.Origin
    translationVector = targetOrigin - sourceOrigin
    #move

    ########
    #drawPlane(sourcePlane, red)
    #rs.ObjectColor(rs.CopyObjects(objects), red)
    ########

    movedObjects = rs.MoveObjects(objects, translationVector)
    movedSourcePlane = sourcePlane
    movedSourcePlane.Translate(translationVector)

    ########
    #drawPlane(movedSourcePlane, blue)
    #rs.ObjectColor(rs.CopyObjects(movedObjects), blue)
    ########

    #rotate1
    xAxisS = movedSourcePlane.XAxis

    xAxisT = targetPlane.XAxis

    anglePlane1 = rs.PlaneFromNormal(targetOrigin, rs.VectorCrossProduct(xAxisS,
xAxisT))

    #angle1 = 2*rs.Angle(xAxisT, xAxisS, anglePlane1)[0]
    angle1 = rs.VectorAngle(xAxisT, xAxisS)
    #rotationAxis1 = rs.VectorCrossProduct(xAxisS, xAxisT)
    rotationAxis1 = rs.VectorCrossProduct(xAxisS, xAxisT)

    rotated1Objects = rs.RotateObjects(movedObjects, movedSourcePlane.Origin, angle1,
rotationAxis1, False)

    rotated1SourcePlane = movedSourcePlane
    rotated1SourcePlane.Rotate(math.radians(angle1), rotationAxis1, movedSourcePlane.
Origin)

    ########
    #rs.ObjectColor(rs.CopyObjects(rotated1Objects), green)
    #drawPlane(rotated1SourcePlane, green)
    ########

    #rotate2
    anglePlane2 = rs.PlaneFromNormal(targetOrigin, targetPlane.XAxis)
    #angle2 = 2*rs.Angle(targetPlane.YAxis, rotated1SourcePlane.YAxis,  anglePlane2)[0]
    angle2 = rs.VectorAngle(targetPlane.YAxis, rotated1SourcePlane.YAxis)

    #rotationAxis2 = targetPlane.XAxis
    rotationAxis2 = rs.VectorCrossProduct(rotated1SourcePlane.YAxis, targetPlane.YAxis)

    rotated2Objects = rs.RotateObjects(rotated1Objects, rotated1SourcePlane.Origin,
180+angle2, rotationAxis2, False)
    rotated2SourcePlane = rotated1SourcePlane
    rotated2SourcePlane.Rotate(math.radians(angle2), rotationAxis2, rotated1SourcePlane.
Origin)

    ########
    #drawPlane(rotated2SourcePlane,pink)
    ########
    return rotated2Objects

#Draw x, y, z axes of a plane for representation
def drawPlane(plane, color = (0,0,0)):
    geo = []
    #print plane
    geo.append(rs.AddPoint(plane.Origin))
    geo.append(rs.AddLine(plane.Origin, plane.Origin+plane.XAxis))
    geo.append(rs.AddLine(plane.Origin, plane.Origin+plane.YAxis*2))
    geo.append(rs.AddLine(plane.Origin, plane.Origin+plane.ZAxis*3))
    for obj in geo: rs.ObjectColor(obj, color)
    rs.AddObjectsToGroup(geo, rs.AddGroup())

#RETURNs midpoint of two points
def midpoint(point1, point2):
    point = rs.PointAdd(point1, point2)
    return rs.PointDivide(point, 2)

def mult(matrix1,matrix2):
    # Matrix multiplication
    if len(matrix1[0]) != len(matrix2):
        # Check matrix dimensions
        print ‘Matrices must be m*n and n*p to multiply!’
        return
    else:
        # Multiply if correct dimensions
        new_matrix = zero(len(matrix1),len(matrix2[0]))
        for i in range(len(matrix1)):
            for j in range(len(matrix2[0])):
                for k in range(len(matrix2)):
                    new_matrix[i][j] += matrix1[i][k]*matrix2[k][j]

        return new_matrix
```

```
import rhinoscriptsyntax as rs
import Rhino
import math
import copy
from globalVariables import *
from miscFunctions import *
#from GeometryClasses import *


class CncRep(object):
    #the geometry in a CncRep has to be such that its basis is worldXY or to its
supershape based on worldXY (at standardPosition)
    instances = []

    inCutColor = red
    outCutColor = blue
    drillPtCOlor = green
    boreCircleColor = yellw
    labelColor = pink

    def __init__(self, contourObject):
        CncRep.instances.append(self)
        self.cObj = contourObject
        self.outCuts = [] #list of ordered lists of points
        self.inCuts = [] #list of ordered lists of points
        self.boreCircles = [] #list of tuples such that (center, radius)
        self.drillPts = [] #list of points for 2mm drilling
        self.labels = []
        self.slaveRep = False
        self.circle = False
    def draw(self):
        if self.slaveRep: return
        idList = []
        for ptLst in self.outCuts:
            idList.append(rs.AddPolyline(ptLst))
            rs.ObjectColor(idList[-1], CncRep.outCutColor)
        for ptLst in self.inCuts:
            idList.append(rs.AddPolyline(ptLst))
            rs.ObjectColor(idList[-1], CncRep.inCutColor)
        for circleDef in self.boreCircles:
            idList.append(rs.AddCircle(circleDef[0].Origin, circleDef[1]))
            rs.ObjectColor(idList[-1], CncRep.boreCircleColor)
        for pt in self.drillPts:
            idList.append(rs.AddPoint(pt))
            rs.ObjectColor(idList[-1], CncRep.drillPtCOlor)
        “””original label version
        for ptLst in self.labels:
            idList.append(rs.AddPolyline(ptLst))
            rs.ObjectColor(idList[-1], CncRep.labelColor)
        “””
        for ptLst in self.labels:
            for point in ptLst[:-2]:
                idList.append(rs.AddPoint(point))
                rs.ObjectColor(idList[-1], CncRep.labelColor)
            print ‘ptLst.........................................’
            print ptLst
            idList.append(rs.AddPolyline(ptLst[-2:]))
            rs.ObjectColor(idList[-1], CncRep.labelColor)


        rs.AddObjectsToGroup(idList, rs.AddGroup())

    def addSub(self,subCncRep):
        if self.slaveRep:
            self.cObj.getSupershape().addSubCncRep(subCncRep)
        else:
            self.addDrillPt(subCncRep.getDrillPts(), True)
            self.addOutCut(subCncRep.getOutCuts(), True)
            self.addInCut(subCncRep.getInCuts(), True)
            for x in subCncRep.getBoreCircles() :
                self.addBoreCircle(x[0],x[1])
            #except : print self.getBoreCircles()

    def configure(self, now = False):
        self.cObj.configureCncRep()
        try:
            supershape = self.cObj.getSupershape()
        except: self.slaveRep = False
        else:
            supershape.addSubCncRep(self)
            self.slaveRep = True

    def addDrillPt(self, pt, listInput = False):
        if listInput: self.drillPts.extend(pt)
        else: self.drillPts.append(pt)

    def getDrillPts(self):
        return self.drillPts

    def addOutCut(self, ptList, listInput = False):
        if listInput: self.outCuts.extend(ptList)
        else: self.outCuts.append(ptList)

    def getOutCuts(self):
        return self.outCuts
```

```
def addInCut(self, ptList, listInput = False):
    if listInput: self.inCuts.extend(ptList)
    else: self.inCuts.append(ptList)

def getInCuts(self):
    return self.inCuts

def addBoreCircle(self, pt, radius, listInput = False):
    if listInput: self.boreCircles.extend(zip(pt, radius))
    self.boreCircles.append([pt, radius])
    self.circle = True

def getBoreCircles(self):
    return self.boreCircles

def addLabel(self, basis, label, labelSize, center = False):
    romans = self.__convertArabToRomanPolyline(label, labelSize)
    if center:
        #move Basis
        offset = romans[1]/2
        vector = rs.VectorScale(rs.VectorUnitize(basis.XAxis) , -1 * offset)
        transMat = rs.XformTranslation(vector)
        basis = rs.PlaneTransform(basis, transMat)
    transMat = rs.XformChangeBasis(basis, rs.WorldXYPlane())
    newLabels = []
    for poly in romans[0]:
        if poly == []: continue
        newLabels.append(rs.PointArrayTransform( poly , transMat ))

    self.labels.extend(newLabels)

def addShardLabel(self, basis, edgePointIDs, labelSize):
    spacing = 10
```

```
        #create edgePointRomans:
        romans = []

        for pId in edgePointIDs:
            romans.append(self.__convertArabToRomanPolyline(pId, labelSize))

        newLabels = []
        basis1 = rs.PlaneFromFrame(rs.PointAdd(basis.Origin, rs.VectorScale(basis.XAxis, 
-1*(spacing/2 + romans[0][1] ))), basis.XAxis, basis.YAxis)
        transMat1 = rs.XformChangeBasis(basis1, rs.WorldXYPlane())
        for poly in romans[0][0]:
            newLabels.append(rs.PointArrayTransform(poly, transMat1))

        basis2 = rs.PlaneFromFrame(rs.PointAdd(basis.Origin, rs.VectorScale(basis.
XAxis,spacing/2) ), basis.XAxis, basis.YAxis)
        transMat2 = rs.XformChangeBasis(basis2, rs.WorldXYPlane())
        for poly in romans[1][0]:
            newLabels.append(rs.PointArrayTransform(poly, transMat2))

        self.labels.extend(newLabels)



    def __convertArabToRomanPolyline (self, label, height):

        roman = “”
        steps = [(1000,’M’), (900, ‘CM’), (500, ‘D’), (400, ‘CD’), (100, ‘C’), ( 90, 
‘XC’), (50, ‘L’), (40, ‘XL’), (10, ‘X’), (9, ‘IX’),(5, ‘V’),(4,’IV’),(1, ‘I’) ]
        tempId = label

        if label == 0:
            roman += ‘+’
        else:
            for step in steps:
                while tempId >= step[0]:
                    roman += step[1]
                    tempId -= step[0]



        x = height
        #points = [ (0,x,0), (x/2,x,0), (x,x,0), (0,x/2,0), (x/2,x/2,0), (x,x/2,0),
(0,0,0), (x/2,0,0), (x,0,0) ]
        #letters = { 'M' : (6,0,4,2,8), 'D' : (0,1,5,7,6,0), 'C':(2,1,3,7,8),
'L':(0,6,8), 'X': [(0,8),(2,6)], 'V': (0,7,2) , 'I':(1,7), '+':(1,5,7,3,1) }
        points2 = [ (0,0,0),(0,2,0),(0,4,0) ]
        letters2 = { 'M' : [0,1,2], 'D' : [0,2], 'C':[1,2], 'L':[0,1], 'X': [2], 'V':
[1] ,'I':[0], '+':[] }
        polylist = []
        """
        for letter, lIdx in zip(roman,range(len(roman))):
            if letter == 'X':
                polylist.extend ( [ [rs.PointAdd( points[idx], (lIdx*x*1.6189,0,0) )
for idx in part] for part in letters[letter] ]  )
                continue
            polylist.append( [ rs.PointAdd( points[idx], (lIdx*x*1.6189,0,0) ) for idx
in letters[letter] ] )
        """



        if not roman == '+':
            for letter, lIdx in zip(roman,range(len(roman))):
                if lIdx == 0: polylist.append([])
                polylist[-1].extend( [ rs.PointAdd( points2[idx], (lIdx*2,0,0) ) for
idx in letters2[letter] ] ) #code
            polylist[-1].extend( [ rs.PointAdd( points2[2], (lIdx*2+1,0,0) ),(rs.
PointAdd( points2[0], (lIdx*2+1,0,0))) ]) #seperator line
```

```
        else:
            polylist.append( [ rs.PointAdd( points2[2], (0*2+1,0,0) ),(rs.PointAdd(
points2[0], (0*2+1,0,0))) ]) #seperator line



        #return [polylist, len(roman)*x*1.618]

        if not roman == ‘+’: length = len(roman)*2+1
        else: length = 1

        return [polylist, length]

    def __obtainRawPointlist(self):
        return self.cObj.getPointlist()
```

```
#Input Constants
#explicitly defined constants are input-parameters; others are derived from these inputs
and should not be changed; units are usually ‘mm’, ‘degrees’ or DIN-Sizes for hardware
materialThickness = 5
##hardware-dictionaries
nutWidth = { ‘M3’:5.4 }
nutThickness = { ‘M3’:2.3 }
screwDiameter = { ‘M3’:3 }
##connectorSection
###connectorSectionTenon
connectorSectionTenonLength = 5
connectorSectionTenonHeightAddition = 0.5
connectorSectionTenonHeight = materialThickness + connectorSectionTenonHeightAddition
#input-derived
connectorSectionTenonThickness = materialThickness #input-derived
###connectorSectionMortise
connectorSectionMortiseLengthClearance = 0.1
connectorSectionMortiseWidthClearance = 0.1
connectorSectionMortiseLength = connectorSectionTenonLength + connectorSectionMortiseLe
ngthClearance #input-derived
connectorSectionMortiseWidth = materialThickness + connectorSectionMortiseWidthClearan
ce
###connectorSectionShoulder
connectorSectionShoulderWidth = 5
connectorSectionShoulderHeight = 5
##shard
###shardScrew
shardScrewDimension = ‘M3’
####shardScrewHole
shardScrewHoleClearance = 0.1
shardScrewHoleDiameter = screwDiameter[shardScrewDimension] + shardScrewHoleClearance
```

```
####shardScrewSlot
shardScrewSlotLength = 3
shardScrewSlotWidthClearance = 0
shardScrewSlotWidth = screwDiameter[shardScrewDimension] + shardScrewSlotWidthClearance
#input-derived
##shardNut
shardNutDimension = shardScrewDimension #input-derived
####shardNutSlot
shardNutSlotSpacing = 5
shardNutSlotLengthClearance = 0.1
shardNutSlotWidthClearance = 0.1
shardNutSlotLength = nutWidth[shardNutDimension] + shardNutSlotLengthClearance #input-
derived
shardNutSlotWidth = nutThickness[shardNutDimension] + shardNutSlotWidthClearance #input-
derived
##connectorPlate
connectorPlateWidth = materialThickness *4
connectorPlateEndCapLength = materialThickness
###connectorPlateSlot
connectorPlateSlotLengthClearance = 0.1
connectorPlateSlotHeightClearance = 0.1
connectorPlateSlotHeight = materialThickness + connectorPlateSlotHeightClearance
###connectorPlateScrew
connectorPlateScrewDimension = ‘M3’
####connectorPlateScrewHole
connectorPlateScrewHoleClearance = 0.1
connectorPlateScrewHoleDiameter = screwDiameter[connectorPlateScrewDimension] + connect
orPlateScrewHoleClearance
####connectorPlateScrewSlot
connectorPlateScrewSlotLength = 3
connectorPlateScrewSlotWidthClearance = 0.1
connectorPlateScrewSlotWidth = connectorPlateScrewSlotWidthClearance + screwDiameter[co
nnectorPlateScrewDimension]
```

```
###connectorPlateKey
connectorPlateKeyLength = connectorPlateScrewHoleDiameter + materialThickness * 2
connectorPlateKeyWidthClearance = 0.1
connectorPlateKeyWidth = materialThickness
###connectorPlateNut
connectorPlateNutDimension = connectorPlateScrewDimension #input-derived
####connectorPlateNutSlot
connectorPlateNutSlotWidthClearance = 0.1
connectorPlateNutSlotLengthClearance = 0.1
connectorPlateNutSlotLength = nutWidth[connectorPlateNutDimension] + connectorPlateNutS
lotLengthClearance #input-derived
connectorPlateNutSlotWidth = nutThickness[connectorPlateNutDimension] + connectorPlateN
utSlotWidthClearance #input-derived



#misc Clearances, Spacings, Distances
shardScrewHoleConnectorSectionMortiseClearance = 5 #this is a MINIMUM-value
shardScrewSlotConnectorSectionTenonClearance = 5 #this is a MINIMUM-value
shardEgdeshardScrewHoleClearance = 5 #this is a MINIMUM-value
shardEdgeShardScrewSlotClearance = 5 #this is a MINIMUM-value
shardNutSlotConnectorPlateNutSlotSpacing = 5

#drill
drillDiameter = 2

#colors
red = (220,20,60)
green = (127,255,0)
blue =  (65,105,225)
pink = (255,20,147)
yellw = (255,255,102)
```

```
#import pydevd
#pydevd.settrace(port=5678)
#pydevd.stoptrace()
import rhinoscriptsyntax as rs
#import Rhino
#import math
#import copy
#from globalVariables import *
#from miscFunctions import *
from GeometryClasses import Mesh
from Contour import *
from CncRep import *

mesh = rs.GetObject('Select Mesh',32)

rs.EnableRedraw(False)

myMesh = Mesh(mesh)

for obj in Contour.registry:
    if True:
        obj.drawPreview(in3d=False)

for cRep in CncRep.instances:
    cRep.draw()

rs.EnableRedraw(True)
```

Image Credits

Figure 1: The Economist, Cover of the Issue April 21st-27th 2012, http://www.economist.com/printedition/covers/2012-04-21/ap-e-eu-la-me-na-uk (viewed May 28, 2015)

Figure 2: Formlabs, http://www.wired.com/images_blogs/design/2012/09/Using-the-Form-1.jpg (viewed May 28, 2015)

Figure 3: MIT Senseable City Lab, "Trash | Track"

Figure 4: MIT Senseable City Lab, "Live! Singapore, raining taxis"

Figure 5: Tesla Motors, http://www.manufacturing.net/sites/manufacturing.net/files/tesla-assembly-line.jpg (viewed May 28, 2015)

Figure 6: Le Corbusier, Maison Dom-ino, http://rubens.anu.edu.au/raid2/no_dgb/pics/4/large/002132922905_96.jpg (viewed November 17, 2014)

Figure 7: Unité D'Habitation, Home Delivery (2008, New York, The Museum of Modern Art)

Figure 8: Rendering Unit´D'Habitation, Home Delivery (2008, New York, The Museum of Modern Art)

Figure 9: Richard Buckminster Fuller, Fly's Eye Dome, Critical Path (1981, New York, St. Martins Press)

Figure 10: Richard Buckminster Fuller, Fly's Eye Dome, Critical Path (1981, New York, St. Martins Press)

Figure 11: Richard Buckminster Fuller, Fly's Eye Dome, Critical Path (1981, New York, St. Martins Press)

Figure 12: Richard Buckminster Fuller, Raleigh Cotton Mill, Critical Path (1981, New York, St. Martins Press)

Figure 13: Richard Buckminster Fuller, Wichita House, Critical Path (1981, New York, St. Martins Press)

Figure 14: Pierre J., "Airbus A380 Final Assembly Line" Uploaded on September 20, 2010 via Flickr, distributed under CC BY-NC-SA 2.0.

Figure 15: Jeff McNeill, "Boeing 747-8 Test Planes in Assembly" July 15, 2009 via Flickr, distributed under CC BY-SA 2.0

Figure 16: Steve Jurvetson, "Tesla Autobots" October 5, 2011 via Flickr, distributed under CC BY 2.0

Figure 17: Steve Jurvetson, "Dance of the robots" October 5, 2011 via Flickr, distributed under CC BY 2.0

Figure 18: Stephen Kieran and James Timberlake, Refabricating Architecture (2004, New York, McGraw-Hll Companies Inc.)

Figure 19: http://i0.wp.com/www.ceec.com.ua/wp-content/uploads/2014/12/vqbnqb.jpg (viewed May 28, 2015)

Figure 20: http://i0.wp.com/www.ceec.com.ua/wp-content/uploads/2014/10/produktion_fh.jpg (viewed May 28, 2015)

Figure 21:

Figure 22: http://www.elk.at/eh-169-sd-35-0 (viewed May 28, 2015)

Figure 23: http://www.freitag.ch/media/stores/zurich (viewed May 12, 2015)

Figure 24: http://www.phooey.com.au/projects/96/children-s-activity-centre (viewed May 12, 2015)

Figure 25: http://en.wikipedia.org/wiki/Habitat_67 (viewed May 12, 2015)

Figure 26: https://outsiderintokyo.files.wordpress.com/2013/03/shiodome-03-nagakin-capsule-tower-kurokawa.jpg (viewed May 12, 2015)

Figure 27: http://upload.wikimedia.org/wikipedia/commons/0/04/International_Space_Station_after_undocking_of_STS-132.jpg (viewed May 18, 2015)

Figure 28: http://upload.wikimedia.org/wikipedia/commons/c/cb/STS-135_EVA_Cupola_and_Tranquility.jpg (viewed May 18, 2015)

Figure 29: Matthias Danzmayr 2015

Figure 30: Matthias Danzmayr 2015

Figure 31: Matthias Danzmayr 2015

Figure 95: Matthias Danzmayr 2015

Figure 96: Matthias Danzmayr 2015

Figure 97: Matthias Danzmayr 2015

Figure 98: Matthias Danzmayr 2015

Bibliography

Le Corbusier, “Mass-Produced Buildings”, trans. Charlotte Benton, Tim Benton, et al. Architecture and Design, 1890-1939: An International Anthology of Original Articles (1975 NewYork, NY: Whitney Library of Design)

Le Corbusier, Towards a New Architecture, trans. Frederick Etchells, (1923, Reprint, Mineova, NY: Dover Publications 1985)

Wikipedia contributors, “Habitat 67,” Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Habitat_67 (accessed May 12, 2015)

Wikipedia contributors, “Nakagin Capsule Tower,” Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Nakagin_Capsule_Tower (accessed May 12, 2015)

Wikipedia contributors, “International Space Station,” Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/International_Space_Station (accessed May 13, 2015)

Wikipedia contributors, “Second Industrial Revolution,” http://en.wikipedia.org/wiki/Second_Industrial_Revolution (accessed May 28, 2015)

Wikipedia contributors, “Neolithic Revolution,” Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Neolithic_Revolution (accessed May 28, 2015)

Wikipedia contributors, “Industrial Revolution,” http://en.wikipedia.org/wiki/Industrial_Revolution (accessed May 28, 2015)

Wikipedia contributors, “Tesla Factory, “http://en.wikipedia.org/wiki/Tesla_Factory (accessed May 28, 2015)

“A third industrial revolution,” The Economist, April 21st 27th 2012

www.airbus.com (accessed May 28, 2015)

Barry Bergdoll and Peter Christensen, Home Delivery (2008, New York, The Museum of Modern Art)

Richard Buckminster Fuller, Critical Path (1981, New York, St. Martins Press)

Stephen Kieran and James Timberlake, Refabricating Architecture (2004, New York, McGraw-Hll Companies Inc.)

Dietmar Offenhuber and Carlo Ratti (Ed.), Die Stadt entschlüsseln (2013, Birkhäuser Verlag GmbH, Basel)

Helmut Pottmann, Andreas Asperl, Michael Hofer and Axel Kilian, Architectural Geometry (2007, Bentley Institute Press, Exton)

National Electrical Manufacturers Association, “About the National Electrical Manufacturers Association,” http://www.nema.org/About/pages/default.aspx (accessed May 15 2015)

National Electrical Manufacturers Association, “All Standards (One Page),” https://www.nema.org/Standards/Pages/All-Standards-One-Page.aspx (accessed May 15 2015)

National Electrical Manufacturers Association, “NEMA Standards Publication ICS 16,” Rosslyn 2001

Arup, “China Television Headquarters,” http://www.arup.com/Home/Projects/China_Central_Television_Headquarters.aspx (accessed May 25, 2015)