nds2004 : caad : arch : ethz : ch

Generation and optimization of complex and irregular construction/structure

on the example of NDS2004 final project

Agnieszka Sowa
nds2004 : caad : arch : ethz : ch

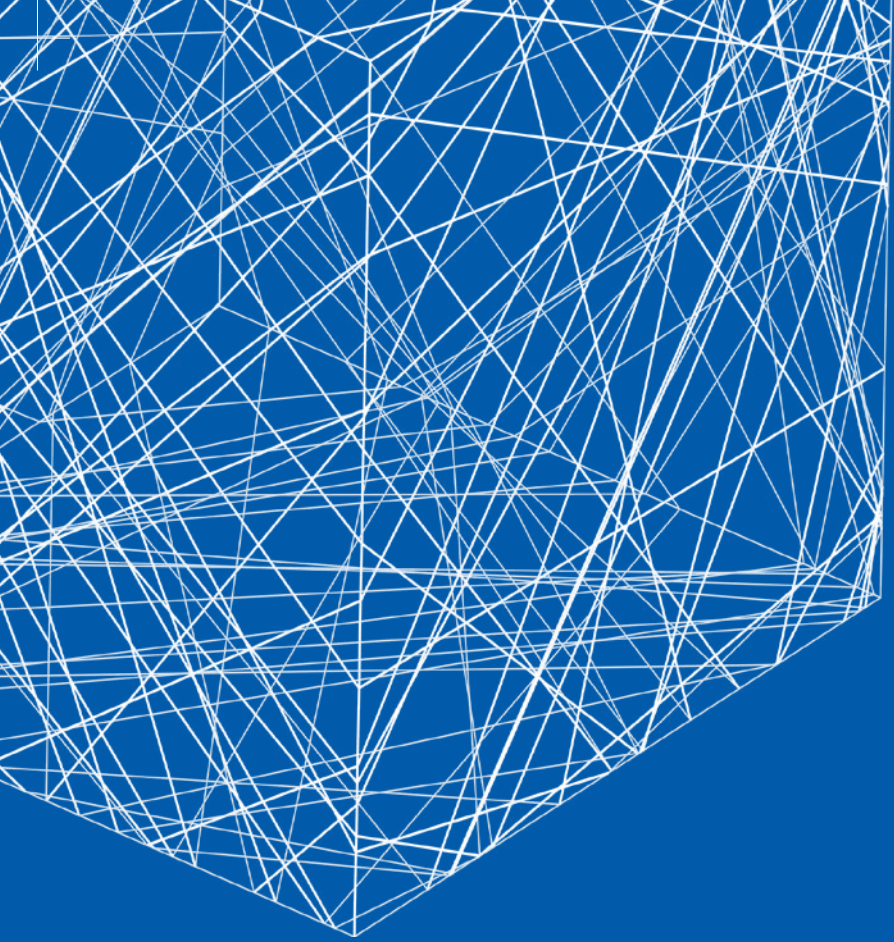
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ETH ZÜRICH
Chair of CAAD
Prof. Ludger Hovestadt
HIL E 15.1
ETH Hönggerberg
8093 Zürich
Switzerland

www.caad.arch.ethz.ch

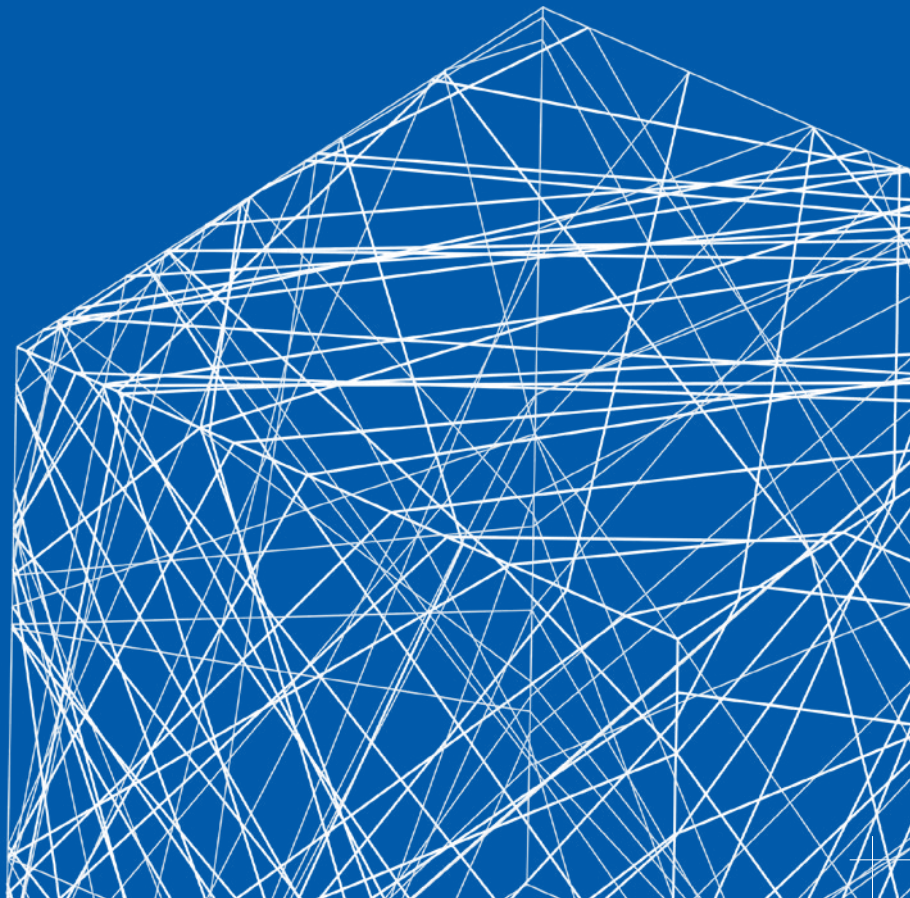
Master Layout | Li-hsuen, Yeh
Text, Drawings | Agnieszka Sowa

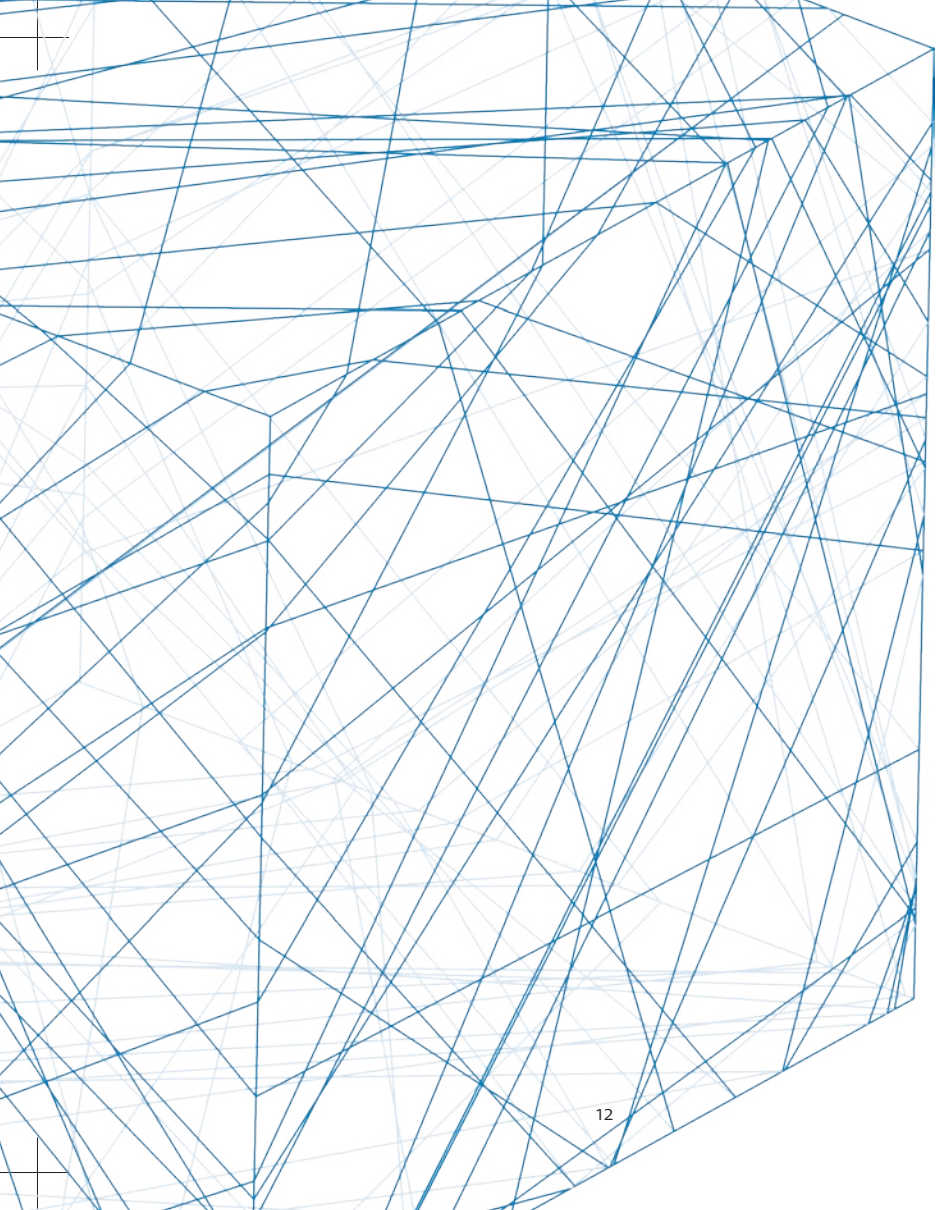
Zürich, October 2004



ACKNOWLEDGEMENTS

Professor of CAAD	Ludger Hovestadt
From CAAD Chair	Sibylla Spycher Oliver Fritz Markus Braach Russell Loveridge Christoph Schindler Karsten Droste Oskar Zieta Odilo Schoch Kai Strehlke
NDS Courses Coordinator	Philipp Schaerer
ETHZ Department of Architecture Research / Postgraduate Studies	Bruno Dobler
NDS2004	Christian Dürr If Ebnöther Jörg Grabfelder Anna Jach Jae Hwan Jung Alexandre Kapellos Irene Logara Michelangelo Ribaudo Hanne Sommer Agnieszka Sowa Detlef Wingerath Thomas Wirsing Li-Hsuen(Alexsandra) Yeh





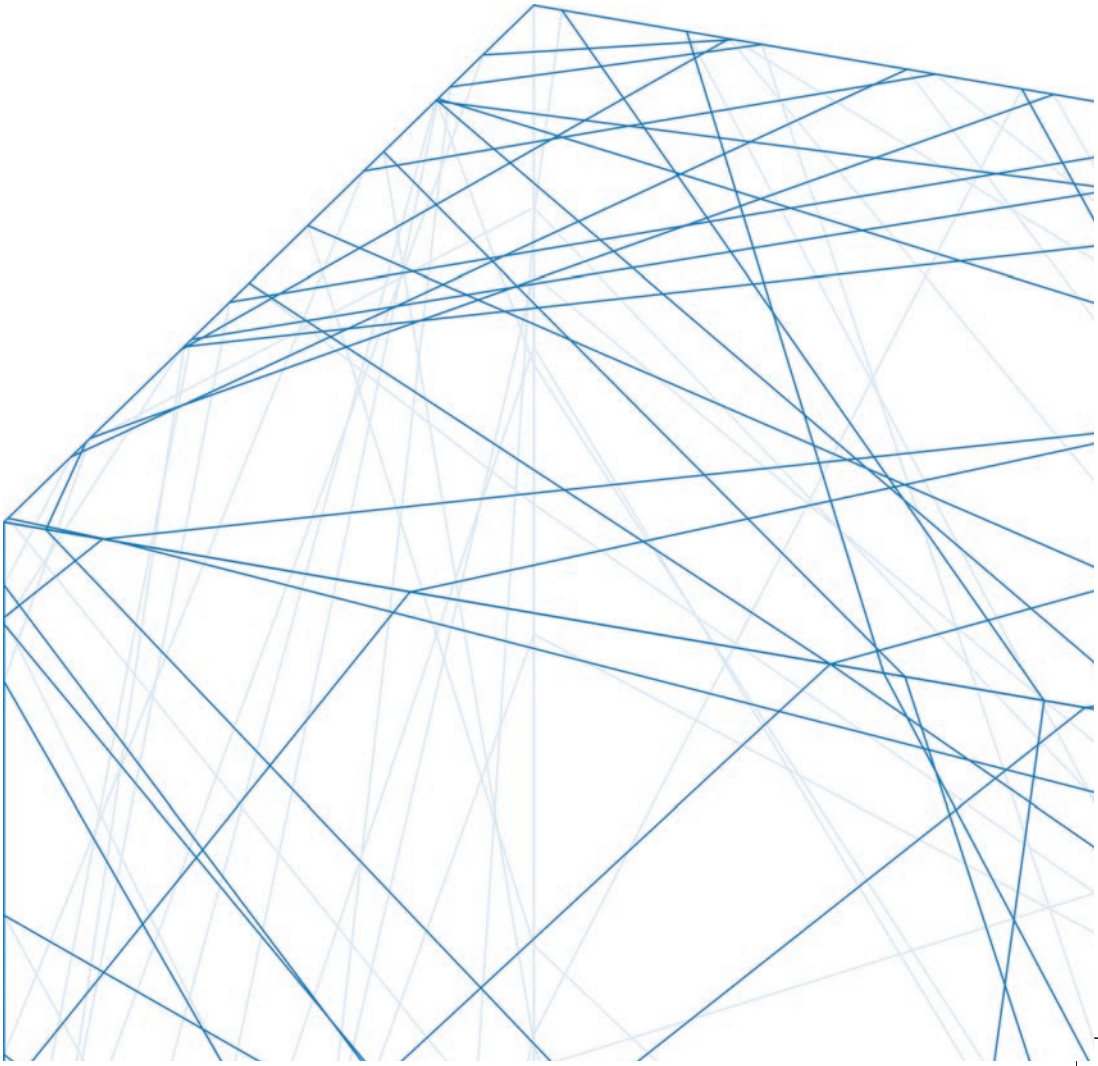
ABSTRACT ENGLISH

Generation and optimization of complex and irregular construction/structure on the example of NDS2004 final project

During the postgraduate studies in CAAD at ETH the research is mainly focused on computer based architectural design and its automatic production. Usually the way from an architectural idea to production starts with creating of a digital model of the structure which is than transformed into data which can be used by CNC machines. The final group work of NDS2004 students also follows this schema. This thesis is centered on its first element – computer aided architectural design.

The aim of the research was to create a programming tool which generate linear construction grids for a cubic form and then optimize them according to given parameters. The data produced during this process is then visualized by digital models which can be evaluated by a designer as ready for production or can be changed in a further design process. Eventually, this data is an input for scripting tools creating production drawings for CNC machines.

The thesis contains information about the mathematical description of the structure, methods of its generation, analysis and optimization. It deals also with problems connected with data exchange and storage. The effect of the work is presented by visualizations of digital models as well as by using rapid prototyping methods. Moreover, the most spectacular result of using tools presented in this thesis is the NDS2004 exhibition structure.



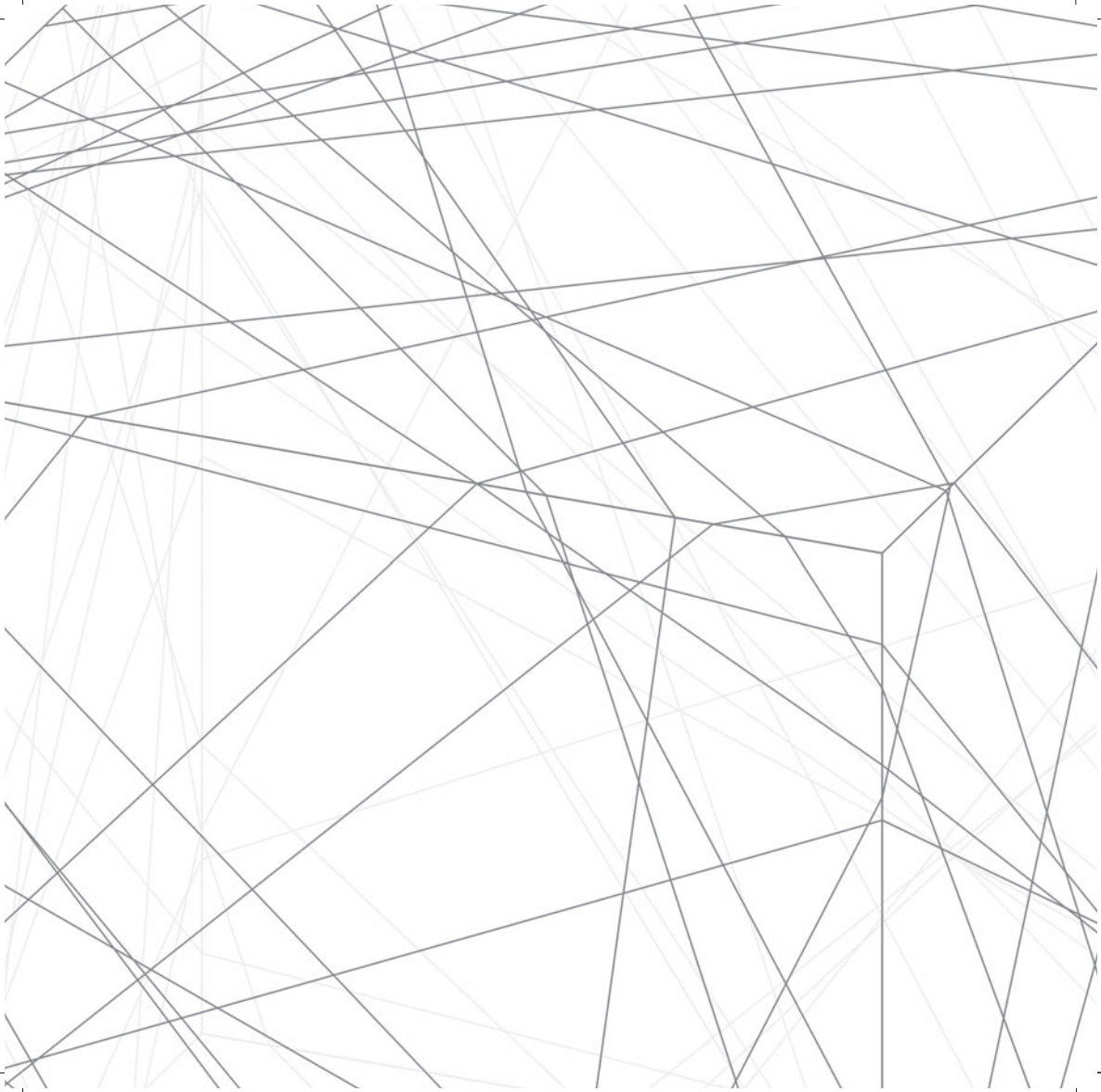
ABSTRACT DEUTSCH

Generierung und Optimierung komplexer und irregulärer Konstruktionen/ Strukturen am Beispiel des Schlussprojektes des NDS2004

Während des CAAD Nachdiplomstudiums an der ETH konzentriert sich die Forschung hauptsächlich auf computerunterstütztes Design und dessen automatische Produktion. In den meisten Fällen beginnt die Umsetzung einer architektonischen Idee mit der Erstellung eines digitalen Modells der Struktur, der dann in Daten umgewandelt wird, die von CNC-Maschinen gelesen werden können. Die Endgruppenarbeit der Studenten des NDS2004 befolgt ebenfalls dieses Schema. Die vorliegende These behandelt in erster Linie computerunterstütztes Architekturdesign.

Ziel der Forschung war die Erstellung eines Programmierwerkzeugs, welches lineare Konstruktionsraster für kubische Formen generiert und diese anhand gegebener Parameter optimiert. Die Daten, die durch diesen Prozess produziert wurden, können dann durch digitale Modelle visualisiert werden. Letztere werden von einem Designer evaluiert und für die Produktion benutzt oder in einem weiteren Design-Prozess verändert. Es ist möglich diese Daten als Input für Scripting-Werkzeuge zu benutzen, die Produktionszeichnungen für CNC-Maschinen erstellen.

Die These enthält Informationen über die mathematische Beschreibung der Struktur, Methoden derer Generierung, Analysen und Optimierungen. Ferner werden auch Probleme bezüglich des Datenaustausches und der Speicherung behandelt. Das Ergebnis dieser Arbeit wird anhand von digitalen Modellvisualisierungen wie auch durch die Benutzung von Rapid Prototyping Methoden vorgestellt. Der NDS2004 Prototyp ist überdies das spektakulärste Ergebnis bei der Benutzung von Werkzeugen in dieser These.



CONTENTS

1. BACKGROUND

1.1 The CAAD Chair at the Swiss Federal Institute of Technology in Zurich

Eidgenössische Technische Hochschule Zürich (ETH)

The CAAD Chair at the department of architecture, ETH Zürich, researches the using of modern information technologies as a development of the concept of architecture. This extended definition encompasses: design support by means of digital media, manufacturing & construction with computer controlled machines and intelligent building services.

(from: http://wiki.arch.ethz.ch/twiki/bin/view/NDS/NdsIntroduction)

1.2 Postgraduate Studies in CAAD

NachDiplomStudium (NDS) in CAAD

The postgraduate studies in CAAD are open to Swiss and international graduates with professional experience from the field of architecture and related disciplines. The main focus of the curriculum is computer based architectural design (CAD) and its automatic production (CAM).

(from: http://wiki.arch.ethz.ch/twiki/bin/view/NDS/NdsIntroduction)

1.3 NDS group work as a crowning of postgraduates studies in CAAD at ETHZ

Apart from the fact that the students are required to produce individual theses to complete postgraduates studies at ETHZ, NDS CAAD students are expected to work as a group on one common, large and complex project. Students of two previous NDS courses designed the so-called pavilions, that is small scale architectural objects without a specified function. Moreover, the structures were not only designed – they were also build in 1:1 scale, using CNC machines and other equipment provided by the CAAD Chair.

After some discussion, NDS04 students also decided to follow this pattern – at least as far as an architectural scale and production methods are concerned. This thesis presents a workflow and effects of the research and programming in the first phase of the design and production process; namly, the stage where the designer's idea concerning the NDS04 structure is translated into a language understandable to machines and optimized so that it can exist not only on the computer screen, but also in the real world.

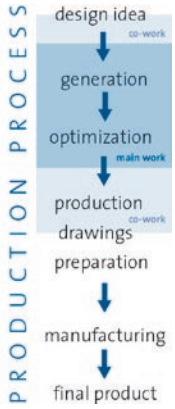
It is important to realize that the presented work is only an example – a particular instance – of all possibilities which the designer is offered by computers. The main idea of the structure can be extended and used also for bigger and more complex architectural forms. Furthermore new features can be added and analyzed. So this work should also be regarded only as an instance of endless variety of all possible solution. Nevertheless, it shows that thanks to using computers an unknown level of complexity and truly integrated design and manufacturing is now more achievable.

2. INTRODUCTION

2.1 content of the thesis on the background of the NDS04 groupwork

The first idea regarding a NDS2004 CAAD group work object could be described quite easily – its outside form is a cube, built by freely rotated planes (on first sketches, they were rotated only around one of the three 3D coordinate system axis). From the structure built by planes an inside shape is subtracted. This work schema results in a structure built by irregular frames.

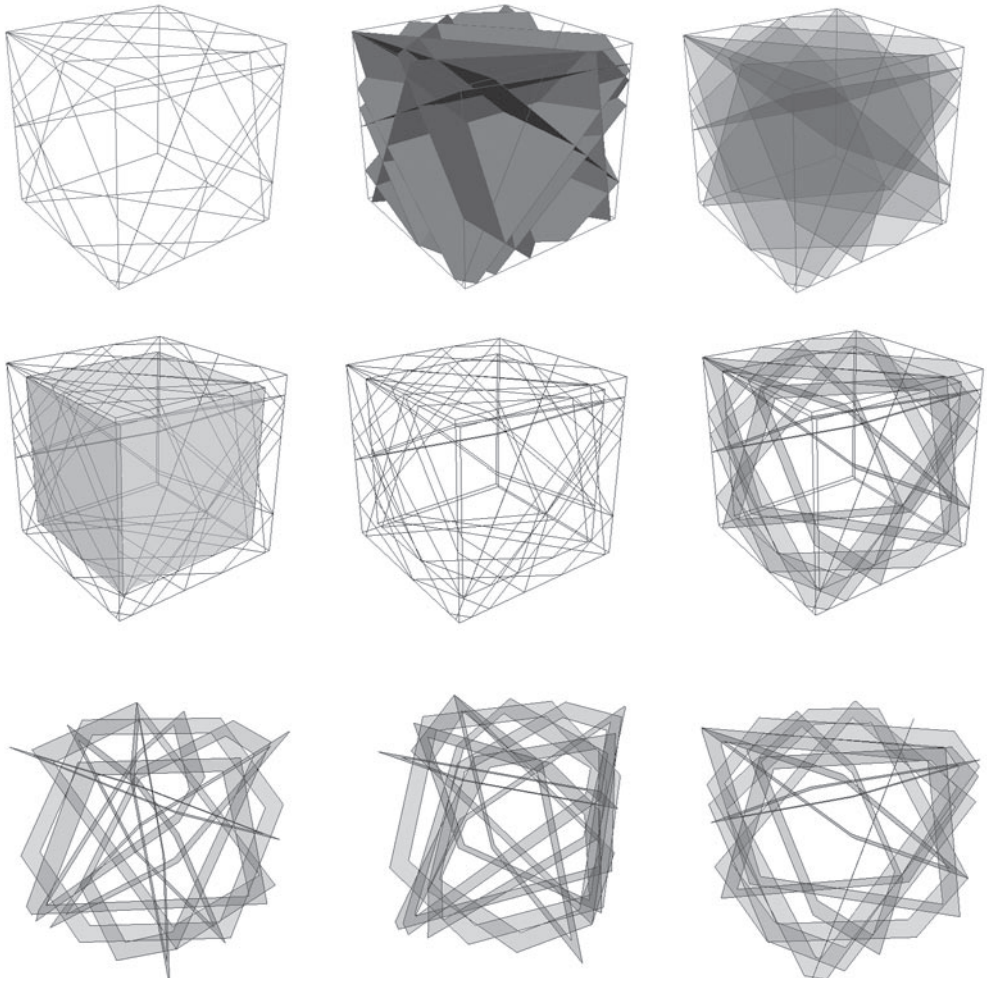
During the programming phase described in this thesis the general idea was being more and more specified and defined. In the final project the planes are rotated freely in all directions. Because of the time-limit (there were only two months between the beginning of the programming and the manufacturing start) the outside and inside shapes remained cubic, but the idea can be also applied to other outside and inside forms.



| 2.1 | the work of the thesis' author against a background of the entire production process.

2.2 structure of the thesis

The goal of NDS students was not only to design the structure and visualize it on the computer screen, but also – and it was a far bigger challenge – to build it using real materials, machines etc. The work described in this thesis fully coveres two phases of "from idea to a final product" process (*fig, 2.1*) – the generation of a wire frame model of an object structure grid and its optimization with regard to diverse esthetic, manufacturing and material features.



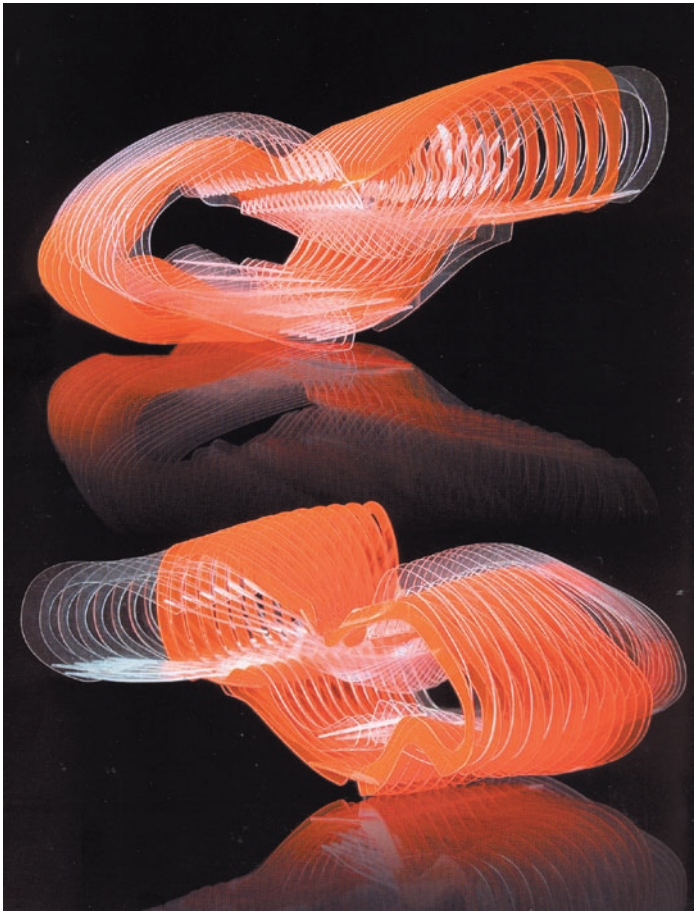
Other examples of computer generation and optimization usage in architectural design are also presented as well as more specified mathematical and theoretical background of the programming. Details of the NDS04 groupwork are also described, followed by a description of used scripting language in ***Appendix 1,*** while the list of created scripts with their parameters and functions can be found in ***Appendix 2***.

2.3 role of CAAD technologies

The idea about the appearance of a form comes from usually the designer during the design process , as a sketch or other form of visualization. In NDS project it is more a verbal "description", where is a place for computer generation and calculation. Consequently, the computer has a big influence on the final form – its construction as a surface structure as well as its elevations.

The computational optimization is also crucial in this case, because the generated structure is so complex and irregular that is seems to be impossible to analyze it without using of a computer. Or, to put it differently, it is hardly possible to find a bulidable solution among all generated propositions without complicated calculations. Also every, even the smallest, change of any structure element entail such complex changes in the entire construction, that it has to be recalculated and reanalyzed. Here the main reason of using a computer in this project can be seen – the structure is simply so complex that it has to be tackled with by a machine.

| 2.2 | schema of the cube creation idea



| 3.1 | Multiplanar surface articulation: through the mutual adaptation of geometric fitness criteria and geometric articulation the morphogenetic process yields an ever-increasing complexity of two co-evolved surfaces that nevertheless remains coherent with the logics of the material system and manufacturing with a laser cutter.

3. COMPUTER GENERATION AND OPTIMIZATION PROCESSES IN ARCHITECTURAL DESIGN

gen•er•a•tion *n.*

1. the formation of a line or geometric figure by the movement of a point or line.
2. the act or process of generating; origination, production, or procreation.

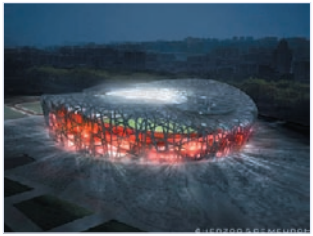
op•ti•mi•za•tion *n.*

The procedure or procedures used to make a system or design as effective or functional as possible, especially the mathematical techniques involved.

Optimization is a process of looking for a solution to a given problem. Its aim is to make structure features as close to the optimum as possible. It does not result with the perfect and the "one and only" solution, mostly because the problems in question are so complex that there is no possibility to check and decide if it is the ideal solution or it is only an approximation. Moreover, the effect of this process is always different, even with the same input data.

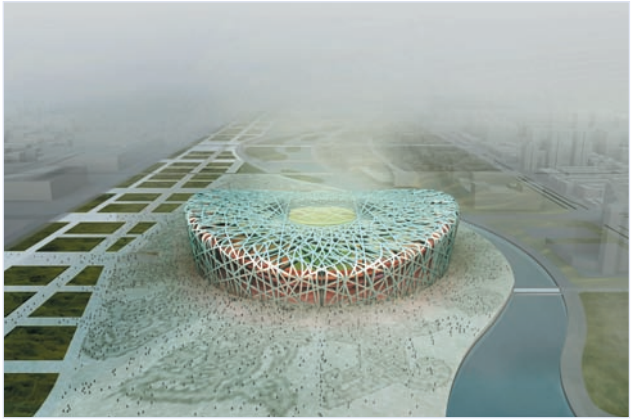
Usually it is difficult to consider in architectural design computer aided generation without optimization and vice versa. Naturally when the aim of an architect is only to create a good looking, pure virtual form – the computer generation could exist without optimization, but structures generated by random need optimization to become buildable. Moreover, evolutionary generating algorithms seem to consist also optimization tools.

The idea of optimization in architectural design is not new. Fabric architects were the first to use computers to optimize form and design in architectural structures – pioneer research was done by Frei Otto in the late 50's. J. Gero and A.D Radford In a book "Design by Optimization in Architecture and Building" were considering different possibilities of using optimization tools already in 1988 – but more in relation to function planning and cost analyses. Examples of structure optimization are not yet so numerous. The most spectacular is probably **Beijing National Stadium**, Olympic Green, China by Herzog and deMeuron. Here the irregular structure elements support each other and build a grid-like formation (*fig. 3.2, 3.3*).

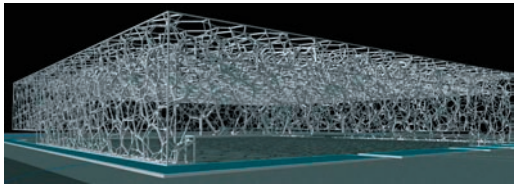


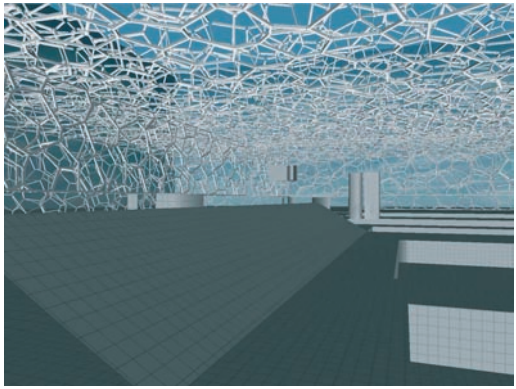
| 3.2 | Beijing National Stadium – structure after optimization

| 3.3 | Beijing National Stadium – structure before optimization, visible non-straight elements



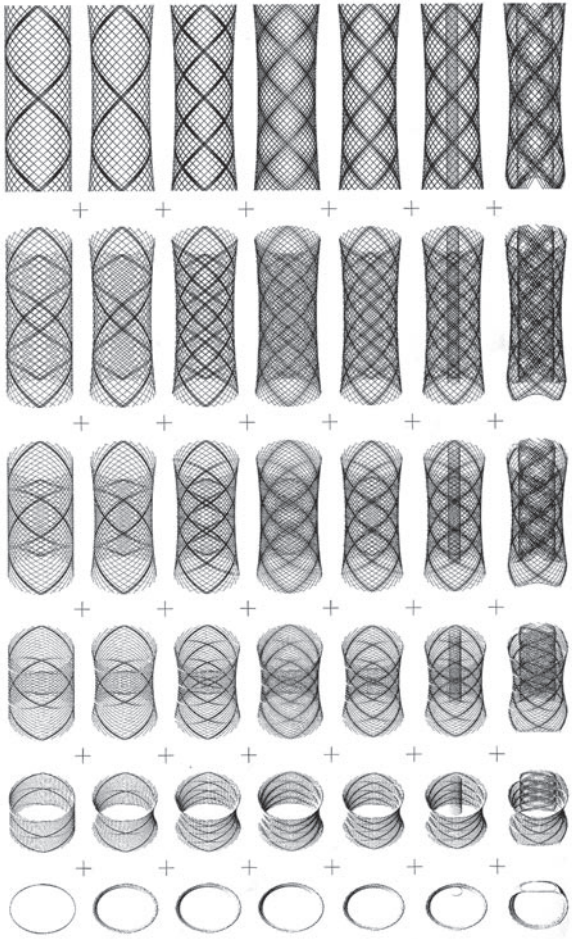
The **"Water Cube"** (*fig. 3.4, 3.5*) was an idea for a design competition entry submitted by China State Construction Engineering(CSCEC) in collaboration with PWT Architects and Arup Sidney. The project is a world class swimming centre in Beijing for the 2008 Olympic games. The structure was inspired by cells and soap bubbles and it is based on a common natural process, the most natural effective subdivision of 3D space. This structure is highly repetitive and buildable while appearing very organic and random. The optimization process meant in this project that the sectional sizes and node diameters were even changing and non – repetitive.



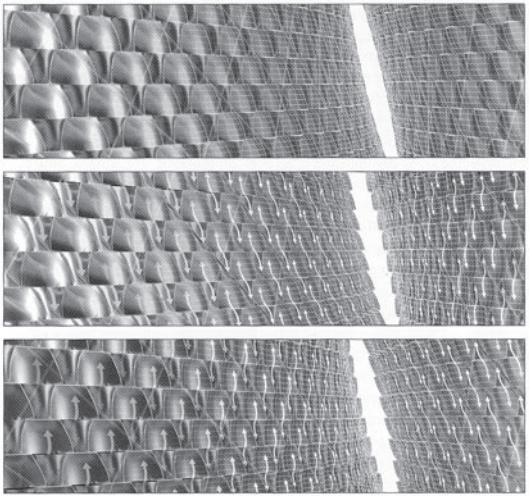


| 3.4 | (above) Water Cube - external steel frame perspective
| 3.5 | (right) Arup realtime screenshot; from

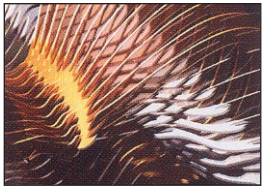
There are also some examples in more theoretical using of evolutionary generation and optimization. The **Emergence and Design Group** (Michael Weinstock, Achim Menges and Michel Hansel) used evolutionary algorithms in research project on high-rise buildings as surface structures (*fig. 3.6, 3.7*). The "seed" – the section of a steel tube was generated at the beginning of the process, it finally resulted with a double-helix structure. the development of the phenotype (the definition – see ***OPTIMIZATION***) was driven by exposure of the geometry to environmental forces. The surface of the structure was covered by panels which were active locally and responding local stimuli, they were capable of modulating the passage of light, heat and air through it in both directions.



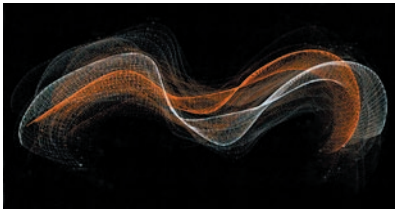
| 3.6 | research project on high-rise buildings as surface structures - helix evolution



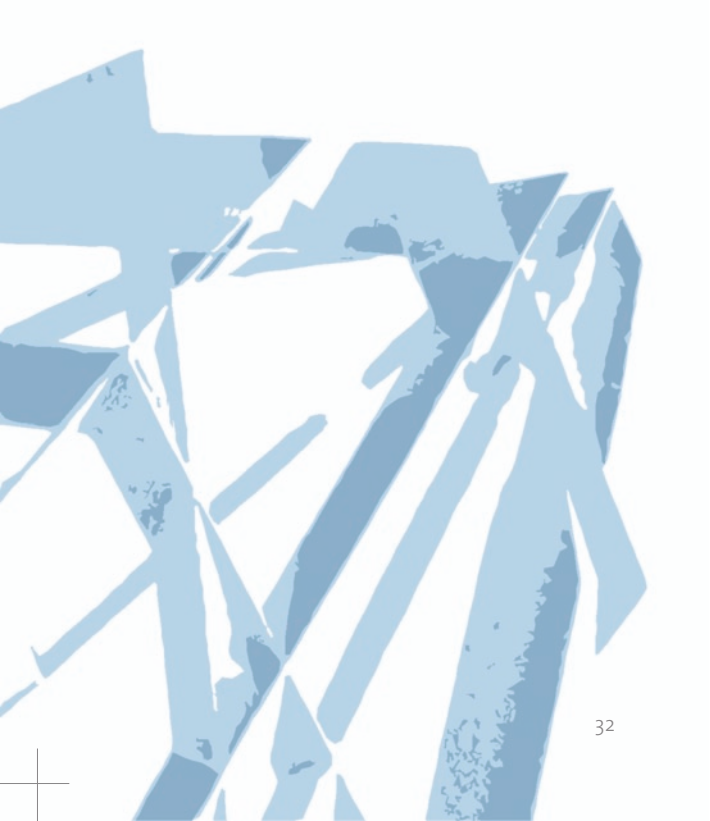
| 3.7 | *research project on high-rise buildings as surface structures skin panel*



| 3.8 | |3.9 | *digital model evolved by a digital morphogenetic process using Genr8.*



Contemporary architects can experiment with evolutionary generation algorithms also using the software developments that operate on the intersection of architecture, artificial intelligence, artificial life, engineering and material science. Here presented example was created with Genre8, a combination of evolutionary computation and generative algorithm. It is implemented as a plug-in for Alias Wavefront's 3D modeler Maya. In experiment (*fig. 3.8, 3.9*) described in (O'Reilly , 2004) two interlocking surfaces evolved through geometric fitness criteria.

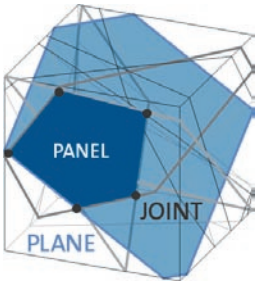


4. GENERATION

4.1 overview

Fig. 4.1 describes definitions which were introduced in the generation and optimization description. As it was mentioned before, the outside form of the NDS2004 object is a cube.

The structure is built by planes, i.e. flat, thin, freely rotated construction elements which are connected with each other so that they form a stable structure. Joints are located where planes intersect. The areas between planes are called panels.



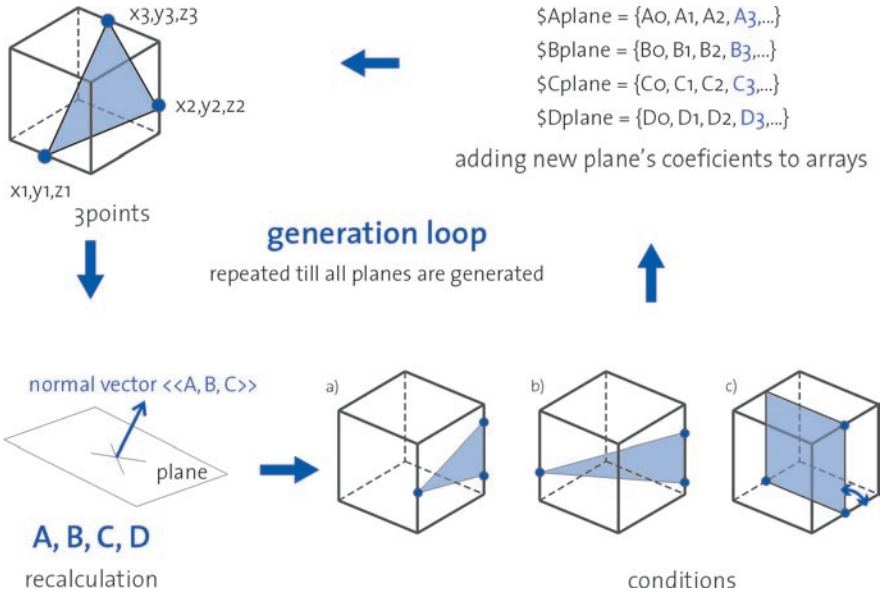
| 4.1 | definition: plane, panel and joint

4.2 structure of the generation script

Generally, the generating script consists of two main procedures. Firstly, the planes are generated randomly – this part is the actual generating script. Secondly, the planes are visualized inside a cube as they build a structure. Of course there are also other procedures used during the process (see ***Appendix2***).

4.2.1 procedure of planes generation

The generation procedure has only one parameter, i.e. the number of planes expected as a result. The planes should satisfy the following conditions: they should be generated randomly in space and be irregular – not parallel to the cube sides. They should also be located in space in a manner that they intersect with the cube. In addition they should build a rigid structure – so intersect with at least few others.

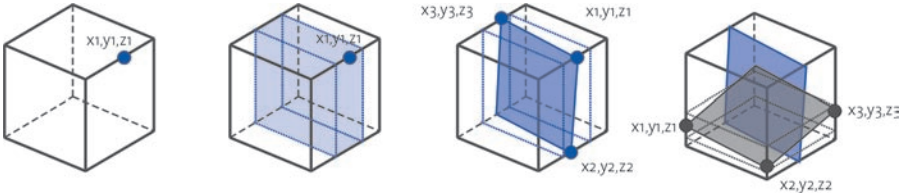


| 4.2 | the schema of generation algorithm

Because of the limitation that already occurres during the generation the planes should follow some rules and fulfill some conditions, like:

- planes are define firstly by 3 points which are randomly choose on edges of the cube (so that they intersect the cube for sure)
- there should be no 3 points on the same side (*fig. 4.2a*)
- there should be no 2 points on the same edge (*fig. 4.2b*)
- the angle between generated plane and 3D coordinate system planes should not be close to 90 in any direction (*fig. 4.2*, see in the chapter ***OPTIMIZATION***).

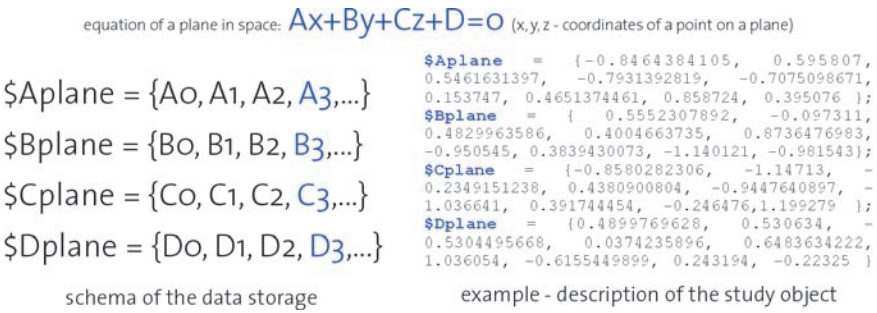
This particularset of rules was formulated after some experiments with different generation patterns. One of these experiments started with an idea that perhaps it would be better if the planes were building a more regular structure already during the generation process – with perpendicular and vertical planes rotated only a little in relation to each other. The script based on a rule that only first point of every plane is created totally at random. Next points coordinates had restrictions – as it is shown on a *fig. 4.3*. Eventhough the idea seemed promising, the visual effect of the experiment did not fulfill the expectation. The structure turned out to be too regular. Secondly, the optimization of the grid using this generation pattern resulted in the structure which had only parallel planes and looked like a regular, straight construction grid.



| 4.3 | schema for generation with restriction algorithm

4.2.2 calculation and storage of data

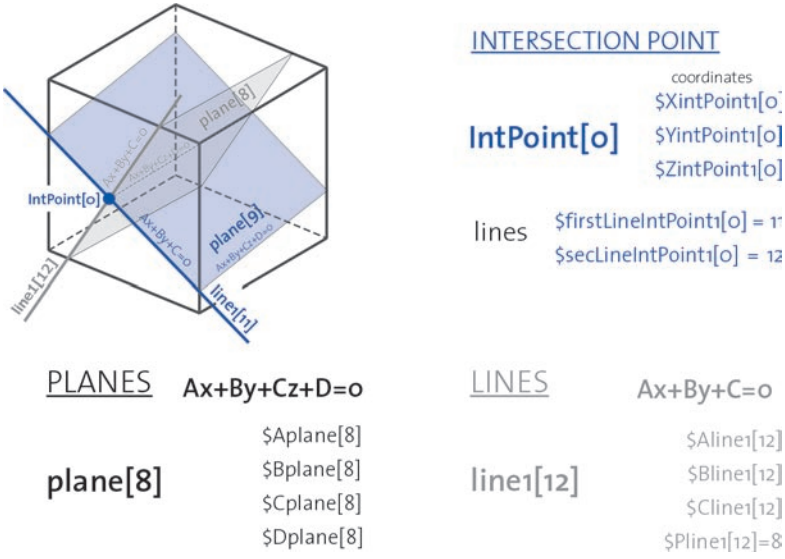
The data about the three points defining plane (as 9 values – x, y, z coordinates of the points) is then already recalculated inside the generation procedure to describe a plane only by 4 values: <<x, y, z>> of normal vector (perpendicular to this plane) and translation in space. Owing to this recalculation the data which is necessary to describe a cube and redrawit again is minimized to a list of four values for every plane (*fig. 4.4*).



| *4.4* | *the structure of the coeficients*

While working with Maya it turned out that the variables used by scripts were not saved during the file saving – so all the numeric data about the generated structure was lost every time, just a 3D wireframe model was stored in .mb file. From the moment it was discovered, all the scripts were structured so that they are able to restore all the information about its variables, coordinates, coefficients etc. only from the list of four digits planes descriptions. Therefore there is no need to store a Maya file with the cube model. Instead it is enough to save text/numeric output in a plain text file.

Because the information about the structure is used by drawing construction procedures and scripts, the data has to be stored in a way that it can be than reused. Particularly the data about joints is crucial for production procedures. From the mathematical point of view joints in the study object are intersection points between two planes and cube side. In other words, every



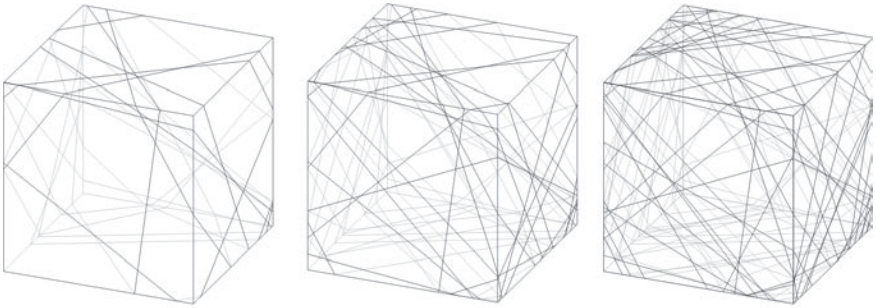
| 4.5 | structure of the data storage

plane has an intersection line with a cube side. If two of these lines (i.e. from two different planes) intersect a joint comes into being.
The data about the joints is stored in arrays (lists of values), where values on the same position refer to the same point. The arrays includes following information:

- the number of the cube side where the point is located (because all calculations are executed on 2D planes of cube faces)
- x ,y, z coordinates of the point
- numbers of two intersecting lines which create this point (thanks this the information it can be recalculated which planes intersect to build this particular joint)

Arrays are also used to store information about lines. Because the lines are always consider as drawn on the cube's sides (also on planes) - they can be described by a equation of a line on a plane, not in space. Information about the lines includes:

- A,B,C – coefficients from the a line equation Ax+By+C=0 (where x, y are coordinates of points on the line)
- the number of the side where the line is located (the third coordinate of every point of this line)
- the number of plane which creates the line



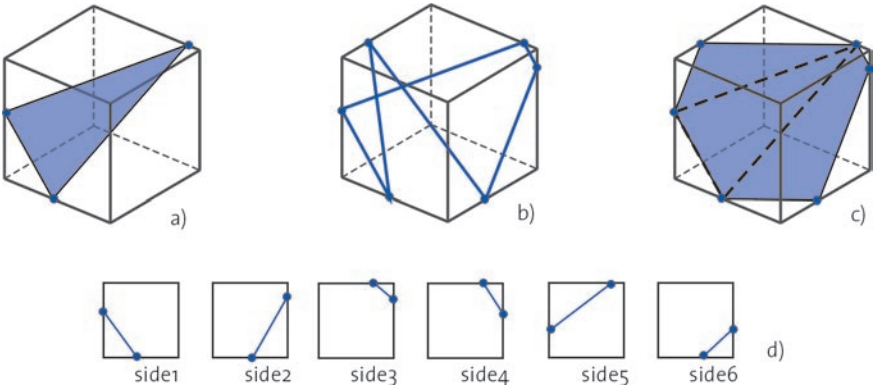
| 4.6 | examples of different structures with different numbers number

4.3 visualization

From the programming point of view there is no need to visualize data during and after the generation process. But a visual analysis of 3D structure model can be useful for both the programmer and for the designer. Firstly, the programmer can evaluate results of scripting and check if there are no mistakes in algorithms' principles. Secondly, the designer can assess an esthetic result of generation and/or change some features manually if it is needed or desired. These are the reasons why (even if no drawings are needed for the generation) already in this phase of work the visualization scripts were created.

The 3D model of the generated construction grid is a linear, wire frame structure. It is drawn in space with lines which connect intersection points of every plane and appropriate cube edges.

The intersection between a plane and a cube is a polygon with 3, 4, 5 or 6 edges (*fig.4.7c*). The coordinates of polygon vertexes are known (as intersection points between the plane and the cube face), but it has to be decided how these points are connected in space (*fig. 4.7b*).

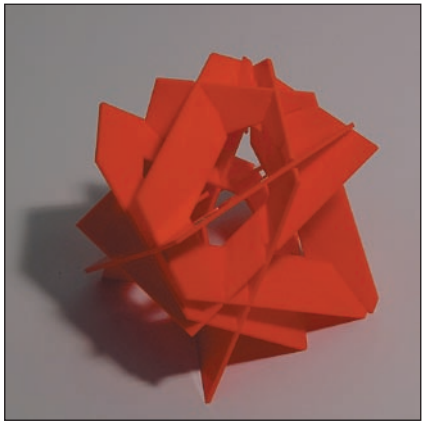


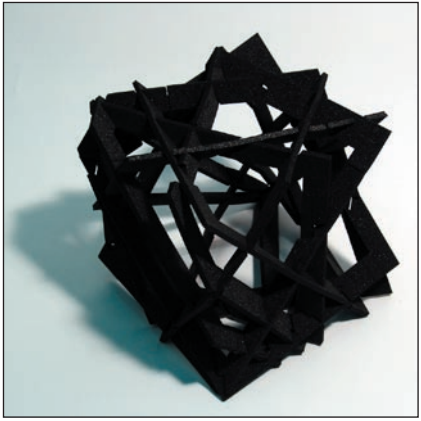
| 4.7 | schema of the drawing algorithm

| 4.8 | example of rapid prototyping model

This problem is solved in a following way: intersection points between the plane and edges of the cube are checked for every side one after the other, so every cube face is analyzed separately(*fig. 4.7d*) but drawn together in the 3D space (*fig. 4.7c*). The idea of separate calculations for every side of the cube is then continued in all scripts. Thanks to this most of the calculation is done on a plane – the cube side– not in space; this makes the procedures easier and they can be executed with the use of less variables.

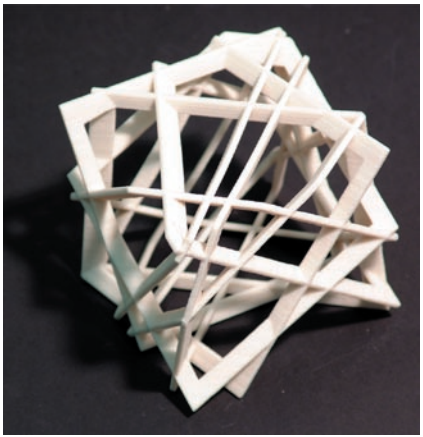
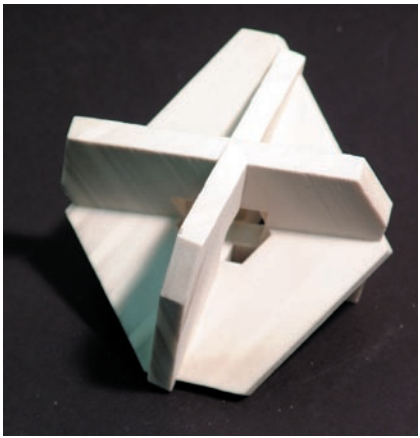
Naturally this simplification works only for a cube or regular cuboid. Other forms (such as irregular form built by flat faces, cones, spheres, cylinders or curvilinear forms "blobs") need scripting with 3D vector geometry description and calculation. During the work on the structure (by production drawing preparation for NDS04, with Michelangelo Ribaudo) some experiments with 3D vectors are already done, so some knowledge about how the work can be continued for more complex forms is already gathered. More considerations about the further work on the generation can be found in chapter entitled ***FURTHER WORK*** in the thesis.

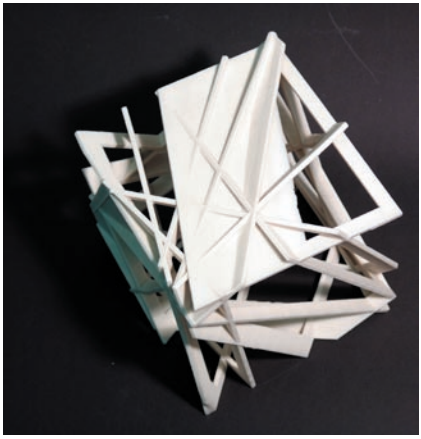
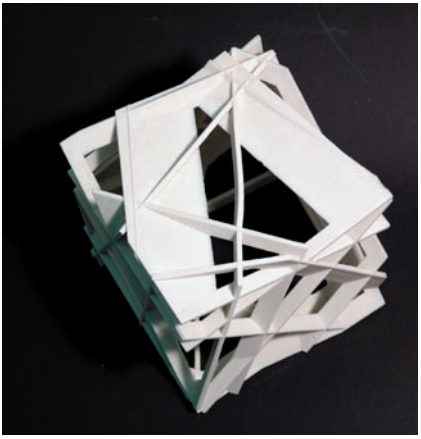


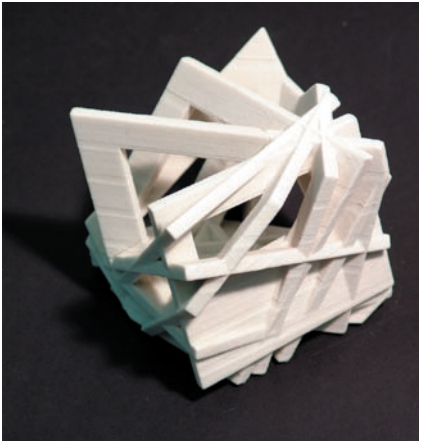


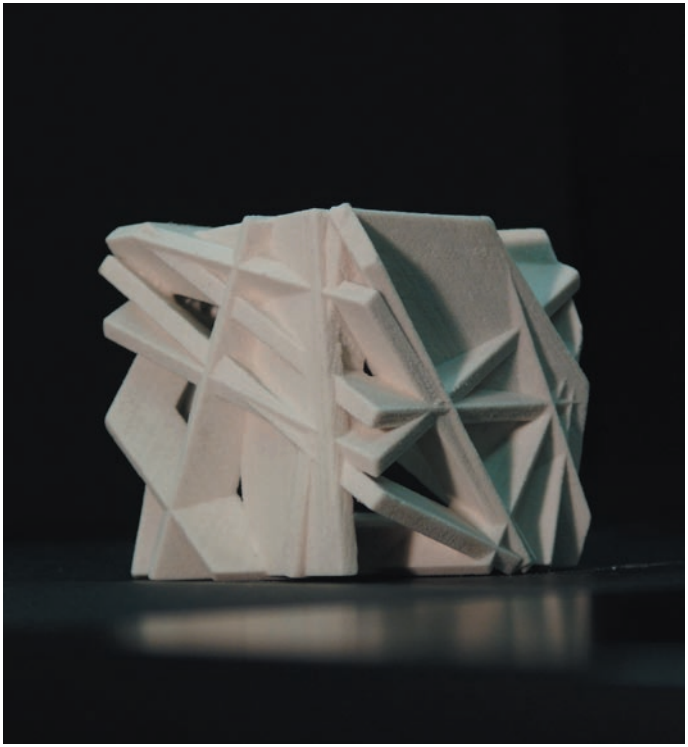
| 4.9 | | 4.10 | (right) examples of the structure small models of different materials

| 4.11 | | 4.12 | (next pages) - examples of models produced by 3D printer





| 4.13 | example of rapid prototyping model

5. OPTIMIZATION

5.1 Overview – the optimization of NDS structure

The optimization process has to take place on the script-generated structure, because it is created by a generation algorithm completely at random so it cannot be used instantly as a supporting construction. Its members should be optimized so that the entire construction fulfills conditions connected with materiality. Generally the variety of these condition could be quite wide: the type and thickness of the construction material influence the structure as well as features of materials used as panels. Machines used to produce structure elements also have their limits which can change structure parameters. All this information should be gathered and then used during the optimization as parameters.

5.2 Optimization methods

5.2.1 Genetic algorithms

gene n.
A hereditary unit consisting of a sequence of DNA that occupies a specific location on a chromosome and determines a particular characteristic in an organism. Genes undergo mutation when their DNA sequence changes.

The idea of optimization can be easily presened on an example of genetic algorithms (GA). The algorithms mimic some methods and ideas similar to the ones from real world in digital world. They were introduced into optimization problem solving area by Holland(1975) and Goldberg(1989) and since then they still demonstrate their usefulness through numerous

applications and theoretical projects (e.g. in research on flooding paths model in metropolitan area, Wen 2004).

The generic algorithms serve as visualisation and optimization tools. In visualization they "(...) can be used to explore the space in which it is impossible for the designer to consider all potential configuration in advance, and only if what results shock or at least surprises."(DeLanda, 2002). In optimization of a structure GA can be used when the structure itself is so complex that it cannot be analyzed and optimized in any traditional way, even with the use of a computer. "Traditional way, but with the use of machine" is used here to mean an analysis of every possible combination. If the number of possible combinations is infinite and when the time needed for calculation is also close to infinity, the use of GA is the best solution.

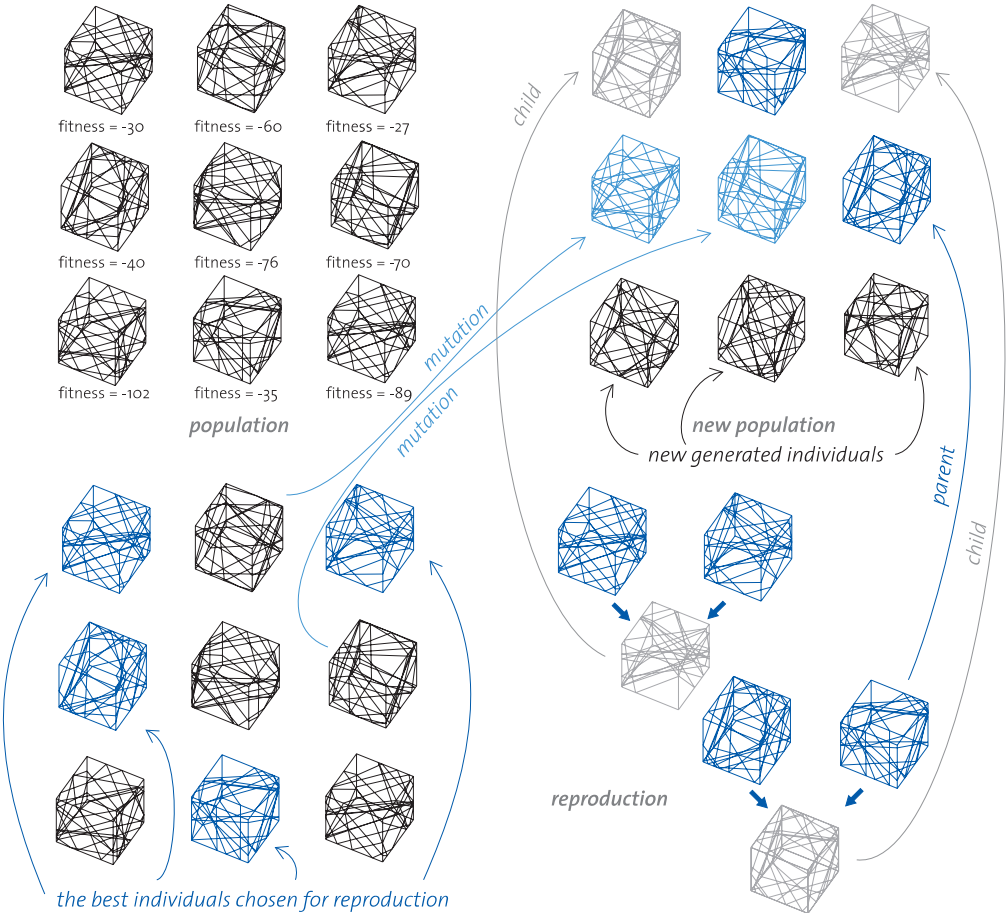
Let us try to imagine how GA can be used to optimize the wireframe of thecube structure.

The process of optimization by GA starts with creation of so called *population*. In our example it is a group of *individuals* – instances, randomly generated cubes with the same or different number of planes. The *phenotype* is simply their visualization the way the cubes look like. The phenotype of every cube is described by a *genotype*. In this case *genes* are data about every plane in a cube.

After creation, individuals' features are verified to check whether they fit to given parameters. The *fitness function* for every cube also is calculated. The value of fitness function informs how close the individual cube is to the optimal solution – the optimal cube. The process resulting in fitness function is based on analisis of good and bad qualities of the structure. Good features can be that a plane in a cube has intersection points with other planes (if it doesn not have, it cannot build a rigid structure). On the other hand, bad features are for example too big or too small distance between planes' joints.

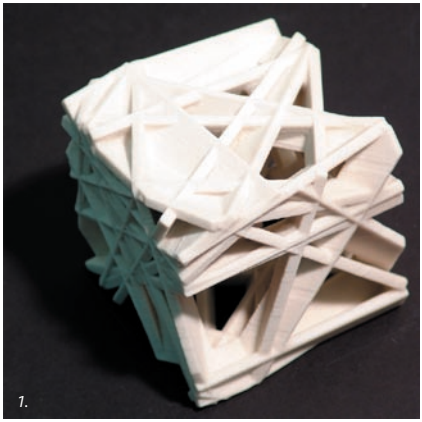
After "good" cubes are chosen, the best of them can reproduce. A *reproduction* is exchanging pieces of the genotype (this process is called the same as in genetics – *crossing over*). The new sets of genes are then genotypes for new cubes – children, instances of the second generation. Some of the parents can also be a part of a new population, also some old cubes with mutations as well as newly generated ones. The whole analysis, reproduction and mutation are then repeated on new population of cubes. (*fig. 5.1*)

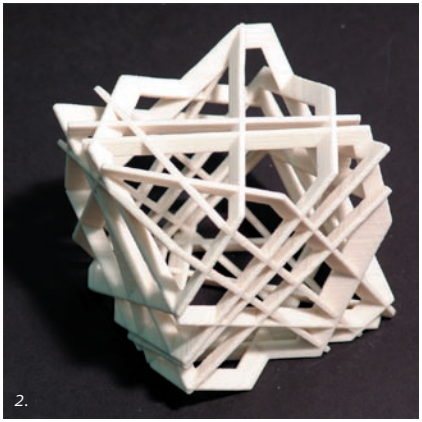
| 5.1 | genetic algorithm on example of cube structure


fitness = -30
fitness = -60
fitness = -27
fitness = -40
fitness = -76
fitness = -70
fitness = -102
fitness = -35
fitness = -89
population
mutation
mutation
child
new population
new generated individuals
parent
child
reproduction
the best individuals chosen for reproduction

At the end the optimized solution is an effect of a long cycle of this random process; every time it can lead to different results, even with the same starting population.

With the population generation as a random process it seems that a designer has not so big influence on the final result. But this impression is only partly true – by manipulation of the fitness function and setting different rules for reproduction, the effect of the process of evolution can be modified significantly.





| 5.2 | the difference between non-optimized (1) and optimized cube (2)

5.3.2 evolutionary algorithms

ev·o·lu·tion n.
1. a gradual process in which something changes into a different and usually more complex or better form.
2. biology:
a. change in the genetic composition of a population during successive generations, as a result of natural selection acting on the genetic variation among individuals, and resulting in the development of new species.
b. the historical development of a related group of organisms; phylogeny.

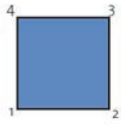
Because of a short time-limit for the programming, finally for the optimization of the structure a simplified version of evolutionary algorithm is used. Evolutionary algorithms are tools that also mimic the natural biological evolution to produce better and better approximation for a solution of a problem, but they are more focused on an individual than a population. The evolution in the natural world occurs mainly as a process of individuals' adaptation to the environment. The individuals which phenotype is better adjusted to the outside condition have better condition for reproduction, so their genes survive. This pattern is also used in architecture for form generation algorithms, where the environmental factors influence the appearance of an individual. By structure optimization the appearance (the form) is given and only the structure evaluate so that at the end of the process it is so close to optimal as possible.

During the fitness function calculation it is important to decide which features are the most important for a good solution (they will influence the structure the most), which are less important and which are only mentioned because of aesthetic point of view. These decisions also influence the final effect of an evolutionary process.

In the example of a cube the main presumption is that every plane in a structure can be analyzed only in environment of other planes, because only in coexistence with others it has bad or good features. Here the fitness function is calculated not for the whole structure, but for every plane separately.

| 5.3 | (next page) schema of the algorithm for panels searching

LOOKING FOR PANELS PROCEDURE



LIST OF EXISTING PANELS

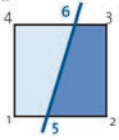
1,2,3,4

starting point - the cube's side - as basis panel

1,2,3,4 - set of vertexes in proper order

(which makes a polygon, not for example 1,3,2,4)

first crossing line



is there the intersection point

between 1 and 2?

YES

YES - its coordinates are added to the list of point coordinates

YES - so there is also the second point on other edge to look for

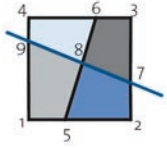
LIST OF EXISTING PANELS

5,8,7,2

1,9,8,5

7,3,6,8

8,6,4,9



the vertexes sequence 1,5,2,3,4

is there the intersection point

between 2 and 3?

NO

NO - next edge has to be checked

when new line is crossing the cube face, the same operation should be repeated for every existing panel

is there the intersection point

between 3 and 4?

LIST OF EXISTING PANELS

1,5,6,4

5,2,3,6

YES

YES - its coordinates are added to the list of point coordinates

YES - this is the second intersection point, so there is no need to check last edge of the basis panel

first starts with first new vertex and end with second new one **5,2,3,6**

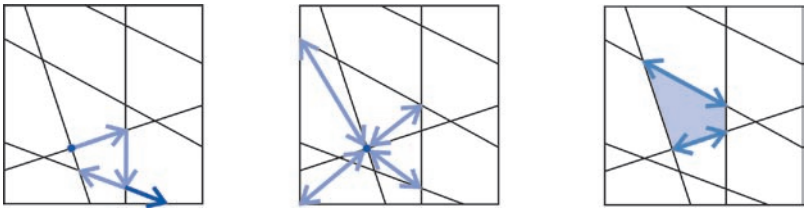
the vertexes sequence 1,5,2,3,6,4

HOW TO FIND TWO NEW PANELS IN THIS SET?

second starts with first basis vertex, continue till first new one, goes then to the second new one, and that to the end of the set **1,5,6,4**

5.3 search for panels

From the construction point of view the most important data is how the intersection points are connected to each other on cube sides, so how they build panels. The data related to them is not produced by the generation/visualization script. The panels are searched during the optimization. Again here all the calculations are done side by side. Even if for a human it is obvious where the polygons are when he/she looks at a linear drawing, it is not so easy for a computer. Lines have intersection points but not every intersection point is a vertex of every polygon. In the analyzed case every point is a vertex of 4 polygons but the polygons have different numbers of edges etc. *(fig. 5.4)*.



| 5.4 | problems occurring during panels searching

Finding polygons is very important because it allows analyzing distances between intersection points. Although the shortest distances could be easily found by checking "every point with every point", there is no way to analyze the longest distances between points it there is no information if they form a panel.

The algorithm used to find panels bases on schema presented on the *fig. 5.3*. When a panel is cut by a line, two new ones are created (or no new ones if the line doesn't cross the panel). Analyzing and calculating is done with arrays: one of them is a list of intersection points coordinates, and others are lists of these point numbers, which form a panel. The idea is that if we have all vertexes of the base polygon and we know how they are connected to each other (in the example we start with cube's face), we can find two new ones quite easily – by analyzing the new vertexes set by its order. The results of the "looking for panels" process are two lists: one with panels' description (sets of their vertexes) and second – list of all the vertexes with coordinates.

In the *fig.5.3* only the general idea is presented. The most important difference between this simplified schema and existing script is that there is not a separate list of panels vertexes which is created during panels searching. Already by generation the list of intersection points is done and this list is used by looking for panels procedure. When a vertex of a panel is found, it is also found on a list of intersection points. Number of a point on this second list is used in panel description. Thus there is less data to store in computer memory.

7.6 experiments with optimization

The experiments with optimization were taking place during the last 3 weeks of programming phase. They were not easy to carry on, mostly because of the poor MAYA calculation possibilities or, in other words, problems with storing a lot of variables needed during panel calculation. Already at the beginning it turned out that it could be a reason why it would be difficult to produce purely calculated solution for the NDS04 structure, without any manual work. That was also why some additional tools came into being – like the script to add planes to a structure (see ***ADDITIONAL TOOLS***).

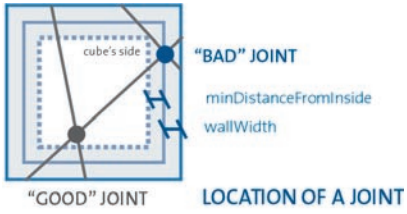
The optimization process gave a big variety of possibilities to carry different experiments. Firstly, different factors and features were analyzed during the optimization, and they had different importance for fitness function calculation. A general idea was that the fitness value is a sum of "minus points". These points were given to the cube planes according to different factors and qualities.

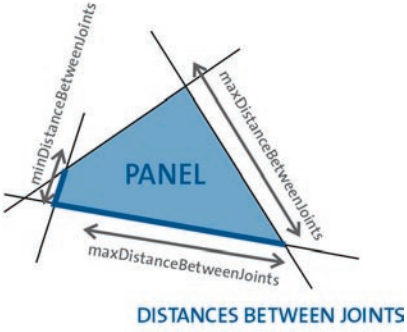
In the NDS04 structure the analyzed factors were *(fig. 5.5)*:

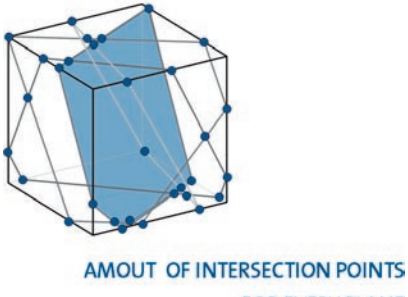
the distance between joints: it shouldn't be too small (difficulties in production) as well as too big (the strength of the material could be too small and deformation could appear); the values for the calculation were: minimum distance = 20cm, maximum distance 120cm. Naturally, the change of these values would lead to different optimization results. The planes were given the "minus points" in following way: the panel edge connected two joints – intersection points. These two points were created by 3 planes – one is common for both. If the edge is too short or too long – all three planes which produced the edge's ends were given "minus points", for example -20 for each, but the common one – twice (because this calculation was carried on for every point separately).

the number of intersection points between planes: at the beginning of the work this feature was not analyzed, what followed to a situation that the optimized cube has only planes almost parallel to each other (because it was better not to build any panels and "risk" that a panel will have too long or too short edges). The final settings for this factor were that if the plane has less than 10intersection points (at least 3 are always present, because as "intersection points" also the intersections between planes and cube edges were considered), the fitness value were decreased by 100; if the number of them is between 10 and 25, -20 is given to a plane.

distance between the cube edges and joints: the distance between the edge of the cube and the joint is also created by the so-called "wall width", i.e. the distance between the outside cube form and inside subtracted one. In reality, no frame has a thickness like the "wall width", because every one of them is rotated in space. Hence, during the optimization this virtual "wall width" was important as the one which build "inside edges" of the cube, from where distances were measured. The distance between cube edges and joints were set as not smaller than 40cm.



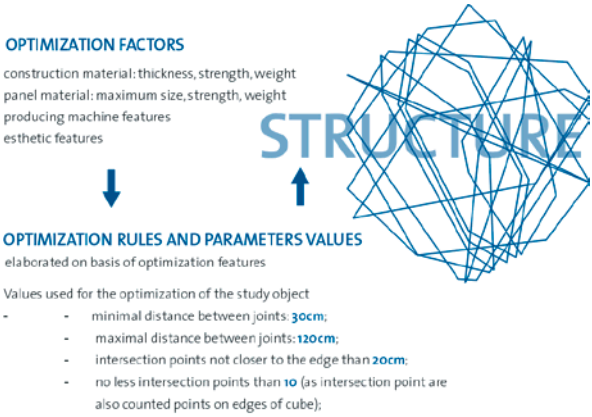




| 5.5 | the features analyzed during the optimization

The optimization as a process should be considered as a repetition of the analyses and structure changes which should be continued so long that at the end every fitness value of the set is 0. With Maya and MEL this condition should be taken only theoretically – because usually there were no possibility to carry on such a large number of repetition that the fitness function were even close to 0, because of the calculation interruption caused by memory lack. Yet the gradual decreasing of the fitness values was visible. The problem was also that there were no information exchange between the plane which was deleted from a structure and the new one which does not learn from the previous one. This caused the situation that the newly generated plane was also the worse in the next analysis and was deleted again.

Because there was no possibility to repeat optimization so long that it will be finished by itself (when the fitness values for every plane would be 0), there could be always the situation that exactly before the process was interrupted, a totally not proper plane was to be add to a structure. On the table 1 and 2 one of author's approaches to optimization is presented. It is also visible that it is not always like that the next cube is better than the previous one – optimization is not a linear process. It is highly possible that better combinations of the cube planes already appeared during the process, but they were changed by next generated planes.



| 5.6 | (left) general schema of optimization process

table.1 (next page) the example of an optimization process - fitness and text data for every cube

table 2. (pages 56, 57) the example of an optimization process with the starting cube the same as in table 1. example

randomly generated structure with 14 planes

fintessPlane -585 -712 -655 -517 -481 -589 -506 -633 -619 -627 -789 -537 -455 -615

$Aplane = {-0.6852115463,-0.7426512167,0.4971806603,-0.7844700563,0.6078703267,0.3035522522,0.5411090797,0.7513749198,0.3776949831,-0.8542221339,0.1253389532,0.8935463422,-0.7316335487,0.5166281294}
$Bplane = {-0.85547919,-0.5257354989,-0.2396018465,0.3761611482,-0.6142055264,0.5274820043,-0.6609640487,0.333143232,-0.2885789413,-0.07795086631,-0.5403049125,-0.8925908984,0.06617009698,0.4894187047};
$Cplane = {0.1140953176,-1,-0.6582924576,0.56206228,-0.9704520426,0.3035417907,0.4628582146,0.4264845831,0.2957041816,-0.9099377541,-0.5144283928,-0.5077448487,0.64859504,-1};
$Dplane = {0.590257952,1.343744045,-0.1698903816,0.3435490279,0.5576517578,-0.6709109629,-0.2504567825,-0.8574096835,-0.2731342375,0.9169581842,0.4725836603,0.1270492236,-0.05269316084,-0.3666677878}

structure after next 2100 repetition of optimization script

fintessPlane -333 -518 -324 -361 -400 -570 -473 -192 -415 -403 -414 -421 -560 -531 -227

$Aplane = {-0.6861576864,-0.8967815676,-0.8159411043,0.1303514828,-0.06723860033,0.3505639473,0.2962542798,0.7513749198,0.3776949831,0.110385696,0.06417568244,0.8001291268,-0.7316335487,0.3943025245,-0.1683425576};
print$Bplane = {-0.2348370697,-0.2310051497,0.2255431864,0.1187075187,-0.333748583,0.4734546185,0.9238132462,0.333143232,-0.2885789413,0.3373776439,0.1887016771,0.1081284548,0.06617009698,-0.8029128584,0.1705221879};
$Cplane = {-0.5787754168,0.5612787313,-1,-0.6881717248,0.8293387158,-1,0.07133735348,0.4264845831,0.2957041816,0.2486815412,0.9153831063,-0.5363352463,0.6485950437,-0.5043305723,-0.1753707825};
$Dplane = {0.397131201,0.1730708389,0.9104710663,-0.1458671241,0.05356957612,0.4764696504,-0.3031986495,-0.8574096835,-0.2731342375,-0.6002091555,-0.9955261385,-0.3709916745,-0.05269316084,0.4401986892,0.02952236604}

structure after next 3000 repetition of optimization script

fintessPlane -289 -375 -425 -407 -381 -265 -531 -632 -485 -398 -433 -395 -356 -482 -414

$Aplane = {-0.6840459908,-0.9169479293,-0.09319688926,-0.1826128985,-0.5487596555,-0.08220094687,0.2962542798,0.1024927641,0.3776949831,0.06417568244,-0.03658594423,-0.1683425576,0.3165069806,-0.9538141297,0.615628249};
$Bplane = {0.5337042875,-0.8163656739,-0.8516940395,-1,-0.00042230666636,-0.1073388793,0.9238132462,-0.07631474529,-0.2885789413,0.1887016771,-0.1463959257,0.1705221879,-1,0.143320981,-0.67433088};
$Cplane = {-0.5378307192,-0.5864130723,0.08146179449,-0.4671026336,0.2895879878,-1,0.07133735348,0.1664231234,0.2957041816,0.9153831063,0.3599932206,-0.1753707825,-0.1950670765,-0.1677668147,0.3012193158};
$Dplane = {0.3189677126,1.334977887,0.8234974581,0.9392271677,0.389845451,0.9007125179,-0.3031986495,-0.05191477092,-0.2731342375,-0.9955261385,0.01122602639,0.02952236604,0.08326195688,0.9305417502,-0.5017104259};

randomly generated structure with 14 planes

fintessPlane=
-585
-712
-655
-517
-481
-589
-506
-633
-619
-627
-789
-537
-455
-615

$Aplane = {-0.6852115463,-0.7426512167,0.4971806603,-0.7844700563,0.6078703267,0.3035522522,0.5411090797,0.7513749198,0.3776949831,-0.8542221339,0.1253389532,0.8935463422,-0.7316335487,0.5166281294}
$Bplane = {-0.85547919,-0.5257354989,-0.2396018465,0.3761611482,-0.6142055264,0.5274820043,-0.6609640487,0.333143232,-0.2885789413,-0.07795086631,-0.5403049125,-0.8925908984,0.06617009698,0.4894187047};
$Cplane = {0.1140953176,-1,-0.6582924576,0.56206228,-0.9704520426,0.3035417907,0.4628582146,0.4264845831,0.2957041816,-0.9099377541,-0.5144283928,-0.5077448487,0.64859504,-1};
$Dplane = {0.590257952,1.343744045,-0.1698903816,0.3435490279,0.5576517578,-0.6709109629,-0.2504567825,-0.8574096835,-0.2731342375,0.9169581842,0.4725836603,0.1270492236,-0.052693160843,-0.3666677878}

structure after 300 repetition of optimization script

fintessPlane=
-391
-364
-188
-346
-413
-276
-365
-499
-372
-177
-303
-361
-376
-323

$Aplane = {-0.6861576864,0.3505639473,-0.3514701599,0.1619538091,0.6078703267,0.3035522522,0.2962542798,0.4138325003,0.3776949831,0.06417568244,0.1024927641,0.8935463422,-0.7316335487,-0.5487596555};
$Bplane = {-0.2348370697,0.4734546185,0.6297872424,0.7529827407,-0.6142055264,0.5274820043,0.9238132462,-0.4049047607,-0.2885789413,0.1887016771,-0.07631474529,-0.8925908984,0.06617009698,-0.0004223066636};
$Cplane = {-0.5787754168,-1,0.4192413675,0.845269367,-0.9704520426,0.3035417907,0.07133735348,-0.2336354248,0.2957041816,0.9153831063,0.1664231234,-0.5077448487,0.6485950437,0.2895879878};
$Dplane = {0.397131201,0.4764696504,0.04683977208,-0.8703285824,0.5576517578,-0.6709109629,-0.3031986495,0.08726453298,-0.2731342375,-0.9955261385,-0.05191477092,0.1270492236,-0.052693160843,0.389845451}

structure after 600 repetition of optimization script

fintessPlane=
-370
-285
-190
-436
-383
-290
-310
-558
-462
-347
-274
-478
-362
-224

$Aplane = {-0.4798258258,0.3505639473,-0.3514701599,0.1619538091,0.6078703267,0.3035522522,0.5151925663,-0.05458522902,0.3776949831,0.9015372746,0.1024927641,-0.6161822505,-0.7316335487,-0.5487596555};
$Bplane = {-0.8158845759,0.4734546185,0.6297872424,0.7529827407,-0.6142055264,0.5274820043,-0.3090030091,-0.8468868293,-0.2885789413,-0.6997724918,-0.07631474529,0.3739638344,0.06617009698,-0.0004223066636};
$Cplane = {0.190850323,-1,0.4192413675,0.845269367,-0.9704520426,0.3035417907,-1,-1,0.2957041816,-0.3754748303,0.16642312340.8417530001,0.6485950437,0.2895879878};
$Dplane = {0.2006321673,0.4764696504,0.04683977208,-0.8703285824,0.5576517578,-0.6709109629,0.6503382267,0.8596720738,-0.2731342375,0.06890150668,-0.05191477092,-0.2847494045,-0.052693160843,0.389845451}

structure after next 900 repetition of optimization script

fintessPlane=-377 -272 -322 -359 -304 -371 -356 -326 -277 -368 -306 -331 -352 -171

$Aplane = {-0.8159411043,-0.5949862424,-0.3514701599,-0.1285593732,0.6078703267,-0.3006255828,0.5151925663,0.8001291268,0.07892000495,0.9015372746,0.1024927641,0.674950266,-0.7316335487,-0.5487596555};
$Bplane = {0.2255431864,-1,0.6297872424,-0.3498810007,-0.6142055264,-0.5119318716,-0.3090003091,0.1081284548,-0.07291287829,-0.6997724918,-0.07631474529,0.76280552,0.06617009698,-0.0004223066636};
$Cplane = {-1,-0.3724308659,0.4192413675,-0.2724308774,-0.9704520426,-0.542433487,-1,-0.5363352463,-0.111364774,-0.3754748303,0.1664231234,-0.593365255,0.6485950437,0.2895879878};
$Dplane = {0.9104710663,1.221026766,0.04683977208,0.2238777612,0.5576517578,0.578314573,0.6503382267,-0.3709916745,0.0951356767,0.06890150668,-0.05191477092,-0.5342073028,-0.052693160840.389845451}

structure after next 1200 repetition of optimization script

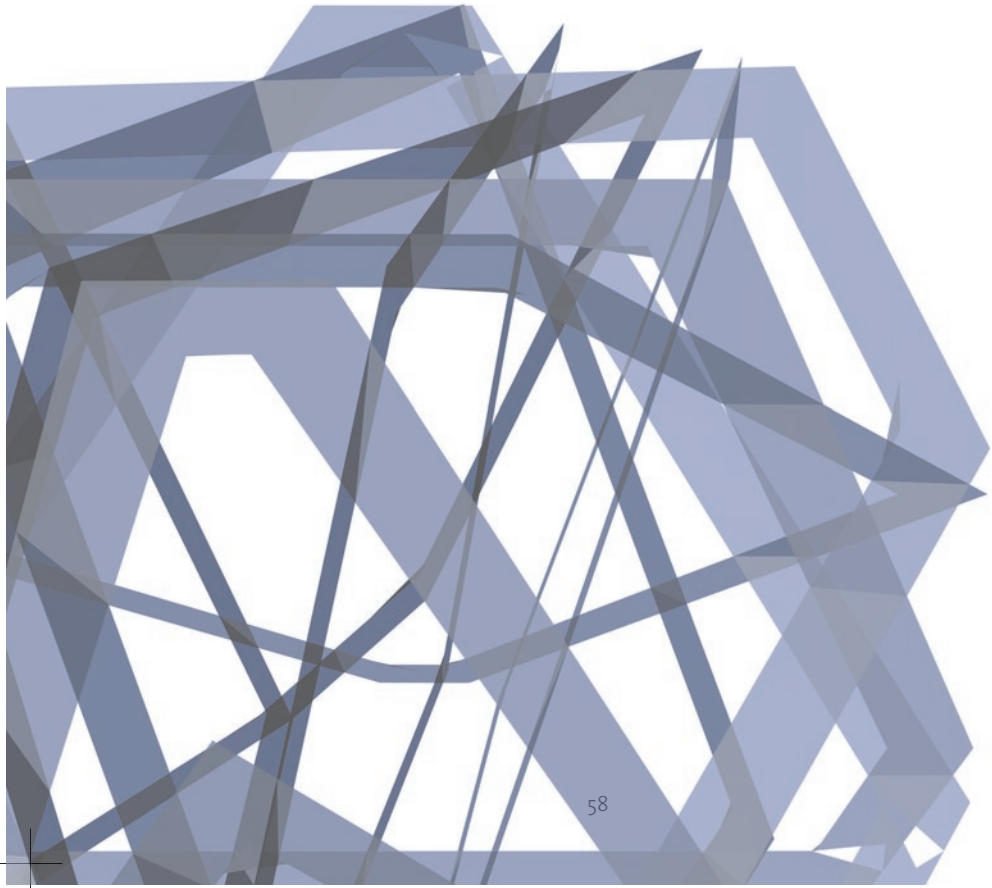
fintessPlane=-367 -319 -360 -382 -395 -369 -393 -392 -396 -264 -275 -446 -233 -311

$Aplane = {-0.8159411043,0.615628249,0.1652141596,0.829298455,0.6078703267,-0.3006255828,0.9094261355,0.8001291268,0.6706505671,0.5852755685,0.1024927641,0.5336387027,-0.7316335487,-0.5487596555}
$Bplane = {0.2255431864,-0.67433088,-0.2534244894,-0.8480783148,-0.6142055264,-0.5119318716,-0.6061760440,0.1081284548,0.0557615581,-0.7924900866,-0.07631474529,0.8949704495,0.06617009698,-0.0004223066636};$Cplane = {-1,0.3012193158,-0.6329559244,0.8384846621,-0.9704520426,-0.542433487,0.7714281778,-0.5363352463,-0.2798229614,-0.9522314486,0.1664231234,-0.8631508242,0.6485950437,0.2895879878};
$Dplane = {0.9104710663,-0.5017104259,0.5607597222,-0.1259884188,0.5576517578,0.578314573,-0.7015569465,-0.3709916745,-0.4829871394,0.427691448,0.05191477092,-0.1619903414,-0.0526316084,0.389845451}

structure after 3000 repetition of optimization script

fintessPlane=-368 -475 -423 -452 -308 -424 -506 -389 -207 -358 -464 -211 -308 -265

$Aplane = {-0.8159411043,0.316506980 6,0.7709898623,0.2760154585,0.6078703267,0.07255871173,-0.3965613290,0.1857705451,-0.9244773472,-0.09319688926,0.1024927641,-0.04166725558,-0.7316335487,-0.5487596555};
$Bplane = {0.2255431864,-1,0.5921964307,0.020566364 17,-0.6142055264,-0.3915411887,-0.3674600869,0.4975711007,-0.3495015359,-0.8516940395,-0.07631474529,-0.07520340767,0.06617009698,-0.0004223066636};
$Cplane = {-1,-0.1950670765,0.75516000609,-0.3156861987,-0.9704520426,0.4424282895,-1,-0.485234844,0.8858919962,0.08146179449,0.1664231234,0.310950727,0.6485950437,0.2895879878};
$Dplane = {0.9104710663,0.08326195688,-0.9439291718,0.2079855636,0.5576517578,0.1457535786,0.960069169,-0.0924340546,0.1660461502,0.8234974581,-0.05191477092,0.08565575975,-0.05269316084,0.389845451};

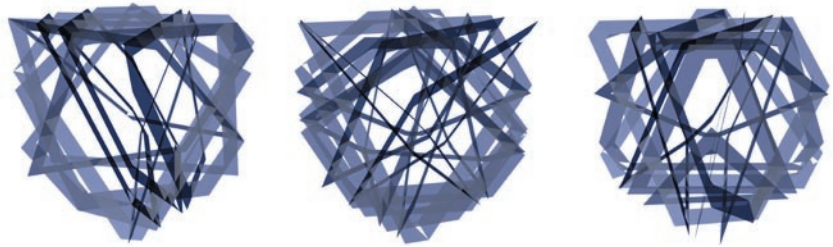


6. WORK FOR NDS04 STRUCTURE

All theoretical considerations and programming work described in this thesis had one main aim – to produce a buildable, wireframe structure for the NDS pavilion. This linear representation should be than used as a skeleton with real materials and it should be firstly visualized in the 3D space on the computer screen and then manufactured.

This kind of work seemed to be a real challenge – both in means of time-limit as also a responsibility and cooperation with other group members:

Michelangelo Ribaudo – who was responsible for production drawings;
If Ebnöther and Jörg Grabfelder – who worked on joints details and manufacturing process,
Anna Jach and Hanne Sommer – whose job was to design the cover structure;
and all other NDS students who also has an influence on the final project.



| 6.1 | three cubes - proposals for further work

The whole design was evaluating all the time during programming phase. These changes and suggestions also influenced GA fitness function or rules for generation algorithm, etc. so the cooperation was a crucial element of work.

Finally, three optimized structures were presenters to the group and some of assistants. One of the proposals was then chosen for further work and production. Some slight changes were needed because of group members' suggestions and ideas. Moreover till the last moment there was an idea of a big information panel incorporated to a structure. at the end, during evaluating the first big cartoon model (*fig. 6.2*), this idea was abandoned because the structure itself seemed to be spectacular enough, Finally, the 9-plane cube was manufactured and exhibited as a result of NDS group research and work(*fig. 6.3, 6.4*).

arrays for three cubes from the *fig. 6.1*

cube1 $Aplane= { -1.072001, 0.6078703267, 0.8001291268, 0.1024927641, -0.5949862424, -0.3006255828, 3.099236 , 0.137652, -0.7203, 1.338953, 1.012666, 2.242515 }; $Bplane= { 0.628609, -0.6142055264, 0.1081284548, -0.07631474529, -1.0, -0.5119318716, -1.794062 , -0.113439, 0.975319, 1.204105, 0.021208, 3.493845 }; $Cplane= { -0.9061, -0.9704520426, -0.5363352463, 0.1664231234, -0.3724308659, -0.5424334487, 3.507812, 0.197528, 1.171913, -0.690224, -0.612726, -1.059611 };

| 6.2 | (left) cardboard model

| 6.3 | | 6.4 | (next pages) NDS04 1:1 structure

$Dplane= { 1.005457, 0.5576517578, -0.3709916745, -0.05191477092, 1.221026766, 0.578314573, -2.48242, -0.085041, -0.461696, -1.17315, -0.193641, -1.66401 };

cube2 $Aplane= {-0.8464384105, -0.342384, -1.085698, 0.5461631397, -0.7931392819, -0.7075098671, 0.8867018132, 0.153747, 0.4651374461, 0.8292579081, 0.858724, 0.395076 }; $Bplane= { 0.5552307892, 1.054175, -0.896179, 0.4829963586, 0.4004663735, 0.8736476983, 0.4361461658, -0.950545, 0.3839430073, 0.8649583705, -1.140121, -0.981543}; $Cplane= { -0.8580282306, 0.939192, -0.914389, -0.2349151238, 0.4380900804, -0.9447640897, 0.236308113, -1.036641, 0.391744454, 0.7720183645, -0.246476, 1.199279 }; $Dplane= { 0.4899769628, -0.851391, 1.370001, -0.5304495668, 0.0374235896, 0.6483634222, -1.075752611, 1.036054, -0.6155449899, -1.011561354, 0.243194, -0.22325 }

cube3 $Aplane= {0.49879, -2.339699, -0.8967815676, -0.7844700563, 0.6078703267, 0.7513749198, 0.3776949831, 0.8935463422,

-0.1683425576, 2.408037, -0.680783}; $Bplane= {0.417268, -2.097016, -0.2310051497, 0.3761611482, -0.6142055264, 0.333143232, -0.2885789413, -0.8925908984, 0.1705221879, 2.685242, 0.585601}; $Cplane= {-0.138892, 1.087785, 0.5612787313, 0.56206228, -0.9704520426, 0.4264845831, 0.2957041816, -0.5077448487, -0.1753707825, 1.002112, 0.580152}; $Dplane= {-0.439046, 2.468681, 0.1730708389, 0.3435490279, 0.5576517578, -0.8574096835, -0.2731342375, 0.1270492236, 0.02952236604, -2.018526, 0.299873};

official cube array

$Aplane= {-0.8464384105, 0.595807, 0.5461631397, -0.7931392819, -0.7075098671, 0.153747, 0.4651374461, 0.858724, 0.395076 };
$Bplane= { 0.5552307892, -0.097311, 0.4829963586, 0.4004663735, 0.8736476983, -0.950545, 0.3839430073, -1.140121, -0.981543};
$Cplane= { -0.8580282306, -1.14713, -0.2349151238, 0.4380900804, -0.9447640897, -1.036641, 0.3917444454, -0.246476, 1.199279 };
$Dplane= { 0.48997696, 0.530634, -0.5304495668, 0.0374235896, 0.6483634222, 1.036054, -0.6155449899, 0.243194, -0.22325 };

7. ADDITIONAL TOOLS

7.1 designer's touch

During the generation and optimization process the interaction between the script/computer and the designer is possible. Not only has the designer set the parameters for these processes: he/she can also make some changes "by hand" – just tune the structure that it satisfies his /her esthetic expectation. Some tools were created to make this interaction easier. The first of them is printCubeArray procedure. In Maya arrays are printed always as lists of values, without any separators and brackets. But when we want to load values from array in Maya, we need the list separated by comas and in {} brackets. ***PrintCubeArray*** is a small but useful tool to print arrays that way that they can be immediately reused.

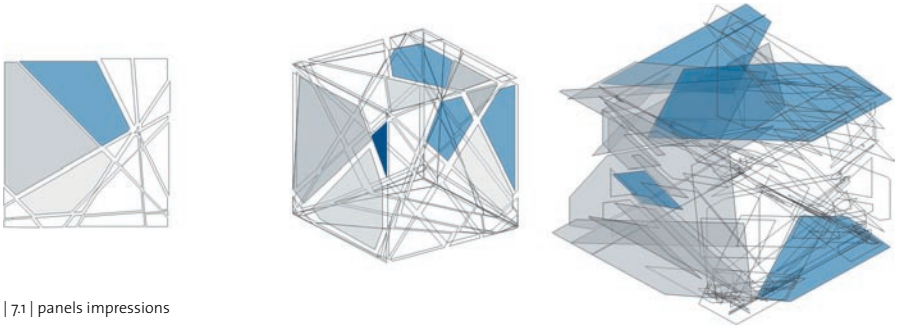
1 2 3 that is how Maya print values	 $array = {1,2,3} that is how the values then have to be loaded.

Next tool is a ***generatePlaneAndAdd*** procedure. After a cube is generated there is sometimes a need to add new plane to the structure. Adding a new plane – it means adding new values to the variable lists in Maya. ***GeneratePlaneAndAdd*** needs as parameters coordinates of three points located on a plane the user wants to add to the structure. After recalculation the coefficients of the new plane are added to the adequate variable lists.

Sometimes there is a need to take a look at fitness values of generated cube without optimizing it (fitness calculation is included in optimization script and normally is not printed – just analyzed and deleted from memory). The ***fitness*** procedure print fitness values for every plane.

7.2 Panels

The information about how intersection points (joints) are connected to each other – or in other words- how panels are located on cube's side is crucial for optimization process. But it can be also useful for production – if the spaces between structural elements will be filled by panels, the shapes of them are already calculated and they can be easily drawn. The output of ***drawingPanels*** procedure execution is 3d model of the cube made of panels which can be deformed or used as a basis for further work on production files.



| 7.1 | panels impressions

All .mel files needed form generation and optimization can be found on the CD ROM attached to this thesis.

8. FURTHER WORK

8.1 generation

While the programming work continued the new, interesting ideas and proposals for further work appeared. They dealed with the generation algorithms as well as with optimization. At the beginning of the work the biggest question was if to use MAYA features supported by MEL (Maya Embedded Language) or to approach in pure mathematical way where MAYA is used only for visualization and the output of the generating process are only points coordinates. Finally, the second approach was chosen because it seemed to be easier transferable to any other programming environment – and this migration was considered quite often (especially after it turned out that MAYA does not support well more complex mathematical calculations).

But of course one of the possibilities for further research is a comeback to a starting point of the NDS groupwork project idea and trying to work on the structure using MAYA features and functions. That means not generating planes by points, but simply by creation and rotation of NURBS planes, not looking for intersection points by calculation, but by MAYA intersection function and so on. Here also the other methods of optimization could be then tested – like springs (one of dynamic MAYA functions).

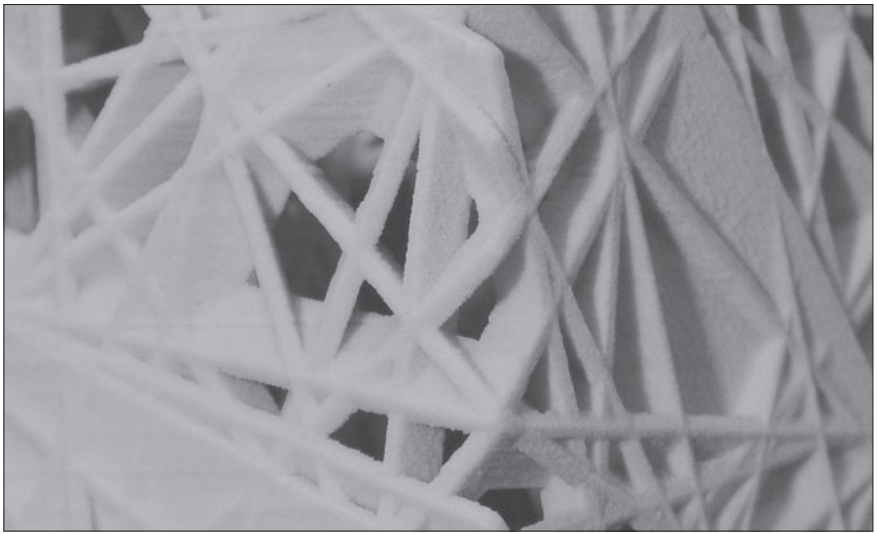
8.2 optimization and genetic algorithms

The optimization is a field which gives the biggest possibilities for further work. Firstly, because it is more complicated and difficult part of the work – so it can simple be improved. Secondly, the experiments with different fitness function settings, methods of reproduction, mutations, generation of new planes in evolutionary process could lead to spectacular and unexpected results. As well the whole implementation of genetic algorithms could be considered. Moreover, also simply new rules could be introduced into already designed optimization algorithm – like for example "opening optimization". This could be a tool which optimize the structure that in a chosen area there will be no joints (intersection points) so an opening will be created.

8.3 other forms - 3D vectors experiments

The fact that the outside and inside form of the NDS04 structure was a cube make some simplification in the programming possible. This means that it was quite easy to carry on all analyses in two dimensions – on planes form by cube's sides. Of course this method would not work for other forms – like blobs, toruses, cylinders – this method will not work.

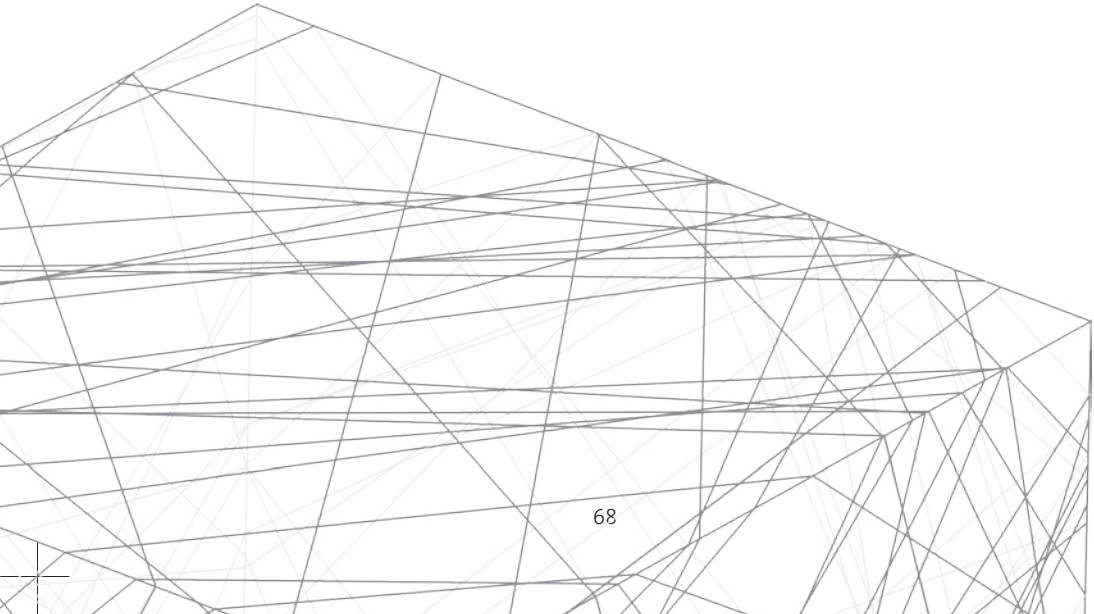
There are two ideas for a further work. Firstly, 3D vectors and Bezier curves could be used as the analyzed elements for optimization algorithms, so the calculation should be then done with their mathematical description. Other possibility is to prepare an approximated structure built by flat planes – for blob or cylindrical form. Then this structure can be analyzed similarly as it was done for a cube and then grids can be projected on the irregular surface. More specified analysis of the panels could also be introduced (as shape, area analysis).



9. CONCLUSIONS

On the whole, the main impression of the author, after this intensive two months of programming, learning about genetic and evolutionary algorithms as well as MEL is that the designed generation and optimization algorithms suggested only a great bunch of new possibilities for further research. To put it briefly – the new horizons were opened up, showing different ways to continue the work on generation (with other programming languages, other MAYA tools, on other forms) and on optimization (new algorithms – especially genetic ones, also new programming environments, experiments with fitness function). Also the chosen tool – MEL – did not come up to the author's expectation. Moreover, eventually it turned out that the generation and optimization could be calculated without any visualization need; output data of these processes – as a text file – can be easily load in any 3D modeling/visualization program. This information is particularly important for planning of further experiments because of Maya/MEL more complicated mathematical calculation are difficult to execute. But even after all simplification and approximation caused by time limit and problems with MEL the work described in this thesis resulted with a design for the structure which was then built by NDS04 students. And this should be considered as a main achievement.

| 8.1 | detal of a structure



APPENDIX 1

Maya and MEL

Because it is very important that there were a good exchange of all data between generation/ optimization scripts and production drawing scripts, the analysis and choice of the proper programming language was done by the author of this thesis together with Michelangelo Ribaudo, who was responsible for automated drawing production.
All scripts needed for generation, optimization and production of the NDS2004 study object were done in MEL (Maya Embedded Language). This scripting language allows a user to create procedures and scripts for custom modeling, animation, dynamics, and rendering tasks as well as customize Maya's interface. In this particular case it was used for calculation and visualization of a wireframe model of cube structure and than production of flat elements' drawings which were than used as a basis for milling and laser manufacturing drawings.

Initially as a scripting language for programming structure of NDS2004 we considered three scripting languages of three different 3D modeling or drafting applications:

VectorScript (for VectorWorks),

advantages: simple, popular in architecture offices, we know basis, possibility of customizing the user interface of the application;
disadvantages: no NURBS, slow 3D engine;

MAXScript (for 3Dstudio MAX)

advantages: 3D studioMAX is more popular than MAYA in architecture offices; possibility of customizing the user interface of the application ;
disadvantages: we have not known it yet – and time we had for programming was short;

MEL (Maya Embedded Language).

advantages: we have known it already, so programming could be started at once; possibility of customizing the user interface of the application; fast 3D modeling;
disadvantages: we did not know them yet;

Now, when the scripting is finished, longer and more detailed list of MEL advantages and disadvantages can be done:

advantages of using MEL:

- no need to compile programs: scripting languages commends can be execute immediately in Maya
- no need to exchange data: generation, optimization and production drawings can be calculated and prepared in one program
- fast visual feedback – the results of calculation can be easily check by a programmer on the screen, which is important in 3D vector calculations
- possible use of additional features of Maya – like springs – for next steps and experiments in optymization
- NURBS surfaces allow fast and accurate visualization of complex forms.

disadvantages of using MEL:

- Maya is not stable enough for executing long and complicated calculations, which were needed for optimization process
- some functions which could be useful doesn't exist in MEL (for example dynamic matrixes);

APPENDIX 2

list of procedures

GENERATION		
name	arguments	do what
makeABCDwithCounter	ABCD of a plane, counter	gives the ABCD calculated for a plane unique namesto do: connect this procedure with calculateABCDofPlane
makePointOnEdge	point number	generated randomly a point of an edge of a cube and gives it a unique name
makePointOnEdgesNEW	-	repeats three time makePointOnEdge procedure, which results with the set of points coordinates
angle_between_plane_ and_coordinate_system	ABCD of a plane	check the angle between a plane described by coefficients ABCD and three planes of coordinate system
generatingPlanes	number of planes	results with 4 arrays of coefficients for a given number of planes, the planes are already analised (see: ***GENERATION***)
CUBE VISUALISATION		
calculateABCofLine	(x1,y1) (x2,y2), number, counter, side number and width of the wall coordinates of 2 points and a number which is a counter for the plane	calculates the ABC of equation of a line Ax+By+C=0 created by 2 points and gives them unique names

intPointABC	ABC of two lines, counters of two lines	calculate intersection point between two lines and gives it unique name
drawCubeFromArray	4 arrays: lists of planes coefficients A, B, C, D	generate a 1/1/1cube with intersection lines on sides;
PANELS		
makingTwoPanelsWithLine	panel, ABC of line, nr of panel, amount of existing panels	calculate two new panels coordinates arrays – these panels appear after division the panel by line;
PanelDrawer1,...., PanelDrawer6	-	repeats the procedure makingTwoPanelsWithLine as long as all panels on a side are found
OPTIMIZATION		
edgesOfPanelFitness1,..., edgesOfPanelFitness6	panel arrayarrays for intersection points coordinatespanel nrside nrminimal distance to checkmaximum distance to check	checking if the edges of a given panels are between minimum and maximum proper value; if not, the procedure changes already the fitness values for planes which build the “bad” edges;
measureDistance	coordinates of two points	gives back the distance between two points
closeToEdge	the minimum distance between the joint and the edge of cube, number of analyzed intersection point, side nr, the width of the wall	checking if the intersection point (joint) is in a proper distance of the cube edge; if not, the procedure changes the fitness function for planes which build the “bad” points
calculateCubeFromArray	4 arrays: lists of planes coefficients A, B, C, Dwidth of the wall	recalculate all the values for a cube – intersection points etc.; generally, do the same as drawCubeFromArray procedure, but without any drawing

optimization		set of instructions, which result with the decision which plane is to be deleted because it is the worse one and with generation of a new one; the process repeated for example 100times can be called optimization
fitness		the same set of instructions as above, but without deleting the worse plane; results with a pront of fitness array values in text editor
TOOLS		
printCubeArray	array	print an array in script editor in such a syntax that it can be then reload to MAYA without any changes
generatePlaneAndAdd	coordinates of three points	add a new plane to an existing set
drawPanel1side, ..., drawPane61side	panel nr	draw a polyline around the panel and close it (so it can be than extrude etc.)
DRAW_PANELS		set of instructions where the drawPanel procedures are repeated so long that it results with the drawing of all panels for all sides

BIBLIOGRAPHY

Bull S., Downing S., (2004) Beijing water cube – the IT chellange, in *The structural engineer,* 2004

Kuo-Chung Wen, Wei-Lung Chen (2004) Application of Genetic Algorithms to Establish Flooding Evacuation Path Model in Metropolitan Area, *CAADRIA 2004 [Proceedings of the 9th International Conference on Computer Aided Architectural Design Research in Asia / ISBN 89-7141-648-3]* Seoul Korea 28-30 April 2004

Hamburg M., O'Reilly U. M. (2004) Evolutionary Computation and Artificial Life in Architecture: Exploring the Potential of Generative and Genetic Algorithms as Operative Design Tools, in *Emergence: Morphogenetic Design Strategies(Architectural Design)*, editors: Micheal Hensel, Mike Weinstock, Achim Menges, 2004

Radford, A. D. and Gero, J. S. (1988) Design by Optimization in Architecture and Building, Van Nostrand Reinhold, New York, 1988

Frei Otto in conversation with the Emergence and Design Research Group (2004),in *Emergence: Morphogenetic Design Strategies(Architectural Design)*, editors: Micheal Hensel, Mike Weinstock, Achim Menges, 2004

Weinstock M. (2004) , Morphogenesis and the Mathematics of Emergence, in *Emergence: Morphogenetic Design Strategies(Architectural Design)*, editors: Micheal Hensel, Mike Weinstock, Achim Menges, 2004

DeLanda, M. (2002), Deleuze and the Use of the Genetic Algorithm in Architecture , in *Contemporary Techniques in Architecture (Architectural Design)*, editor: Ali Rahim, 2002

Weinstock M., Menges A., Hansel M. (2004), Fit Fabric: Versatility Thought Redundancy and Differentiation, in *Emergence: Morphogenetic Design Strategies(Architectural Design)*, editors: Micheal Hensel, Mike Weinstock, Achim Menges, 2004

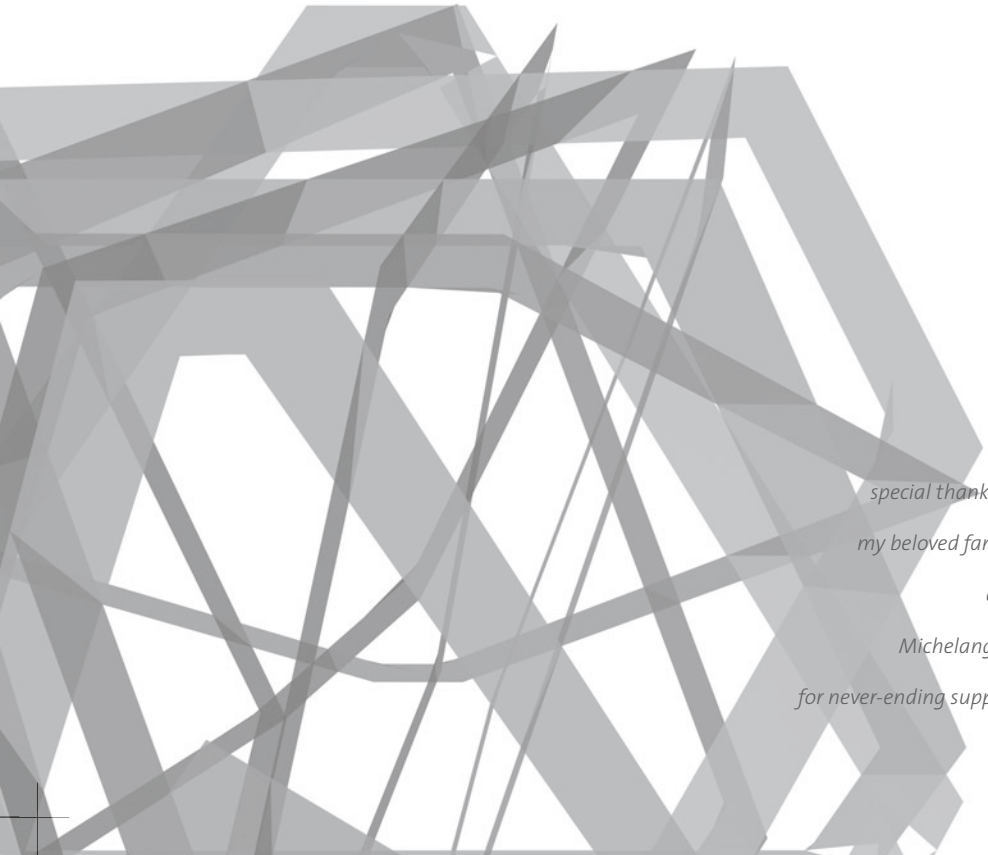
www.arup.com

FIGURES:

| 3.1 | | 3.2 | | 3.3 | | 3.6 | | 3.7 | | 3.8 | from: *Emergence: Morphogenetic Design Strategies(Architectural Design)*, editors: Micheal Hensel, Mike Weinstock, Achim Menges, 2004

| 3.4 | | 3.5 | from: Bull S., Downing S., (2004) Beijing water cube – the IT chellange, in *The structural engineer,* 2004

photos of rapid prototyping models and nds2004 structure: thanks to NDS04



special thanks to

my beloved family

and

Michelangelo

for never-ending support

Agnieszka Sowa
qleczka@intelcom.pl
+48 602 49 02 51
GG: 2222986
AIM: QllaAgu
MSN: qleczka@msn.com