Gennaro Senatore

MSc Dissertation
Emergent Technologies & Design
The Architectural Association of London

Tutors:
Michael Weinstock
Michael Hensel
Achim Menges

Contents

Acknowledgments

I would personally like to thank the **AA Architectural Association** and the **Center for Evolutionary Computing in Architecture CECA** at the **University of East London UEL** for supporting this work since its very beginning. I am particularly grateful to my tutors at the **AA** *Michael Weinstock, Michael Hensel, Achim Menghes* whose advices have been essential to the making of this research. Special thanks go to *Paul Coates and Christian Derix,* at the **CECA,** whose work paved the way for mine and constitutes the foundation upon which my work can stand.

Aim and Scope

The research here presented aims to design a methodology whereby the utilization of artificial intelligence techniques is systematically and rigorously applied within the architectural design process. The main justification for the development of such a methodology lies in the yet unsolved issues derived from the new performance-based approach to design. This approach seeks to integrate generation of forms and evaluation of their performances in order to design spatial configuration whose morphology is emergent rather than being super-imposed. This implies the generation of forms that are the result of a negotiation between their inherent topology and real design constraints. Although this philosophy has been applied by many architectural practices, there is an evident lack of an organic methodology that can allow the shift from evaluation of performances post-facto, to generation of forms by means of the evaluation of their performances. In layman terms, simulating the behaviour of an artefact under certain conditions has been used so far in a passive way after the artefact has already been designed. Apart from pure tensile surface structures, whose shape is not designed but is the result of a form-finding process, there are no others relevant examples. Very sophisticated computational tools have always been used for proving a good design idea and not as integral part of the process that brings to the formulation of the idea. Along with this issue, a performance-based approach to design requires the utilization of different simulation software for examining various performance aspects. Their integration, which is vital for the accuracy of the simulation and for avoiding continuous remodelling , requires the creation of a common system for the exchanging of data and a common framework [1].
It is within this context that artificial intelligence techniques find their main justification. The introduction of these problem solving methods might lead to create a decision support environment that can assist the designer over the architectural design process. The way these techniques has been investigated in this research, indeed, follows the idea of considering the design process objective oriented, where the objectives are defined by performances that have to be evolved. Stochastic methods such as Evolutionary Algorithms and Simulating Annealing, amongst the others artificial intelligence techniques, offer the possibility of integrating within one framework different performance simulations whereby a truly form finding process can be achieved. Their development begun two decades ago and their contribution have been relevant to the introduction of computational techniques for embracing complexity and trying to instrument its effect [1][2].
The other perspective for deploying this techniques, lies in the consideration that design optimization, using performance simulations, can be also an aid for stimulating the designer's creativity. Generating forms and having a simultaneous feedback on their behaviours, under real design constraints, would also help considering non conventional configurations of space [1][3]. Such a design evolution method would allow for an efficient exploration of alternatives while proposing solutions that are consistent with design constraints.
This body of research mainly refers to the work of Paul Coates and Christian Derix at University of East London, the work of John Frazer at the Architectural Association, the work of David E. Goldberg, the work of Peter Bentley and Uma O' Reilly , the intellectual framework first pioneered by John Holland and Richard Dawkins.
Reilly and Bentley developed an interacting software based on evolutionary algorithm and agents system called Agency Gp tool. EifForm, developed by Kristina Shea, is a software based on simulating annealing whereby it is possible to generate design topology and have the possibility of transforming them while maintaining a valid structural system. Gner8, developed by Martin Hemberg is an interacting software based on 3D map L-System whereby generating and evolving surfaces. Paul Coates has been working on Genetic Programming developing numerous applications amongst which the so called "Domino House". *Christian Derix* has been working on Neural Network and their application within the architectural design process. John Frazer, one the pioneer in the application of artificial intelligence in architecture, attempted to evolve the rules of 3d cellular automata using evolutionary techniques.
These are some of the relevant developments made in the field upon which this body of work can build its foundations.

Contribution

This research focuses its contribution on the development of a methodology where the integration of different performance-based simulation techniques, by means artificial intelligence techniques, gives birth to an active design space in which performance assessments move from evaluative to be generative. As already mentioned above, the lack of a common framework for the integration of those tools undermine the possibility of using them for generating forms rather than just evaluating them. This work aims to build the basis for the creation of such a framework in the attempt of reaching a higher level of integration within the design process.
It is worth saying that most of the performance-based software available at the moment require a high level of detailing, expert knowledge from users and are very expensive in terms of computational resources and time.
With regard to this issue, this work also aims to develop a lower resolution of some of them for increasing the possibility of their integration within one framework and their utilization at conceptual level.

Intro

We begin by presenting the description of some of the developed techniques of the field. Their utilization and further development, within the architectural design process, is focussed on the aid that they might provide when elaborating spatial configurations. Considering that the qualities of a space can not been fully understood until this space is used, the generation of spatial configurations has to undergo a reposition in the design process due to the limitations of deterministic approach. A space can be fully regard as a "complex system". This implies that the repercussion of design choices can be foreseen only to a certain extent using a conventional approach. The instrumenting of techniques that can embrace complexity can lead to a better understanding of the problem of emergence and its contextualisation within the architectural design process [4].

Cellular Automata

Cellular Automata are system based on cellular entities whose states depends on their previous state and on the one of their neighbours. This system performs complex outcomes by implementing simple rules that affect only local relations of their components. The system is usually described as a grid in 1,2 or 3 dimensions which might have any number of cells. Each cell has a neighbourhood which is constituted by a selection of finite number of other cells that affect its state. The rules are applied to the whole grid for each cell in the same way but go through the system only by means of the interaction between neighbours. First developed by Stainslaw Ulam and John von Neumann in the 1940s these system have been explored by many others amongst who Knorad Zuse and Stephen Wolfram. Probably the most famous CA (Cellular Automata) is the one developed by John Conway which was called Game of Life. Although the rules governing the state of the cells were elementary simple, the system presents an almost infinite amount of behaviours going from random to ordered patterns. My exploration of these systems focuses on 2 dimensional and 3 dimensional grids where the local rules affect the coordinates of nodes or the density of pixels of which they are made respectively.
Fig1 shows the final outcome for a 3D CA which starts from a grid of pixels. Every iteration each cells check in its 3D neighbourhood (the 27 cells surrounding it) if a certain fixed value of volumetric density has been reached. If the answer is negative it will create a smaller copy at a random position in its surroundings which will be subtracted to it by means of boolean operations. This repeats for each cell in the whole grid until the threshold has been reached. Once again a very simple rule produces impressive complex outcome(*fig1)*. Fig1 shows also the model obtained with 3D prototype technique which was sponsored by the *AEDAS* an international architectural practice.

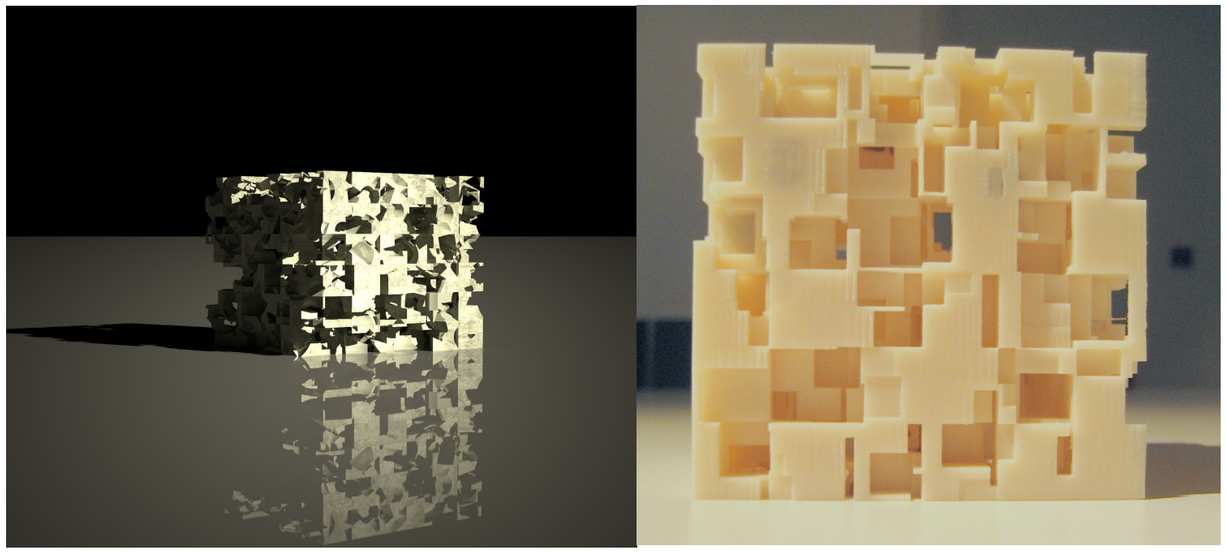


Fig1 3D CA render 3D prototype

Agent based System

Agents can be thought as small algorithms based on a set of rules whereby different reactions can be simulated when encountering different situations. There are two main type of agents : dumb agent ,and intelligent agent. The first one behaves according to the some rules and cannot modify them, the second one is able to learn from the environment in which is placed and infer decisions.
There are several type of agents which are classified according to the type of mechanism that drives their behaviour. This can be constituted by simple movement rules, trail formation or even physical particle properties.
One of the most interesting type of autonomous agents are what what is called "Boids", first conceived by Craig Reynolds. Every entity of this system behaves according to only three rules: cohesion within the swarm, alignment to the direction of flock members, and repulsion when another entity comes too close in order to avoid collision. *Fig3* shows the lines traced by the swarm during the simulation which truly resembles the flock of birds.
Another interesting type of agent is the one whose behaviour is governed by only simple rules of attraction and repulsion. There are some agent that can attract the others entities who can only repel each other. By tuning these simple rules according to the configuration of the system many interesting outcomes can be observed such as the formation of spontaneous Voronoi Diagram, both 2D and 3D, and the configuration shown in *fig4.*
Agent systems can be used for simulating complex people interactions such as pedestrian flow or crowds effects. *Alasdair Turner* at the **UCL** developed several applications using axial analysis and agents based system. *Paul Coates* and *Christian Derix* recently designed **SSSP** which stands for smart solution for spatial planning part of the Urban Buzz iniziative.

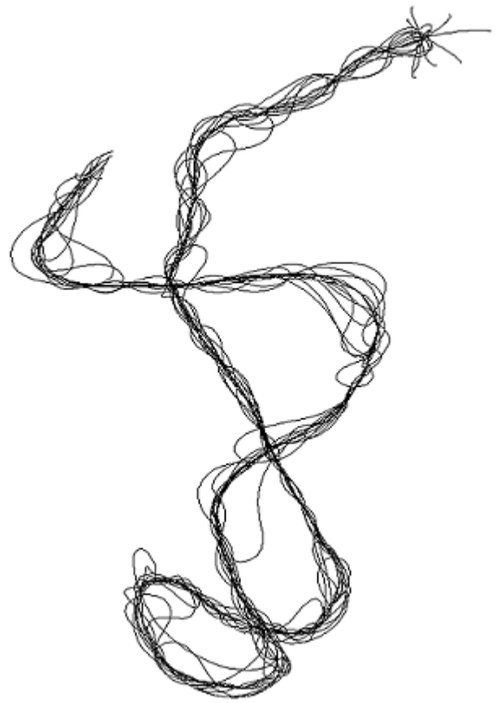


Fig2 Boids

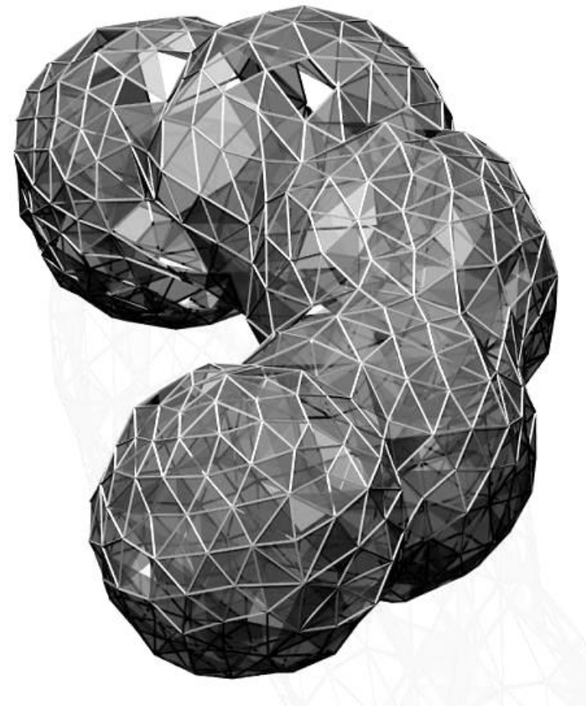


Fig3 attract and repel

Evolutionary Strategies

These strategies was first introduced by Rechenberg in1963 as optimization tools for aereodynamic wing design. They mimic Darwin's Theory of Evolution by evolving generations of configurations under the pressure of an environment.
They mainly use mutation and selection as search operators which are applied iteratively until termination criteria are met. At each iteration an entire set of configurations is generated and evaluated according to specified criteria. The best performing individuals (configurations) are selected and their genome, which is the set of information whereby they are represented, is mutated. The selection operator in these techniques is deterministic because is based on the fitness ranking and not on the fitness value of the individual. The fitness is in general a value that represent the performance of the configuration under certain design constraints. Every time an individual is selected, its genome , which in general is constituted by a vector of numbers, can mutate and only if the mutant has an higher fitness value it becomes the parent for next generation.
ES can tackle multi objectives optimization problems where there is not one best solution that can be achieved, but rather a set of optimum solutions, which is the reason why it operates evolving set of configurations.

Genetic Algorithm

The structure of this technique is mainly based on the same operators of Evolutionary Strategies with the difference that the generated configurations are not only selected and mutated but also recombined. This entails a more sophisticated encoding of the set of information that constitute their genome because thy have to be recombined in a coherent manner. Useful information have to be preserved in order to be transmitted to next generations. The operation of recombining the genome of selected individuals is called crossover. In addition the selection procedure is not deterministic but rather stochastic. There are several technique of selection which share the same principle of giving high probability to the fitter configurations to be chosen but also leaving a certain probability to the less fit ones. This is mainly due to the fact that some informations encoded in the less fit configurations can turn to be useful over generations.
Genetic Algorithm are more efficient in complex search spaces and have less problem in encountering local maximum respect to Evolution Strategies which is mainly due to the cross over operation.
We will examine in the details this technique, which constitutes the core of our system.

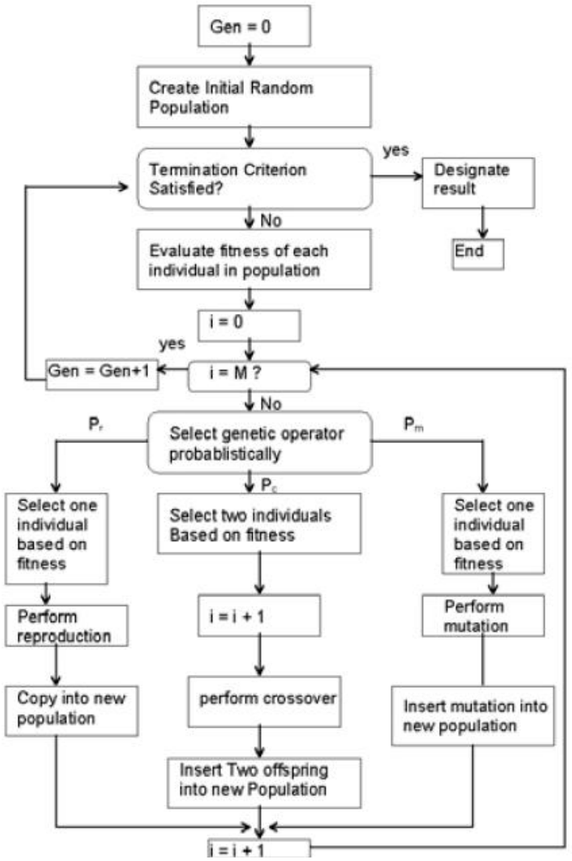


Fig1 Genetic Algorithm structure

Genetic Programming

One of the main limit of Genetic Algorithm is the lack of clear distinction between genotype and phenotype. The phenotype in GA is the direct decoding ,filtered by different procedures, of the information enclosed in the genome. This entails the impossibility for the developmental process, which is the set of rules or procedures that operate on the genome for producing the phenotype, to evolve. Genetic Programming might be the answer to overcome this problem. Although can be considered as a generalization of Genetic Algorithm, they work evolving instead that set of configurations, the procedures that generate the configurations. The individuals are usually represented as a tree structure where each node has an operator function and each terminal node has an operand. The tree structure replaces the concept of genome in Genetic Algorithm and it is constituted by the rules that define the developmental process of the configuration. In this way there is not direct encoding of it and the system can develop its own hierarchy autonomously. When cross over is performed for two selected individuals, one of the node is switched with another node from another individual replacing an entire branch *(fig1,* taken from the forthcoming book **Programming Architecture** written by **Paul Coates***).* It is easily imaginable how different can be individuals from one generation to the next one.

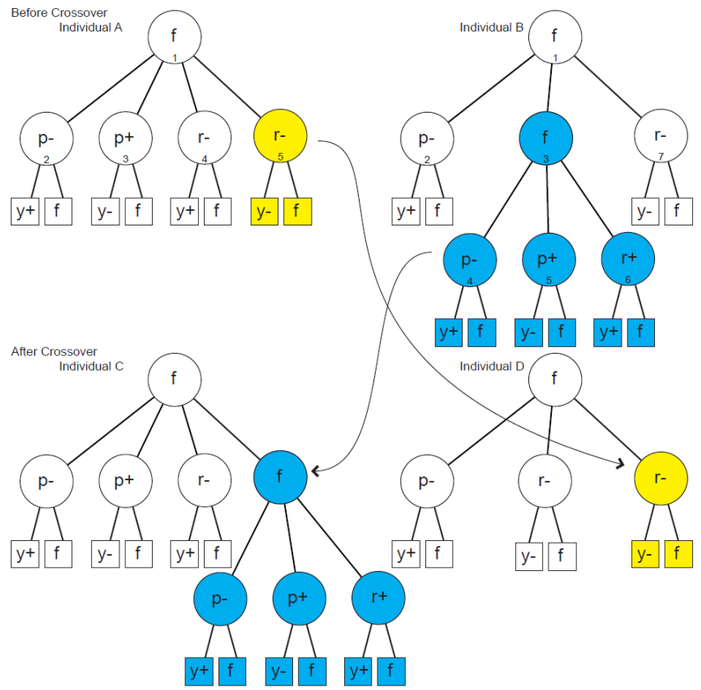


Fig1 crossover Genetic Programming

Artificial Neural Network

Artificial neural networks are computer systems based on a connectionist approach to computation. Simple nodes are connected together to form a network of nodes. Artificial neural networks are quite different from the brain in terms of structure. Like a brain, however, a neural net is a parallel collection of small and simple processing units. However in terms of scale a brain is massively larger than a neural network and the units used in a neural network are typically far simpler than neurons as well as the learning algorithms. A typical neural network consists of a set of nodes. Some of these are designated input nodes, some output nodes, and those which are neither are referred to as hidden nodes *(fig2)*. There will be connections between the neurons and a weight is associated with each connection. When the network is in operation, values will be applied to the input nodes; these are then passed through weights and a simple computation is performed in each node. These results are then passed through each node in turn until it reaches the output node (fig2).
Typically the weights in a neural network are set to small random values; this represents the network knowing nothing. As the training process proceeds, these weights will converge to values allowing them to perform a useful computation. When a neural net is first started, it is nothing but a set of input nodes, hidden nodes, and output nodes.
A node is just the term for one of the pseudo-neurons. An outside system (environmental sensors, or perhaps some other program) provides the input by placing values in the input nodes. By performing a set of calculations upon those nodes, the internal nodes are calculated, and then the output nodes.
Multi-layer perceptron networks use a variety of learning techniques, the most popular being back propagation. Here the output values are compared with the correct answer, and through various techniques the error is fed back through the network, which adjusts the calculation performed by each node to make it slightly closer to correct. It is provable that a multi-layer perceptron network (given sufficient nodes) is capable of learning any continuous real function to arbitrary accuracy. Neural Network can be also classified in two general categories according to the type of learning algorithm.
Supervised learning which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During the learning process global information may be required. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning. An important issue concerning supervised learning is the problem of error convergence, the minimization of error between the desired and computed unit values. The aim is to determine a set of weights which minimizes the error. One well-known method, which is common to many learning paradigms, is the least mean square (LMS) convergence.
Unsupervised learning uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. We will describe in details an unsupervised neural network SOM (Self Organising Map) in the discussion of the final experiment.

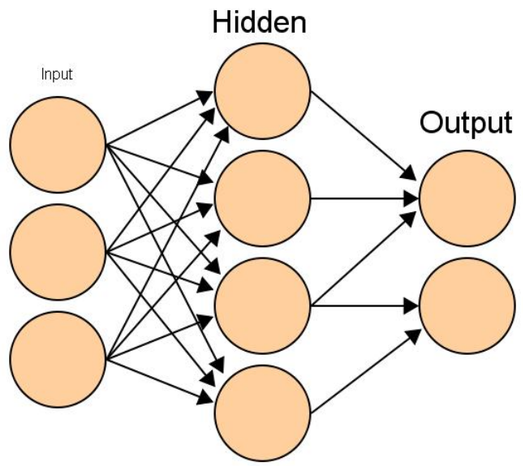


Fig2 Neural Network's structure

Abstract

The following part of this document presents the description of three experiments whereby we attempted to lay the foundations of our methodology. The diagram shown in next page represents, very synthetically, the main steps for our design system.

- *topology*

shifting from the design of spatial configurations to the design of systems that can generate them, we start defining the topology that we are interested to explore. The study of properties of the topology and the main features of what will be its environment lead to define the principles that guide the whole process (fitness criteria).

- *method of representation*

once topology and design constraints have been defined, we need to find a method that allow the abstract representation of the family of spatial configurations that belongs to the chosen topology. This can be done by building a system that operate on geometric operations, it can be based on others techniques such as Cellular Automata and Agent Based System or combination of them. This system, which can be regarded as the generative system, has to be encoded algorithmically in a formal language in order to generate considerable amount of configurations in reasonable amount of time. Out of the description of the method of representation come the definition of the design variables and their correspondent solution domain.

- *evolution*

the strategy that has been adopted for searching through the solution domain is Genetic Algorithm. Before designing the layout of such a technique, we have to translate the design variables in a grammar that can be understood by this "machine". The crucial part in this procedure is the definition, or better saying, the translation in the formal language of the design principles (fitness function). This is the way by which the generated configuration will be evaluated and return a feedback that has to guide their evolution. The designer has to customise this procedure according to type of architectural scenario in which he/she operates.

- *design brief*

the exhaustive exploration of the solution domain that belongs to the chosen family of spatial configuration lead to a set of possible solutions for an n-dimentional problem as the design process can be considered to be. The generation of forms by means of the evaluation of their behaviours under specified design constraints, can lead to consider patterns that we would have not come across if we had proceeded with traditional method. This is mainly due to the impossibility for us to design more than one layout at time and to explore all possible combinations of the variables. Navigating through the final outcome of such a system can help the designer in the formulation of a possible design brief.

The central part of my research is the practical implementation of the proposed design system where design has been mainly considered as an objective driven task. The first experiment shows the unfolding of the techniques necessary to govern and customise the Genetic Algorithm. The second experiment presents the implementation of the *method of representation* based on a system different from a traditional geometric definition in order to show other possible ways to define the developmental process. The third experiment is where I tried to deploy the possibilities offered by an evolutionary technique at most and to position the design system towards an architectural scenario.
Although encountering difficulties in handling the complex interactions between the design parameter and the variables involved in the process, these experiment demonstrate the possibility offered by such a system and allow to speculate to its further implementation.

Topology

Preliminary study of topology features

Material system

Site condition

Construction techniques

Define fitness criteria

t

Topology Representation

Geometric definition

Cellular Automata description

Agent based system description

Select key features for method of representation

Encode key features

Encode method of representation

Select design variables

r

Encode design variables in Genetic Algorithm grammar
Making genome for each individual in population

Choose the dimension of the domain to be explored

Preliminary set of weights for fitness values

Preliminary set of mutation rate

Decode geometric features – reconstruction of body plan

Generation(i)

yes

satisfied

no

i=i+1

Termination Criteria

Structural feedback

Environmental feedback
Wind influence
Sun influence

Spatial analysis
Geometrical analysis
Circulation analysis

Rescale weights for fitness

Evaluate fitness each individual in population

Chose method for Selection

Natural selection - roulette

Rank selection

Tournament selection

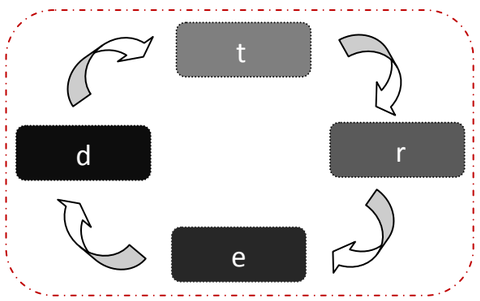
Artificial Neural Network

Selection

Crossover – encoding genome for next generation

Mutation

e



Aesthetic considerations

Representing the topology /Critical considerations

Fitness Criteria /Critical considerations

Termination criteria/ Critical considerations

Further analysis/development (Artificial Neural Networks)

Design Brief

d

t → topology r → representation e → evolution d → design brief

1st experiment

This discussion of this first experiment means to give a general understanding of the logic of the "machine" that governs the core of our proposed methodology. It can be regarded as a system in which spatial configurations are created and evolved under certain design constraints. The deep knowledge of the computational scheme that builds this machine is a fundamental requirement if one want to speculate on a possible design brief resulting from it. The experiment here reported, which was my first early attempt to engage with artificial intelligence techniques, illustrates step by step the basic procedures for the development of a Genetic Algorithm whose aim is to find a balanced solutions (often called optimum) to a problem which might have n-indipendent variables. In order to understand if a combination of the values of this variables constitutes a good or a bad solution of our problem, we could proceed in a step by step iteration of all possible combinations. However, even when the number of variables and the one of the values of each of them is relatively small, it would require a great amount of time. We will see, over the course of the three experiments, that if properly instrumented a Genetic Algorithm reduces to a reasonable amount the required time, while offering the opportunity to make an exhaustive exploration amongst all the possible combination of configurations.
We leave at later time (2nd and 3rd experiment) the explanation of the developed framework that surrounds and interacts with the "machine" and the unfolding of the strategies to contextualise our proposed methodology in an architectural scenario.

Encoding the body plan

As already said in the introduction, a genetic algorithm works evolving entire sets of configurations which are so called populations or generations. The members, that build up the generations, are what we call "individuals". Each individual is generated starting from a very compact kit of informations that, resembling the biological terminology, is named genome. The individuals can be regarded as the output of a system, the generative system, that receives as input their genome and produces their physical representation. There have been some works on the development of generative systems that allow the evolution of the developmental process, which is the set of laws whereby the phenotype is created. However, for the time being, this goes beyond the scope of our research.
Our generative system is made of two simple procedures. The first one in the encoding of the values, for the variables of our problem, in strings of binary numbers (0&1) that represent the genome of the individuals [2]. The second is the formulation of a set of rules whereby these information can be represented.
Fig3, next page, shows the body plan of an individual whose morphology is one of the possible representation of the topology that we are interested to explore. In this way the variables of our problem are the position of points whereby the geometry of the individuals can be described and their values constitute pieces of the genome.
The body plan has been obtained by drawing 4 points each section (*fig2,* first floor) and afterwards building triangulated panels between two consecutive floors *(fig2*, truss structure). If the height of the floors is fixed, it needs 2 numerical values each point. Therefore, if the number of floors is 10, the solution domain is a permutation of 80 independent values which is *factorial 80*. The size of the solution domain depends also by the range of values that each variables has. We can choose this range according to how vast we want the exploration to be and the available computational recourses. In this example I choose to encode these values in strings of length 5 which means that, as we will see in next paragraph, there are 31 possible representation for each gene. *Fig1* shows a piece of the genome that represents an entire section of the body plan of the individual.

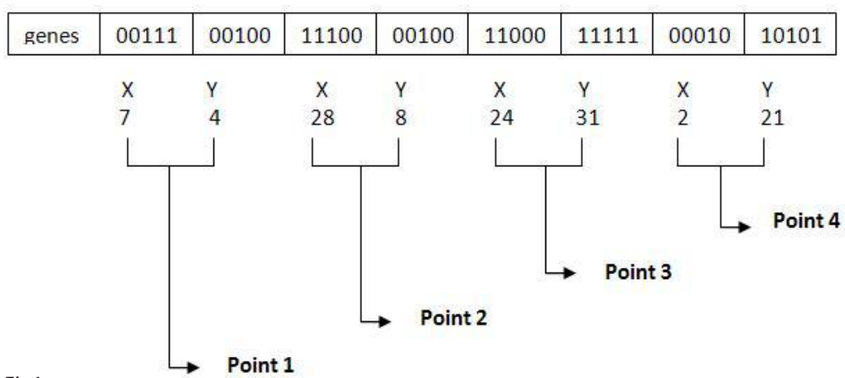


Fig1

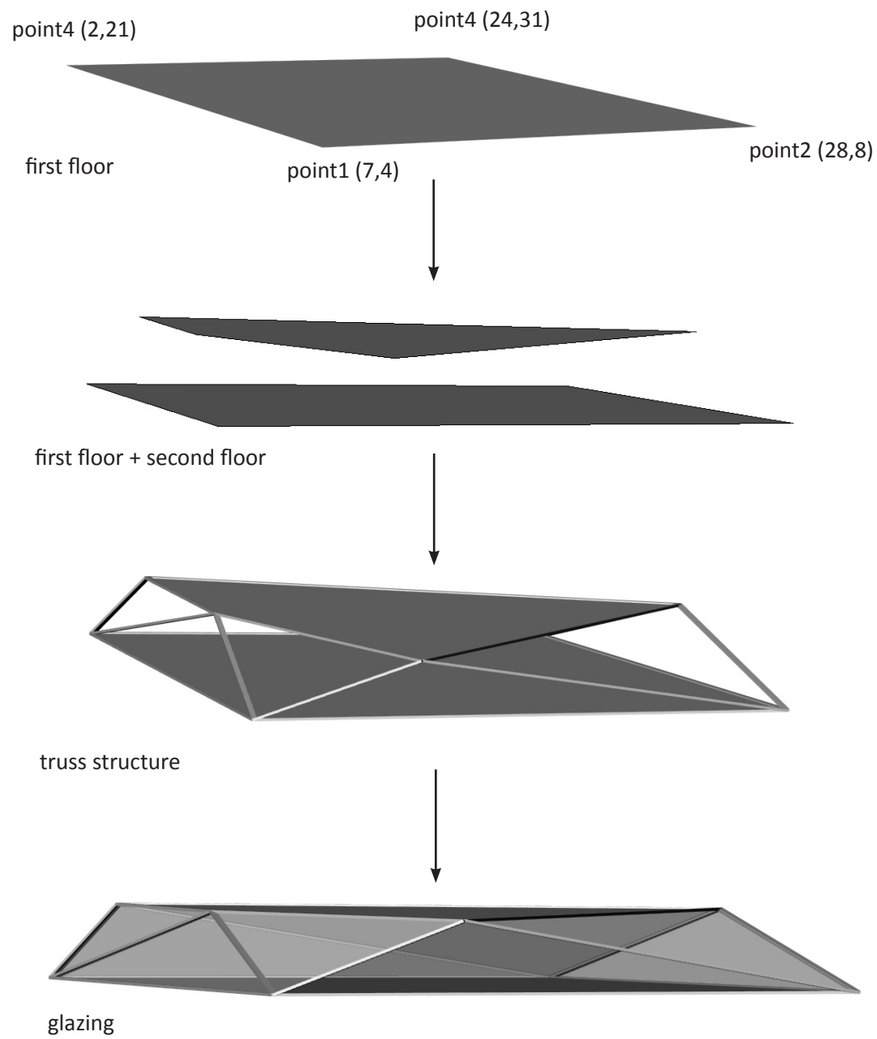


Fig2

Decoding the genome

The decoding system is simple and can be explained in three steps:

- once the size of the solution domain has been decided, strings of binary number in random order are generated. This is the first step where the first generation is initialised.

- according to the size of the solution domain a string representing one of the gene can have a certain length. We start it reading it from left and multiply each number by 2 *(fig2)*. According to the position in the string, 2 is raised at a certain power starting from 0 for the first position (right end) and ending with a number which is equal to the length of the string minus 1 (left end). The sum of this products represents the value of the variables which in this case are coordinates (x,y,z) and 3D points respectively [2][2].

- once all the genes have been decoded they are ready to be used in order to build the virtual representation of the individual. *Fig2*, previous page, shown the sequence of building for one of the floor/section. The main justification for the encoding-decoding procedure lies in the advantage that strings of binary number offer when performing cross-over between the genome of the selected configurations (see page 13).

The range of values that it is possible to decode depends on the length of the strings. If the string representing the coordinate of the points is made of 5 bits, the maximum value is 31; if the length of the string was 6, then the maximum value would be 63 and so forward.

Fig2

The whole genome

Considering that in this example each gene (string of 0&1) is made of 5 bits and there are 80 genes per individual, the number of possible permutations is (80*5) factorial [3]. This number can be considered as infinite as it would be the time that such a big permutation would require if it were done in a step by step iteration. Genetic algorithms instead operate searching the solution domain by quickly discarding configurations that do not respect specified design criteria. Successively, the genome of the selected individuals are manipulated by breeding and mutation in order to reach better configuration [3][2]. It is evident that the definition of design constraints and the development of a system, whereby these morphologies can be evaluated, is crucial for driving the evolution. *Fig1* shows the entire genome that encloses the necessary information for reproducing the body plan of the individuals.

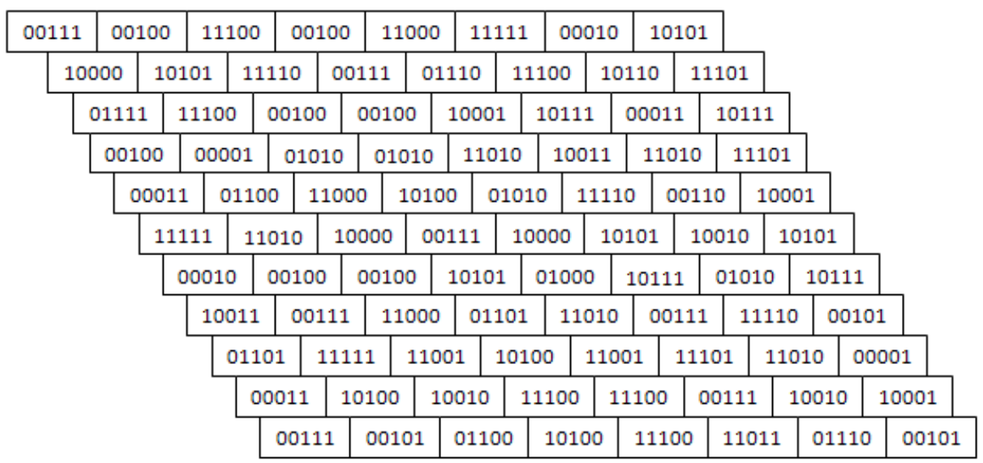


Fig1

decoding

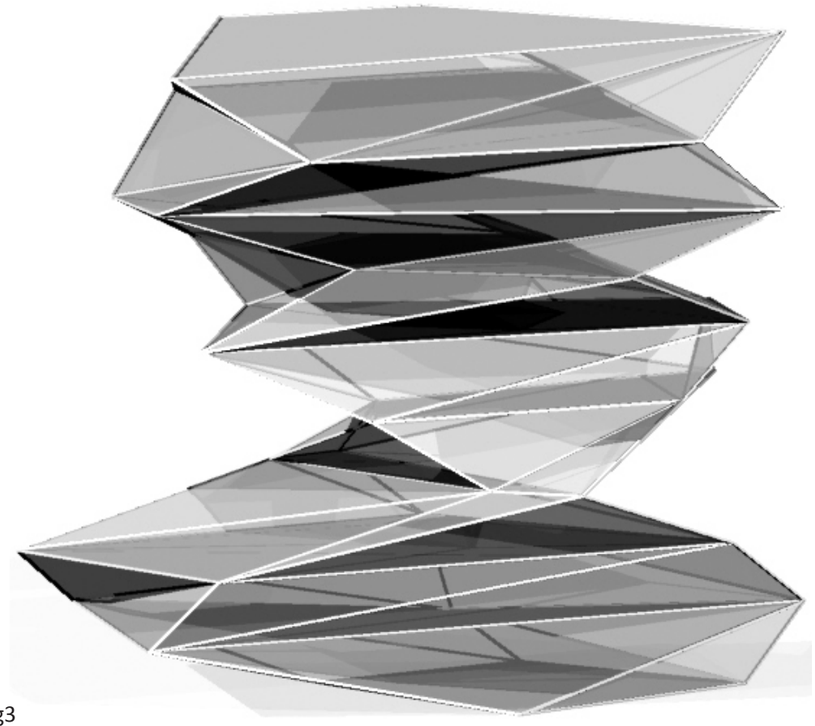


Fig3

Fitness Criteria

The fitness function evaluates the individuals of the population and assigns a numerical value according to the objectives to be optimised. When evolving spatial configurations there are many criteria that can be used for influencing their development: structural, spatial organization, daylight exposure, to mention a few.
Those are the fitness criteria that will guide the algorithm through the searching of spatial configurations which respond positively to their evaluation. By continuously testing the individuals over generations, the fitness criteria filter the information that are useful for the individuals to behave according to the environment in which they are placed. It is an indirect control on their morphology as well as abstract is the representation of their topology that we encoded. The way the fitness criteria are implemented, in order to combine their values and make the individual's fitness, is called fitness function. This can be regarded as the representation of the "environment" that we referred before. In this experiment we drive the evolution taking into account only one criteria but there may be several of these. When having more than one parameter, it is clear that they have to be comparable in order to build a value that represent the fitness of the individual as a whole. In general, most of the times, fitness parameter have different dimension and magnitude and, therefore, they need to be scaled and weighted. We will see in the 2nd and 3rd experiment, where the number of fitness criteria goes up to seven, how these procedures can be embedded in the fitness function, along with several techniques that can be deployed for implementing it.

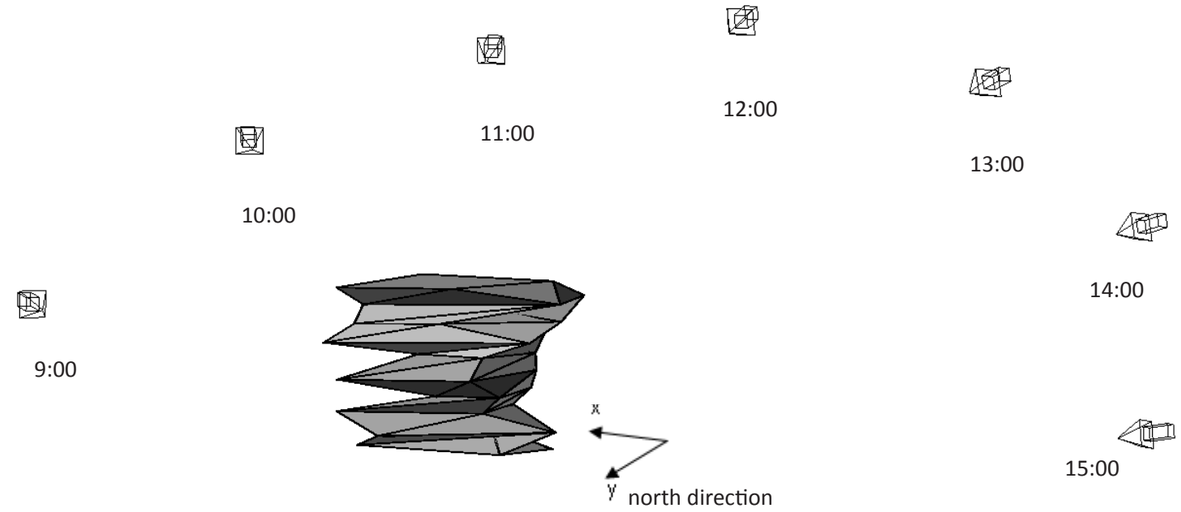


Fig1 north facing wall

sun path 21st December 51.4879° Latitude -0.178° Longitude, London

Solar gain

The criteria of evaluation for the configuration of this experiment is the maximization of the interception of daylight.
For performing this kind of analysis there are many available commercial software, such as Ecotect, but considering the number of simulations that should be done each generation, this would require an amount of time that does not match our recourses. In addition, using an external software adds another layer of complexity to the implementation of the procedure which has been made in a single computational environment. The computational schemes that I developed are supported only by Cad packages which are Autocad for this experiment and Rhino3D for 2nd and 3rd . This implication entails limitations in the accuracy of the analysis that we can perform, which are mainly based on vector calculation. However, the aim of our work implies the utilisation of such performance evaluation tools at a conceptual level, integrated in a common decision support environment. From this point of view, a low resolution of these tools serves the scope of our research.
The aim of this procedure is to produce a fitness value related to solar gain. For doing so, site conditions have to be simulated. The north is represented by the y-axis of the world coordinate system *(fig1)* in order to orient our individuals respect to sun. In this way, taking the vectors that represent the direction of the sun at specified hours and for a specific day, it is possible to understand how the south facing wall should be oriented for maximizing solar gain. *Fig1* shows the north facing wall of one of the configurations and the path that the sun describes on the 21st of December from 9:00 to 15:00 for latitude and longitude of London.
The chosen day is the 21st of December because we want to maximize solar gain during the winter and in the shortest day. The angle *alfa (fig3)* that each sun direction makes with the normal vector to each panel can be used for analysing the exposure of the individuals to sun. By averaging the sum of this angle for each panel for the number of sun directions, we have a mean value of the degree of the exposure of each face to sun. The sum of such a value for all the faces of the mesh, returns the degree of exposure of the individual for the whole day. It goes without mentioning that this value is the fitness parameter.
The same angle can be used for visualising the map of the exposure to sun by simply playing on the RGB value for each mesh face. In this case I use only red and blue value that gives a gradation of 255 tones. As can be seen in the *fig4* when the *alfa* is close to 0 degree it means that sun direction and normal to face lie parallel and have opposite versus (maximum exposure). When they lie orthogonal alfa is 90 degree (no exposure). For values of angles that are bigger than 90 degree the faces are shadowed.
The detailed explanation of the procedure can be found in 3rd experiment in the paragraph "solar gain".

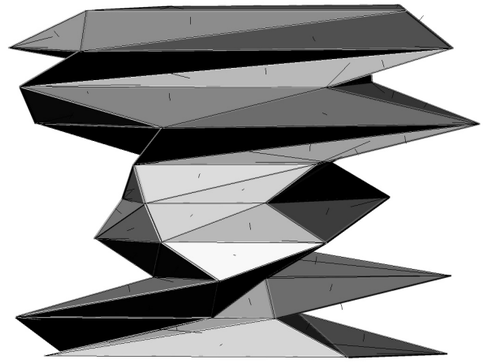


Fig2

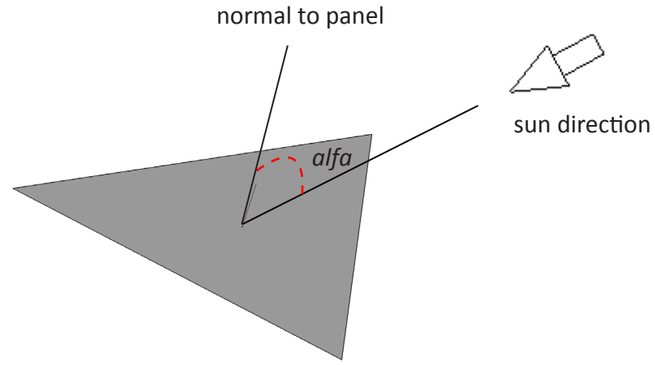


Fig3 mesh face

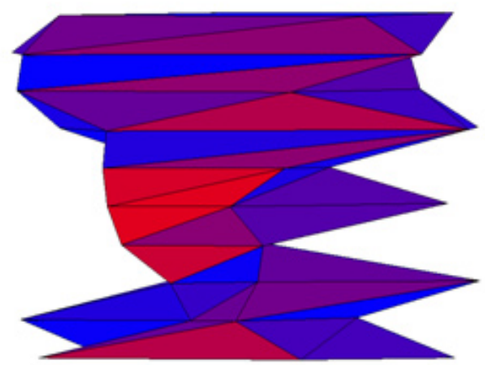
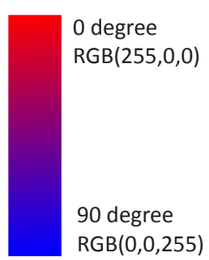


Fig4 south facing wall



solar gain map

Selection

The effectiveness of a genetic algorithm depends on the ability of preserving useful information (genes) while discovering others good qualities by mean of breeding and mutation over generations. The preservation of good traits entails a method of selection that is able to discern to which extent an information is good or not. If the choice of the best performing configuration might seams reasonable, it might lead to too fast convergence without neither reaching an optimum set of configuration nor using the potential of the system. A more balanced criteria of selection between performance and heterogeneity gives the possibility to make a full exploration of the solution domain reaching better results.
There are several techniques that can be used for implementing the selection procedure that can be divided in:

- **roulette selection**

this strategy provides a method of selection where the probability for an individual to be chosen is proportional to its fitness value [2][3]. It gives a high probability to fitter individuals but also leaves a certain possibility to the less fit. The method is represented in *fig1*[*] where the grey rectangles, at the left, represent fitness values of individuals and the long rectangle at the right is the sum of these ones. A random fraction of this sum is taken, acting as a threshold, and the sum of the fitness values, until the threshold is reached, is performed again. At the last values added corresponds the configuration to be selected, favouring in this way individuals that have bigger fitness values (the threshold is reached faster by adding big values) than less fit ones but leaving a certain probability also for them. This is the strategy that i adopt in the system and it is explained in details in the last part of this chapter "pseudo code".

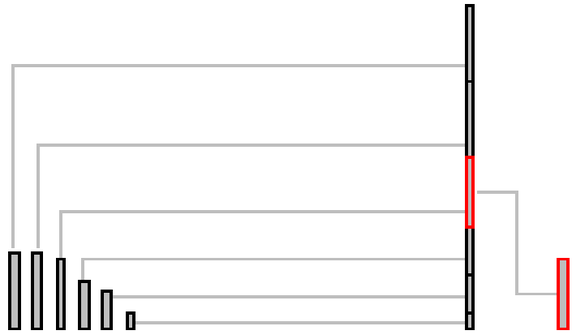


Fig1 Roulette Selection

- **rank selection**

rank selection is not based on the actual distribution of fitness but rather on fixed range of values that determine the classification of the individuals. If the fitness value for a configuration lies in between the limits of a rank it will be assigned the fixed score for that rank *(fig2).* This method ensures the tendency towards the better members but does not allow the discernment of small similarities amongst the individuals in terms of fitness.

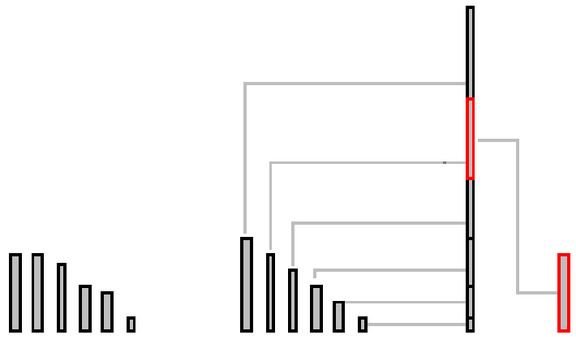


Fig2 Rank Selection

- **tournament selection**

this strategy combines the random selection and performance based evaluation. First there is a tournament of individuals that are selected randomly, amongst which the best performing is selected *(fig3).*

- **elitism**

in contrast to the previous strategy, elitism entails the copying of a certain number of best performing members into the new generation. In this way good qualities will not be lost through mutation and cross over (breeding). These members remain unaltered until better performing individuals have been found acting as a sort of source material. The problem of this method is that it leads to too fast convergence and high risk of local maximum. A local maximum is a condition in which the genetic algorithm reaches a set of configuration that is better than the previous one but is not the optimum possible.

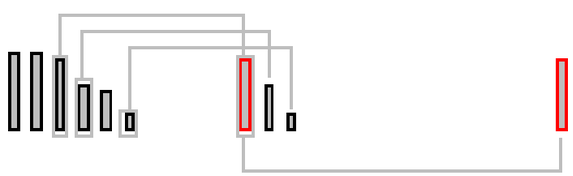


Fig3 Tournament Selection

[*] *fig1, fig2, fig3* are taken by the unpublished thesis "*Design code*" written by Kramer and Kunze [4][2].

Breeding|Cross over

After being evaluated and selected the members are ready to breed. The breading procedure is done by crossover their genome. The selection proceeds by selecting and pairing two individuals at time whose genome are split in a random position and swapped over. Eventually two new configurations are created [5].
This method of manipulating the good qualities of selected configurations works because it allows to transmit good genetic heredity to next generations [6] . The splitting of a genome can be done in a random position, in this way each time new genes are created when recombining them *(*3rd step *fig1)* [2] [4]. If the morphology of the configuration requires coherence they can be split respecting the length of the genes (length of the strings).

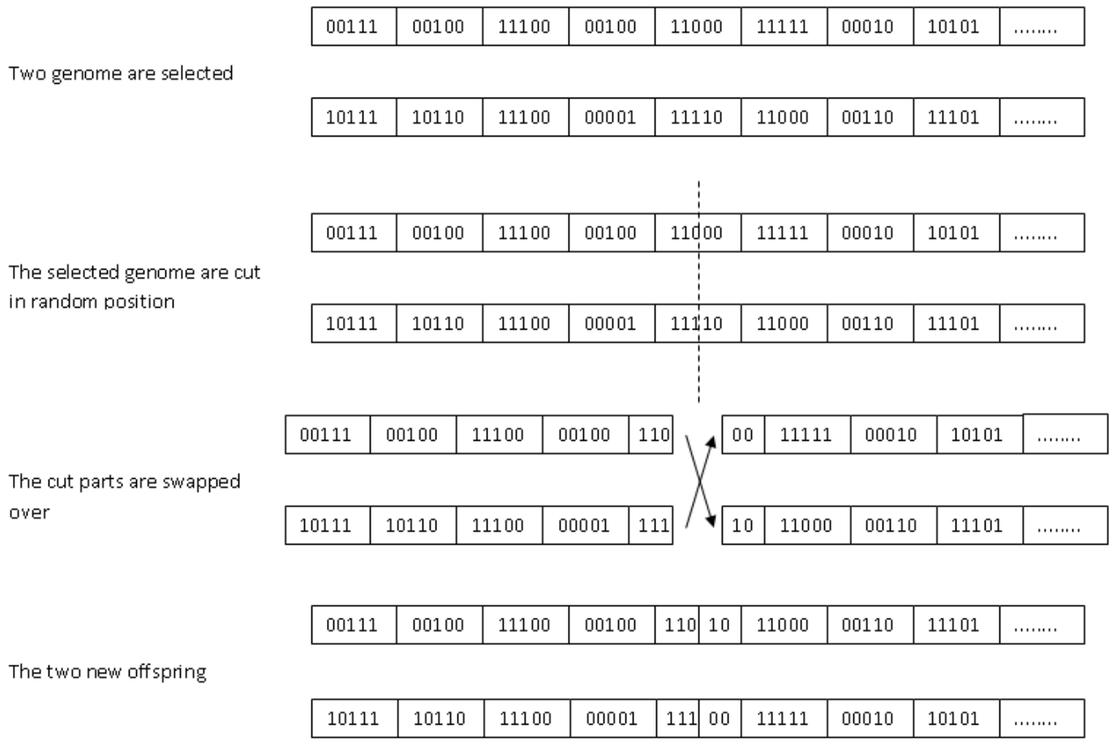


Fig1

Mutation

The possibility of having a certain degree of mutation is given by simply flipping one or more bits chosen at random positions within a genome. Usually it is preferable to have high mutation rate at the outset of the searching for ensuring maximum exploration of the solution domain [2][5]. The rate can be set to decrease over generation or to respond dynamically to the trend of the fitness (see 3rd experiment “pseudo code”, set mutation rate)

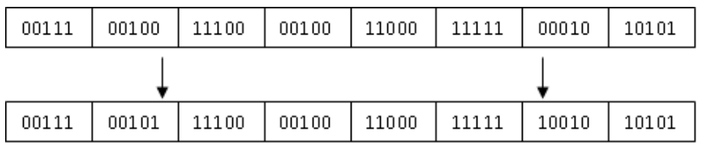
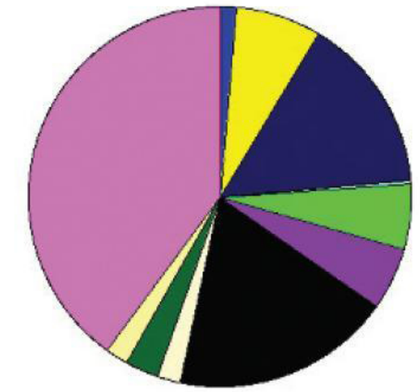


Fig2



Goldberg’s weighted roulette wheel - the size of the segment is proportional to the fitness of an individual.

Evolution

The steps that have been explained in the previous pages repeat in a loop until termination criteria have been satisfied (see 3rd experiment, driving the evolution). Termination criteria can be simply constituted by the fact that there is no further improvement after a certain number of generation has been reached. Usually when monitoring the trend of the fitness, there is a step increase until a maximum which is followed by a decrease that lasts only for few generations until it stabilises around a certain value. Other ways for implementing termination criteria is to drive the evolution until the performance of the individuals is in a certain range from a target value or distribution of values.
The chart in *fig1* shows the trend of the mean individuals' fitness value (generation fitness) for each generation. It has an upward trend from the initial minimal value to its maximum at generation 50. The reason why this trend does not conform to the typical trend observed in many other application lies in the size of the chosen solution domain. Its vast dimension would have required a bigger number of generation for reaching a stable solution instead than 50. This gives me the opportunity to set the termination criteria in a different way for next experiments. Instead than setting a fix number of generation after which the algorithm should stop, I will monitored the trend of the fitness setting the termination of the procedure when there is no more appreciable difference between a given number of generations fitness values.
The chart in *fig2* shows the comparison between the fitness values of individuals of generation 1 and the ones of generation 50.It is clearly visible that the values of the 50th generation are constantly higher than the ones of the 1st generation.
The reason why generations rather than single individuals have been evolved lies in the fact that in multi-objective optimization process there is not one solution to the problem but a set of possible solutions. At certain point in the process the individuals start to converge towards similar morphologies but even at the very last generation a good degree of variety is present. With regard to this experiment, the individuals slowly modify the position of the points that describe their geometry, keeping their topological relations, in order to catch the more daylight is possible *(fig3).* Considering the fact that two individuals might perform the same but have different morphology, the designer is able to explore a very large set of transitions. These are, together with the possibility of exploring very large set of configurations and evolving more than one fitness parameter at the same time, the main justification for using this method in a design process.

Critical observations

In this first early experiment I decided to cope with only one parameter to be evolved. This application was meant to be an exercise to engage with the logic of genetic algorithm. In next experiments I will tackle the task of evolving more than one fitness parameter in parallel process, which is what Genetic Algorithm has been invented for, trying to instrument the influence of those on the morphologies of the individuals.
Although its incompleteness, this first exercise gave me the opportunity to set the core procedure of the design methodology. Next experiments will be focused on different way of encoding the body plan and on the design of the fitness function which is the part where the designer interacts more.

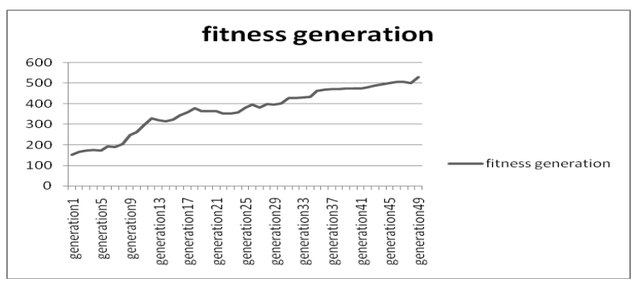


Fig1

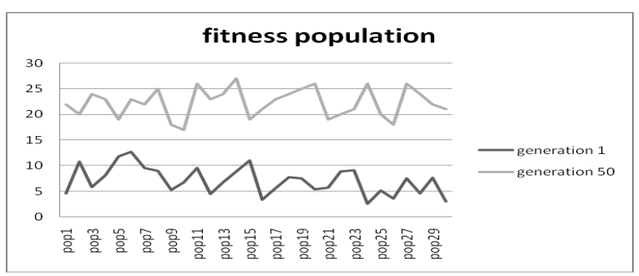
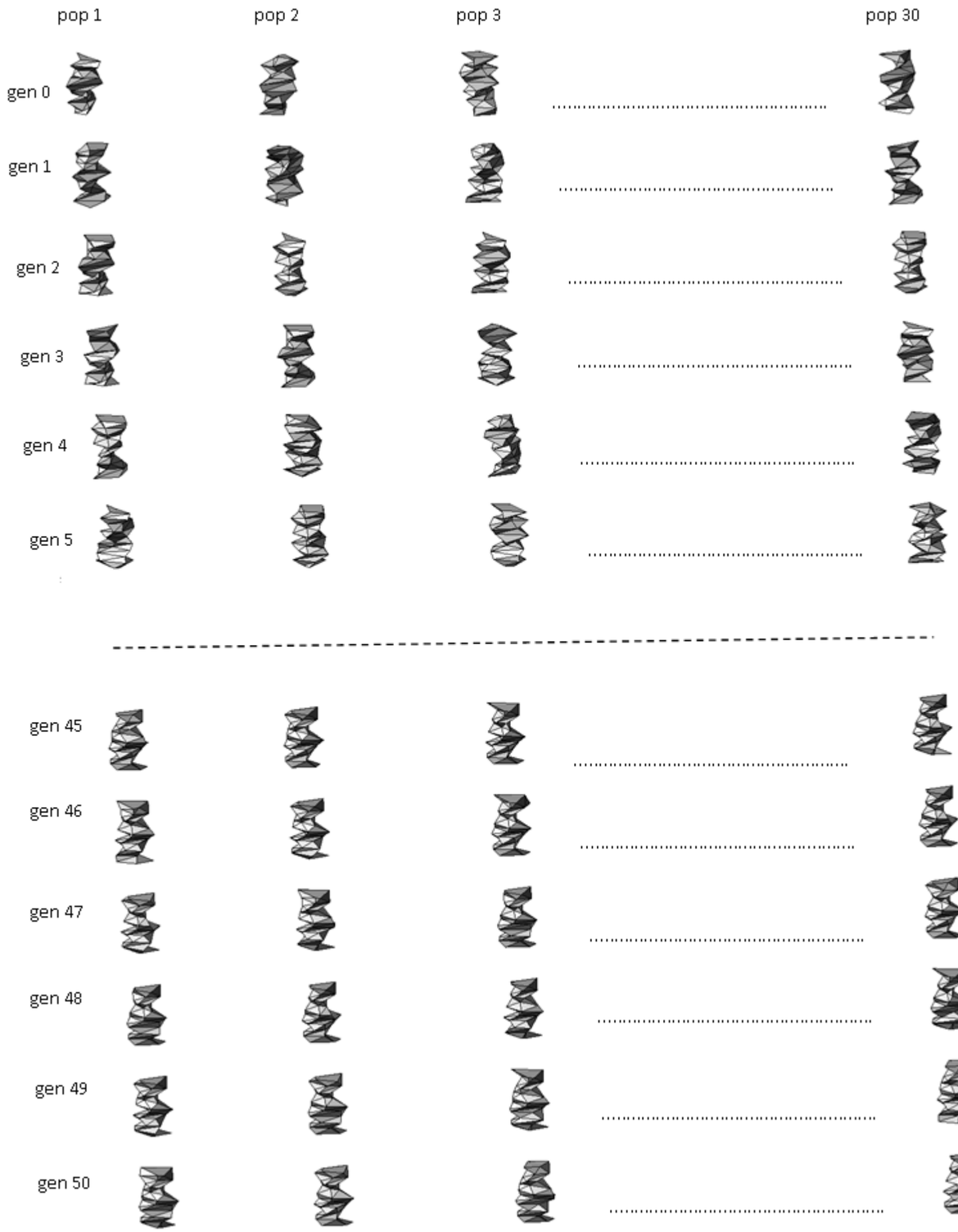


Fig2



Genetic Algorithm|pseudo code

In this paragraph I describe the main sub-routines that build Genetic Algorithm. We will refer to these procedure many times over the description of next experiments which is why I think it is worth to give same details. The general pseudo code for this technique is shown in *fig1* and its detailed explanation can be found in 3rd experiment|"pseudo-code" where we also describe how performance based evaluation tools can be integrated in the procedure.
The main steps in a Genetic Algorithm are :

- *making the genome*
strings of binary number are randomly generated according to the variable that they have to represent and the size of the chosen solution domain [2][6]. The "" in the code *(line 528 to 540)* stand for strings variable in Vbscript or VBA language.

- *decoding*
this is the translation in Vbscript of the mathematical procedure that we previously explained in this chapter. The way it works is similar to what we would do by hand, working backwards through the string. The name "words" *(line 542 to 568)* stands for variables and word_length is the size of the string in which the value of the variable is encoded. There are as many words as the variables are. In addition I set a minimum for the values that a variables can assume *(line 563).*
In this way we can reduce the size of the domain if we are not interesting in exploring a certain range of values. For instance, here the min_value *(line 563)* is set to 15 which means that the points that describe the sections of our individual (*fig2,* page 9) can not have a coordinate whose value is smaller than that threshold. This lead not to have too narrow sections which may affect the stability/equilibrium of the "building" as we will see in 3rd experiment (gravity).

- *roulette selection*
a fraction of the generation's fitness (sum of the fitness for all the individuals) is set as a threshold *(line 582).* The individuals' fitness value are summed until this threshold is reached. The individual whose the last fitness value is added to the sum, is selected [2][7] *(line 584 to 588)*.

- *crossover*
the genome of two individuals amongst the selected ones are taken, split in a random position and afterwards swap over *(line 594 to 603)* [2][8]. This repeats iteratively until all the individuals have been recombined.

- *mutation*
according to the current mutation rate one of the bits in the genome of an individual is changed from its state. If its values is 0 it becomes 1 and viceversa [2][9].

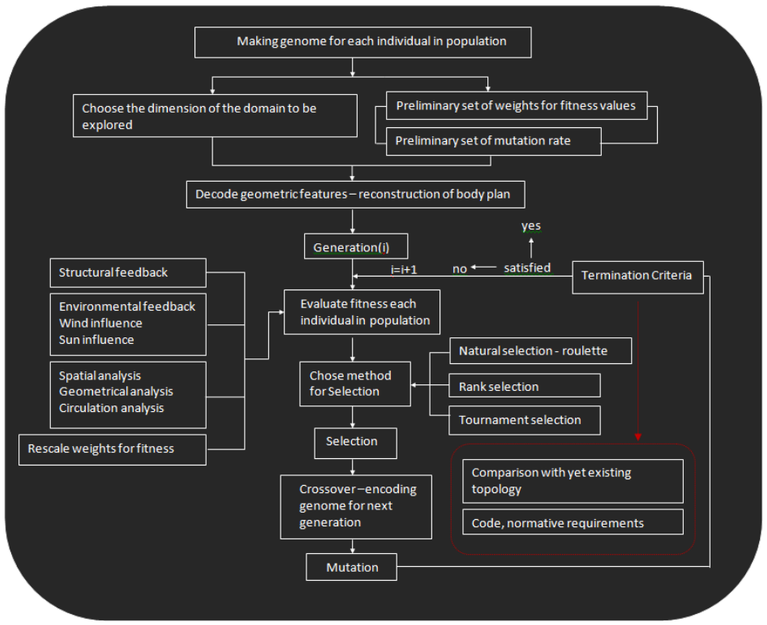


Fig1

Ga pseudo-code

```
528 Sub make(ByRef genome)
529
530     Dim j
531     genome = ""
532
533     For j = 1 To gene_length
534         If Rnd < 0.5 Then
535             genome = genome + "1"
536         Else
537             genome = genome + "0"
538         End If
539     Next
540 End Sub
541
542 Sub decode(ByRef genome,ByRef values())
543
544     Dim temp , pos , endpos
545     Dim i, j,stringvalue
546
547     For i = 1 To words
548         temp = 0
549         endpos = i * word_length
550
551         'work backwards through string
552         For j = 0 To word_length-1
553             pos = endpos - j
554
555             stringvalue= Mid(genome,pos,1)
556
557             If stringvalue = "1" Then
558                 temp = temp + 2 ^ j
559             End If
560
561         Next
562         'set a minimum value for the dimension of the domain
563         If temp <min_value  Then temp = min_value
564         values(i) = temp
565
566     Next
567
568 End Sub
569
570 Function random(lower , upper )
571     random = Int((upper - lower + 1) * Rnd + lower)
572 End Function
573
574
575 Function roulette(ByRef sumfitness,ByRef oldpop_fitness())
576
577     'select a single individual via weighted roulette wheel selection
578     Dim treshold , partsum , j
579     partsum = 0
580     j = 0
581
582     treshold = Rnd * sumfitness
583
584     Do
585         j = j + 1
586         partsum = partsum + oldpop_fitness(j)
587     Loop Until ((partsum > treshold) Or (j = max_pop))
588     roulette = j
589
590 End Function
592 '------- crossover splice chooses a random split point in the genes and swaps them over
593 '-------  a b , c d > a d , c b
594 Sub crossoverSplice(ByRef mum ,ByRef dad,ByRef newmum ,ByRef newdad )
595
596     Dim start
597
598     start = random(1, gene_length - 1)
599
600     newmum = Left(mum, start) + Mid(dad, start + 1)
601     newdad = Left(dad, start) + Mid(mum, start + 1)
602
603 End Sub
604
605 Sub mutate(ByRef genome)
606     Dim pos
607     Dim stringvalue
608
609     pos = random(1, Len(genome))
610     stringvalue=Mid(genome, pos, 1)
611
612     If stringvalue = "0" Then
613
614         genome=left(genome,pos-1)+"1"+Mid(genome, pos+1)
615     Else
616         genome=left(genome,pos-1)+"0"+Mid(genome, pos+1)
617
618     End If
619
620 End Sub
```

The encoding of the body plan for the 2nd experiment is done by using a very simple version of what in artificial intelligence is called Cellular Automaton. This computational scheme, borrowed from *Christian Derix (CECA),* consists of a set of basic rules applied to a grid of points in order to average their position according to given external inputs *(fig4&5)* .We can think at the grid of nodes, shown in *fig4,* as a network where the points are linked together according to local rules. As soon as one of these nodes moves away from its initial position, all the nodes in the grid start looking at their local environment in order to receive information about the position of their neighbourhoods. This can be constituted by 4 (Moore neighbourhood) or 8 (Van Neumann neighbourhood) other nodes *(fig1)*. Each of the nodes will then iteratively assign itself its new coordinate, which is the average of its neighbourhoods' coordinates, until the configuration imposed by the boundary condition has been reached. What comes out of this process is a smooth configuration of points that can mimic a surface whose degree of curvature depends from the given inputs. One of the most interesting and important steps in the process is ensuring simultaneity when assigning the new coordinates. This is due to the fact that in general in any non linear dynamic process things happen at the same time. In so saying the process has to be frozen each iteration for computing the averaging of nodes' coordinate at the same time. Simply speaking means that if after averaging the first neighbourhood the current node assigns itself immediately a new coordinate, it will be updated before the other nodes can receive information regarding its current position. This will cause wrong outcomes. The creation of what is often called in the terminology of the field "limbo word" allow to "fake" the simulation of a parallel process.

Why cellular automata ?
The topology that in this experiment i want to explore can be described as a continuous surface that touches the ground in certain points supporting itself with "legs" or appendices that come out of it. With the regard to Cellular Automata, it is worth saying that it offers a very convenient way for generating and describing such as space. The spatial configuration that comes out of the process in the emergent result of the interaction of the actions of single nodes in their local environment. In this way the only input that have to be given are the boundary conditions.

i - 1 j + 1	i j + 1	i + 1 j + 1
i - 1 j	i j	i + 1 j
i - 1 j - 1	i j - 1	i + 1 j - 1

Fig1 Von Neumann Neighbourhood

The whole genome

As far as the Genetic Algorithm goes, using Cellular Automata for describing the morphology of the individuals offers the possibility of economizing on the size of the genome. This time, for having variety of shapes, it is enough to assign for each individual different boundary conditions whose encoding will constitute the whole genome for the individuals.
As already done in the previous experiment the coordinate of the boundary conditions can be encoded in string of *0&1 (fig2)*. In this case the z coordinate of the three chosen nodes in the network are put to -10 as shown in *fig6.*

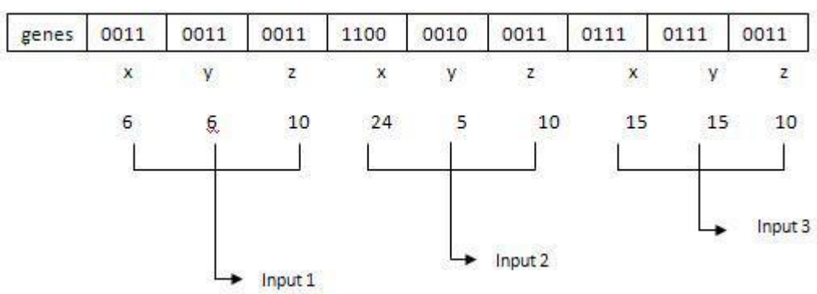


Fig2

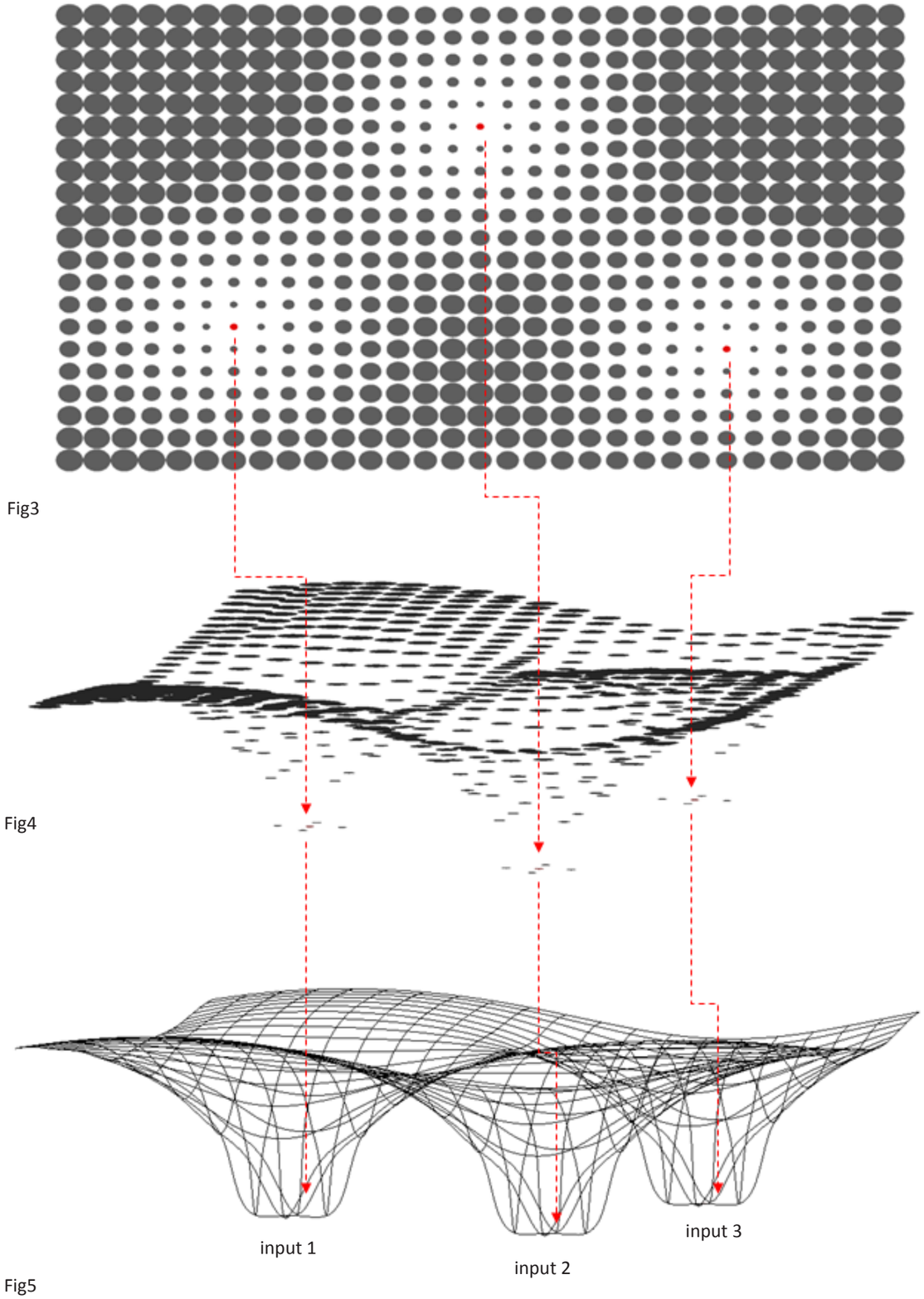


Fig3

Fig4

Fig5

Fitness Criteria

The main criteria, whereby the solution domain is explored, is the equilibrium of the individuals. Let's consider that the surface touches the ground in two points having two "legs". If we assume that the surface is not constrained but only lies onto the ground, the line that goes between these two points is a rotation axis for the structure. If the weight of the parts that projects out the rotation axis, in one of the two sides, is the same the equilibrium is assured. If the surface has more than two "legs" we can extend what said above for each rotation axis (represented by the inner red lines in *fig3*). *Fig4* shows the areas that project out each axis of rotation and the inner area delimited by them. We call the last mentioned area "support plant" and "Gps" its baricenter *(fig3).* The first condition of equilibrium is that "G" the baricenter of the whole area has to fall inside the "support plant". The best condition for equilibrium is when all the areas have the same size and are minimized as much as possible.

Normalizing the fitness parameters

In order to encode the fitness criteria we need more than one parameters. The differences between the areas have to be tested and for this reason, in case the "legs" are 4, there are 6 main possible permutations. In addition, there is another parameter that needs to be taken into account which is related to the size of the area of the inner region defined by the rotation axes, "support plant". For equilibrium reason has to be maximized as much as possible.
When having more than one parameter to evolve we need to assure that the evolution of one does not hinder the evolution of the other ones especially when they tend to balance out. For doing so, each parameter has to be normalized respect to the its maximum and minimum value. In this case we divide the difference between each couple of areas over the biggest one. Once they are normalized they all come in a range of 0-1 and with the same dimension. It is worth mentioning that, most of the times, parameters having different dimension have to be combined for making the individual's fitness, in this case the normalization is an essential procedure for having a stable and meaningful fitness value (see 3rd experiment, "normalising the fitness parameter" and "understanding the weights") .
Normalized parameters can be also weighted according to the importance that we want to give them. In this case the parameters related to the difference of the areas are weighted with a factor of 1 while the parameter related to the size of the area of "support plant" with a factor of 10 *(fig1).*

```
fit1 = (1 - (area(0) - area(1)) / area(0)) ^ 2
fit2 = (1 - (area(0) - area(2)) / area(0)) ^ 2
fit3 = (1 - (area(0) - area(3)) / area(0)) ^ 2
fit4 = (1 - (area(1) - area(2)) / area(1)) ^ 2
fit5 = (1 - (area(2) - area(3)) / area(2)) ^ 2
fit6 = (1 - (area(1) - area(3)) / area(1)) ^ 2

fit7 = area_plant_support / area_plant

w1 = 1
w2 = 1
w3 = 1
w4 = 1
w5 = 1
w6 = 1
w7 = 10

tot = w1 + w2 + w3 + w4 + w5 + w6 + w7
population(who).fitness = (w1 * fit1 + w2 * fit2 + w3 * fit3 + w4 * fit4 + w5 * fit5 + w6 * fit6 + w7 * fit7) / tot
```

Fig1

In addition raising at the power of 2 each normalised parameter would make easier for the selection procedure to appreciate small differences between individuals that have close fitness values *(fig2).*

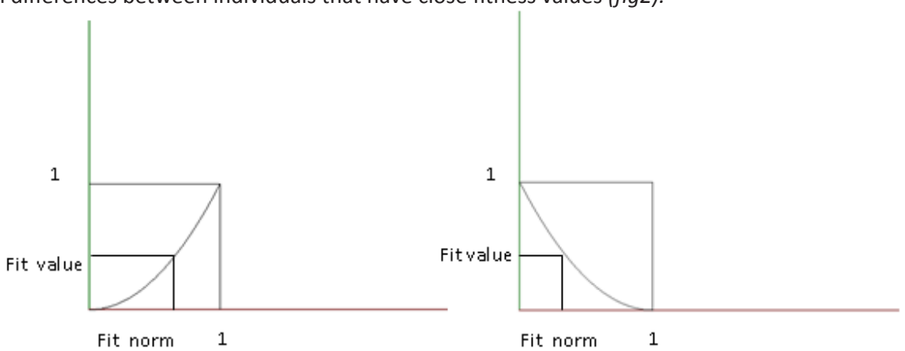


Fig2

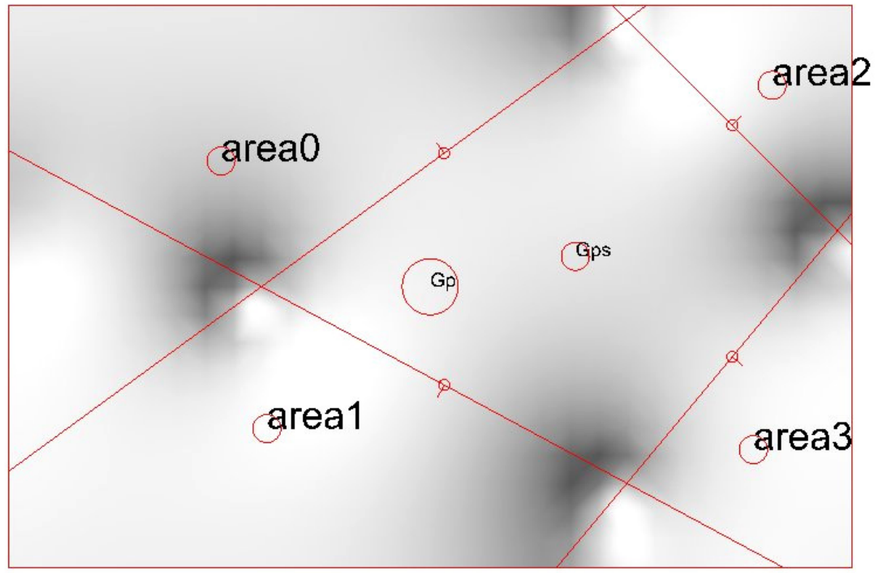


Fig3, palnt view

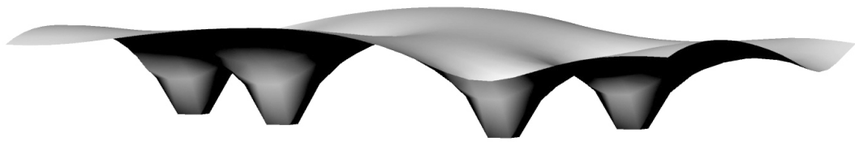


Fig4,elevation

Fig5, generation over generation

Site condition

Fig3 shows the plant of a site located in "Valle di Diano" which is an area close to Salerno, Italy. Starting from the procedure developed for a squared plant it is possible to extend it to a more general boundary contour as the one shown in *fig3.* The main activity this space has been designed for is housing agro industrial fairs. One of the criteria that guides the finding of this shape is to use all the available area of the site and minimizing the impact on the ground. This traduces in taking the contour that define the perimeter of the site as boundary condition and at the same time find a condition of equilibirim with the minimum possible number of "legs".

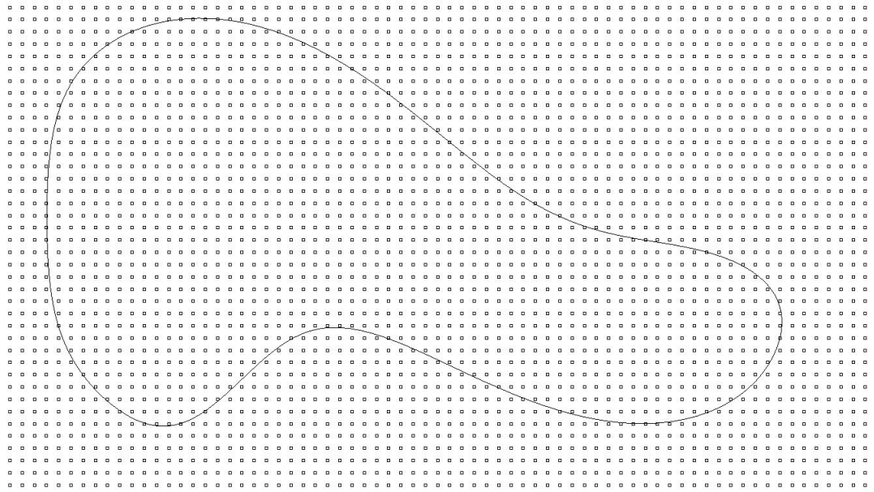


Fig3, boundary condition

Results

The chart in *fig1* shows the fitness generation (sum of the fitness for all individuals in generation). It has an upward trend from the initial minimal value until it reaches its maximum at generation 15. After reaching this peak it levels off.
The chart in *fig2* shows the comparison between the fitness values of individuals of generation 1 and the ones of generation 15 where the fitness is expressed in percentage terms (0 is the minimum, 1 is the maximum). It is clearly visible the values of generation 15 are constantly higher than the ones of generation1.
The individuals, generation over generation, try to equalise the size of the area that project out the axis of rotation as illustrated in previous page. Along with this criteria, the fitness value is given favouring the configuration that develop the minimum number of appendices (legs) which in this experiment can vary from 2 to 5. Considering the extension of the boundary contour, they manage to find a good balance touching the ground in 4 points *(fig4).*

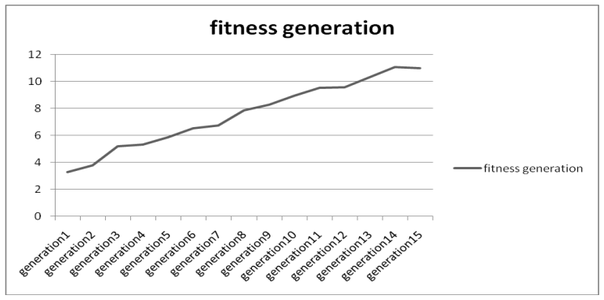


Fig1

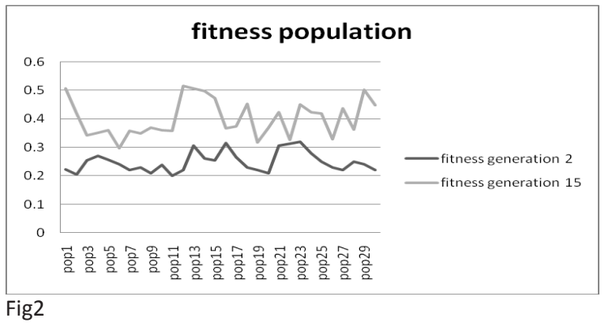


Fig2

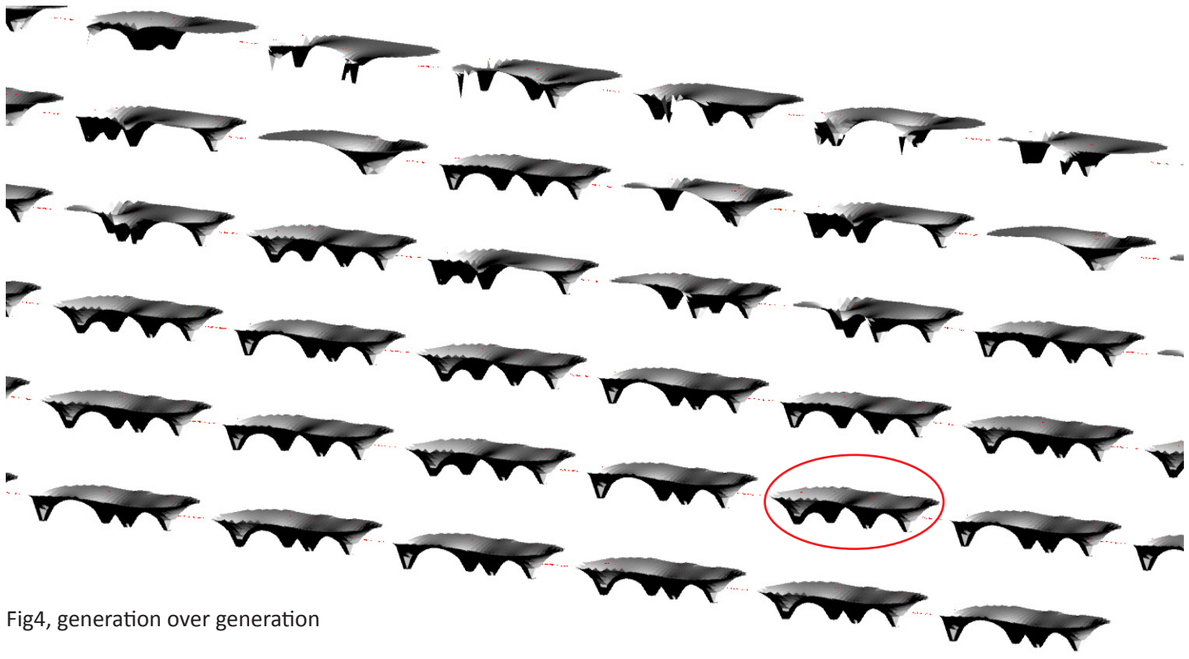


Fig4, generation over generation

Navigate through generations

Fig4 shows some of the individuals generated in the simulation. After the procedure terminates, the designer have to decide which of the proposed configurations has the right qualities to be considered a possible design brief. In this experiment we have driven the evolution with condition of equilibrium try to minimise the impact to the ground without taking into account any other criteria. The choice of one or more configuration can be made, therefore, according to other criteria that have not been encoded in the main procedure. In this case, I chose the one shown in *fig5* because it performs very well in terms of stability (4th best performing amongst the individual of penultimate generation, highlighted with the red ellipse in *fig4*) and because of the slenderness of its shape. This step will be much more elaborated in 3rd experiment where we examine the repercussion of working with entire set of configurations at time, and strategies for orienting the choice amongst the final outcomes are discussed (see 3rd experiment, navigate through generations).

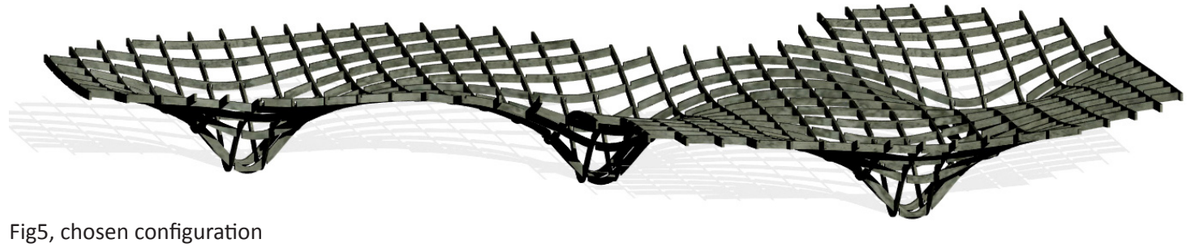


Fig5, chosen configuration

Further development

For further investigating the possibility of building such a structure, I developed a separate algorithm whereby it is possible to make grooves in the ribs in order for them to interlock each others. For minimising the use of mechanical joint as much as possible, the pattern of grooves is not uniform but follows a defined hierarchy that is adaptable to any other similar topology. The ribs touching the ground in at least two points are considered "structural" *(fig1),* common sense would be choosing those ribs in the direction where they are shorter. Their cutting pattern will be oriented in order to support the other ribs meeting them. The grooving pattern for ribs touching the ground in only one point (*fig1*, cantilever ribs), is oriented for supporting the ribs of the other direction and changes when meeting one of structural rib in order to be sustained (*fig1*, red ellipse). By means of a series of boolean operators the algorithm is able to discern what typology of ribs is working on and orienting in the appropriate versus its correspondent grooving pattern.

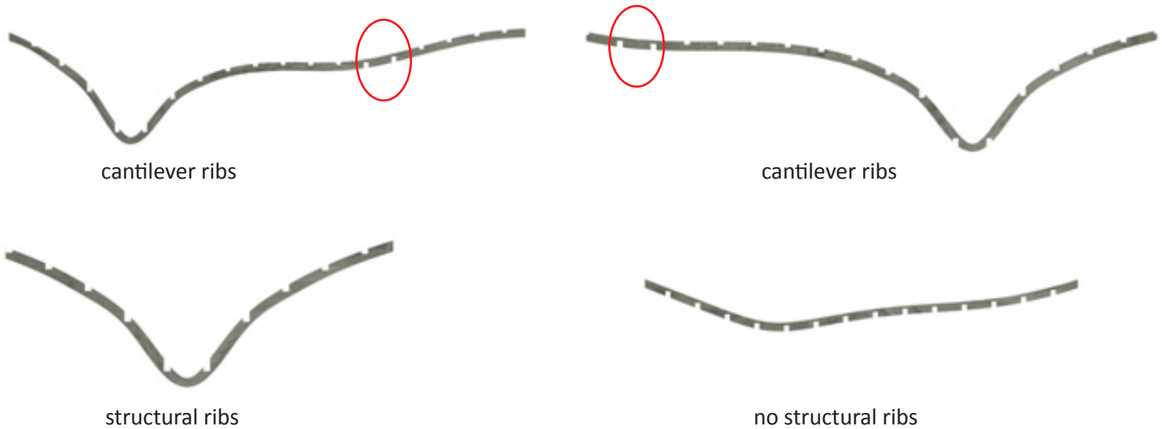


Fig1, ribs

Making the model

The model was produced using laser cutting technique which suits the task because of the big number of grooves to be done *fig2*.
When using the laser, different intensities and velocities have to be set in order to engrave the labels and cut the profiles.
Labelling the ribs in an ordered series of numbers, which respect the above mentioned hierarchy, is crucial in order to recognise the assembling order.
As long as the laser finishes working, the cut ribs can be carefully extracted from the acrylic sheet and the model is ready to be assembled *fig5*. In the next page, are reported some of the images of the model.

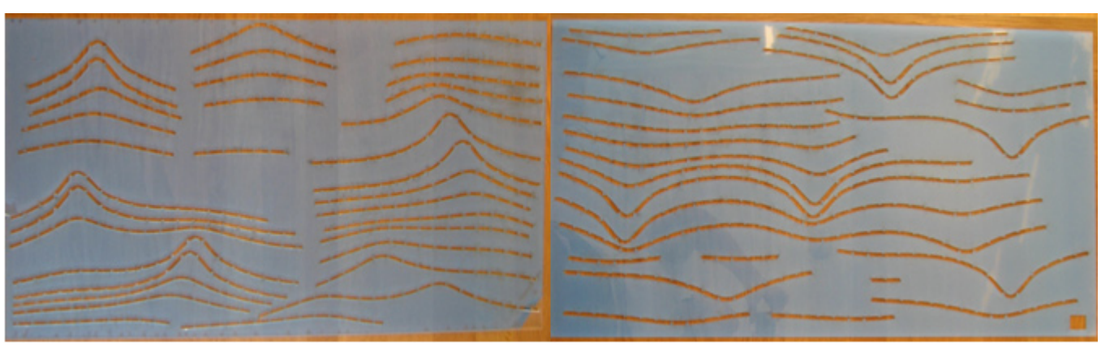


Fig2, laser cutting

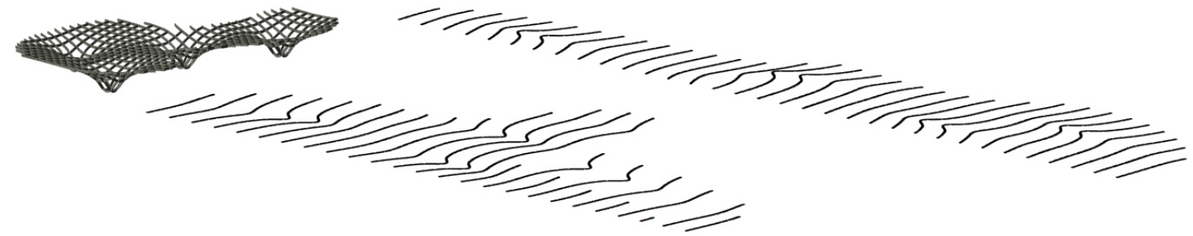


Fig4, positioning the ribs

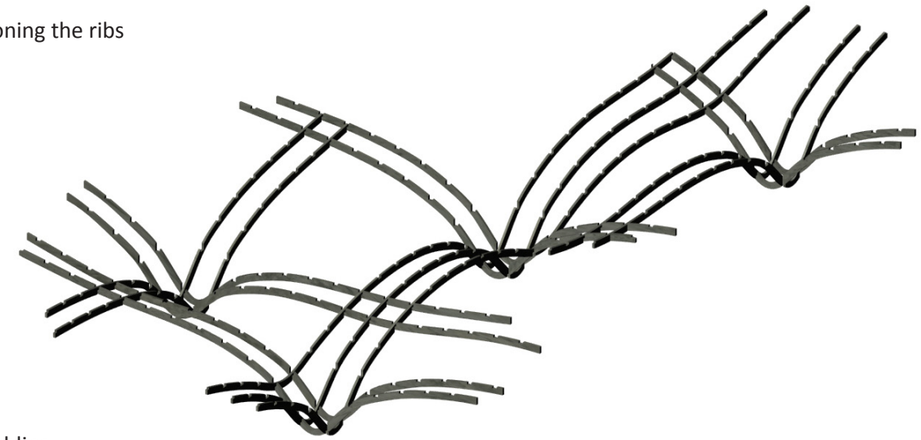


Fig5, assembling

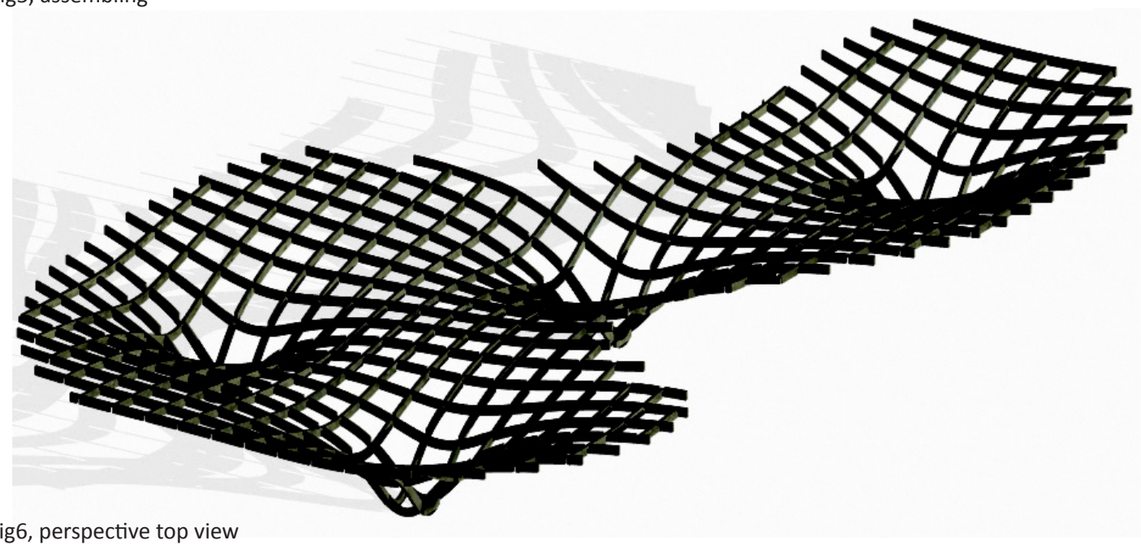
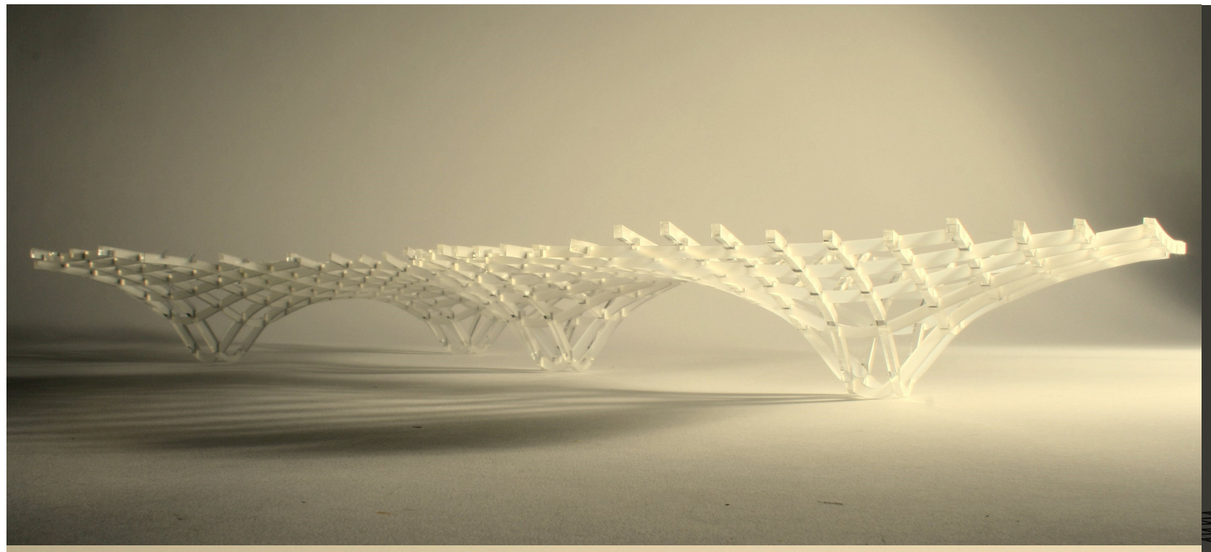
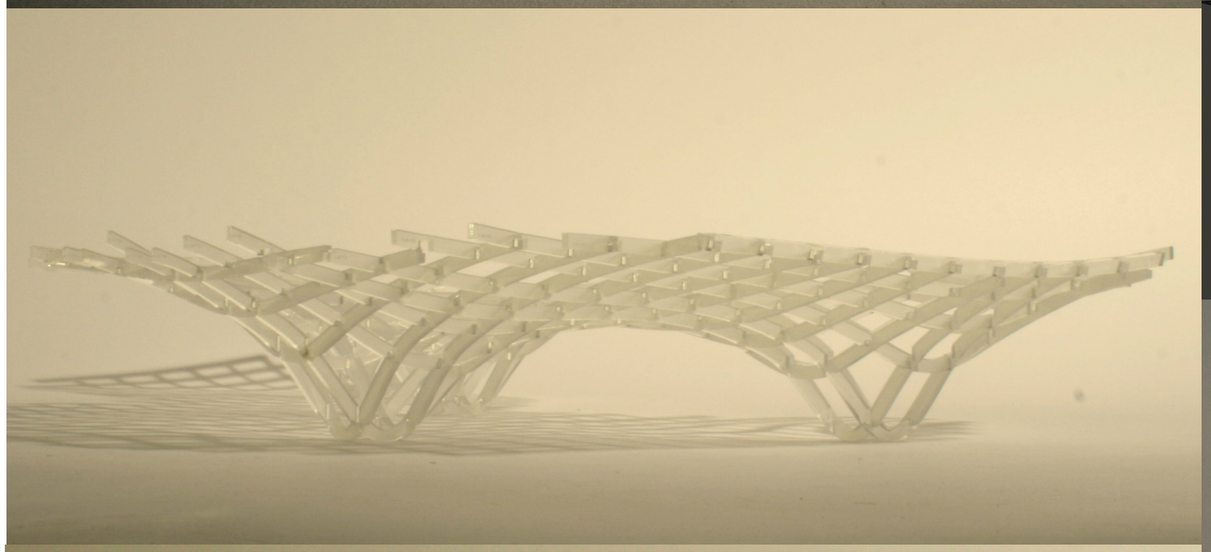
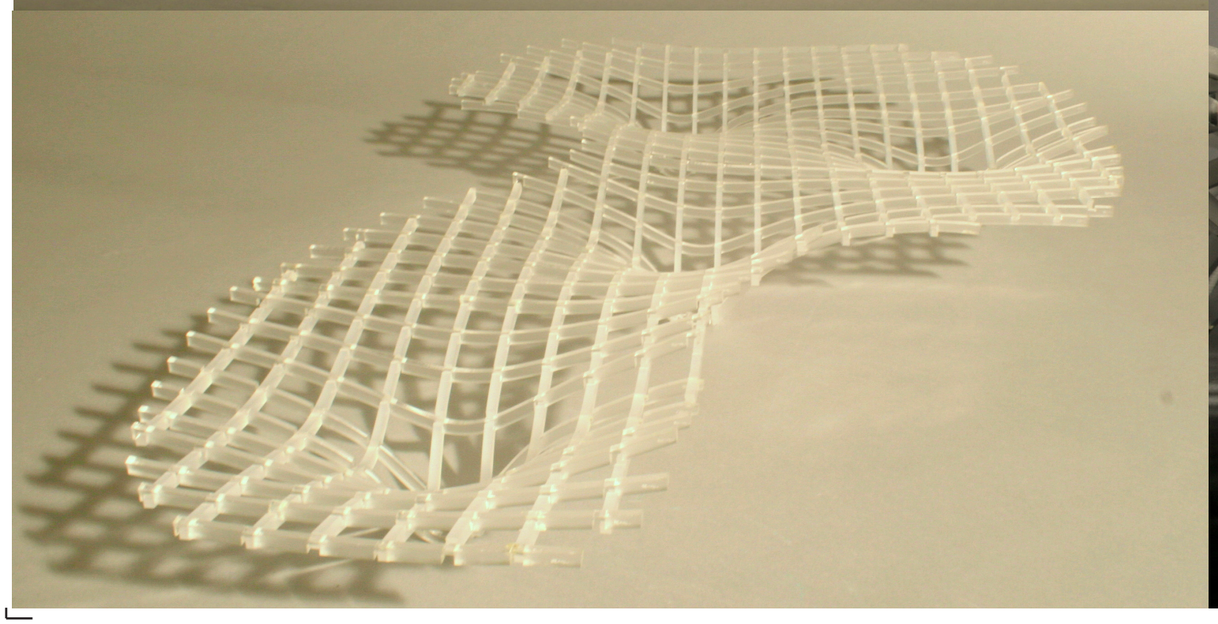


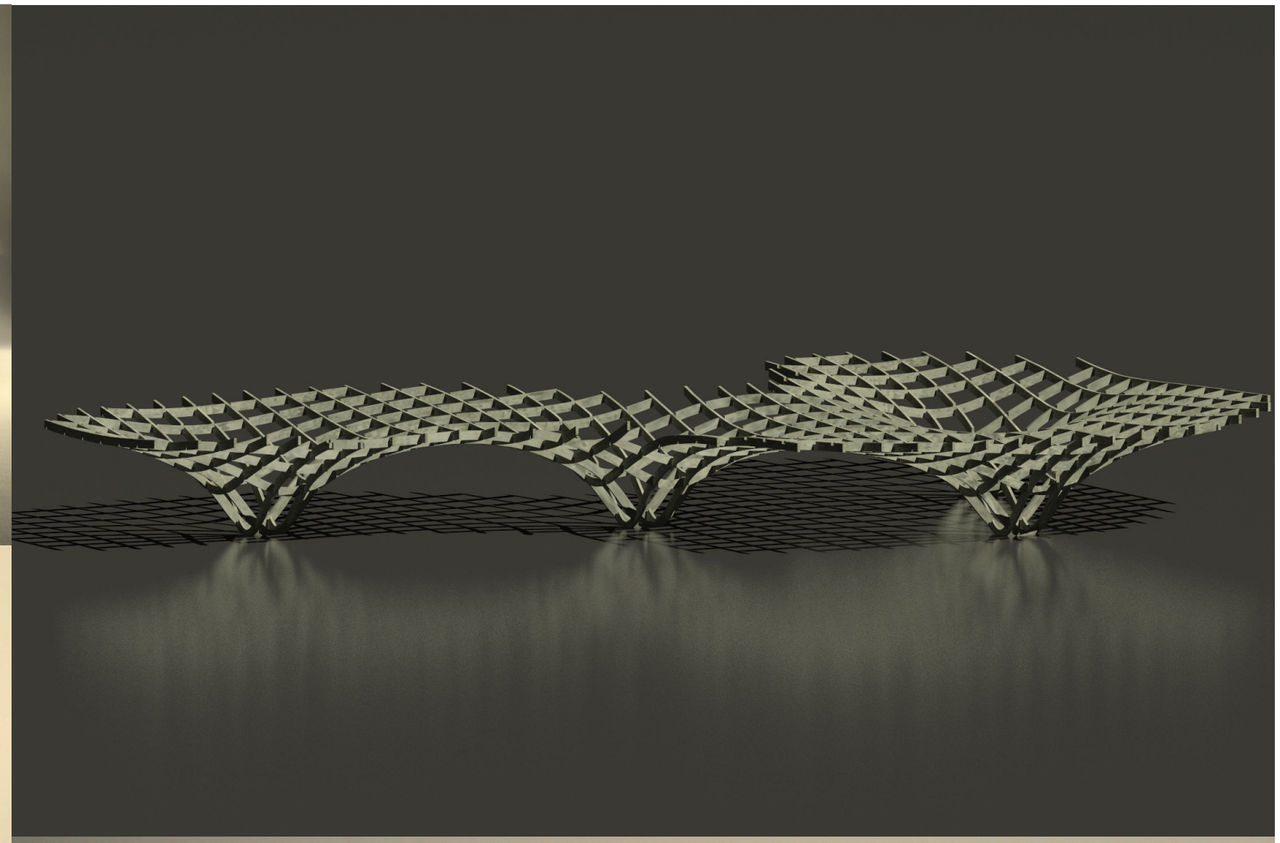
Fig6, perspective top view

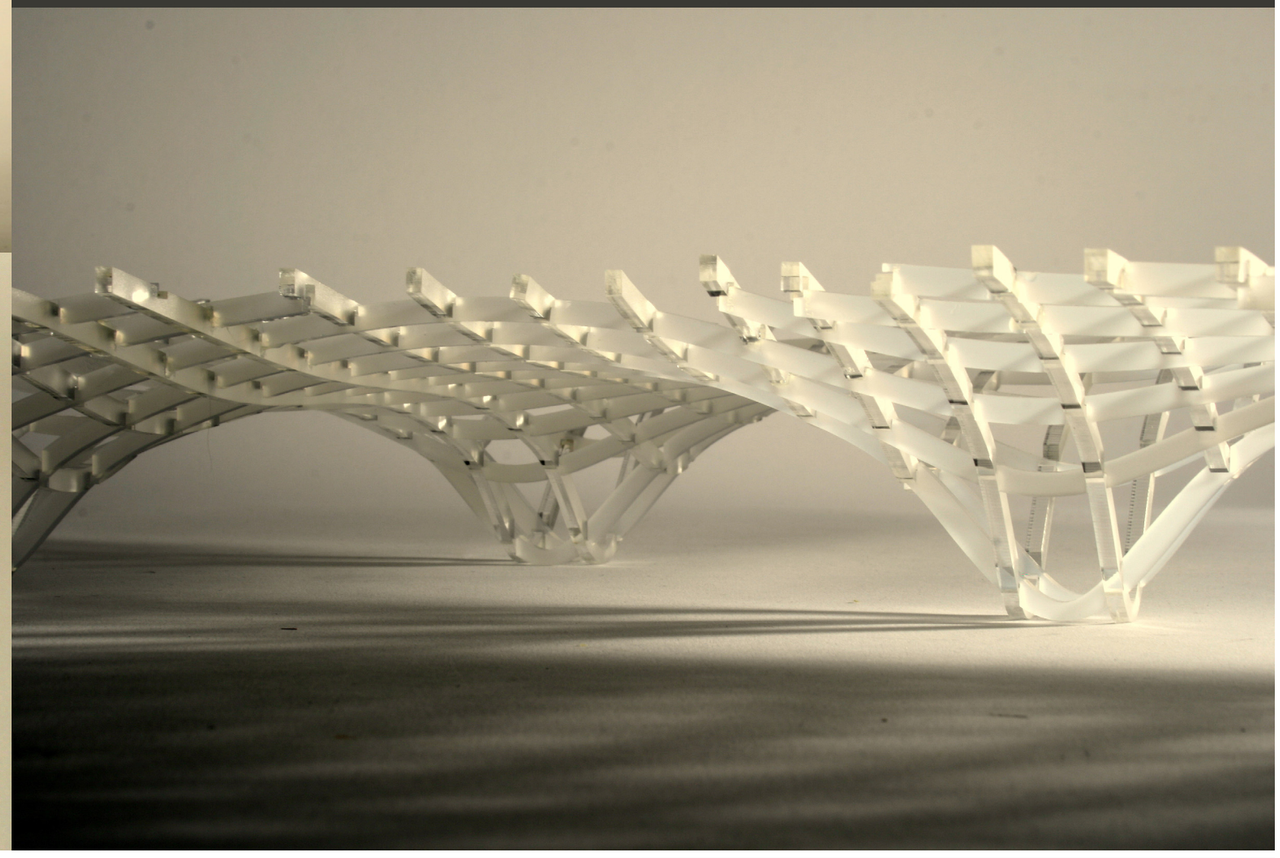
Fig7, elevation











3rd experiment|encoding the body plan

The aim of the 3rd experiment is to deploy the possibility offered by the proposed methodology for reaching a higher level of integration in a design process. Supported by the developed set of tools, I try to build the basis for a design method whereby shapes and patterns are the emergent result of a negotiation between their topology and the environment in which they are placed.
The environment is simulated with different techniques taking into account gravity, wind, sun. Geometric parameters referring to the volume and surface of individuals will complete it for their evaluation. The criteria chosen for guiding the evolution of our individuals are the logic representation of the key features of the environment.
The topology that we are interested to explore, shown in *fig 3*, can be described as Nurbs surface that encloses a volume. Once again our design variables are the position of the points whereby the geometry is described. In this way the three- dimensional dominium of possible positions of these points is explored in order to obtain a set of solution that are consistent with the design criteria.
The way for representing this geometry has been done by assigning the position of points through which spline curves are drawn. The coordinates of these points, that represent the design variables, are encoded in string of 0&1 as already explained for 1st and 2nd experiment *(fig1)*. In general we can have any number of points for describing the surface, in the example shown in *fig4* we have 8 points per section which are interpolated with Nurbs curves creating 6 sections.
The sections are then lofted and a cap that respects the continuity of the surface is created *(fig4).*

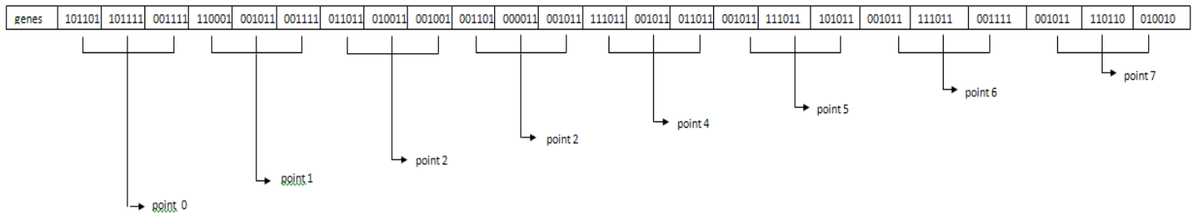


Fig 1

This time we also need a base point respect to whom the positions of the points, defining the sections of the body plan, are given *(fig2).*

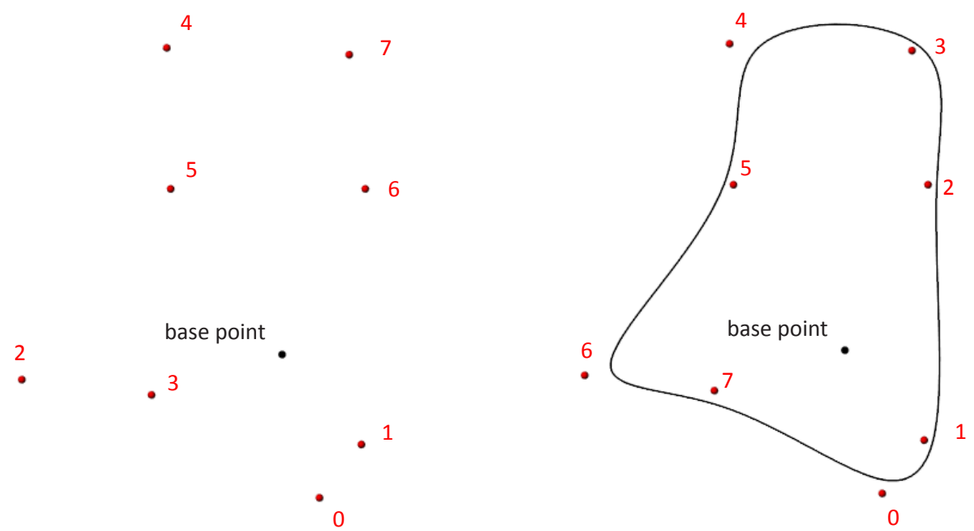


Fig 1

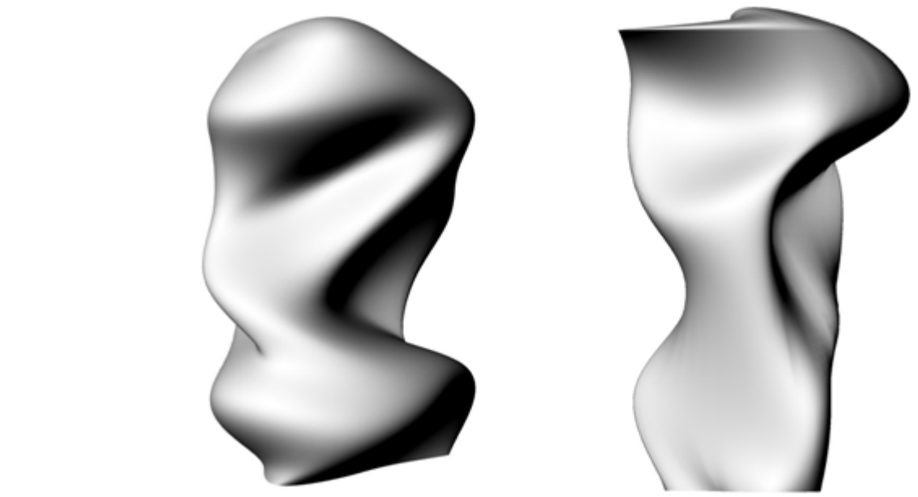


Fig 3

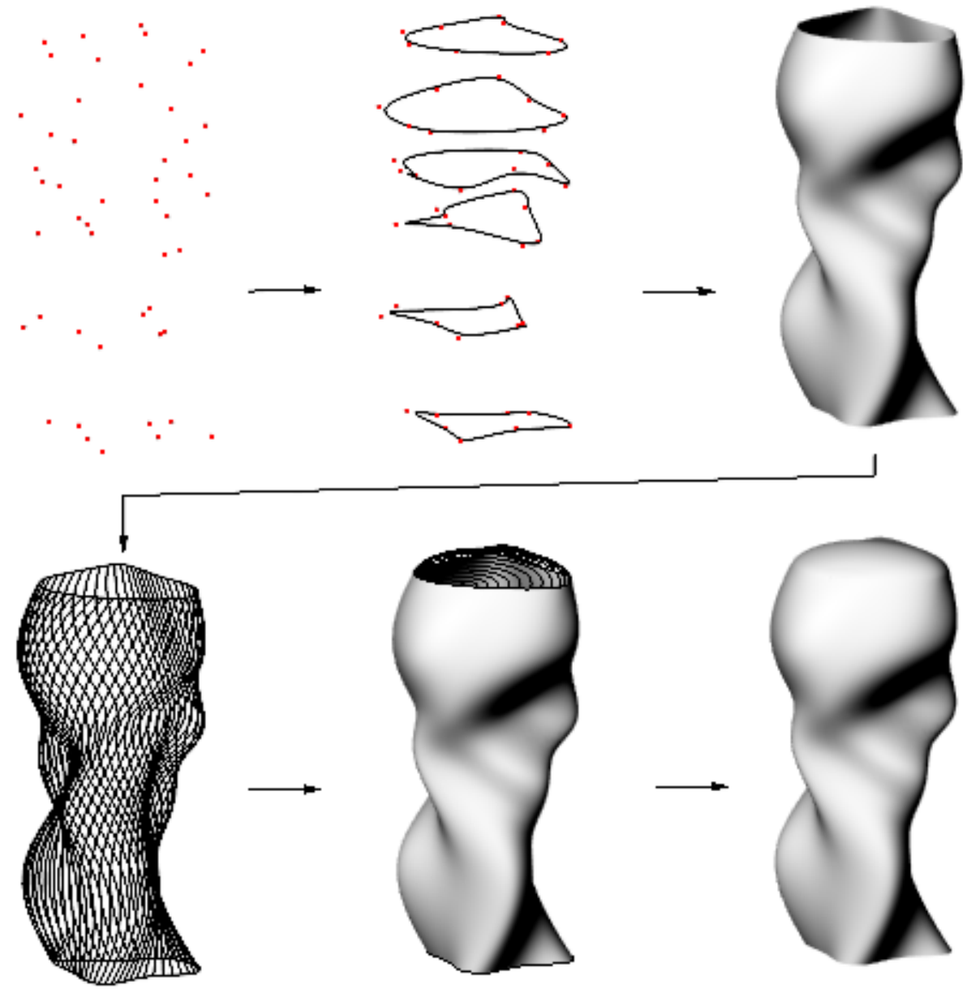


Fig 4

building the body plan

The making of nurbs curve from random points is based on vector calculation. Vectors are calculated from the position of the base point around which their positions have been assigned. By using cross product and dot product it is possible to order them in anticlockwise or clockwise way respecting the geometry of a closed concave region *(fig2).* In this way the curves will not self intersect.
For closing the surface in a way that G2 continuity is assured in every point, I developed a procedure that starts by extracting the top contour of the lateral surface. Getting its domain it is possible to rebuild a series of points at curve parameters from "t to t/2" and "t-1 to t/2-step" where t it is just a variable describing the curve. Each point lying on one side of the curve (t to t/2) corresponds another one lying on the other side (t-1 to t/2-step). It is possible then to use these points for extracting the isocurves of the lateral surface at their location in order to calculate the value of their tangent. After doing this, a series of curve is drawn whose start point and end point are the points extracted from the aforementioned curve. The last condition is that the tangent and the curvature (simulated with accumulation of knots points) at the start and the end is equal to the one of the lateral surface using the values previously extracted from the isocurves *(fig3).*
The procedure can be found in Appendix A page 66 (line 862 to 1008).

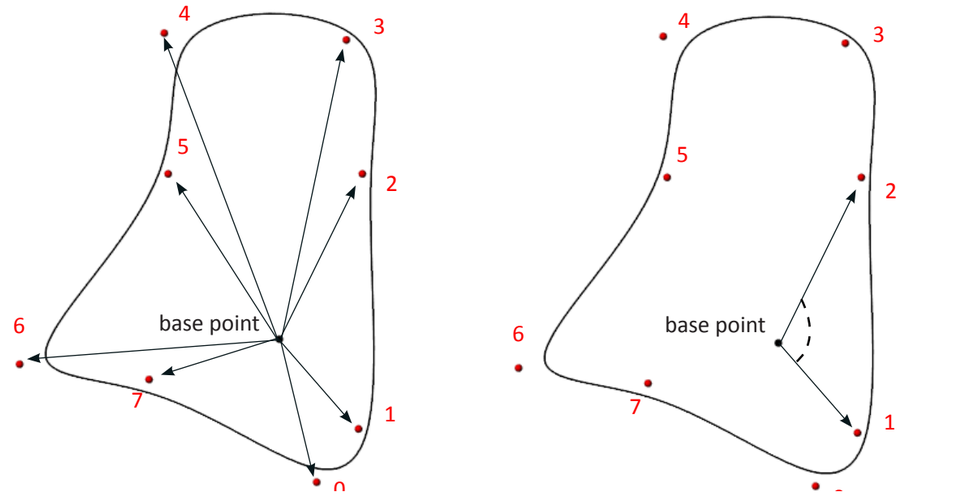


Fig 2

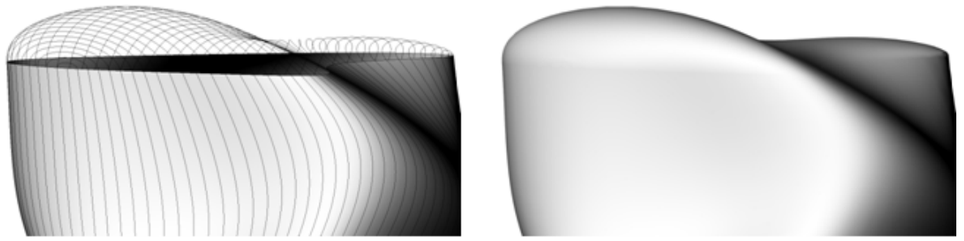


Fig 3

The whole genome

In this 3rd experiment the size of the solution domain is far more bigger than the other two ones. This is due to the number of points (design variables) that are encoded which may vary from 8 points per section to 14. If the number of sections for describing the whole shape is 6, it means that we need 6x8x3=144 values to be encoded. Considering that each values has itself 63 possible way to be expressed (in layman terms it goes from 1 to 63 because I encoded each value in 6 length string of bits), there are *144x6 factorial* combinations. Once again, the number of possible permutations that one should do, for exploring the solution domain in a step by step iteration, would take an infinite amount of time which is one of the main reason that justifies the use of Genetic Algorithm [3][3]
Fig1 shows how the points are encoded making the whole genome.

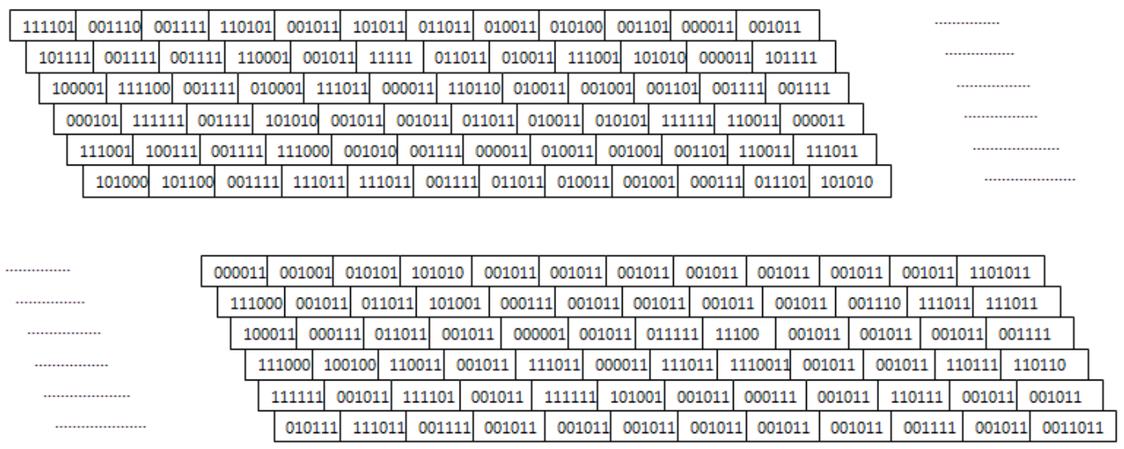


Fig 1

decoding

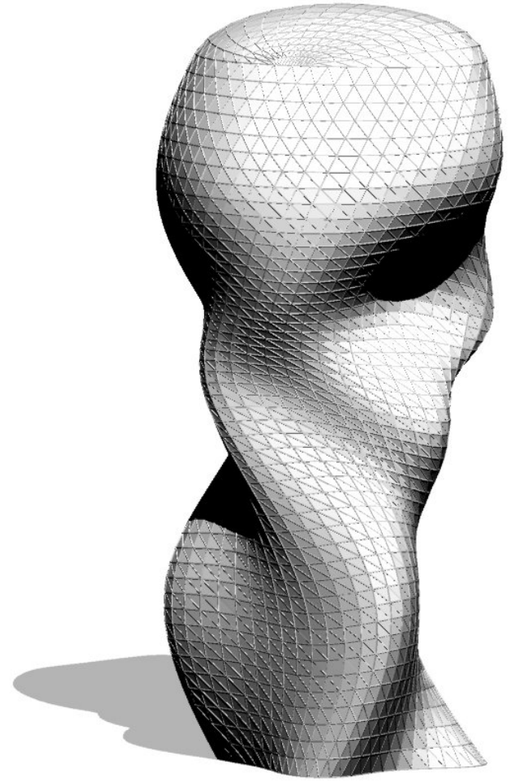


Fig 4

individual

Fitness Criteria

The key point that defines the aim of this research is the development of a methodology whereby design concepts, represented by forms, can emerge from the negotiation of internal traits and their surrounding environment. Within this context the concept of performance assumes multiple meanings. One of the main reasons why the performative approach to architecture has become increasingly accepted and constantly pursued, lies in the recent developments in technology and environmental sensibility. In this framework performance can be broadly defined touching multiple realms from pure technical aspects like structural, energetic, financial, to spatial and social [1][4]. The aim of the proposed methodology is to consider the performances of an artefact not individually but rather simultaneously from the very early stage of the design process. Building performances become guiding design principles for an approach to design that sees a radical shift from making forms to finding forms. Qualitative and quantitative values of the behaviours of the buildings, given by simulation tools, become the basis for a new approach to design [1][5].
It is worth mentioning that this approach should not be intended simply as a way for deriving a set of possible solution to a n-dimensional problem. It is the understanding of the hidden relations between different performative aspects, often counteracting each others, the key challenge of this body of work and its future development.
Consideration should be given to the design space that one want to explore when applying evolutionary algorithm. The size of the solution domain has to be sufficiently bigger for having unpredictable results. The unpredictability is ensured due to impossibility to consider all the potential configurations in advance. For having this, we must go towards the idea of evolving entire set of possible solutions, the so called “populations”. A population is the visual representation, the phenotypes, of the information enclosed in the genome. The evolution goes slowly ahead as genes propagates in a population, which happens at different rates and at different times, until a newly created form emerge from it [7].
As Manuel Delanda points out, it might seem that in this way the role of the design has been relegated to a final choice amongst the proposed solutions but the truth could not be more distant from this statement. The repercussion of such a shift from the conventional way of designing, is that the designer becomes the creator of the generative system. The emergence of forms and patterns is driven by intensive and extensive evaluation of their behaviours under specified design constraints[7] [2]. Simultaneously inventing and interpreting computational scheme, whereby topological configurations are continuously modelled under the action of an optimization procedures, goes beyond the role of a spectator to whom at the end of the play is asked which form he/she would prefer.
Having said that, the other repercussion that such a shift has brought about, is the need for the designer to develop a new sensibility and multiple-skill knowledge in order to able to examine, evaluate and choose, amongst the proposed configurations.
One of the main difficult step to design within the scope of this method, is the development of the “termination criteria”. These criteria should not only set a threshold beyond which the procedure must terminate but should also inform the procedure with further details for preparing and guiding the choice of one or more individuals. It is worth remembering that, due to the yet incomplete structure of the proposed method and to the low-resolution fashion of developed performance evaluation tools, this procedure can only provide a first, rough selection through the almost infinite possible configurations of the solution domain. Let’s assume for the moment that the completeness of the whole procedure was reached by the development of computational scheme that could introduce the evaluation of intensive measurements such as structural vector flow (which can be done by linking the algorithm with FEM based software), or circulation flow (speculation can be done on the so called “agent-based system” for their simulation).
Even if this was realized, the designer will still have to front the fact that, at the end of the procedure, he/she must make a judgement on the proposed forms, patterns whose morphologies will share relevant traits and have differences. The similarity of traits between the individuals commences after a certain time over the course of the procedure. This is strongly connected the size of the initial population and the size of the chosen solution domain. The bigger are these two dimensions, the less similarity the individuals will share and more time is needed for having the convergence towards a specific set of morphologies. In general the richness of the initial population and solution domain ensures great variety throughout all the process. Even when convergence is at its maximum and the values of fitness parameters are at their highest level, the individuals, although sharing similar traits and common topology, have significant differences.
This perfectly matches the expectations of our method, which were to create a system whereby it is possible to make an exhaustive exploration amongst all the possible configurations of the solution domain, discarding the ones that do not respond positively to the pressure of the environment. In so doing at the end of the procedure there will be one, or more, set of possible configurations that have to be subjected to further analysis and judgment based on the criteria that have not been possible to encode. The magnitude of the differences between similar morphologies amplifies the more the scale of observations get closer to the individuals. Before outlining a possible strategy for navigating amongst the configurations, resulting from the proposed methodology, I will explain in the details the “fitness criteria”.

The environment

With regard to the representation of the environment, it can be structured by encoding computational schemes whereby simulating different performative aspects. This evaluation will then feed the so called “fitness function” for the assignment of a value for each individual in population .The computational schemes that I have been able to develop are mainly based on vector fields for the simulation of sun and wind, the others such as spatial organization rely on geometric evaluations. Sun analysis and wind analysis with the relative pressure map that is possible to retrieve from it, is the only case where intensive measurement is performed. In this case the vector evaluation is linked to the empirical laws given by the British Standard Normative ENV 1991-2-4 whereby the geometric configuration is tested.
The representation of the environment can be described as follows :

- **gravity**, which is simulated as a condition of equilibrium checking the position of the volumetric centroid
- **sun**, whose action is instrumented in order to maximize solar gain on the 21st of December
- **wind**, whose action is instrumented for evolving shapes that minimize its impact on them
- **geometric limits**, which is an index of feasibility of their structure
- **spatial organization**, which influences both the external morphology and the internal layout.

Taking into account the organization of space is mainly expressed by three parameters whose optimization tend to maximise the allocation of volume at higher position, minimize the footprint, minimise the size of the external surface (Facade area) and maximise the total area of floors.
It is worth mentioning that when having more than one fitness criteria the development of an efficient and rigorous fitness function is crucial for the effectiveness of the procedure. In this experiment 7 parameters with different dimension need to be weighted for making one fitness value for each individual. For not losing the contribution of any parameter especially when having some of them whose value counteracts others value, they first need to be normalized. After doing this, the designer can assign a set of weights in order to drive the evolution according to the importance that each of those has in the design process.
We will see, when describing in details the normalization procedure, that understanding the influence of each fitness value, when having more than one criteria, is not an easy task. The normalization helps to discern the contribution of each parameter to the whole fitness but the influence that each of those has on the resultant morphology can only be inferred by simulating a set of experiments where they can be evaluated singularly. Only after gaining this knowledge it is possible to manipulate, via the weighting procedure, the contribution to the total fitness of the individuals in order for our system to respond to different purposes. The other method whereby is possible to tackle with many fitness parameters at time is to use one of the five state of the art *Multi Objective Evolutionary Algorithm* (such as *SPEA II, NSGA II, DMOEA* etc.) which mare mainly based on the concept of **Pareto Optimality**. However, the implementation of such methods goes beyond the scope of this research and we leave it to future steps.

Solar gain

Great advances have been recently done in this field whose results can be summarized in few key elements. Creating large areas of south-facing glazing walls and heavily insulated north facing ones, allows to maximise solar energy gain and daylight while minimizing thermal losses. In this way the southern tract can house space that have to be used for a longer period as office or residential which would also have open view through wide areas of glazing [8].
By keeping heating or cooling energy in floor slabs that have a high thermal capacity, it is possible to release this energy at a later time for reducing temperature extreme and , in so doing, achieving a balanced indoor climate [8][2].
As already explained in the 1st experiment the chosen day is the 21st of December because we want to maximize solar gain during the winter and in the shortest day.
Fig3 shows the sun path from 9:00 to 15:00 and the north facing wall of one of the individual where the north direction is represented by the y axis in the world coordinate system. *Fig2* shows the encoding of the vectors representing the direction of the sun at different time of the day.
In order to keep the algorithm as light as possible I developed a procedure whereby the nurbs surface is replaced by a mesh from which normal vectors to each of its faces are extracted.
By calculating the angle (*fig1 alfa*) that each vector, representing a sun direction, makes with the normal vector to each panel, it is possible to retrieve information regarding the exposure of the individual. Manipulating this information lead to determine the percentage of solar gain and to draw a map that shows the average exposure of the individual.

```
'London
'Latitude              51 degree     30 minutes      0    seconds
'Longitude              0 degree      6   minutes   59.76 seconds

'                      21st december
'                      time 07:00       0.8844,-0.439,-0.1587
'                      time 9:00        0.6546,-0.7593,0.0961
'                      time 10:00       0.4525,-0.8722,0.1859
'                      time 11:00       0.2305,-0.9425,0.2418
'                      time 12:00      -0.0072,-0.9655,0.2601
'                      time 13:00      -0.2443,-0.9396,0.2395
'                      time 14:00      -0.4649,-0.8666,0.1814
'                      time 15:00      -0.6537,-0.7514,0.0898
'                      time 16:00      -0.7981,-0.6018,-0.0292
'                      time 17:00      -0.8881,-0.4282,-0.1673
'                      time 18:00      -0.9176,-0.2422,-0.3152
'   Source vectors     21st june
'                      time 07:00       0.8882,0.0669,0.4546
'                      time 09:00       0.6539,-0.2564,0.7118
'                      time 08:00       0.7982,-0.1068,0.5928
'                      time 10:00       0.465,-0.3717,0.8035
'                      time 10:30       0.3578,-0.4139,0.8371
'                      time 12:00       0.0073,-0.4707,0.8823
'                      time 13:00      -0.2304,-0.4477,0.864
'                      time 16:00      -0.791,-0.1167,0.6006
'                      time 17:00      -0.8844,-0.0559,0.4633
'                      time 18:00      -0.9176,0.2416,0.3157
'                      time 19:00      -0.8883,0.4276,0.1677
'                      time 20:00      -0.7984,0.6014,0.0294
```

Fig 2

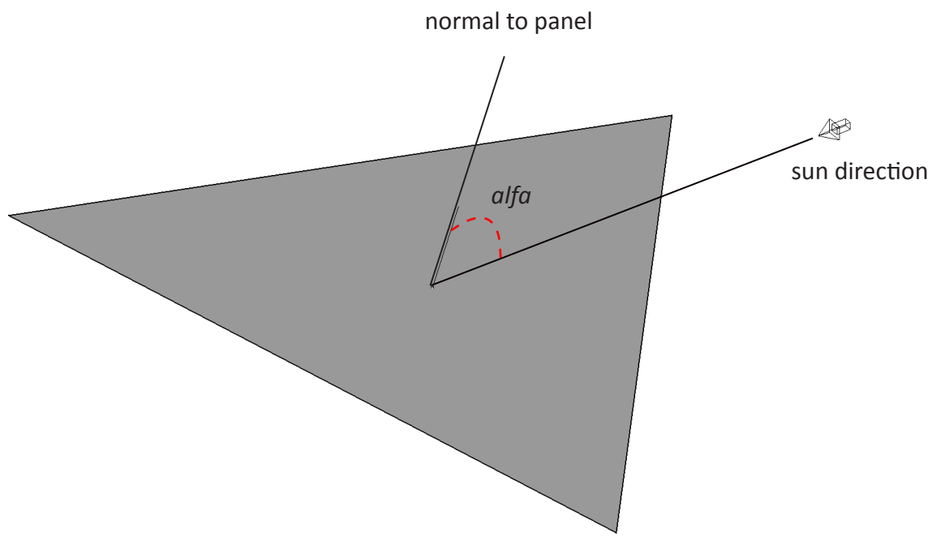


Fig 1 mesh face

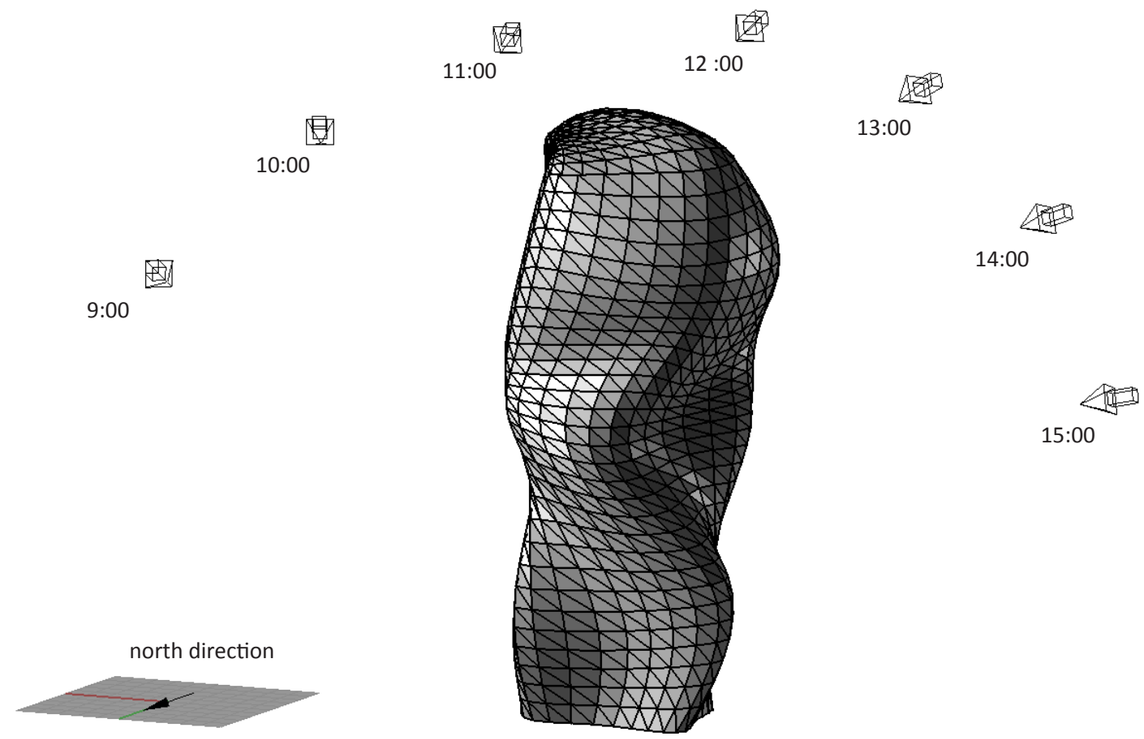


Fig 3

sun path north facing wall

The following paragraph describes the procedure that I developed for building a regular mesh on any sort of Nurbs surface and visualise the degree of exposure to sun for each face. Starting from the surface that describes the geometry of the individual there are 5 main steps :

- surface domain
the domain of the surface is examined in order to determine position of points in the world coordinate system at specific parameter (u,v). By changing the value of the parameters we can follow the isocurves that describe the surface and specify a set of rules for building triangulated faces *(line 1079 to 1103).*

- meshing
after storing in an array the position of all the points that we are interested to have, according to the resolution of the mesh, a set of rules for the kind of desired mesh has to be formulated. Our intent in not to have separated triangulated faces but one mesh object made of smaller faces. This will considerably increase the speed of the process. In order to do this, a sequential array of number that describe the order of all the vertices for all the faces of the mesh have to be generated *(line 1105 to 1123).*

- getting the normals
after the mesh has been drawn, it is possible to analyse the normal vector to each panel and store them in an array which afterwards is used for computing the angle *alfa* that each of the normal vector make with each of the sun direction *(line 1128).*

Fig2 illustrates the sequence and anticipates the map that is drawn after computing the values of *alfa* for each panel.

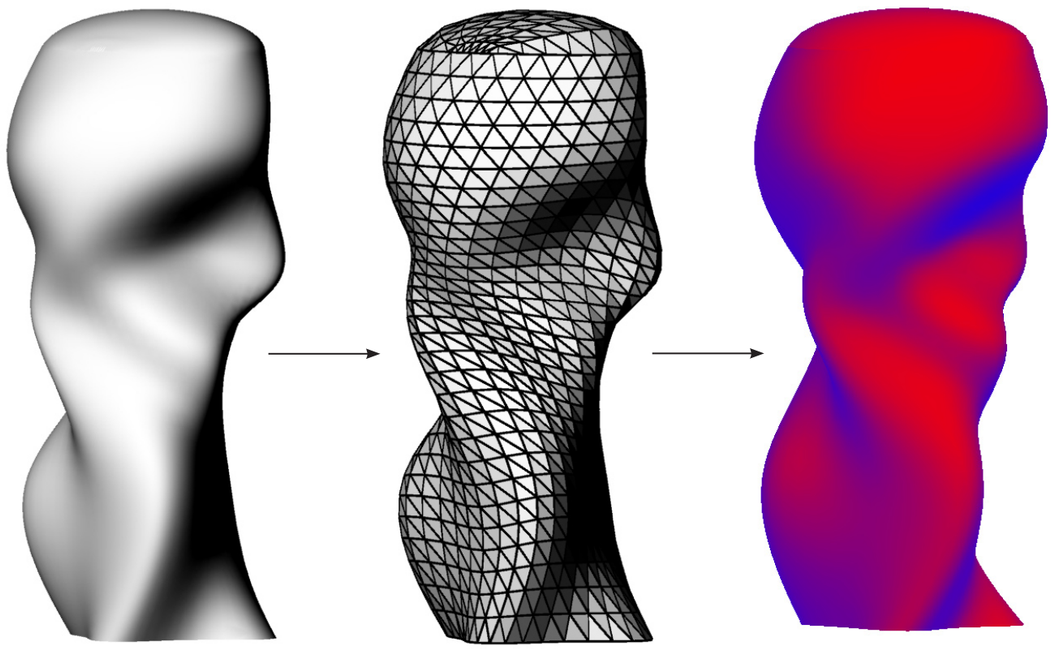


Fig 2 surface mesh map

```
1074 Sub tri_facets (ByVal id_surf,ByRef panels_body,ByRef panels_norm_body(),num_u,num_v,ByVal startpoint)
1075
1076     Dim domU, domV,uMin, uMax, vMin, vMax
1077     Dim uInc, vInc
1078
1079     domU = Rhino.SurfaceDomain(id_Surf, 0)
1080     domV = Rhino.SurfaceDomain(id_Surf, 1)
1081
1082     uMin = domU(0)
1083     uMax = domU(1)
1084     vMin = domV(0)
1085     vMax = domV(1)
1086
1087     uInc = (uMax-uMin)/num_u
1088     vInc = (vMax-vMin)/num_v
1089
1090     Dim i, j,mesh_points(),mesh_face_vertices()
1091     ReDim mesh_points(0)
1092     Dim counter:counter=0
1093
1094     For i = uMin To uMax+uinc/2 Step uInc
1095         For j = vMin To vMax-vInc/2 Step vInc
1096
1097             mesh_points(counter)= Rhino.EvaluateSurface(id_Surf, Array(i, j))
1098             ReDim Preserve mesh_points(ubound(mesh_points)+1)
1099             counter=counter+1
1100         Next
1101     Next
1102
1103     ReDim Preserve mesh_points(ubound(mesh_points)-1)
1104
1105     Dim header
1106     counter=-1
1107     For i=0 To num_u
1108         For j=0 To num_v-1
1109
1110             header=i*(num_v)+j
1111             counter=counter+2
1112             ReDim Preserve mesh_face_vertices(counter)
1113
1114             If j=num_v-1 Then
1115                 mesh_face_vertices(counter-1)=array(header,header+1,header+num_v,header+num_v)
1116                 mesh_face_vertices(counter)=array(header,header-num_v+1,header+1,header+1)
1117             Else
1118                 mesh_face_vertices(counter-1)=array(header,header+num_v+1,header+num_v,header+num_v)
1119                 mesh_face_vertices(counter)=array(header,header+1,header+num_v+1,header+num_v+1)
1120             End If
1121
1122         Next
1123     Next
1124
1125     If isarray(mesh_points)=True And isarray(mesh_face_vertices)=True Then
1126
1127         panels_body=    rhino.AddMesh (mesh_points,mesh_face_vertices)
1128         panels_norm_body = Rhino.MeshVertexNormals(panels_body)
1129
1130     Else
1131
1132         panels_body=False
1133
1134     End If
1135
1136 End Sub
```

- dot solar

an easy way for computing the angle that normal vector makes with one of the examined sun directions, is using “dot product”. It is important to normalise these vectors before computing the dot-product in order not to do any further calculation for retrieving the angle *(line 1211).*

The closer the angle is to 0 degree or dot product equal to 1 , which means that sun and normal have same direction, the more the panel is exposed to sun. For an angle of 90 degree, or dot product equal to 0, sun direction is orthogonal to the normal and it can be regarded as it were no exposure. For values that are bigger than 90 degree , it means that the panel lies on the north facing wall or they are shadowed.

For this reason, the angle itself represents the degree of exposure of each panel. Because there are 7 vectors representing the sun (from 9:00 to 15:00 step 1 hour) each angle is recorded and afterwards they summed. Finally we divide this value by the total number of sun directions (7) for having a mean of the exposure over the whole day which is called “panels_heat” *(line 1220).* It is worth saying that even if we play only with “blue” and “red” which are at the extreme of the RGB gradation we have 255 possible values to use in order to enhance smooth transition of exposed area and shadows.

- mapping

using these values and RGB colours it is possible to visualise the degree of exposure of each panel to the sun *(line 1222 to 1225).*

solar gain|fitness

In order to have a numerical value that is index of the efficiency of the analysed shape with regard to solar gain, we use the information gained in the previous analysis. As already said, the value “panels_heat” represents the exposure for each panel throughout the whole day and it comes in a range from 0 to 1. The closer this value is to 1, the more exposed to sun is panel. Therefore, checking if the value of panels_heat is bigger than a specified threshold represents a measurement of the percentage of the exposure to sun (line 1239 to 1251). By counting for how many faces of the mesh this value is bigger than a certain threshold, we can assign a consistent fitness to the individual that regards the solar gain. In this experiment the threshold has been set to the 60 percent of the thermal radiation that would be possible to accumulate in one day if the a panel lay always perpendicular to the sun .

```
1200 Sub dot_solar(ByVal panels,ByVal panels_norm(),ByRef panels_heat(),ByVal source_vectors_sun(),ByVal generation)
1201
1202     Dim i,j
1203     Dim rgb_value
1204     Dim red,green,blue
1205     Dim incident_angle
1206
1207     For i = 0 To UBound(panels_norm)
1208
1209         For j = 0 To ubound(source_vectors_sun)
1210
1211             incident_angle =rhino.VectorDotProduct(panels_norm(i),source_vectors_sun(j))
1212
1213             If (incident_angle >= 0) Then
1214
1215                 panels_heat(i) = panels_heat(i) + incident_angle
1216             End If
1217         Next
1218
1219         'average panel exposure angle
1220         panels_heat(i)=panels_heat(i)/(ubound(source_vectors_sun)+1)
1221
1222         red=255 * panels_heat(i)
1223         green=0
1224         blue=255 - 255 * panels_heat(i)
1225         arrcolors(i)=rgb(red,green,blue)
1226
1227     Next
1228
1229 End Sub
1230
1231 Sub get_sun_solar_gain(ByVal panels_heat_body(),ByVal panels_heat_cap(),ByRef sunfitness)
1232
1233     Dim i
1234     Dim panel_fitness
1235     Dim max
1236
1237     max=ubound(panels_heat_body)+ubound(panels_heat_cap)+2
1238
1239     For i=0 To ubound(panels_heat_body)
1240         If panels_heat_body(i)>0.6 Then
1241             panel_fitness=1
1242             sunfitness=sunfitness+panel_fitness
1243         End If
1244     Next
1245
1246     For i=0 To ubound(panels_heat_cap)
1247         If panels_heat_cap(i)>0.6 Then
1248             panel_fitness=1
1249             sunfitness=sunfitness+panel_fitness
1250         End If
1251     Next
1252
1253     sunfitness=sunfitness/max
1254
1255 End Sub
```

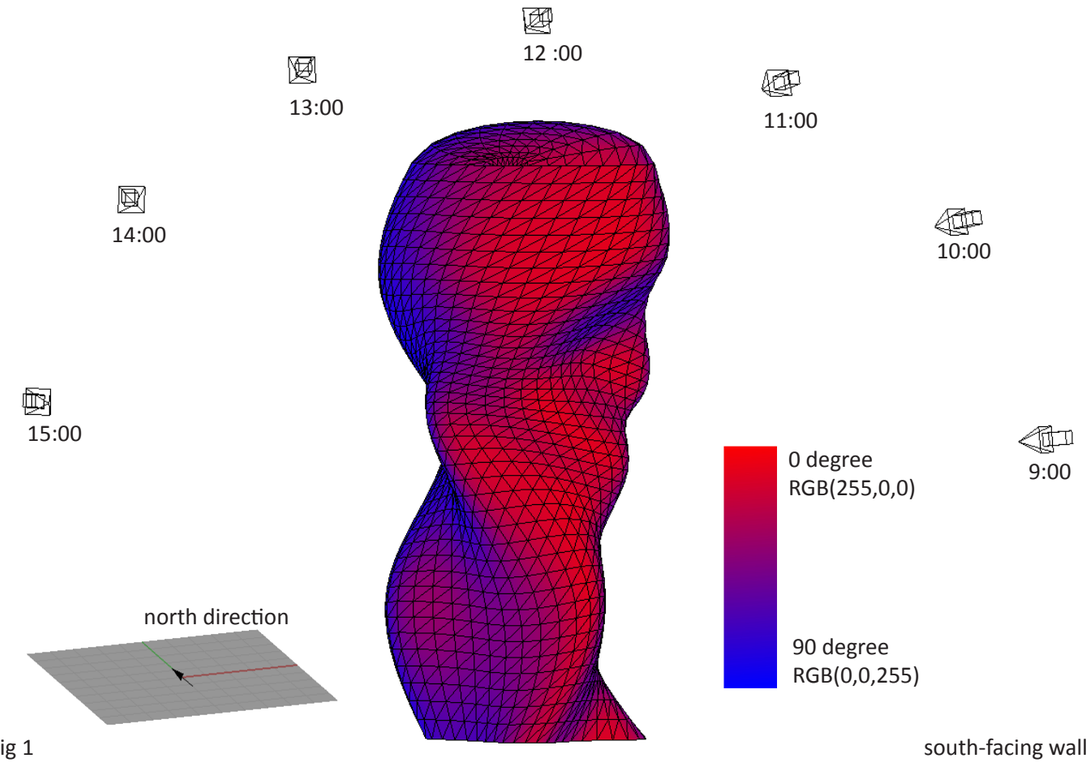


Fig 1

Wind analysis

The procedure here presented relies on the British standard normative ENV 1991-2-4 for wind analysis which takes into account, amongst other parameters, the direction of the wind and the orientation of an exposed surface. Although the results that come from this procedure are approximated, compared to the ones that would come from a CFD simulation, they are sufficient for the scope of this research. The main limit is that turbulence phenomena that occurs when the height of the building is more than 200 meters are not taken into account

The reason why this algorithm has been developed is mainly due to computation resource limits. If one wants to analyse each individual, with a software that performs CFD analysis, it will slow down the whole process of an unreasonable amount of time . It might even be impossible for a normal laptop or desk pc to perform such a great number of analysis. It is worth mentioning that the size of each population for having good variety of shapes should be between 20 and 50 (the more the better) and the number of generations , for having convergence, should be bigger than 5. This means that at least there are 100 individuals to be analysed.

Fig1 shows the meaning of reference pressure and the wind map for the United Kingdom. The reference pressure is in layman terms pressure that the wind would exercise on a surface if this lay perpendicular to it (without taking care turbulence and other dissipative effects). The meaning of it that the kinetic energy owned by the wind is transformed in pressure when finding a surface that obstacles its flow. In the formula shown in *fig1 ρ* stands for the density of the air at 25 Celsius degree and V for velocity of the wind.

The prevailing direction of the wind, taken from the normative, is 240 270 degree, while the velocity is around 20m/s *(fi2).* There are many other coefficients that should be taken into account for calculating a more reliable velocity of the wind for a specific site. The coefficient are given by the roughness of the area, as well as by other factors that regard the surroundings. I decided to take a conservative value of 30 m/s leaving at later time the analysis of these , when a precise site will be specified.

Fig3 and fig4 show the behaviour for a duo-pitch roof building which is exposed to wind. As we will describe over the course of this paragraph, it implies the knowledge of the normal vector for each of the face of the building as well as the upwind and downwind edge of it.

Section 7 Reference wind

7.1 Reference wind pressure

(1)P The reference mean wind velocity pressure, q_{ref}, shall be determined from:

$$q_{ref} = \frac{\rho}{2} v_{ref}^2 \qquad (7.1)$$

where:

v_{ref} reference wind velocity as defined in 7.2

ρ air density

The air density is affected by altitude and depends on the temperature and pressure to be expected in the region during wind storms. Unless otherwise specified in annex A, the value of ρ shall be 1,25 kg/m³.

Fig 1

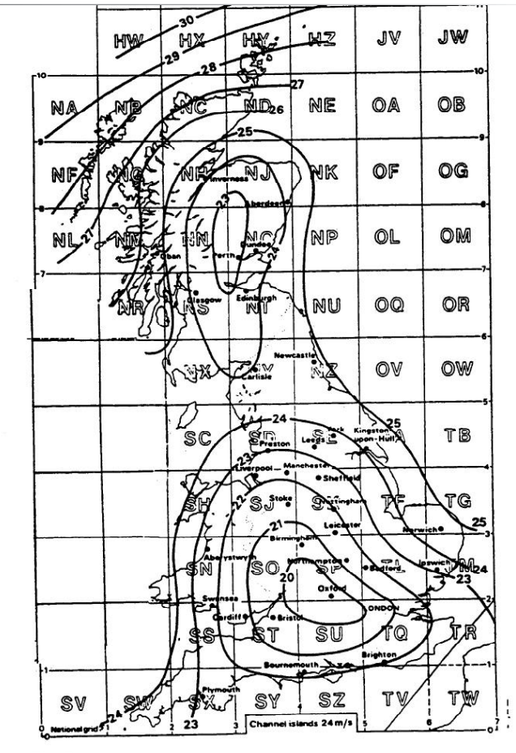


Fig 2

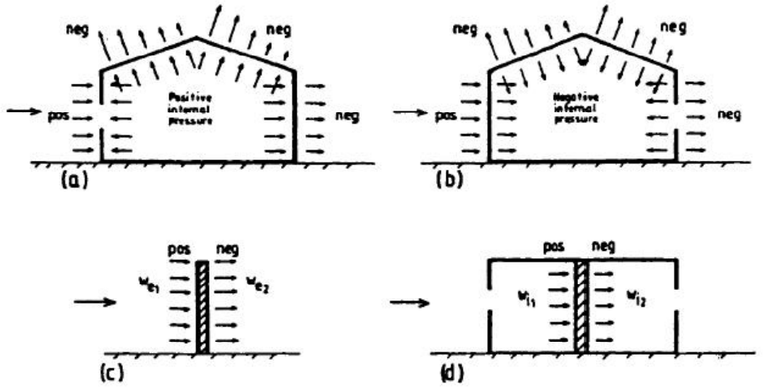


Fig 3

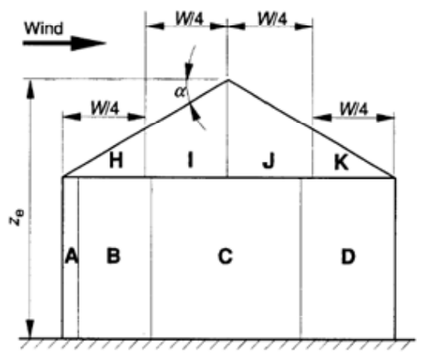


Fig 4

a) Duopitch roof

Using the normative

The mesh generated in “sun analysis” is used again here for simulating the impact against the wind. The solid angle that the wind, represented by a vector, makes with the normal to each panel can be divided in two angles *teta* and *alfa* (*fig2 & fig 3)*. *Teta* is the angle that the wind makes with the normal in the xy plane and *alfa* (or to be more precise 90-*alfa*) is the angle between the two vectors in one of the orthogonal plane to the xy plane of the world coordinate system. If we assume the wind blows horizontally, the last mentioned angle represents also the inclination of the face respect to the ground. If it is 90 it means that the wall is vertical, if it is 0 the wall lays horizontally. Basically *teta* and *alfa* are the component of the solid angle that the wind makes with the normal to the face.
With the same technique used in sun analysis for calculating these two values (normalization and dot-product), it is possible to take from tables, given by the normative, the value of the corresponding pressure coefficient. This is multiplied for the reference pressure obtaining the pressure on the panel.
In addition to these two values the upwind and downwind edge have to be determined. This is crucial for having the right value of pressure coefficient which can vary dramatically between two equally oriented panels that are in two different zones of the building. The procedure for obtaining the size of the different zones of the building is given by the ENV 1991-2-4 as shown in *fig1 & fig2.*
According to the typology of the building, its edges are divided in one or more zones which are determined by taking into account the distance of them from the upwind edge *(fig1)*. There is a different set of coefficient for each of these zones.
In addition, the size and the number of the zones in which the building is divided vary also depending on the direction of the wind. If the wind invests the building perpendicularly its main dimension, it is likely that the laterals side will be divided in not more than 2 zones *(fig1 case d<e).* This is mainly due to the fact that the wind causes wide area of negative pressure on the faces whose correspondent *teta* has values from 75 to 130 degree. In general the wind creates positive pressure on the face where impacts and negative pressure on the connected lateral faces *(fig 1 zone A)*. If the length of the lateral surface is sufficiently extended the pressure will gradually become positive along it *(fig1 zone B and C)* and experiences another abrupt change in *zone E* which is the back side of the building.
The same can be said for a roof *fig3.* A roof that is completely flat *(alfa=0)* will be exposed to negative pressure for almost all its area while instead a roof whose *alfa* value is 90 can be considered as a wall. If *alfa* is in between these two extremes, the roof *(fig3 mono-pitch roof)* experiences a first *zone H* where the pressure is mainly negative and a *zone I* where the pressure gradually becomes positive again.

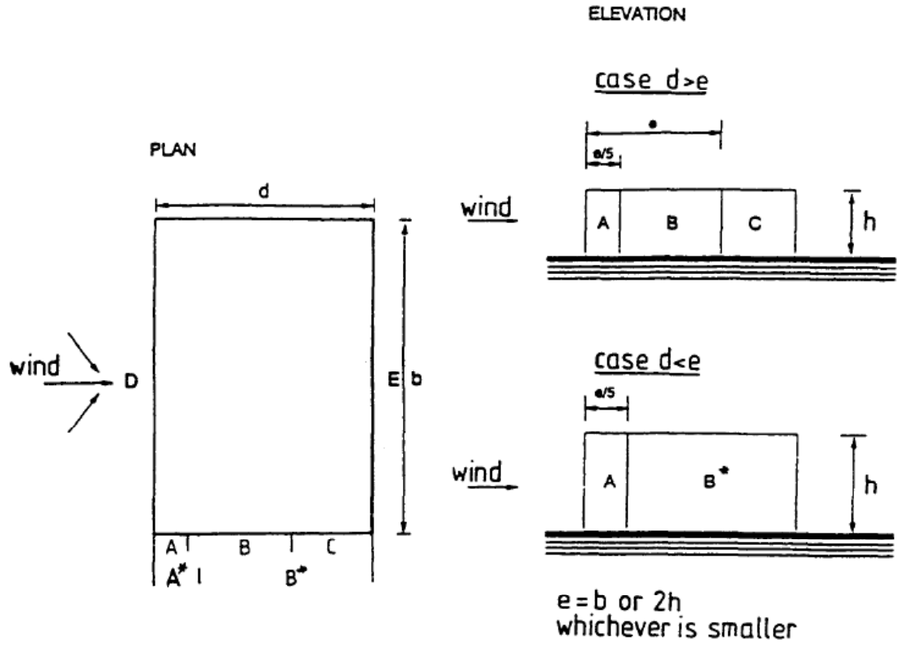


Fig 1

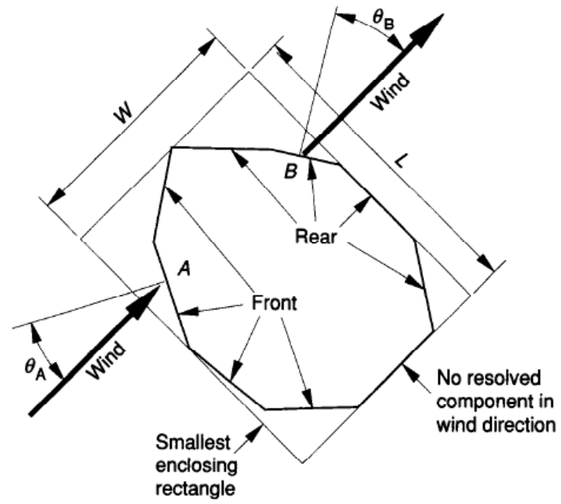


Fig 2

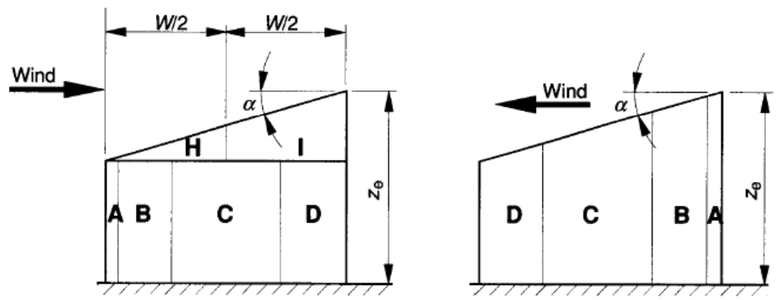


Fig 3

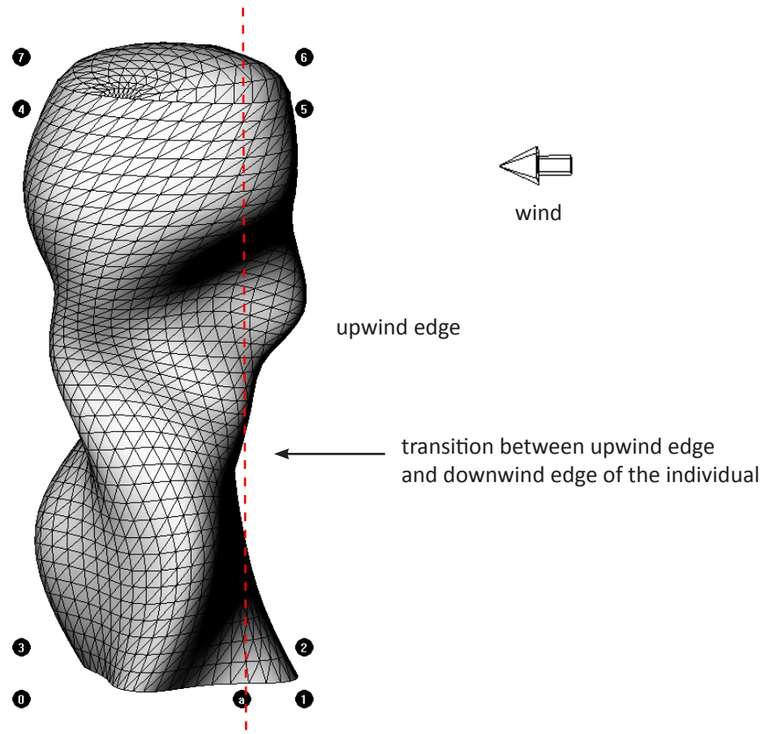


Fig 4

The determination of these values for a geometry such as the one of our individual is far more complex but, generally speaking, those are the main principles. In order to overcome the problem of dividing the building into zones according to its orientation to wind and retrieving the correspondent pressure coefficient, I developed a procedure that holds for any sort of surface and mainly consist in :

- using the normative

We need to transfer the table given by the normative *(fig3)* in three double array (7,12) shown in *fig1* each made of 7*12 elements *(fig1)*. These are nested arrays and are composed of a first one, where is encoded the value for the angle *alfa* (0 to 90 degree), and a second nested one that has two values. The second value of this array is the angle *teta* (0 to 180 degree) and the first one is the correspondent pressure coefficient that a face would have according to the aforementioned angles. There are three different arrays (called cpe) one for each zone of the building A,B,C. We do not take into account zone D *(fig2)* which occurs only in the case of extremely long buildings.

- determining upwind and downwind edge

after computing the bounding box for each individual(*fig4* previous page, bounding box represented by the 8 dot points) it possible to determine on which side of it the wind impacts and follow the rules given by the normative to subdivide the building into different zones. In *fig4,* previous page, the dot *a* and the *dashed red line* represent the threshold between the upwind and downwind edge as already seen for *fig1 & fig2* (previous page).

- meshing

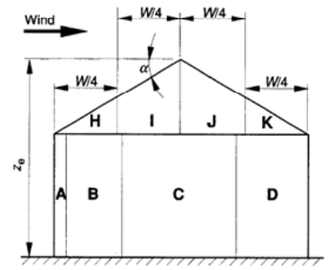
with regard to the mesh, we use the same one created for “sun analysis” and the same array of normal vectors of its faces. Once the values of *teta* and *alfa* have been calculated and the upwind edge of the building has been determined, it possible to take the corresponding pressure coefficient.

```
cpe_a(0,0)=array(90,array(1,0))
cpe_a(0,1)=array(90,array(1.07,15))
cpe_a(0,2)=array(90,array(1.10,30))
cpe_a(0,3)=array(90,array(1.12,45))
cpe_a(0,4)=array(90,array(0.54,60))
cpe_a(0,5)=array(90,array(-1.10,75)
cpe_a(0,6)=array(90,array(-1.30,90)
cpe_a(0,7)=array(90,array(-0.8,105)
cpe_a(0,8)=array(90,array(-0.63,120
cpe_a(0,9)=array(90,array(-0.50,135
cpe_a(0,10)=array(90,array(-0.34,15
cpe_a(0,11)=array(90,array(-0.30,16
cpe_a(0,12)=array(90,array(-0.34,18

cpe_a(1,0)=array(75,array(0.58,0))
cpe_a(1,1)=array(75,array(0.71,15))
cpe_a(1,2)=array(75,array(0.85,30))
cpe_a(1,3)=array(75,array(0.82,45))
cpe_a(1,4)=array(75,array(0.78,60))
cpe_a(1,5)=array(75,array(-1.10,75)
cpe_a(1,6)=array(75,array(-1.21,90)
cpe_a(1,7)=array(75,array(-0.8,105)
cpe_a(1,8)=array(75,array(-0.63,120
cpe_a(1,9)=array(75,array(-0.50,135
cpe_a(1,10)=array(75,array(-0.34,15
cpe_a(1,11)=array(75,array(-0.30,16
cpe_a(1,12)=array(75,array(-0.34,18

cpe_a(2,0)=array(60,array(0.50,0))
cpe_a(2,1)=array(60,array(0.63,15))
cpe_a(2,2)=array(60,array(0.77,30))
cpe_a(2,3)=array(60,array(0.68,45))
cpe_a(2,4)=array(60,array(0.59,60))
cpe_a(2,5)=array(60,array(-1.10,75)
cpe_a(2,6)=array(60,array(-1.21,90)
cpe_a(2,7)=array(60,array(-0.8,105)
cpe_a(2,8)=array(60,array(-0.63,120
cpe_a(2,9)=array(60,array(-0.50,135
cpe_a(2,10)=array(60,array(-0.34,15
cpe_a(2,11)=array(60,array(-0.30,16
cpe_a(2,12)=array(60,array(-0.34,18
.....
```

```
cpe_c(0,0)=array(90,array(0.83,0))
cpe_c(0,1)=array(90,array(0.68,15))
cpe_c(0,2)=array(90,array(0.49,30))
cpe_c(0,3)=array(90,array(0.34,45))
cpe_c(0,4)=array(90,array(0.26,60))
cpe_c(0,5)=array(90,array(0.23,75))
cpe_c(0,6)=array(90,array(0.20,90))
cpe_c(0,7)=array(90,array(-0.26,105))
cpe_c(0,8)=array(90,array(-0.29,120))
cpe_c(0,9)=array(90,array(-0.33,135))
cpe_c(0,10)=array(90,array(-0.32,150))
cpe_c(0,11)=array(90,array(-0.28,165))
cpe_c(0,12)=array(90,array(-0.24,180))

cpe_c(1,0)=array(75,array(0.81,0))
cpe_c(1,1)=array(75,array(0.82,15))
cpe_c(1,2)=array(75,array(0.83,30))
cpe_c(1,3)=array(75,array(0.68,45))
cpe_c(1,4)=array(75,array(0.55,60))
cpe_c(1,5)=array(75,array(0.37,75))
cpe_c(1,6)=array(75,array(0.20,90))
cpe_c(1,7)=array(75,array(-0.26,105))
cpe_c(1,8)=array(75,array(-0.29,120))
cpe_c(1,9)=array(75,array(-0.33,135))
cpe_c(1,10)=array(75,array(-0.32,150))
cpe_c(1,11)=array(75,array(-0.28,165))
cpe_c(1,12)=array(75,array(-0.24,180))

cpe_c(2,0)=array(60,array(0.80,0))
cpe_c(2,1)=array(60,array(0.70,15))
cpe_c(2,2)=array(60,array(0.62,30))
cpe_c(2,3)=array(60,array(0.50,45))
cpe_c(2,4)=array(60,array(0.35,60))
cpe_c(2,5)=array(60,array(0.27,75))
cpe_c(2,6)=array(60,array(0.20,90))
cpe_c(2,7)=array(60,array(-0.26,105))
cpe_c(2,8)=array(60,array(-0.29,120))
cpe_c(2,9)=array(60,array(-0.33,135))
cpe_c(2,10)=array(60,array(-0.32,150))
cpe_c(2,11)=array(60,array(-0.28,165))
cpe_c(2,12)=array(60,array(-0.24,180))
.....
```

```
cpe_b(0,0)=array(90,array(1,0))
cpe_b(0,1)=array(90,array(1.07,15))
cpe_b(0,2)=array(90,array(1.10,30))
cpe_b(0,3)=array(90,array(1.12,45))
cpe_b(0,4)=array(90,array(0.54,60))
cpe_b(0,5)=array(90,array(-0.73,75))
cpe_b(0,6)=array(90,array(-0.80,90))
cpe_b(0,7)=array(90,array(-0.73,105))
cpe_b(0,8)=array(90,array(-0.63,120))
cpe_b(0,9)=array(90,array(-0.50,135))
cpe_b(0,10)=array(90,array(-0.34,150))
cpe_b(0,11)=array(90,array(-0.30,165))
cpe_b(0,12)=array(90,array(-0.24,180))

cpe_b(1,0)=array(75,array(0.81,0))
cpe_b(1,1)=array(75,array(0.82,15))
cpe_b(1,2)=array(75,array(0.83,30))
cpe_b(1,3)=array(75,array(0.69,45))
cpe_b(1,4)=array(75,array(0.55,60))
cpe_b(1,5)=array(75,array(-0.73,75))
cpe_b(1,6)=array(75,array(-0.80,90))
cpe_b(1,7)=array(75,array(-0.73,105))
cpe_b(1,8)=array(75,array(-0.63,120))
cpe_b(1,9)=array(75,array(-0.50,135))
cpe_b(1,10)=array(75,array(-0.34,150))
cpe_b(1,11)=array(75,array(-0.30,165))
cpe_b(1,12)=array(75,array(-0.24,180))

cpe_b(2,0)=array(60,array(0.57,0))
cpe_b(2,1)=array(60,array(0.63,15))
cpe_b(2,2)=array(60,array(0.79,30))
cpe_b(2,3)=array(60,array(0.68,45))
cpe_b(2,4)=array(60,array(0.47,60))
cpe_b(2,5)=array(60,array(-0.73,75))
cpe_b(2,6)=array(60,array(-0.80,90))
cpe_b(2,7)=array(60,array(-0.73,105))
cpe_b(2,8)=array(60,array(-0.63,120))
cpe_b(2,9)=array(60,array(-0.50,135))
cpe_b(2,10)=array(60,array(-0.34,150))
cpe_b(2,11)=array(60,array(-0.30,165))
cpe_b(2,12)=array(60,array(-0.24,180))
.....
```

Fig 1



a) Duopitch roof

Fig 2

Table D.13 External pressure coefficients for pitched roof zones A to J

Pitch angle α	Local wind direction Θ	Zone A	B	C	D	E	F	H	I	J
−45°	0°	−0.61	−0.58	−0.56	−0.41	−0.76	−0.78	−0.62	−0.79	−0.94
	±30°	−0.53	−0.50	−0.49	−0.55	−0.55	−0.81	−0.52	−0.58	−0.58
	±60°	−1.11	−1.29	−1.36	−0.96	−0.97	−0.91	−1.05	−0.97	−1.17
	±90°	−1.25	−0.81	−0.62	−0.42	−0.77	±0.20	−1.48	−1.05	−0.97
−30°	0°	−0.76	−0.68	−0.60	−0.50	−0.76	−0.63	−0.76	−0.85	−0.90
	±30°	−1.13	−1.02	−0.89	−0.79	−0.84	−0.76	−1.17	−0.87	−0.73
	±60°	−2.06	−2.33	−2.17	−1.22	−1.03	−0.80	−1.69	−1.18	−1.21
	±90°	−1.28	−0.94	−0.70	−0.37	−0.70	±0.20	−1.54	−1.10	−1.01
−15°	0°	−1.08	−1.05	−0.97	−0.92	−0.88	−0.82	−1.10	−0.96	−0.83
	±30°	−2.64	−2.37	−1.71	−1.00	−0.93	−0.85	−2.75	−1.66	−1.11
	±60°	−2.25	−2.15	−1.85	−1.02	−0.76	−0.72	−2.44	−1.60	−1.07
	±90°	−1.22	−0.79	−0.58	−0.31	−0.60	−0.20	−1.51	−1.15	−1.10
−5°	0°	−1.49	−1.13	−1.19	−1.12	−0.83	−0.82	−1.47	−0.91	−0.67
	±30°	−2.36	−2.21	−1.63	−1.04	−0.82	−0.77	−2.24	−1.30	−0.91
	±60°	−1.85	−1.57	−1.28	−0.77	−0.65	−0.54	−2.10	−1.67	−1.09
	±90°	−1.30	−0.79	−0.58	−0.27	−0.59	−0.20	−1.65	−1.15	−1.10
0°	0°	−1.47	−1.25	−1.15	−1.15	−0.69	−0.71	−1.43	−0.75	−0.52
	±30°	−2.00	−1.70	−1.38	−1.03	−0.66	−0.67	−1.70	−1.24	−1.10
	±60°	−1.70	−1.24	−1.10	−0.64	−0.61	−0.42	−2.00	−1.70	−1.38
	±90°	−1.43	−0.75	−0.52	−0.24	−0.62	±0.20	−1.47	−1.25	−1.15
+ 5°	0°	−1.39	−1.24	−1.11	−1.19	−0.56	−0.59	−1.39	−0.69	−0.43
	±30°	−1.78	−1.64	−1.34	−1.09	−0.62	−0.60	−1.75	−1.02	−0.76
	±60°	−1.67	−1.33	−1.12	−0.71	−0.64	−0.42	−2.05	−1.51	−1.05
	±90°	−1.21	−0.83	−0.55	−0.25	−0.61	±0.20	−1.48	−1.15	−1.10
+15°	0°	−0.91 ±0.20	−0.83 ±0.20	−0.78 ±0.20	−0.81 ±0.20	−0.21 ±0.20	−0.31 ±0.20	−0.90 ±0.20	−0.36 ±0.20	−0.30 ±0.20
	±30°	−0.84 ±0.20	−0.88 ±0.20	−0.82 ±0.20	−0.83 ±0.20	−0.21 ±0.20	−0.37 ±0.20	−0.63 ±0.20	−0.35 ±0.20	−0.32 ±0.20
	±60°	−1.27 ±0.20	−0.86 ±0.20	−0.70 ±0.20	−0.61 ±0.20	−0.54 ±0.20	−0.33 ±0.20	−1.57 ±0.20	−1.21 ±0.20	−0.93 ±0.20
	±90°	−1.20	−0.84	−0.58	−0.27	−0.64	±0.20	−1.42	−1.15	−1.10
+30°	0°	−0.38 +0.50	−0.50 +0.50	−0.50 +0.50	−0.50 +0.50	±0.20 +0.39	−0.25 +0.40	−0.60 +0.20	−0.30 +0.20	−0.25 +0.20
	±30°	−0.50 +0.75	−0.50 +0.55	−0.50 +0.40	−0.50 +0.45	±0.20 +0.41	±0.20 +0.26	−0.40 +0.55	−0.30 +0.50	−0.25 +0.47
	±60°	−0.14 +0.50	−0.50 +0.43	−0.45 +0.30	−0.40 +0.25	±0.20 +0.20	±0.20 +0.20	−0.81 +0.40	−0.89 +0.40	−0.83 +0.33
	±90°	−1.13	−0.94	−0.77	±0.20	−0.60	±0.20	−1.25	−1.15	−1.10
+45°	0°	+0.52	+0.50	+0.50	+0.60	+0.49	+0.70	+0.42	+0.40	+0.35
	±30°	+0.80	+0.78	+0.48	+0.55	+0.45	+0.45	+0.65	+0.60	+0.55
	±60°	+0.60	+0.45	+0.35	+0.30	+0.28	+0.21	+0.50	+0.50	+0.50
	±90°	−1.17	−0.76	−0.86	−0.33	−0.55	−0.28	−1.25	−1.15	−1.15
+60°	0°	+0.57	+0.57	+0.57	+0.80	+0.57	+0.80	+0.50	+0.50	+0.50
	±30°	+0.80	+0.79	+0.59	+0.62	+0.59	+0.62	+0.77	+0.77	+0.77
	±60°	+0.70	+0.47	+0.37	+0.35	+0.37	+0.35	+0.59	+0.59	+0.59
	±90°	−0.44	−0.44	−0.44	±0.20	−0.44	±0.20	−1.21	−1.21	−1.21
+75°	0°	+0.81	+0.81	+0.81	+0.81	+0.81	+0.81	+0.58	+0.58	+0.58
	±30°	+0.83	+0.83	+0.83	+0.73	+0.83	+0.73	+0.85	+0.85	+0.85
	±60°	+0.55	+0.55	+0.55	+0.41	+0.55	+0.41	+0.78	+0.78	+0.78
	±90°	−0.43	−0.43	−0.43	±0.20	−0.43	±0.20	−1.21	−1.21	−1.21

NOTE 1. Interpolation may be used between values of the same sign.
NOTE 2. Pressure change rapidly from negative to positive with increasing pitch between α = 15° and α = 30° and values for both signs are given.
NOTE 3. When interpolating between α = 15° and α = 30°, interpolate between negative values to give load case for upward load and interpolate between positive values to give load case for downward load.

Fig 3

- dot wind

once the upwind and downwind edge have been defined it is possible to examine each face of the mesh and determine the angles *alfa* and *teta (line 1671 to 1683).*

The way by which the pressure coefficient is assigned for each of the face of the mesh is described from line 1687 to 1707. First the position of the face is checked for determining in which zone of the building it lies. This is done simply by checking if its vertices fall beyond or behind the plane (or the planes) that divide the building into zones according to the normative *(fig1).* In this way we know in which "cpe double array" (a,b,c fig1 previous page) the pressure coefficient should be. *Alfa* and *teta* and relative cpe double array *(a,b,c line1695 t0 1702)* of the face are given to a sub routine called "get_cpe" in which two sorting mechanisms are performed. The first one will check which is the closest value to the current *alfa* value in the correspondent cpe double array. In this way it knows which of the seven nested arrays has to be scanned for getting the pressure coefficient. Performing again the same sorting mechanism, looking this time at *teta* value, will eventually allow to find the searched value *(line 1724 to 1748).*

This value is afterwards multiplied for the reference pressure in order to compute the pressure on the face.

Using the values of pressure for each face, it is possible to play with the RGB gradation and obtain the map of pressure on the entire building (line 1711 to 1720).

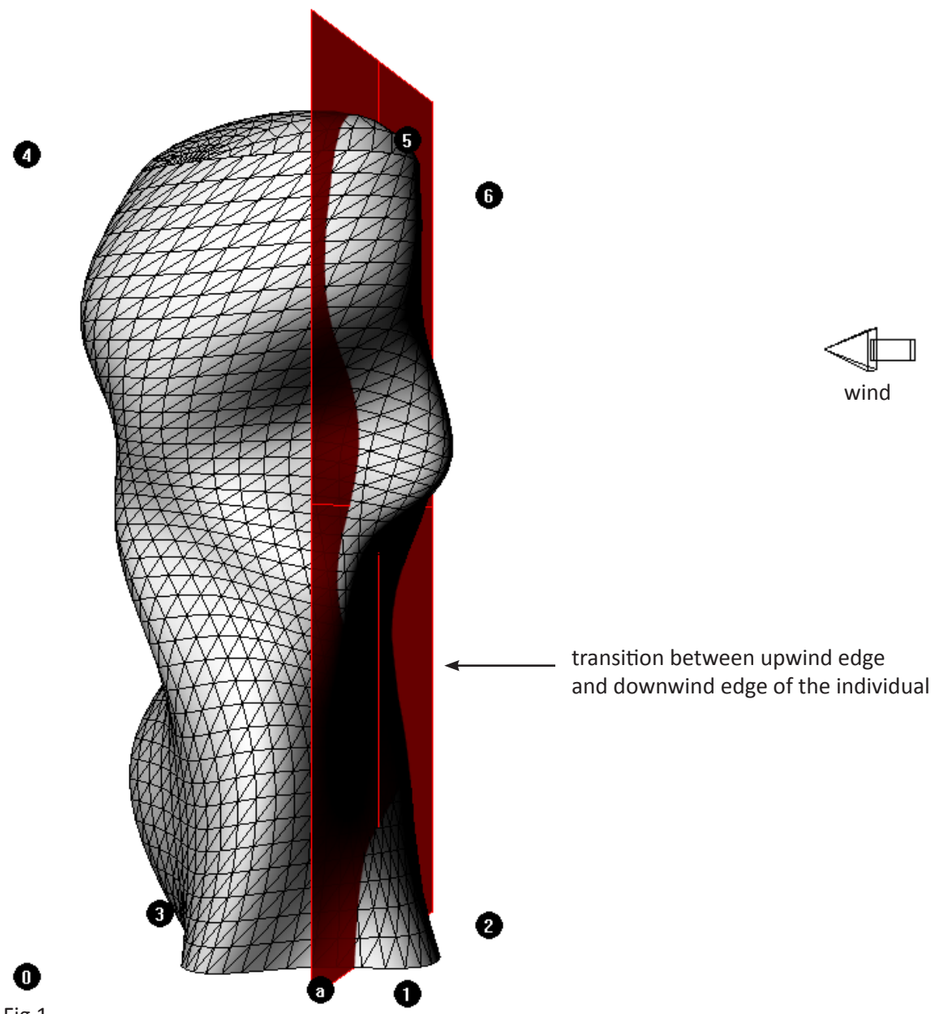


Fig 1

```
1664 Sub dot_wind(ByVal panels,ByVal panels_norm(),ByRef panels_pressure(),ByVal source_vectors_wind()_
1665     _,ByVal cpe_a,ByVal cpe_b,ByVal cpe_c(),ByVal a,ByVal b,ByVal index,ByVal generation)
1666
1667     Dim i,j,rgb_value
1668     Dim red,green,blue,incident_angle_beta,incident_angle_teta
1669     Dim cpe_value,alfa,beta,teta,panels_norm_teta(2),vertices
1670
1671     vertices=rhino.MeshVertices(panels)
1672
1673     For i = 0 To UBound(panels_norm)
1674
1675         panels_norm_teta(0)=panels_norm(i)(0)
1676         panels_norm_teta(1)=panels_norm(i)(1)
1677         panels_norm_teta(2)=0
1678
1679         For j = 0 To ubound(source_vectors_wind)
1680
1681             incident_angle_teta =rhino.VectorDotProduct(panels_norm_teta,source_vectors_wind(j))
1682             alfa=rhino.ACos(panels_norm(i)(2))*180/rhino.Pi
1683             teta=rhino.ACos(incident_angle_teta)*180/rhino.Pi
1684
1685             If isempty(b)=False Then
1686
1687                 If vertices(i)(index)>=a(index) Then
1688                     get_cpe  panels_norm(i),cpe_a,alfa,teta,cpe_value
1689                 Else
1690                     get_cpe  panels_norm(i),cpe_b,alfa,teta,cpe_value
1691                 End If
1692
1693             Else
1694
1695                 If vertices(i)(index)>=a(index) Then
1696                     get_cpe  panels_norm(i),cpe_a,alfa,teta,cpe_value
1697                 End If
1698                 If (vertices(i)(index)<a(index) And vertices(i)(index)>b(index)) Then
1699                     get_cpe  panels_norm(i),cpe_b,alfa,teta,cpe_value
1700                 End If
1701                 If vertices(i)(index)<=b(index) Then
1702                     get_cpe  panels_norm(i),cpe_c,alfa,teta,cpe_value
1703                 End If
1704
1705             End If
1706
1707             panels_pressure(i) =panels_pressure(i)+cpe_value*dynamic_pressure
1708
1709         Next
1710
1711         panels_pressure_norm=(panels_pressure(i)+2.75*dynamic_pressure)/(1.12*dynamic_pressure+2.75*dynamic_pressure)
1712
1713             If panels_pressure(i)<-0.79*dynamic_pressure Then
1714                 arrcolors(i)=rgb(0,0,0)
1715             Else
1716                 red=255 * panels_pressure_norm
1717                 green=255-255 * panels_pressure_norm
1718                 blue=255 * panels_pressure_norm
1719                 arrcolors(i)=rgb(red,green,blue)
1720         End If
1721     Next
1722 End Sub
1724 Sub get_cpe (ByVal panels_norm(),ByVal cpe(),ByVal alfa,ByVal teta,ByRef cpe_value)
1725
1726     Dim i,j,dif(),index_alfa,index_teta
1727     ReDim dif_alfa(7),dif_teta(12)
1728
1729     If panels_norm(2)>0 Then
1730
1731         For i=0 To 7
1732             dif_alfa(i)=array(abs(alfa-cpe(i,0)(0)),i)
1733         Next
1734
1735         sort_number dif_alfa
1736         index_alfa=dif_alfa(0)(1)
1737         Else
1738         index_alfa=0
1739     End If
1740
1741     For j=0 To 12
1742         dif_teta(j)=array(abs(teta-cpe(index_alfa,j)(1)(1)),j)
1743     Next
1744
1745     sort_number dif_teta
1746     index_teta=dif_teta(0)(1)
1747     cpe_value=cpe(index_alfa,index_teta)(1)(0)
1748 End Sub
```

- get fitness

in order to have a fitness value for each individual that describe the behaviour of this respect to wind, the pressure that acts on each of its face is compared with the value of the reference pressure. The faces whose absolute pressure value is bigger than a certain percentage of the reference pressure are counted. This number is divided over the total number of faces in the mesh for having a normalised value. The reason why is the absolute value to be taken into account is because the wind creates large area of negative pressure when investing a building.

Determining the efficiency of an individual respect to wind, not only implies the examining of the area where the wind impacts but also lateral and back side where the pressure in mainly negative. The so determined fitness represents, therefore , the percentage of faces whose absolute pressure value is bigger than a specified threshold giving information that regard the efficiency of the individual as a whole *(line 1848 to 1870).* To be more precise the “wind_fitness” of the individual is the result of the difference between 1 (which is the maximum) and the above mentioned value because our aim is to minimise the effect of the wind on the individual.

Fig1 (current page) *& fig2* (next page) show behaviours for buildings having different morphologies. The back side is the part where pressure in mainly negative with higher value of depression where the colour becomes black. On the upwind side, where the wind impacts, the colour is magenta representing a value close to the dynamic pressure (positive pressure). Where the colour is green|gray we are in transition zone from positive to negative . Negative pressure might occurs also at the upwind edge depending on the orientation of the face respect to wind.

It is rewarding to see how close is this map with the one obtained from a CFD simulation in Ansys as shown in *fig2 & fig3* next page.

```
1848  Sub get_wind_fitness(ByVal panels_body,ByVal panels_cap,ByVal panels_pressure_body(),_
1849      _ByVal panels_pressure_cap(),ByRef individual_wind_fitness,ByVal generation)
1850
1851      Dim i,j,k,l
1852      Dim max
1853
1854      max=ubound(panels_pressure_body)+ubound(panels_pressure_cap)
1855
1856      For i=0 To ubound(panels_pressure_body)
1857          If (abs(panels_pressure_body(i))>0.6*dynamic_pressure)  Then
1858              individual_wind_fitness=individual_wind_fitness+1
1859          End If
1860      Next
1861
1862      For i=0 To ubound(panels_pressure_cap)
1863          If (abs(panels_pressure_body(i))>0.6*dynamic_pressure)  Then
1864              individual_wind_fitness=individual_wind_fitness+1
1865          End If
1866      Next
1867
1868      individual_wind_fitness=individual_wind_fitness/max
1869
1870  End Sub
```

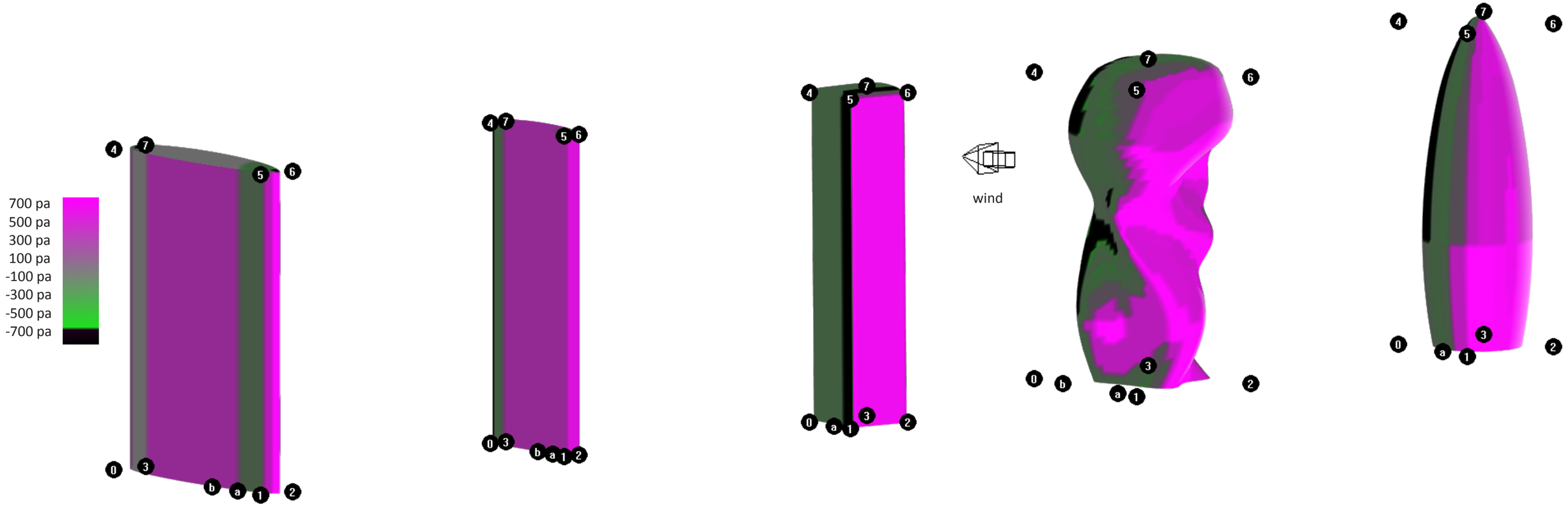
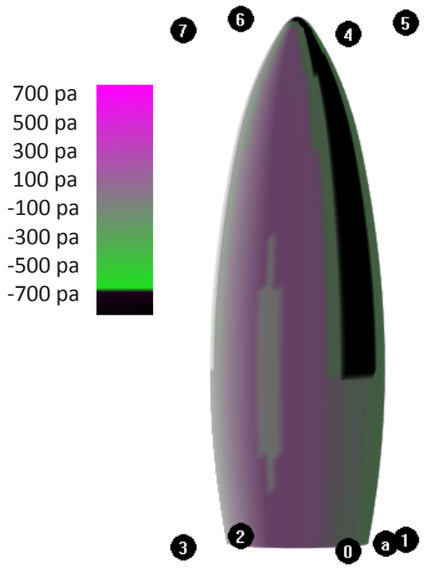
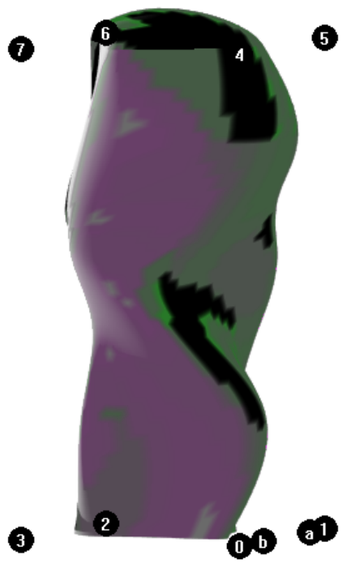
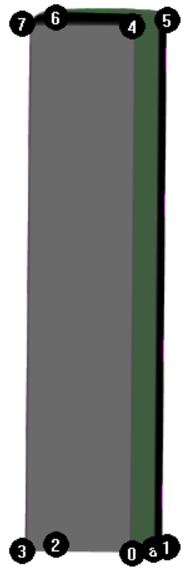


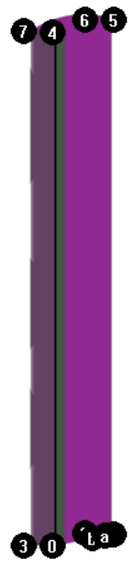
Fig1

perspective front view









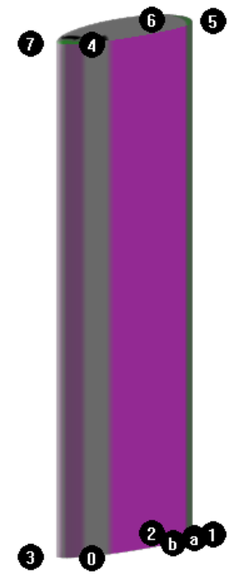
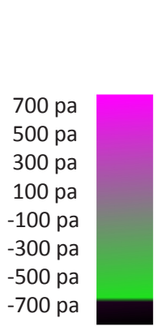
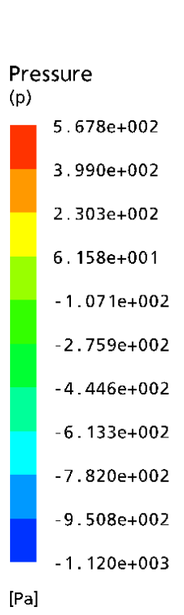


Fig1

perspective back view





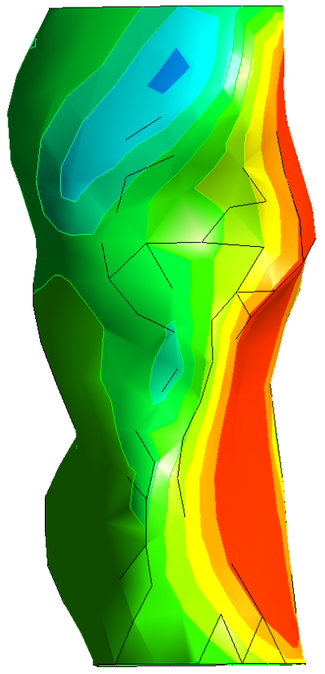


Fig2

perspective back view

perspective front view

Fig 3

front view, ansys CFD simulation

Normalizing the fitness parameters|fitness function

Considering that a genetic algorithm makes a parallel search into the solution domain working with an entire set of solutions at time, we need to avoid the risk of comparing values that are not mathematically comparable. The definition of a fitness function for leading the Genetic Algorithm in a specific direction is crucial. This function performs the evaluation of the individuals giving them a score which is afterwards translated into a probability to be selected. There are two main ways for doing this translation. The first one is called "relative fitness scaling" where the analysis values of the individuals are replaced by values relative to the distribution of fitness within the population. In this way the relative fitness values indicate how individuals perform respect to the population average. Each relative fitness is , after being weighted, recombined with the others for making the total individual's fitness. This procedure has the advantage that the over-importance of a parameter respect to others is avoided. However, due to the fact that the weights of the parameters are constantly varying, the searching of the algorithm might experience dramatic fluctuation.
The second procedure instead, which is the one that I adopt, provide the fitness function with a set of reference values. The fitness parameters of the individuals can be expressed as multiples, fraction or percentage of these.
Normalizing each fitness parameter respect to these reference values allows to evaluate in the correct way their contribution to the individual's fitness. In order to normalize them, we first need to understand what are the boundaries of the chosen solution domain. If the coordinate of each point is encoded in a string of 0&1 of length 6, it means that its value can vary from 1 to 63 (as already explained in 1st experiment). If the position of the points are given from a base point and they all happen to have the maximum value of 63, they will describe the section shown in *fig4*. In addition, considering that the distance between two sections it is controlled by genes, it can vary from 1 to 63 as well. If all the sections are spaced of the maximum value, we can calculate the maximum volume and maximum surface area possible for the size of the chosen solution domain *(fig4).*
This time for avoiding to have over pronounced kinks when lofting two sections that are very close to each other, the minimum distance between consecutive sections is set to 15. Similarly, the minimum value for the coordinates of points is set to 20 to avoid the individual to be too "skinny" in just one or two sections *(fig3).*
With the same consideration given for the maximum volume, it is possible to retrieve, from the chosen solution domain, all the information that we need for normalizing the fitness parameters:

- min and max area footprint
- min and max lateral surface area (Facade area)
- min and max volume
- min and max height of the volumetric centroid

$$(V/F\_A)min < V/(F\_A) < (V/F\_A)max$$

$$\frac{V}{F_A} - \left(\frac{V}{F_A}\right)min < \left(\frac{V}{F_A}\right)max - (V/F\_A)min$$

$$\text{Fitness norm} = \frac{\left(\frac{V}{F_A}\right) - \left(\frac{V}{F_A}\right)\min}{\left(\frac{V}{F_A}\right)max - \left(\frac{V}{F_A}\right)\min} < 1$$

$$\text{Fitness value} = \left(\frac{\left(\frac{V}{F_A}\right) - \left(\frac{V}{F_A}\right)\min}{\left(\frac{V}{F_A}\right)max - \left(\frac{V}{F_A}\right)\min}\right)^{x}$$

Fig1

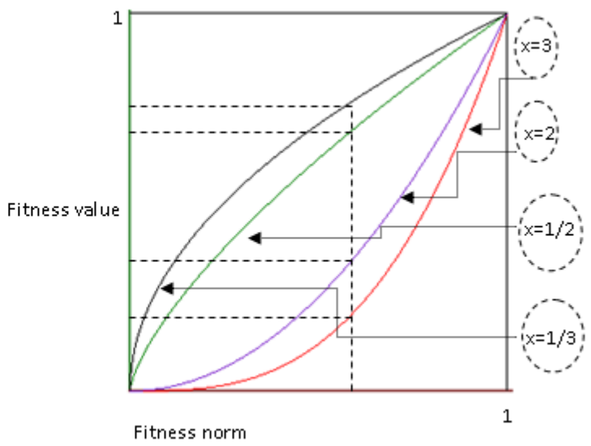


Fig2

The knowledge of these values allows for normalizing fitness parameters such as the ratio between Volume and Facade area as shown in *fig1.* Similarly this procedure can be applied to all the other fitness parameters and, in so doing, have them all in a range of 0&1 which is very useful for evaluating their contribution in percentage terms. It goes without saying that this procedure is vital when comparing parameters that have different dimensions and whose values would be not mathematically comparable.
After being normalized, fitness parameters are also raised at the power x which varies according to how sensible we want the value to be (*fig2)*. Raising at the power of 2 the normalized parameters will give the opportunity of making the fitness values sensible to small improvements when having high values of normalized fitness *(fig2).*
Although this procedure is very efficient for linking fitness parameters to solution domain, its major drawback is that the importance of certain parameters could be overemphasized respect to others in dependence of the magnitude of the reference value to which they are scaled (see page 39 "minimising|maximising").

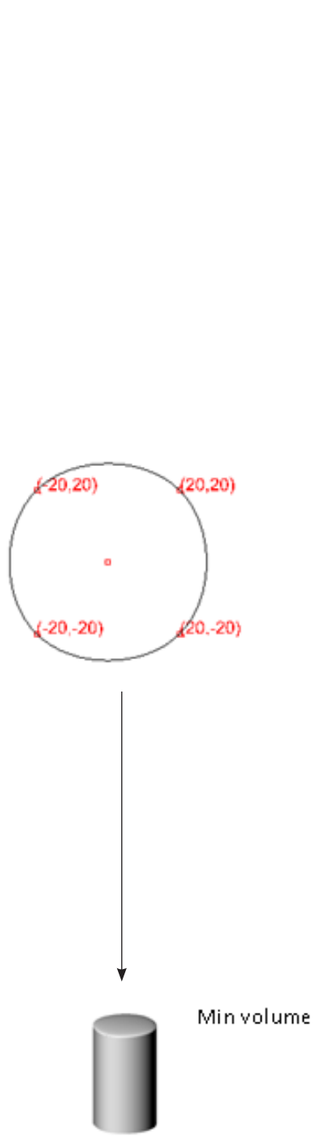


Fig3

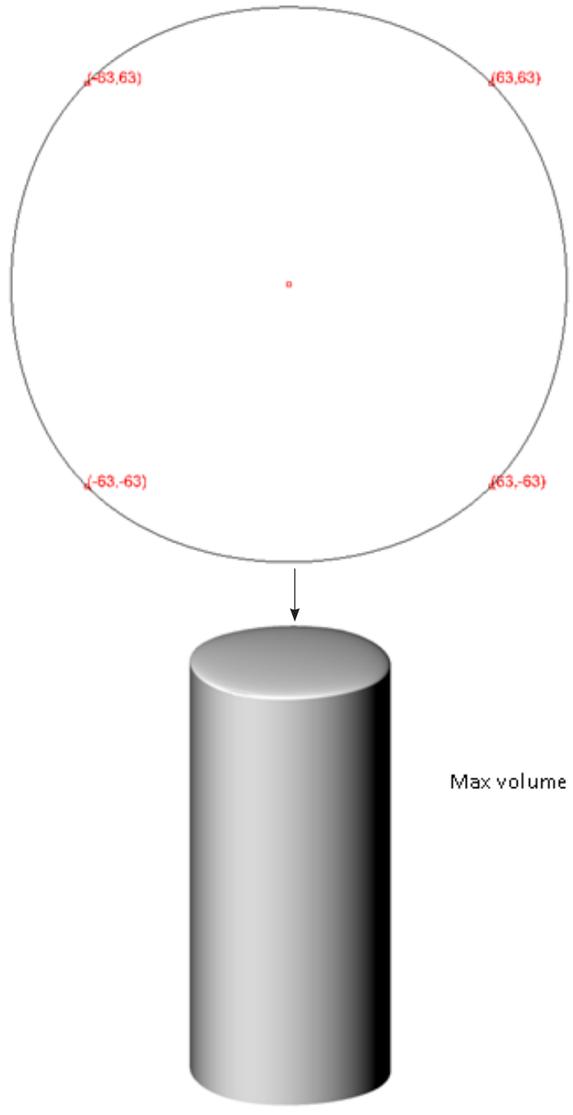


Fig 4

Volume over Footprint

The ratio between the volume of the individuals and their footprint is a fitness parameter whose optimization leads their morphology to assume traits that are similar to the ones shown by the truncated cone in *fig1*. Because of the influence of the other parameters they will never get to that shape though. The reason why I introduced this ratio is because its evaluation leads to have smaller footprint areas and the allocation of more volume at higher position (Manhattan function). Similarly to what we have seen for the ratio Volume over Facade area, the normalization of this parameter is done by dividing the difference between the value of the individual and the minimum one (according to the size of the chosen solution domain) over the difference between the maximum and the minimum value *(fig1* previous page). In this way it comes in a range of 0&1 which is vital for evaluating its contribution to the whole fitness.

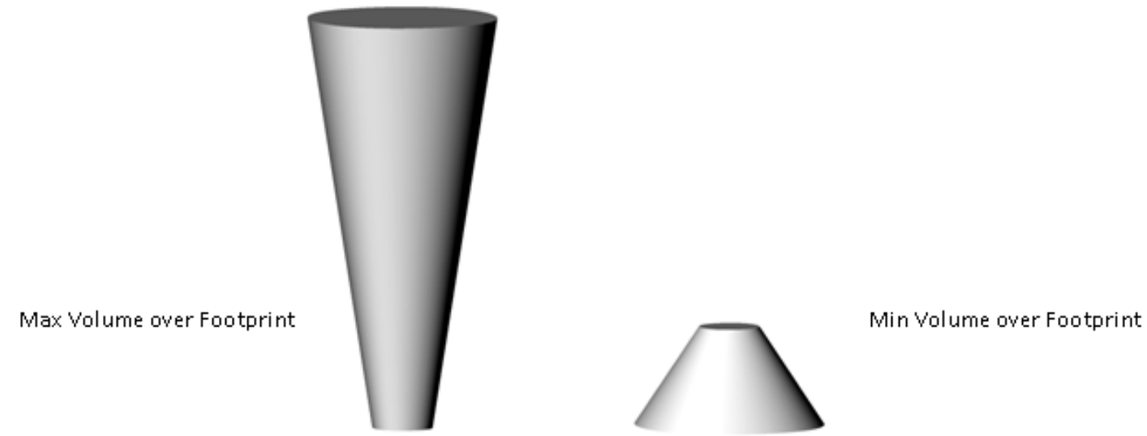


Fig1

Volume over Facade area

The ratio between the volume of the individuals and their lateral area, which can be regarded as the facade area, is a fitness parameter whose optimization, would lead their morphology to become as a sphere. Because of the influence of the other parameters and the way the points are assigned they will never get to a sphere though. The reason why I introduced this ratio is because it is an index of the efficiency of the individuals' morphologies. Considering that for a given volume there are many different shapes that can enclose it, Volume over Facade area gives an evaluation of the amount of resources (facade area) that an individual requires for doing it.

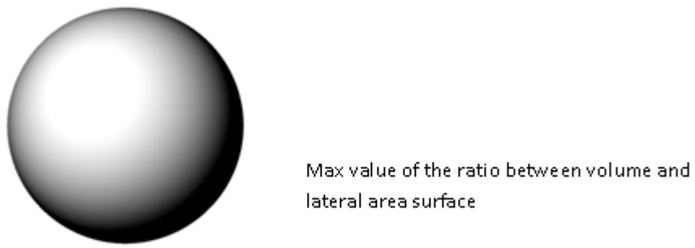


Fig2

Facade to Floors ratio

Facade to Floors ratio is the ratio between the total floors area, which is the sum of the area of each floor, and the relative facade area. It is essentially an index of economy efficiency being facades the most expensive part to build. It gives, therefore, a quick evaluation of the efficiency of the individuals' morphology. The meaning of this parameter can be also explained saying that it represents how much facade area is needed for enclosing a square meter of usable floor space [9].
This ratio mainly depends on the distance between two consecutive floors and on the shape of the building that define the facade area. Varying this distance influences to a great extent this ratio whose changes help to understand how a particular morphology performs from this point of view.
In this experiment I encoded the distance between two consecutive floors in the genes in order to have a set of values that starts from 3 meters and ends at 5 meters with interval of 0.25 meters. For taking into account the presence of cores, the total floors area has been reduced by 20 per cent of its original value.
Fig3 shows how the ratio Facade to Floors (FA_R) varies for different morphologies having the distance between two consecutive floors fixed at 5 meter sand a number of floors of 55.

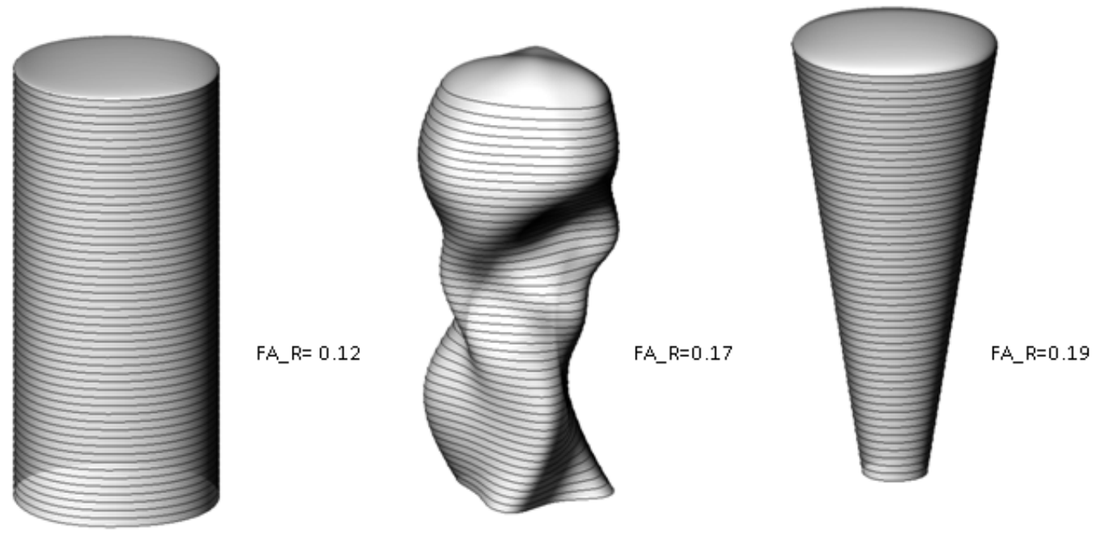


Fig3

Gravity

For simulating gravity as a condition of equilibrium the volumetric centroid of each individual is calculated. Once the position in space of the centroid is known, it is possible to check if the projection of this point falls within the first contour, which is the one the lays on the ground (*fig1).* If the projection of the centroid is not outside this contour, it means that the individual does not tend to lean over. Although this is a not a sophisticated procedure for equilibrium condition, it is very efficient because it decreases considerably the size of the solution domain, quickly discarding all the members whose morphology does not satisfy this criteria.

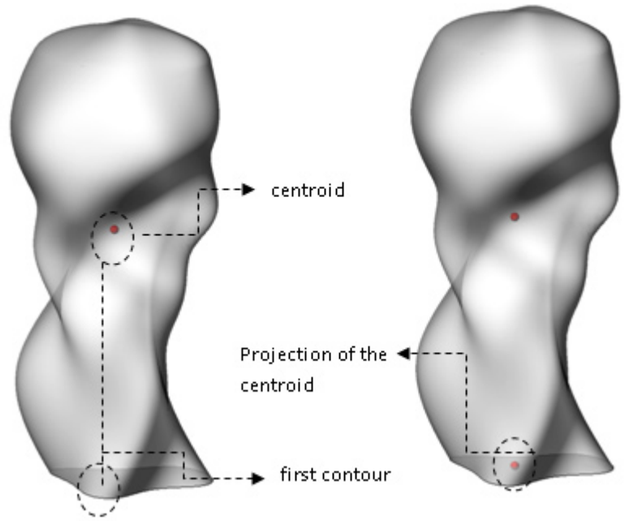


Fig1

Height of volumetric centroid

The height of the volumetric centroid is a parameter that evaluates how the volume is allocated along the height of the individual. As already explained for the others fitness parameters there is a minimum and a maximum height for the centroid according to the chosen size for the solution domain. It is obvious to say that the more the centroid is high, the more the volume is allocated at higher position. It is a very simple and effective way for ensuring that individuals will tend to increase their height and the allocation of volume at the upper side. *Fig2* shows the morphology that present minimum and maximum value for this parameter.

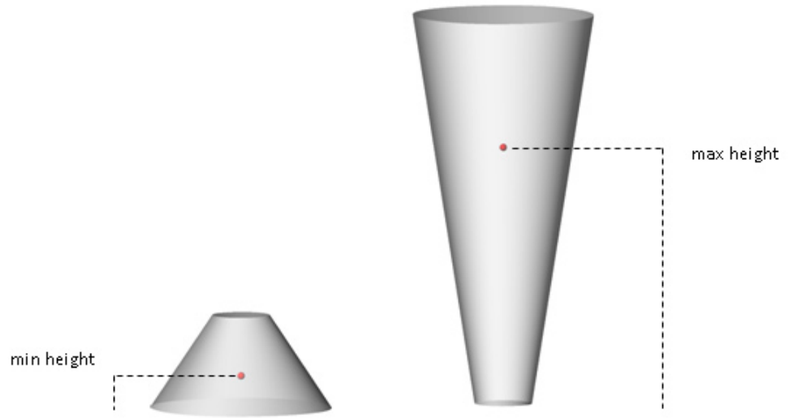


Fig2

Minimum radius of curvature

Evaluating the minimum radius of curvature of the surface of the individuals allows checking whether there are over pronounced local discontinuities on it. The main reason why I introduced a parameter related to this value lies in the fact that too small radius of curvature cause difficulties when meshing the surface. The other reason is that in terms of feasibility, it would be surely impossible to build a truss like structure that follows the curvature of the surface if there are kinks or very small radii of curvature.
Because there is no limit to such a value that can be derived from the size of the solution domain, I fixed this value to be not smaller than 0.1 meters (10 cm) which is a reasonable value when thinking of connections.
Fig2 shows a map representing values of minimum radius of curvature on the surface of an individual. Red colour correspond to high values and blue to small ones.

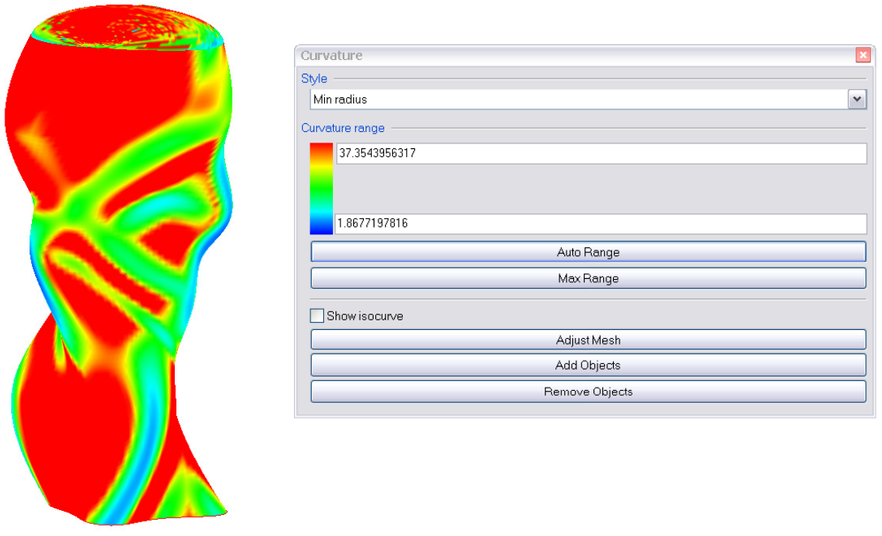


Fig3

The algorithm|pseudo code

In this paragraph I comment the main subroutine of the code which is made by other several subroutines that perform particular actions and return the necessary information to the main. The explanation of the main gives a general understanding of the whole procedure while the rest of code can be found at the end of the document in appendix A. Part of the subroutines, that i refer to in this paragraph, have already been commented in previous experiment or in the previous pages of this chapter. This general comment can be regarded as the *"pseudo code"* for the procedure.

pseudo code

- declaration of the variables and encoding of the necessary information for "sun analysis" and "wind analysis" such as latitude, longitude of the site - orbit of the sun - wind direction, reference pressure etc.

- *making the genome*
for all the individual in the first population by assigning array of strings of random 0&1 numbers whose size depends on the dimension of the chosen solution domain *(line 276).*

- *generation 1 to max number of generations*
after this point we are in the main loop which starts by redefining all the parameters every time a new generation is created in order to free memory and set them for the current generation *(line 280).*

- *individual 1 to max number of individuals*
nested loop in which each individual of the current generation takes shape out of its genome *(line 308).*

- *forms builder*
this subroutine is linked to a several other subroutine whereby it is possible to draw the form of the current individual. Everything starts from a point cloud which is generated by using the information encoded in its genome *(line 320).*

- *first selection|gravity*
the position of the volumetric centroid of each individual is examined in the way we explained when describing the fitness parameters. The result of this analysis are translated in a boolean variable, that has two values True and False, with which we alter the state of the individual. If its morphology responds negatively its state is set to False which means that will be excluded by any other performance evaluation. It will obviously not transmit its genetic information to next generations *(line 327).*

- *surface analysis*
the curvature of the surface that describe the individual is analysed for checking the location and the magnitude of the minimum radius of curvature *(line 342)*. This values is used for assigning the fitness parameter that concerns the geometric limits for the individual.

- *Facade to Floors Ratio*
given the distance between two consecutive floors, the area for all the floors of the building is calculated. This value is afterward used for computing the **FA_R** ratio and assigning the correspondent fitness criteria *(line 350)*. It is worth outlining that the area of the floors is computed without drawing the surface that would describe them. This would take to much memory and slow down the whole procedure. One of the **theorem of the divergence** formulated by **Gauss** is used for accomplishing this task. With this method we need only the positions of points that describe the contours of the floors and through a series of operations ,based on vectorial calculus, we are able to retrieve the desired value. This procedure is illustrated in Appendix A page 67 line (1009 to 1067)

```
270     Randomize (seed)
271     startpoint=array(0,0,0)
272
273     '----------- making the gemome ----------
274
275     For i=1 To max_pop
276         make pop_genome(i)
277     Next
278
279     generation=1
280     For generation=1 To maxgens
281
282         rhino.AddText "generation"&generation,array(startpoint(0)-300,startpoint(1),startpoint(2)),25,,1
283
284         ReDim population(max_pop)
285         ReDim population_temp(max_pop)
286         ReDim population_state(max_pop)
287         ReDim population_volume(max_pop)
288         ReDim population_base_area(max_pop)
289         ReDim population_heightofcentroid(max_pop)
290         ReDim population_height(max_pop)
291         ReDim popfitness(max_pop)
292         ReDim population_cap(max_pop)
293         ReDim population_min_curv_radius(max_pop)
294         ReDim population_facade_area(max_pop)
295         ReDim population_solar_gain(max_pop)
296         ReDim population_wind_fitness(max_pop)
297         ReDim population_floors_tot_area(max_pop)
298         ReDim Vol_over_Facade(max_pop)
299         ReDim vol_over_Footprint(max_pop)
300         ReDim HeightCentroid(max_pop)
301         ReDim Facade_to_Floors_Ratio(max_pop)
302         ReDim panels_body(max_pop)
303         ReDim panels_cap(max_pop)
304
305         sumfitness(generation) = 0
306
307         i=1
308         For i=1 To max_pop
309
310             popfitness(i)=0
311             population_state(i)=True
312
313             '----------- decoding the genome ----------
314             decode pop_genome(i), values
315
316             startpoint(0)=startpoint(0)+1000
317             rhino.AddText "pop"&i,array(startpoint(0),startpoint(1)-180,startpoint(2)),20,,0
318
319             '----------after decoding the genome it is possible to draw the elements of the population using different methods --------
320             shapemaking startpoint,values,population_temp(i),population_height(i),population_state(i),generation
321
322             If population_state(i)=True Then
323
324                 population(i)=population_temp(i)(0)
326                 '----------- evaluation-fitness_function ----------
327                 first_selection population(i),population_state(i),population_volume(i),population_footprint(i),population_vol_centroid(i)
328
329                 If population_state(i)=True Then
330
331                     cameraheight=rnd*8
332                     camera_x_rnd=rnd*4
333                     camera_y_rnd=rnd*5
334                     For j=0 To 90 Step 1
335                         camera(0)=startpoint(0)-camera_x_rnd*100+800*cos(2*j*rhino.Pi/180)
336                         camera(1)=startpoint(1)-camera_y_rnd*100-1200*sin(j*rhino.Pi/180)
337                         camera(2)=startpoint(2)+500*sin(j*rhino.Pi/180)
338                         Rhino.ViewCameraTarget ,camera,startpoint
339                     Next
340
341                     capsurf population(i),population_cap(i)
342                     analysis population(i),population_volume(i),population_facade_area(i),population_min_curv_radius(i)
343
344                     '''''''' solar analysis''''''''''
345                     tri_facets  population(i),panels_body(i),panels_norm_body,num_u_body,num_v_body,startpoint
346
347                     If panels_body(i)<>False Then
348
349                         get_floors panels_body(i),startpoint,population_floors_tot_area(i)
350                         Facade_to_Floors_Ratio(i)=population_facade_area(i)(0)/population_floors_tot_area(i)
351
352                         tri_facets_cap  population_cap(i),panels_cap(i),panels_norm_cap,num_u_cap,num_v_cap
353
354                         ReDim panels_heat_body(ubound(panels_norm_body))
355                         ReDim panels_heat_cap(ubound(panels_norm_cap))
```

- mesh

the surface that describe the geometry of the individuals is replaced by a mesh by means of the subroutine called “tri_facets” *(line 345)* which has already been examined in this chapter when explaining sun and wind analysis. The mesh will be fundamental for retrieving geometric information such as the normal vectors for each triangulated face and perform the analysis for solar gain and wind exposure. In addition, working with mesh objects speed up the whole procedure because they are much lighter than surface objects in terms of required allocated memory.

- solar_gain

after the mesh has been done, we can perform the analysis for solar gain and assign the value for the correspondent fitness parameter *(line 357)*. Throughout the whole procedure the array in which we store information such as normal vectors to the faces of the mesh, their correspondent pressure or heat value have to be redefined for allocating new memory for next individual *(line 362 or 384 etc.)* .

- wind_exposure

using the same mesh wind analysis is performed. As already explained, it is first necessary to determine the position of the upwind and downwind edge with the subroutine called “bounding box” and divide the body of the individual in different zone following the British Starndard ENV 1991-2-4. It is afterwards possible to perform wind analysis and assign the value for the correspondent fitness criteria *(line 370 to 382)*.

- fitness function

once all the necessary information for computing the 7 fitness parameters have been gathered, we can use these to feed the “fitness function” where they are combined in order to compute the “individual’s fitness” *(line 395 to 402).* I think it is worth to examine this subroutine in the details and, therefore, I remind later its description after different methods, whereby it can be implemented, will be analysed.

- indvidual after individual

these steps repeat iteratively until the maximum number of individual is reached (which is decided by the user). In this way we create our first generation and assign for each of its individual the correspondent fitness value.

- preparing for selection procedure

we need to prepare the field for selection. For doing this, the values of the individuals’ fitness have to be summed *(line 427)* in order to make the fitness of the whole generation. This value is used by the “Goldberg roulette wheel” for choosing the individuals stochastically as already seen in the 1st experiment.

- preparing for breeding procedure

the allocation of array in which fitness values and genome (array of strings of 0&1) are stored in order to free them for next generation and prepare for the breeding procedure *(line 429).*

- checking

in order to avoid that the computer crashes, we have constantly check that everything is stored in correct order and in the appropriate array. Sometimes there might be missing information that causes serious troubles for the stability of the procedure. This is avoided by the introduction of several check operators that will act as filters not allowing incomplete information to be computed. In addition, along with the possibility of having missing information due to problems with built-in methods of the software, the individual might not respect some of the conditions imposed by gravity simulation, curvature limits or might self intersect. Also in this case the check operators quickly discard the wrong configuration.

```
357             dot_solar panels_body(i),panels_norm_body,panels_heat_body,source_vectors_sun,generation
358             dot_solar panels_cap(i),panels_norm_cap,panels_heat_cap,source_vectors_sun,generation
359
360             get_sun_solar_gain panels_heat_body,panels_heat_cap,population_solar_gain(i)
361
362             Erase panels_heat_body
363             Erase panels_heat_cap
364
365             '''''''''wind analysis''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
366
367             ReDim panels_pressure_body(ubound(panels_norm_body))
368             ReDim panels_pressure_cap(ubound(panels_norm_cap))
369
370             bounding_box panels_body(i),source_vectors_wind,a,b
371             dot_wind panels_body(i),panels_norm_body,panels_pressure_body,_
372                 _source_vectors_wind,cpe_a,cpe_b,cpe_c,a,b,index,generation
373             dot_wind panels_cap(i),panels_norm_cap,panels_pressure_cap,_
374                 _source_vectors_wind,cpe_a,cpe_b,cpe_c,a,b,index,generation
375
376             get_mesh_faces_area panels_body,panels_cap,mesh_faces_area_body,_
377                 _mesh_faces_area_cap
378             get_wind_fitness panels_body(i),panels_cap(i),panels_pressure_body,_
379                 _panels_pressure_cap,population_wind_fitness(i),generation
380             get_wind_force  panels_body,panels_cap,panels_pressure_body,_
381                 _panels_pressure_cap,panels_norm_body,panels_norm_cap,_
382                 _mesh_faces_area_body,mesh_faces_area_cap,num_u_body,num_v_cap
383
384             Erase panels_norm_body
385             Erase panels_norm_cap
386             Erase panels_pressure_body
387             Erase panels_pressure_cap
388
389             Erase a
390             Erase b
391
392             ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
393             'calcualte fitness
394
395             fitness_function popfitness(i),population_volume(i),population_footprint(i),_
396                 _population_vol_centroid(i),population_height(i),_
397                 _population_facade_area(i),population_min_curv_radius(i),_
398                 _population_solar_gain(i),population_wind_fitness(i),_
399                 _Facade_to_Floors_Ratio(i),i,generation
400
401             Vol_over_Facade(i)=population_volume(i)(0)/population_facade_area(i)(0)
402             vol_over_Footprint(i)=population_volume(i)(0)/population_footprint(i)(0)
403             HeightCentroid(i)=population_vol_centroid(i)(2)
404
405             rhino.AddText "Fitness="&CStr(popfitness(i)),_
406                 _array(startpoint(0),startpoint(1)-250,startpoint(2)),20,,0
407             rhino.AddText "Facade_to_Floors_Ratio="&CStr(Facade_to_Floors_Ratio(i)),_
408                 _array(startpoint(0),startpoint(1)-300,startpoint(2)),20,,0
409             rhino.AddText "Wind_Fitness="&CStr(population_wind_fitness(i)),_
410                 _array(startpoint(0),startpoint(1)-300,startpoint(2)),20,,0
411
412         Else
413             popfitness(i)=0
414             rhino.AddText "wrong solution",array(startpoint(0),startpoint(1)-200,startpoint(2)),20,,1
415         End If
416
417     Else
418         popfitness(i)=0
419         rhino.AddText "wrong solution",array(startpoint(0),startpoint(1)-200,startpoint(2)),20,,1
420     End If
421
422 Else
423     popfitness(i)=0
424     rhino.AddText "wrong solution",array(startpoint(0),startpoint(1)-200,startpoint(2)),20,,1
425 End If
426
427 sumfitness(generation)=sumfitness(generation)+popfitness(i)
428
429 oldpop_genome(i)=pop_genome(i)
430 oldpop_fitness(i)=popfitness(i)
431
```

- free memory
once again keeping it light it is imperative for the stability of the procedure, which is why after having done the mesh and the analysis we delete the surface that describe the geometry of the individual. Considering that we still have the mesh describing its geometry and that we can always rebuild the nurbs surface this seams a reasonable choice *(line 433 to 449)*.

- set mutation_rate
the mutation rate plays a very important role when trying to avoid local maximum which is a common problem in genetic algorithm. The development of the mutation rate is also crucial for having a good degree of differentiation within a generation. The way I set this rate is made for ensuring maximum exploration in the solution domain at fist generations and afterwards let the rate follow the fitness trend and respond dynamically to its changes. If the fitness trend goes always upward, the mutation rate will slowly increase in order not to ruin good solutions, while if the trend becomes downward, it will decrease for allowing further exploration in the solution domain. We can play with different function for doing this and set how sensible we want the mutation rate to be according to the variation of the fitness trend *(line 454 to 470)*. It is worth saying that, for the way I implemented this function, the higher is the value of the mutation rate the smaller in the probability of having mutation in one of the genes and vice versa.

- natural selection
as already seen in the first experiment, the individuals that have to transmit their genetic information to next generation are chosen stochastically by using Goldberg weighted roulette wheel. In this way the fitter individuals are more likely to be chosen but also the less fit ones have a probability to be selected. The main reason for this is because they might have informations that can turn to be useful generation over generations. The individuals are chosen two per time until the maximum number of them is reached *(line 473 to 478)*.

- crossover|breeding
in the breeding procedure genome of the chosen pairs of individual are chopped in a random position and swapped over for creating the genome of new individuals *(line 480)*.

- mutate
after having set the mutation rate, mutation might occur in the genome of the newly created individuals *(line 484)*.

- generation over generation
everything said above repeats as many time as the maximum number of generations is or it stops when a certain stability in the fitness is reached. If after a certain number of generation there is no further improvement, the procedure will stop automatically.

- monitoring
in order to monitor the trends of all the fitness parameters and other kind of information such as mutation_rate, size of the solution domain etc. can be exported automatically to an excel sheet. This is essential for understanding the history of the evolution and for analysing the solutions.

```
433         If (generation<maxgens)  Then
434
435             If isempty(population(i))=False  Then
436                 If isnull(population(i))=False Then
437                     rhino.DeleteObject population(i)
438                 End If
439             End If
440
441             If isempty(population_cap(i))=False  Then
442                 If isnull(population_cap(i))=False Then
443                     rhino.DeleteObject population_cap(i)
444                 End If
445             End If
446
447         End If
448
449     Next
450
451     If generation < maxgens Then
452         i=1
453
454         '''''''''''''''''''' set mutation_rate''''''''''''''''''''
455         If generation>1 Then
456
457             If sumfitness(generation)>sumfitness(generation-1)  Then
458                 mutation_div=sumfitness(generation)
459                 operator=1
460             Else
461                 mutation_div=sumfitness(generation-1)
462                 operator=-1
463             End If
464
465             mutation_rate=mutation_rate+operator*((abs(sumfitness(generation)-sumfitness(generation-1)))/mutation_div)^mutation_sens_factor
466
467             If mutation_rate>1 Then mutation_rate=1
468             If mutation_rate<0 Then mutation_rate=0
469
470         End If
471
472         '---------- natural selection Goldberg roulette ----------
473         For i = 1 To max_pop - 1 Step 2
474
475             mum = roulette(sumfitness(generation),oldpop_fitness)
476             dad = roulette(sumfitness(generation),oldpop_fitness)
477
478             '---------- crossover ----------
479
480             crossoverSplice oldpop_genome(mum), oldpop_genome(dad), pop_genome(i), pop_genome(i + 1)
481
482             '---------- mutation  --------------
483
484             If Rnd > mutation_rate Then
485                 mutate pop_genome(i)
486                 mutate pop_genome(i + 1)
487             End If
488
489         Next
490
491     End If
492
493     For j=0 To max_pop*10
494
495         camera(0)=startpoint(0)+300-80*j
496         camera(1)=startpoint(1)-1000
497         camera(2)=startpoint(2)+300
498         target(0)=800
499         target(1)=startpoint(1)
500         target(2)=startpoint(2)
501         Rhino.ViewCameraTarget ,camera,target
502
503     Next
504
505
506     rhino.Print "gen="&CStr(generation)&"   gen fitness="&CStr(sumfitness(generation))
507     rhino.print "mutation_rate="&CStr(mutation_rate)
508
509 Next
510
511 excel_output sumfitness,Vol_over_Facade_mean,vol_over_Footprint_mean,Facade_to_Floors_Ratio_mean,_
512     _HeightCentroid_mean,min_curv_radius_mean,solar_gain_mean,wind_fitness_mean,mutation_rate,words,scale_noise_factor
513
514 rhino.EnableRedraw True
515
516 End Sub
```

Driving the evolution

The evaluation of performances of the generated configurations produces a set of informations (fitness parameters) which are used for giving the Genetic Algorithm a direction in its searching. There are several ways by which this can be achieved according to the context of application. Controlling the direction of optimization could mean not only to maximise or minimise a parameter but also to examine its intermediate values. It is worth saying that the most creative step in which the designer should interact with this system is in the formulation of these parameters, the way they are used and the understanding of their interconnections. At the highest level of abstraction the definition of the fitness parameter and the design of means for influencing their influence can be considered as qualities that we want to embed in the examined topology, which will be represented by their emergent morphology.

Minimazing | Maximizing

The simplest operation to control the direction of a fitness parameter is to decide whether it should be maximised or minimised. It has to be said that, because we have reference values for scaling these parameters, some of them may be constantly scaled to less significance respect to other ones [4][3]. This is due to the fact that when normalizing the fitness parameters, they are scaled according to their correspondent reference value which is directly related to the solution domain. In this way it can happen that due to the dimension of its reference value, a parameter is scaled to a number (percentage of fitness) that are much smaller respect to the others. Therefore, their weight in the building of the individual's fitness value can be underestimated. In order to overcome this drawback we can assign a weight factor for this parameter higher than the other ones and raise its normalized value at the power of "x" which will further increase it *(fig1 x=2).*
For minimizing a normalised parameter we use the function "1-fit_norm" and afterwards this sum is raised at the power of x. This would give high fitness value when having small value of the relative parameter.

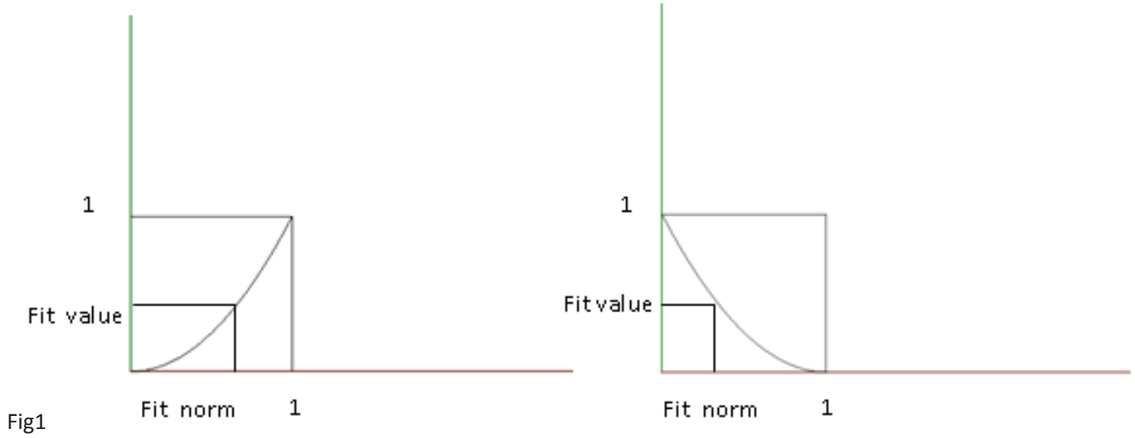


Fig1

Target

The objective is not always to minimize or maximise a parameter but also to observe the behaviour of the individuals respect to a specific condition. This condition can be represented by a numerical value, target, against which the performance of the individual can be compared. It goes without saying that there might be several targets, each of these related to its correspondent parameter. The fitness value in this case is the higher when the analysis value lies closer to the target. It is like observing the solutions with a Gaussian function and try to redistribute their qualities around a mean and having a certain deviation.

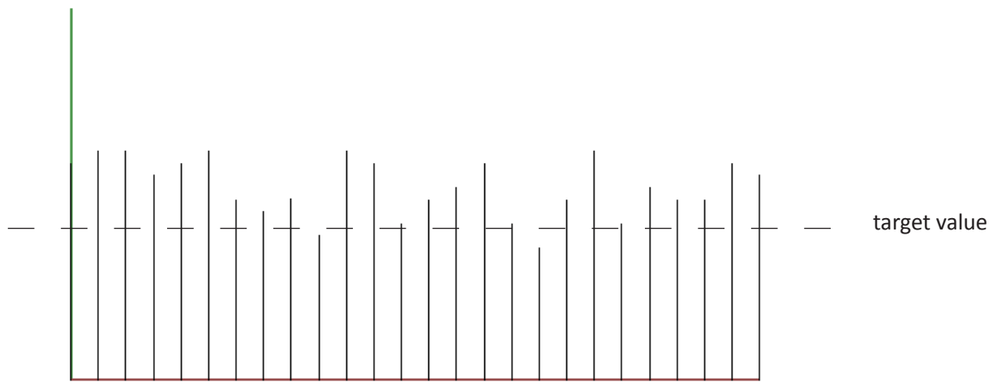


Fig2

Standard deviation

For evaluating the uniformity of fitness parameters or its variation from an average value we can build a Gaussian curve using the value of each individual in a population *(fig1)*. The deviation can be maximised or minimised according to how wide we want the distribution to be. Having a small deviation might ensure good performance for a small group of individual over generation but will enhance the risk of reaching local maximum. A local maximum is a condition where the Genetic Algorithm hardly manages to step out. In this condition, it is not able to understand whether the current configuration performs maximum qualities respect to its solution domain or only respect to previously reached values. If the standard deviation has to be minimised, narrowing the solution domain for having faster convergence, it should be coupled with a high mutation rate for not "playing always with the same cards" [3][4]. The mutation rate can be also set to be dynamically following the trend of the fitness. In this way high rate of mutation are provided when fitness is at low values and small rate when fitness is at high values for not ruining good solutions.

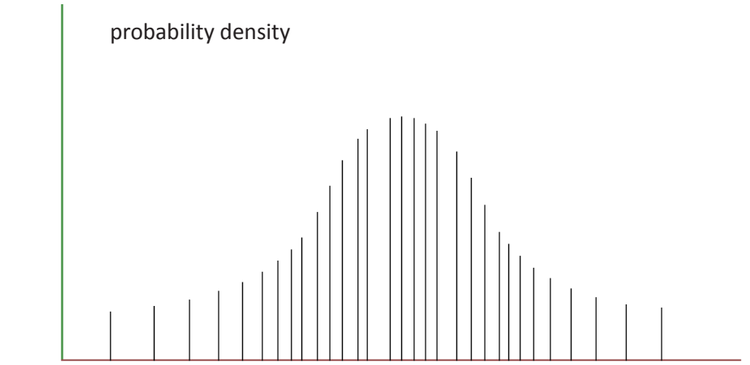


Fig 1

Distribution Graph

For controlling the fitness, a predetermined target distribution function can be encoded. The values of fitness are compared with the ones of the target distribution and the sum of the differences is evaluated as measure of deviation from it. This sum, which in the graph can be represented as the area in between the current distribution and the target one, can be minimised in order to be closer to desirable values (*fig2)*. This procedure can be also used to set a dynamic scaling of the weight for the fitness parameters which can be , in so doing, weighted according to specified range of values.

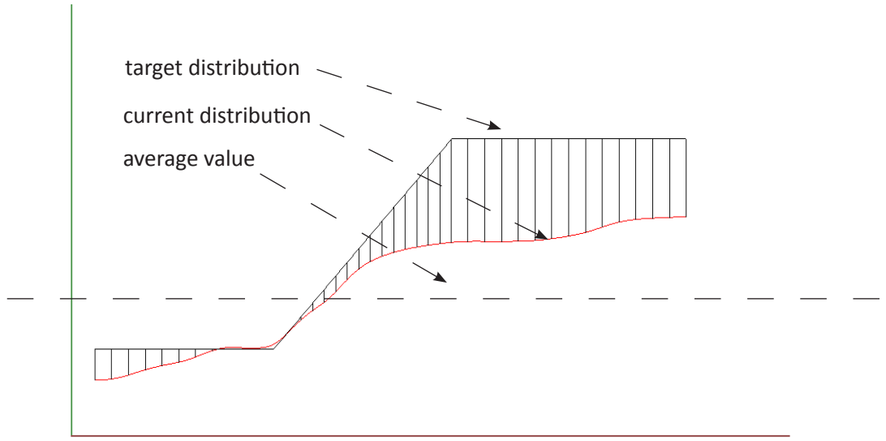


Fig 2

Distribution Map

Extending the distribution graph in two dimension lead to have a three dimensional function for the fitness value. In this way 2 dimensional array of target value can be specified which allows for studying the relation between two fitness parameters at time. Also in this case the values of the normalized fitness parameters are compared with the ones of the target distribution and the sum of the differences, which in this case represents a 3 dimensional space, is evaluated. This sum can be minimised in order to for the fitness to be closer to desirable values (*fig3)*.

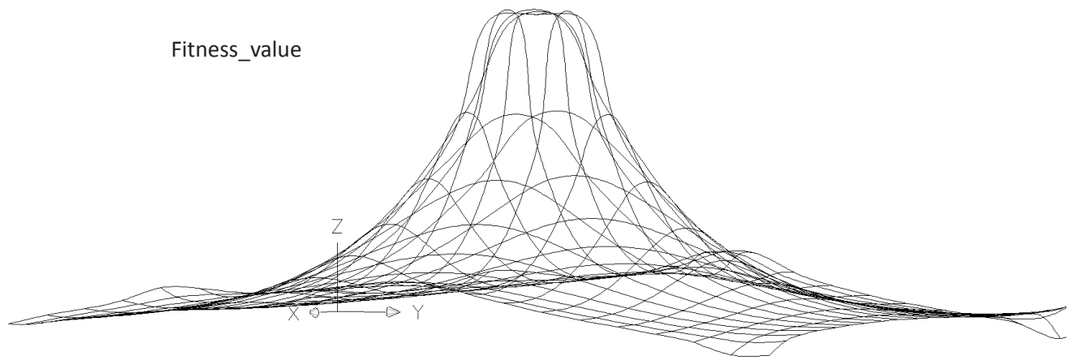


Fig 3

Fitness Function

As already anticipated when describing the “pseudo-code”, we present the detailed explanation of the fitness function, which is probably the most important part to be implemented.
Once all the fitness parameters have been computed, they are given to the fitness function in order to determine the individual’s fitness. There are three main steps :

- *normalise*
reference values for each parameter is calculated in a separate procedure before running the algorithm. Being the strategy for driving the evolution the one of minimising or maximising the fitness parameters, we have to calculate the minimum or maximum value for each of those. These values vary according to the dimension of the solution domain that we want to investigate. The size of the solution domain should be decided according to the characteristics of the topology and the available computational recourses. In general too vast domains are not advisable while a good understanding of the appropriate size for it should be undertaken before running the system.
Using this reference values allows to normalise *(line 1988 to 2002)* the fitness parameters as already seen in the paragraph “normalising the fitness parameters”.

- *sensibility*
in order to have more control on the parameters we can raise their normalised value to the power of “x” *(fig1 & fig2)* , which give us the possibility of helping the ones that are scaled to less significance to bring their contribution and to make their variation sensible in a desired range *(line 2004 to 2026).* For instance, after running several experiment one might notice that the normalised value of a certain parameter falls always in a low range of values. In this case we want to make its variation sensible in this range in order to appreciate small differences between individuals whose fitness is very close. For serving this scope we can use values for x such as 1/2 or 1/3 when the fitness norm (normalised fitness parameter) has very small values generation over generation, x=2 or x=3 in the opposite case *(fig1 & fig2).*

- *weighting*
the fitness values are ready to be weighted which is the main control that we have for driving the evolution of our individuals under specific purposes *(line 2028 to2030)*. The assignment of an appropriate set of weights is not a trivial task though. A possible strategy for their determination is explained in the paragraph “understanding the weights”. Generally speaking the weights can be set for enhancing the contribution of a parameter (which represent its performance under a specific design criteria) while minimising other ones for studying to what extent the morphology of the individual is affected by it.

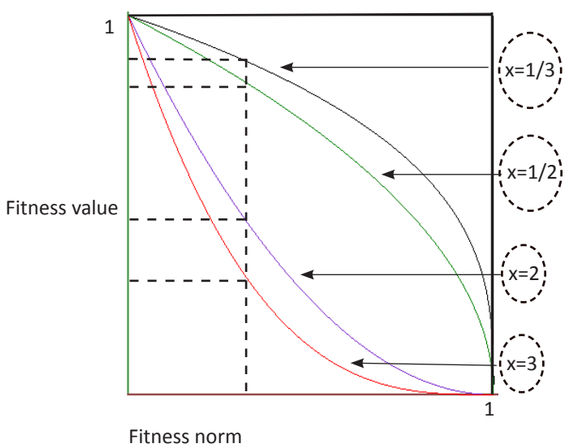


Fig1

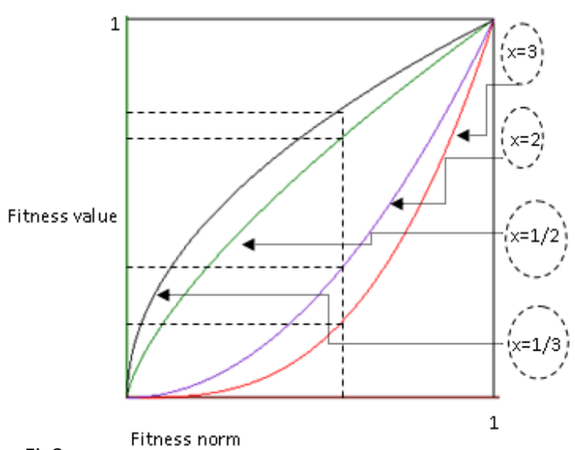


Fig2

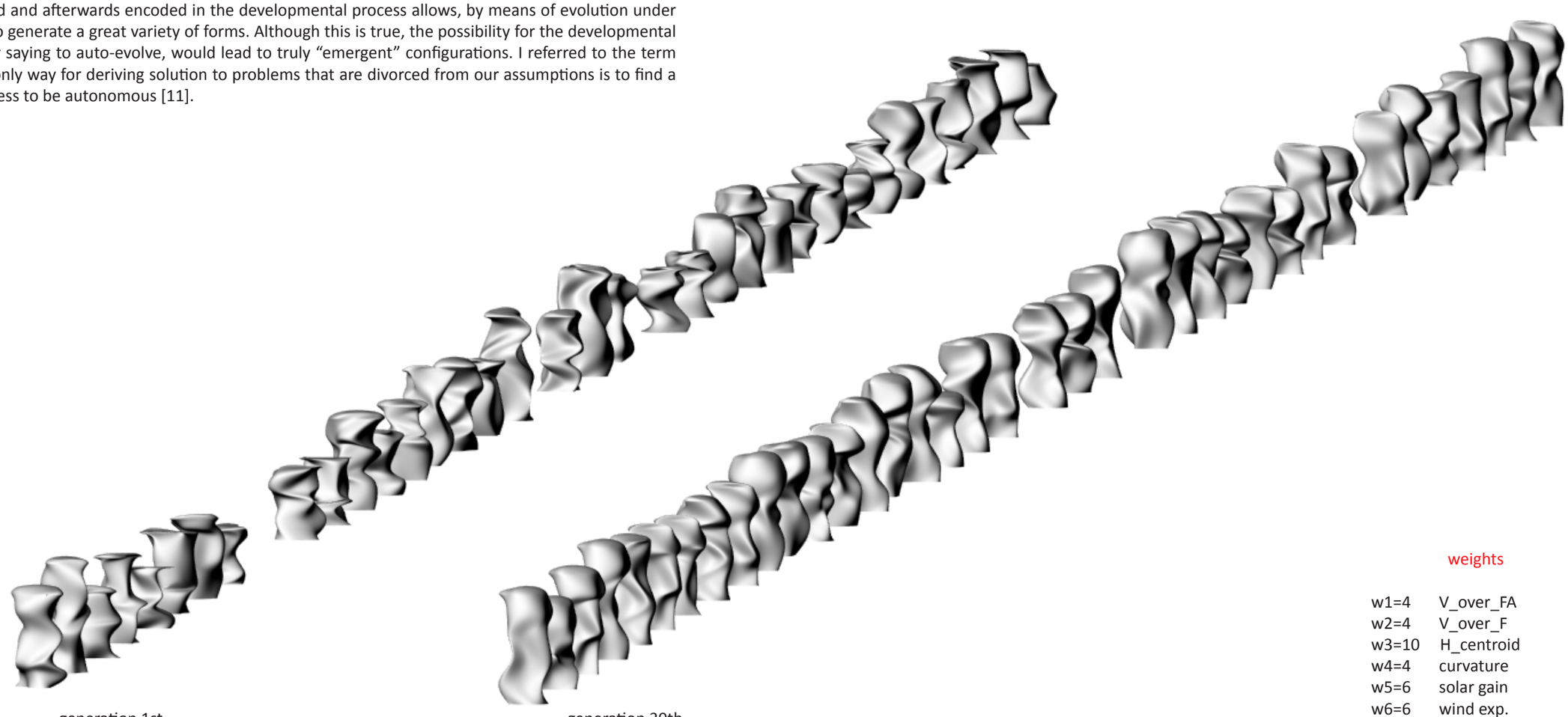
```
1976  Sub fitness_function (ByRef individual_fitness,ByVal individual_volume,ByVal individual_footprint,_
1977      _ ByVal individual_vol_centroid,ByRef individual_height,ByVal individual_facade_area,_
1978      _ ByVal individual_min_curv_radius,ByVal individual_solar_gain,ByVal individual_wind_fitness,_
1979      _ ByVal Facade_to_Floors_Ratio,ByVal who,ByVal generation)
1980
1981      Dim V_over_FA,V_over_FT,V_over_FA_norm,V_over_FT_norm
1982      Dim vol_norm,footprint_norm,facade_area_norm
1983      Dim centroid_height_norm,centroid_height
1984      Dim height_norm,max_curv_norm,max_curv
1985      Dim facade_to_floors_ratio_norm,tot
1986      Dim f1,f2,f3,f4,f5,f6,f7
1987
1988      'norm height_centroid
1989      centroid_height_norm=(individual_vol_centroid(2)-min_volume_centroid_height)/_
1990          _(max_volume_centroid_height-min_volume_centroid_height)
1991      'norm curvature
1992      max_curv=1/individual_min_curv_radius
1993      max_curv_norm=max_curv/max_curv_limit
1994      'norm FA_R
1995      facade_to_floors_ratio_norm=facade_to_floors_ratio/facade_to_floors_ratio_max
1996      'norm height
1997      height_norm=individual_height/max_height
1998
1999      V_over_FA=individual_volume(0)/individual_facade_area(0)
2000      V_over_FT=individual_volume(0)/individual_footprint(0)
2001      V_over_FA_norm=(V_over_FA-V_over_facade_area_min)/(V_over_Facade_Area_max-V_over_Facade_Area_min)
2002      V_over_FT_norm=(V_over_FT-V_over_Footprint_area_min)/(V_over_Footprint_area_max-V_over_Footprint_area_min)
2003
2004      f1=(V_over_FA_norm)^0.5
2005      f2=(V_over_FT_norm)^0.5
2006
2007      ''''''Height of Centroid '''''
2008      f3=centroid_height_norm^2
2009      ''''' max_curv ''''''
2010      If abs(max_curv_norm)<1 Then
2011          f4=(1-abs(max_curv_norm))^2
2012      Else
2013          f4=0
2014      End If
2015
2016      ''''' solar_gain''''''
2017      f5=(individual_solar_gain)^2
2018      ''''' wind_fitness'''''''
2019      f6=(1-individual_wind_fitness)^2
2020
2021      '''' Facade_to_Floors_ratio
2022      If facade_to_floors_ratio>1 Then
2023          f7=0
2024      Else
2025          f7=(1-facade_to_floors_ratio_norm)^2
2026      End If
2027
2028      ''' percentage fitness'''''
2029      tot=w1+w2+w3+w4+w5+w6+w7
2030      individual_fitness=((w1*f1+w2*f2+w3*f3+w4*f4+w5*f5+w6*f6+w7*f7)/tot)*100
2031
2032  End Sub
```

Evolution

Fig1 shows the first and the last generation of one of the numerous experiments that have been run over the course of this research. What is clearly evident, looking at the picture, is the striking difference between traits of members of the 1st generation and the ones of the 30th generation. Although they all share the same topology the characteristics traits of the last generation can not be inferred from the ones of the first one. The final morphologies are the result of a continuous remodelling of the form of the individuals under the indirect influence of the fitness parameters. Their traits have not been consciously encoded but are the result of a process that starts with an abstract representation of their topology. After several generations it manages to translate the influence and the contrasting relations of the parameter that govern the evolution into shapes. Looking at the set of weights that has been assigned for this experiment, it can be seen that the influence of parameters concerning spatial organization such as “Height_of_Centroid” are dominant over the others. The height of the individuals of the 30th generation is bigger than the one of the 1st generation and the majority of them features a pronounced increase in volume along the height. In the majority of them the abrupt changes in the curvature featured by the individuals of the 1st generation are no longer present. The reason for that can be found in the influence of the above mentioned parameters whose main action is to favour the allocation of volume at the top side of the individuals as well as an increase in height.

Although this can be regarded as a successful result, the understanding of the influence of the fitness parameter and their interconnections have yet to be explored. The type of optimization that this procedure seeks to reach, does not lead to the fulfilment of the optimum for the parameters singularly taken. It tries to gain a balanced compromise between these which, most of the times, tend to balance out as the improvement of one lead to spoiling another one.

Consideration should also be given to the way I implemented the generative system. The developmental process is embedded in the algorithm and can not be subjected to neither modification nor evolution. The abstract representation of this topology that I first imagined and afterwards encoded in the developmental process allows, by means of evolution under certain design constraints, to generate a great variety of forms. Although this is true, the possibility for the developmental process to evolve , or better saying to auto-evolve, would lead to truly “emergent” configurations. I referred to the term “auto-evolve” because the only way for deriving solution to problems that are divorced from our assumptions is to find a way for developmental process to be autonomous [11].



generation 1st

generation 30th

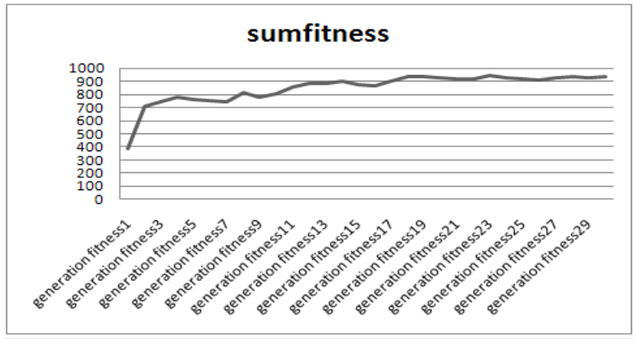
weights

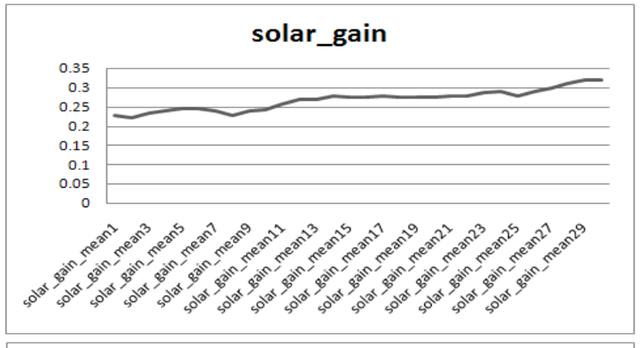
w1=4	V_over_FA
w2=4	V_over_F
w3=10	H_centroid
w4=4	curvature
w5=6	solar gain
w6=6	wind exp.
w6=8	FA_R

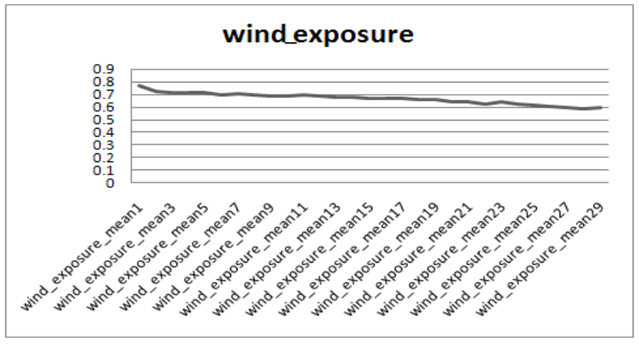
Monitoring the evolution

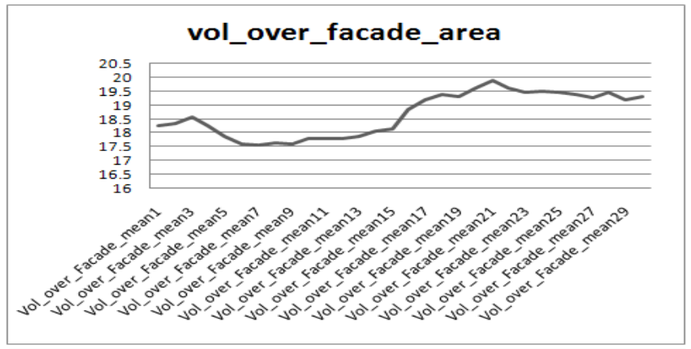
Are here reported the monitoring of the trends for the fitness parameters over generations. The trends shows the mean values of the fitness parameters for each generation representing in this way the overall progress. The individuals for this simulation have a planar configuration of sections. This means that there are only two genes or numerical value each point which are x and y coordinates. The domain has been set to *"words 7"* which means that each coordinate is encoded in a string constituted by 7 bits. After its decoding , therefore, it can assume a value from 1 to 255, as already explained in the 1st experiment.

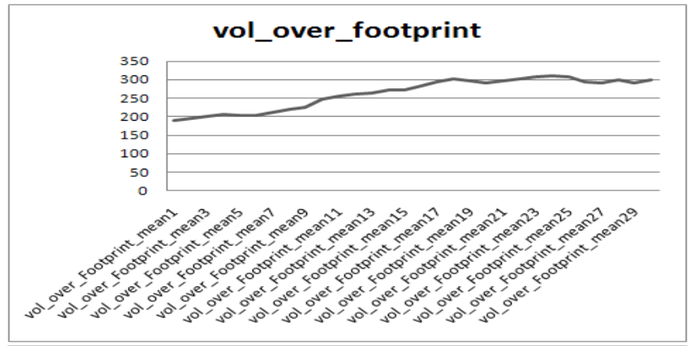
The mutation rate starts from a value of 0.1 for the initial generation which means that there is the 90 per cent of probability of having mutation within a gene. The rate follows the trend of the *sumfitness* adjusting dynamically its value according to it. *Sumfitness* is the mean for each generation of the individuals' fitness resulting from the weighting procedure. When the fitness trend stabilises, around the 15th generation, it assumes a value of 1 which means that there are no more possibility of having mutation. In this way there is a maximum exploration of the solution domain at the outset of the searching rather than at the end when a mutation might ruin a good found solution.

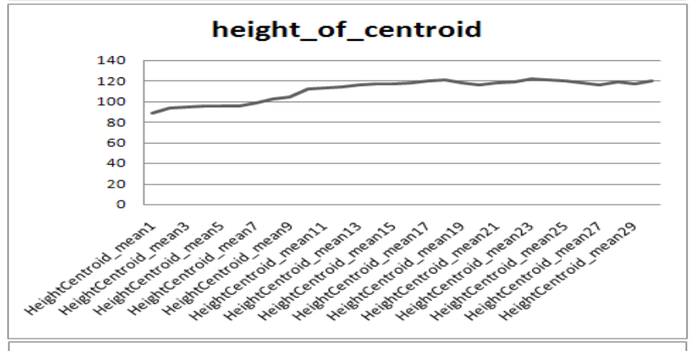


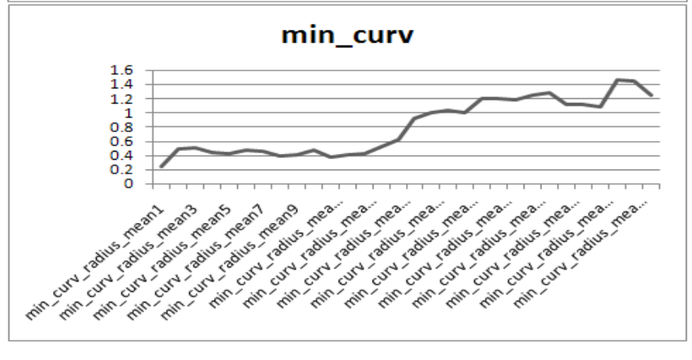












Understanding the weights

When having different fitness parameters the designer needs to decide a proper set of weights according to their design criteria. A weight is just a number that can be used for guiding the evolution with the fitness criteria that are more important for the designer. Before assigning the set of weights we should know , within the scope of a specific experiment, to what extent a fitness parameter contributes to the whole individual's fitness. When starting exploring a topology it is worth doing a set of experiments with one fitness parameter at time in order to draw a benchmark against which will be possible to confront the performance of the individuals when having more than one parameters. For instance, *fig1* shows a set of weights where the parameters related to the influence of the physic environment are dominant over the course of the evolution.

weights

w1=5	V_over_FA
w2=2	V_over_F
w3=3	H_centroid
w4=2	curvature
w5=10	solar gain
w6=8	wind exp.
w6=3	FA_R

Although the weighting procedure allows the system to be responding to different architectural scenario and to guide the evolution under different design constraints, the decision of the value of the weights can not be made without the knowledge, at least at qualitative level, of the degree of the influence of the correspondent fitness parameter. It is necessary to be able to examine and understand the interconnections between the fitness criteria in order to navigate safely the solution domain. In case there were only two parameters, it exists a range of best solutions according to the importance that each parameter has in relation to the individual's fitness value. In layman terms, there is a combination of their correspondent weight that can generate very good solutions. This consideration becomes much more relevant when having several parameters whose influence often balances out the contribution of the other ones. There are two main strategies that we can deploy for deciding the set of weights :

- dynamic response weighting
- fractional factorial design

With the first strategy it is possible to vary the intensity of the weights when the algorithm is running. This can be done in a way that, each iteration, the weights are scaled in reference of their mean value. In so doing we help the fitness parameter that are scaled to less significance in the normalization procedure. In addition it is possible to assign some rules to the weighting. For instance if we are interested in studying the response of our individual to a sub-set of fitness parameters, their weights can be coupled with rules concerning the time of their activation, proportionality or inverse proportionality [4][4]. As already outlined the best way for performing a Multi objective optimization would be to use a *Multi Objective Evolutionary Algorithm* procedure **MOEA** such as **NSGA II, SPEA II, DMOEA** etc. This goes for now beyond the scope of this research.
The second strategy, explained in the details in next paragraph, is based on a filed of statistics that regards the analysis of problems where a set of independent variables is connected to one or more global outputs of the process in which they are defined. The aim of this analysis is to attempt to understand which are the most influent variables related to a particular outcome that we want to study. For instance in the case of this 3rd experiment we have 7 fitness parameters with 7 correspondent weights whose intensity and mutual relations determine different result in individual's fitness. In addition, although this variables are formally independent, there are strong relations between some of those that yield to have different unpredictable results in the global outcome. For instance parameter such as "Volume_over_Footprint" of "Height_of_Centroid" might have a positive or negative influence on others parameter such as "Wind_exposure" or "Solar_Gain" or viceversa. The attempt of this analysis is, therefore, to gain the knowledge of the dependency of the outcomes from these variables, which ones are the most significant and which ones produce noise disturbing the direction of the evolution. It is worth saying that with this strategy it is possible to examine not only the mutual interconnections between fitness parameters in relation to the individual's fitness but also in relation to a particular fitness criteria. This allows the system to have flexibility according to the particular scenario in which it might be used.
In the next two pages I introduce , very briefly, the fundamental concept of "fractional factorial design" and explain how this method can be used as supporting tool in order to decide an appropriate set of weights which suits the realm of different context of application.

Factorial Fractional Design

Factorial design is a way of conducting an experiment when there are two or more independent variables that have a set of discrete possible values. These are tested in all possible combination in order to understand the their effects on the response of the system [12]. Experimenting all possible combinations would obviously give a full understanding of the main effects and of the internal relations between two or more variables but if the numbers of factors is high this method is not feasible time wise. To overcome this problem it is possible to carefully choose subset or fraction of the domain of a full factorial design for gaining an understanding of the most important features of the problem studied and limiting the amount time that this requires. The most important steps in experimental design which was first formulated by Sir Ronald Fisher are:

- *Comparison*

Factorial design is mainly based on the result that come out of a set of experiment in order to understand the effect of the variables which imply the creation of a standard environment to refer to, over the course of the analysis [12][2].

- *Randomization*

The philosophy of this method is to study the behaviour of a system not deterministically but rather probabilistically. This can yield to an understanding of it that goes far beyond any predefined theory whose constraints might limit the exploration of it. However, this method implies the running of a set of experiments that has a cost in terms of time and resources [12][3].

- *Replication*

The replication of the same experiment leads to an understanding of the non controllable factors that can cause noise in relation of the global outcome [12][4].

- *Blocking*

The arrangement of similar subset of experiments within the total set of runs that have to carried out yield to the understanding of the irrelevant source of variation between the variables. If there is a particular subset of variables of interest it is possible to design a block of experiment where the nuisance factors are held constant and in so doing the variation of the factor of interest can be studied with more precision [12][5].

- *Orthogonality*

Orthogonality in experimental design refers to the possibility of having each experiment completely independent from the other ones which ensure the gaining of different information for each of those. It is not always possible to have orthogonality and the fundamental requirement is that the values of the variables have to be normalized [12][6].

- *Analysis of variance*

The analysis of the variance of the global outcome can be done partitioning its domain into components directly related to specific factor in order to test the influence of this on the outcome. This is the step where we eventually understand the contribution of different factors separately or in combination which give a map of information regarding the most significant and less relevant variables involved in the process [12][7].

Important contribution to this field was made by a Japanese engineer Genichi Taguchi who introduced new ways of conceptualizing an experiment, such as parameter design and tolerance design, along with fractional factorial design methods that have been extensively used by U.S, India and Japan in a wide range of manufacturing industries.
In this method I see the possibility of gaining a qualitative understanding of the relations between our fitness parameters and the global outcome. This understanding act as a support when choosing the set of weights for the parameters in relation to the particular scenario that one want to investigate.

From full to fractional

Our task is the gaining of a qualitative understanding of the effects of the fitness parameters and their internal relations on the individuals' fitness. In order to understand how this method can be applied for our task, I describe in this paragraph how the number of runs in an experiment where there are n independent variables can be fractionated to a small number with the utilization of factorial fractional array.
The simplest full factorial design consist in having three variables (factors) with two values each +1 and -1 representing the high value and the low value respectively. A design with all possible combination require 2^k runs, where k is the number of factors and 2 the number of values for each factor, which means that for our simple design we need to perform 8 runs. The number of runs required using full factorial can increase very fast if the number of factors and values should augment. Considering that in our case we have 7 weights to assign with 10 possible values each ,the number of runs would be 10^7 which would take an infinite amount of time considering the resources that we have [12][8].
Going back to our simple experiment we can represent a full factorial as shown in *fig1* where X1,X2,X3 are the factors, the arrows show the direction of increase for the values of the factors and the numbers stand for the order of runs. For instance at run 1 all three factors are at their low level and instead at run 8 they all are at their high level [12][9].
If we run all the possible combination and we store all the observation values and the level of the factors we can build a table as shown in *fig2* where Y is the observation. The right-most column of the table indicate the responses measured for the experiments. If a first order effect *C1*, such as the influence of X1 has to be investigated, we have to compute the average of response (observations) at all runs with X1 at the high level minus the average response of all runs with X1 at low level :

$$c_1 = (1/4)\,(y_2 + y_4 + y_6 + y_8) - (1/4)(y_1 + y_3 + y_5 + y_7)$$
or
$$c_1 = (1/4)(63+57+51+53) - (1/4)(33+41+57+59) = 8.5$$

For reducing the numbers of run to do we can build trough a technique called "blocking" a fractional array out the the full one as shown in *fig3.* In this array we do not take into account the runs represented by the full black circles in this way reducing of a half the number of the experiment to do[12][10].
Computing the same first order effect on the observation as previously done with the full factorial gives:

$$c_1 = (1/2)\,(y_4 + y_6) - (1/2)\,(y_1 + y_7) \text{ or}$$
$$c_1 = (1/2)\,(57+51) - (1/2)\,(33+59) = 8.$$

The values of the effect for X1 are very similar which demonstrates the effectiveness of the factorial fractional. It is worth saying that although we economise on the number of runs there is also a price that we have to pay. This price is called in the terminology of the field "confounding" and means that when doing a fractional factorial some of the main effects are confused with some of the second order effects such as the interaction between X1 and X2 often represented with the symbol X1*X2 [12][11]. Considering this we have to carefully choose our fractional factorial according to the purpose of the investigation and possible assumptions that can be done before setting the experiment. Usually lower resolution fractional factorial are used for examining first order effects considering them more important than the others, while higher resolution ones allow to consider also second order effects.
Fractional factorial are available on the internet and in many books where one can find even the description of the "confounding" and , therefore, choose the most appropriate array for their problem [12][12].
With regard to our experiment we have to cope with 7 factors 10 levels each. In order to economise further more on the number of runs to do, we have to abandon the idea of using all possible values of our factors and reduce them to at most 3 level. For each factor (weights) we chose to have three possible value 1,5,10 which are the minimum, medium and maximum value. Even with this assumption the number of experiment that would take for performing a full factorial would be 3^7 or 2187 different runs which is not feasible for our resources. Considering that we are interested in investigating principally main effects of the weights on the global outcome and that we can always shift the observation from the individuals' fitness to one of the other fitness criteria, a three-level fractional factorial seems the appropriate choice. This design requires 27 runs after which screening operation mainly based on *Taguchi* methods allows to understand main effects of our factors (weights) on the individuals' fitness, or to be more precise on the mean of these values for each generation. These information will be afterwards used in next experiments for tuning the weights according to the purpose of the exploration.

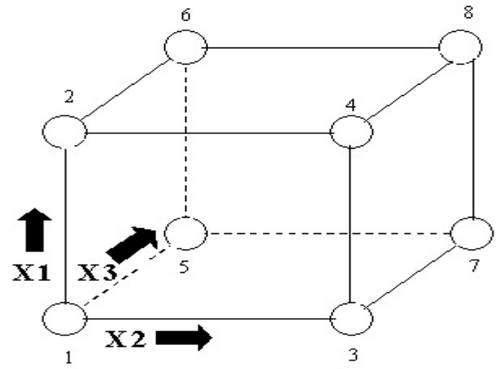


Fig1

	X1	X2	X3	Y
1	-1	-1	-1	$y_1 = 33$
2	+1	-1	-1	$y_2 = 63$
3	-1	+1	-1	$y_3 = 41$
4	+1	+1	-1	$Y_4 = 57$
5	-1	-1	+1	$y_5 = 57$
6	+1	-1	+1	$y_6 = 51$
7	-1	+1	+1	$y_7 = 59$
8	+1	+1	+1	$y_8 = 53$

Fig2

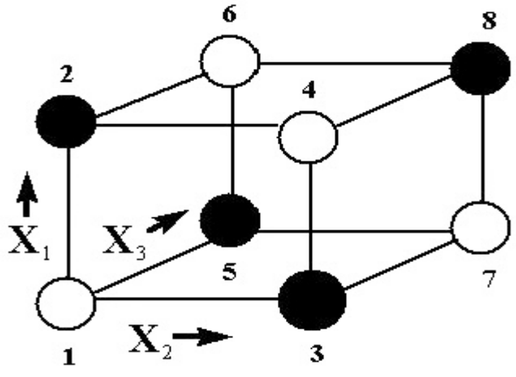


Fig3

As already said in the previous paragraph our aim is to investigate the main effects of the fitness parameters on the global outcome represented by the mean of the individuals' fitness for each generation. This will derive a set of possible combinations of their correspondent weights that yields to optimum solutions. With regard to our procedure it must be said that there are some noise factors that are not controllable, which are mainly caused by the mutation rate. Our program can generate same sequence of random numbers which are the initial values for our design variables. It is worth remembering that the design variables are the coordinates of the points that are used for building the geometry of our the individuals each generation. Even if two experiments share the same sequence of initial variables, it is not ensured that the result will be exactly the same. The reason that causes this phenomena lies in the continuous adjustment of the mutation rate which follows the trend of the fitness trying to avoid local maximum. In order to evaluate the main effects of the weights we need to take into account the presence of this nuisance. One of the main contribution of *Taguchi* to the field of experimental design was the introduction of the noise factors as integral part of the observation when experimenting the combination of the level of the controllable factors. Noise factors are responsible for causing the functionality of the process to deviate from target value. Instead that using the standard deviation as a measurable value in order to find the set of combination of the values of the controllable factors that yields to optimum solutions, *Taguchi* introduced the **Signal to Noise ratio** [13]. This is an index of how the nuisance can affect the performance and its evaluation free our design experiment from the presence of noise. However, also in this case there is a price that we have to pay. In order to evaluate our process with S/N (Signal to Noise ratio) we have to repeat each single run more than one time. Considering our resources, each run can be repeated not more than three times which brings the numbers of simulation to be done at 27*3=81. Each simulation takes approximately 15 minutes (with 30 individuals each population and a maximum of 30 generations) with other additional 5 minutes for examining the results and calculating the Signal to Noise we reach 20 minutes. 20 minutes*81 times = 1620 minutes which are 27 hours. Even if this number might seem excessive it is worth remembering that we are trying to understand the main effects of the parameters of a 7-dimensional problem whose solution domain is a permutation of 120 (number of variables for individual) * 5 (size of the string of bits in which each variable is encoded) = **600 factorial** which can be considered an infinite number.

Going back to the meaning of S/N ratio in practice its main advantage is that its evaluation allows to minimise the deviation when keeping the mean on target while usually when the deviation decreases, the mean decreases as well.

The S/N ratio can be divided in three main categories:

$$S/N = 10\log\frac{\bar{y}}{s_y^2}$$ nominal is the best

$$S/N = -10\log\frac{1}{n}\sum y^2$$ smaller the better

$$S/N = -10\log\frac{1}{n}\sum\frac{1}{y^2}$$ larger the better

where $\bar{y}$ is the average of observed data; s_y^2 is the variance of y; n is the number of observations and y is observed data.

The one that suits our task is the "larger the better" because we want to understand the combination of weights that yield to maximise the mean of the individuals' fitness for each generation, which is our observation (y) [13][2].

To summarize, we run 27 experiment following the Taguchi fractional factorial array (*fig1)* for the assignment of the values (levels) of the factors (weights), repeat each experiment 3 times. We then compute the S/N ratio for each set of repeated experiments using the observed data which is the mean of the fitness of the individuals at the best performing generation. Eventually we have 27 S/N ratio values that can be used for evaluating the contribution of each weight at each level.

For doing this we need to average the sum of the S/N ratio for each weight at each level and afterwards evaluate their differences. The bigger the average value for a weight at a level, the more important is that level. The bigger is the difference between the maximum average value and minimum average value for a weight, the more important is the weight.

This can be understood by saying that because we want to maximise the individual's fitness, the bigger is the change in this value, when changing the level of a factor, the more important is this factor.

In this way we can rank the weights in order of importance listing which weight at which value contribute the most to the global outcome. *Fig2* shows very synthetically the explained procedure and two very interesting and unexpected results. The parameter "Vol_over_Footprint" and its relative weight are the most influencing ones and their contribution is maximum when the weight is set to the minimum value. This means that his parameter affects negatively the whole process which is something that happens when trying to **co-evolve** criteria that balance each other out. Same can be said about "min_curvature" which is the second in order of importance and its contribution is maximised when the value of its weight is at the minimum level. In this way we have the whole map of influence of the fitness parameter and their correspondent weights in respect to the observed data. This will tell us directly how we should set the array of weights according to our purpose. The application of this set of experiments, shifting the observed data from "total fitness" to one of the fitness parameters will give other informations regarding the appropriate set of weights that we need to assign in case we want to change the direction of evolution. In layman terms this procedure provide a method for using our proposed methodology that can satisfy the needs of different purposes contextualised within different architectural scenario.

factors

run	w1	w2	w3	w4	w5	w6	w7
1	1	1	1	1	1	1	1
2	1	1	1	1	2	2	2
3	1	1	1	1	3	3	3
4	1	2	2	2	1	1	1
5	1	2	2	2	2	2	2
6	1	2	2	2	3	3	3
7	1	3	3	3	1	1	1
8	1	3	3	3	2	2	2
9	1	3	3	3	3	3	3
10	2	1	2	3	1	2	3
11	2	1	2	3	2	3	1
12	2	1	2	3	3	1	2
13	2	2	3	1	1	2	3
14	2	2	3	1	2	3	1
15	2	2	3	1	3	1	2
16	2	3	1	2	1	2	3
17	2	3	1	2	2	3	1
18	2	3	1	2	3	1	2
19	3	1	3	2	1	3	2
20	3	1	3	2	2	1	3
21	3	1	3	2	3	2	1
22	3	2	1	3	1	3	2
23	3	2	1	3	2	1	3
24	3	2	1	3	3	2	1
25	3	3	2	1	1	3	2
26	3	3	2	1	2	1	3
27	3	3	2	1	3	2	1

Fig1 Taguchi 3 levels fractional factorial

Average Signal to Noise ratio **Main effect**

factors	Level 1	Level 2	Level 3	L_{max}-L_{min}	**optimum combination**
w1 (Vol_over_FacadeArea)	26	31	28	5	w1=5
w2 (Vol_over_Footprint)	36	31	24	12	w2=1
w3 (Height_of_Centroid)	34	39	42	8	w3=10
w4 (min_curvature)	24	18	14	10	w4=1
w5 (solar_gain)	26	27	30	4	w5=10
w6 (wind_exposure)	29	30	35	6	w6=10
w7 (FA_R)	26	27	25	2	w7=2

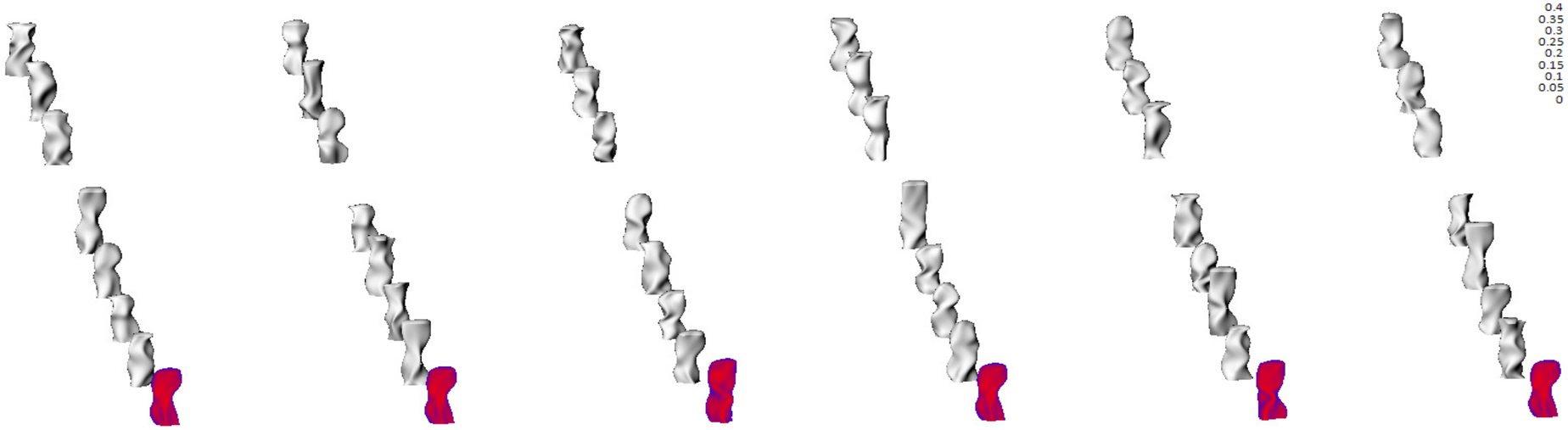
Fig2

Solar gain

Fig2 & Fig3 show some of the individuals of the 20th generation for an experiment where all the fitness parameters , except the ones that concerns solar gain, are set to zero. At the and of the procedure they all have a morphology whose most relevant trait is to have the main dimension oriented perpendicularly to the mean of the directions of the sun(9a.m. to 3p.m. 21st of December) . This happens because they try to gain the highest value of solar detection possible within the size of their solution domain. The individuals shown in *fig2 & fig3* are obtained starting from a different set of random numbers(initial position of points which are the genes of individual) but having same weights. It interesting to see, when looking at the trend for the fitness parameters, that they all benefit from this effort to detect as much daylight as possible although their correspondent weight has been set to zero. The value of solar_gain stabilises around **0.34** which means that the 34 percent of the mesh faces manage to capture more then the 60 percent of the daylight (see solar gain|fitness at page 26). A **cylinder**, having same height and same area, performs a value of **0.24** *(fig4).*

weights

w1=0	V_over_FA
w2=0	V_over_F
w3=0	H_centroid
w4=0	curvature
w5=10	solar gain
w6=0	wind exp.
w7=0	FA_R



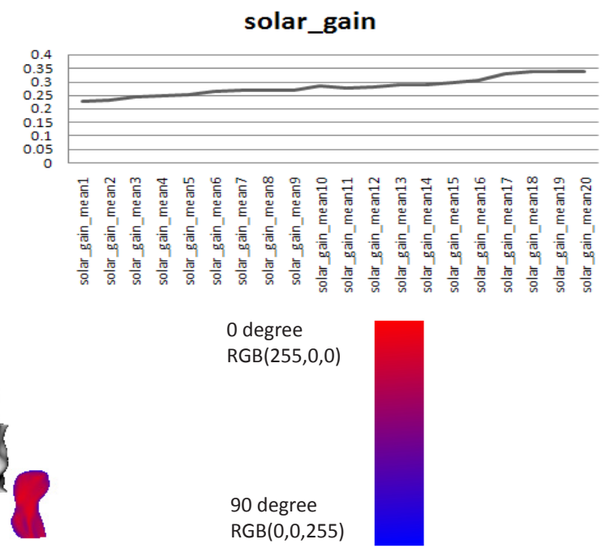
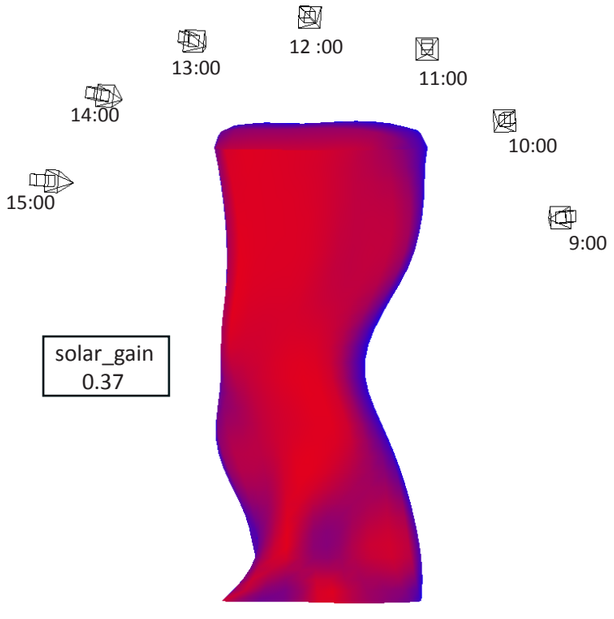


Fig1

genertion over generation



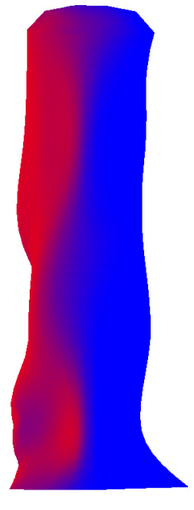
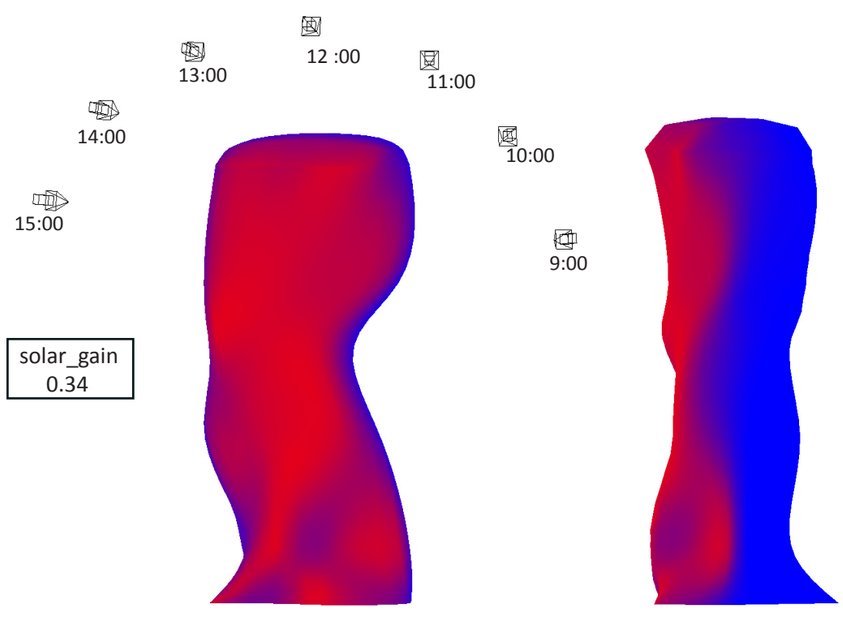


Fig2 south facing wall front view



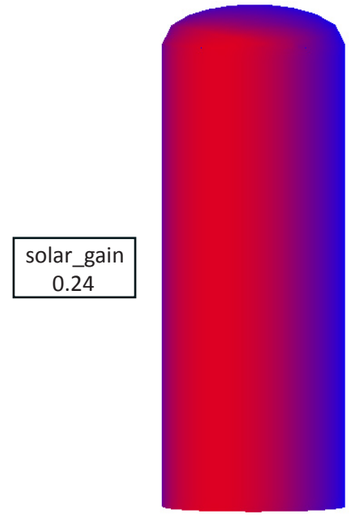
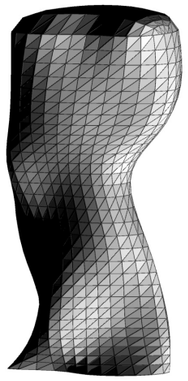


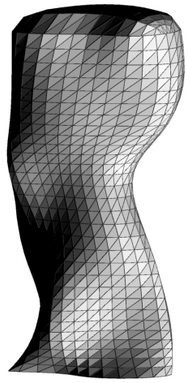
Fig3 south facing wall front view

Fig4

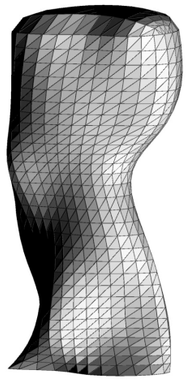
9:15



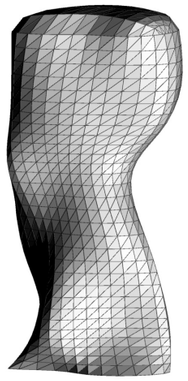
9:30



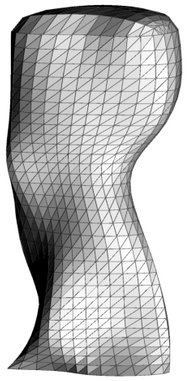
09:45



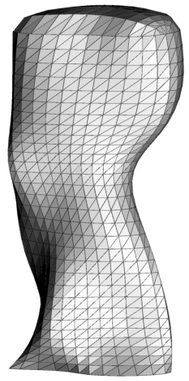
10:00



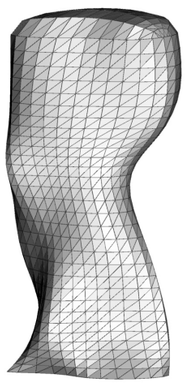
10:15



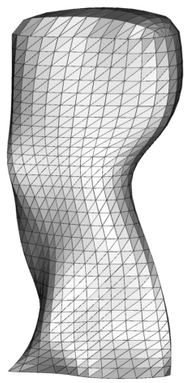
10:30



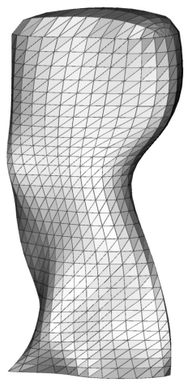
10:45



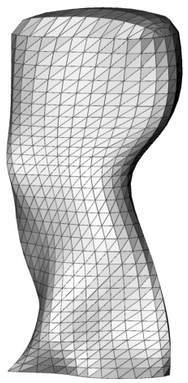
11:00



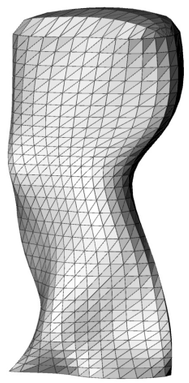
11:15



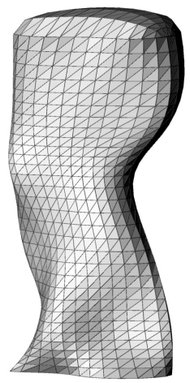
11:30



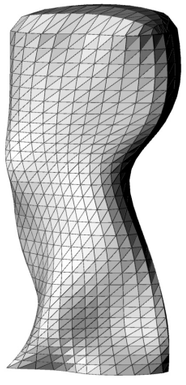
11:45



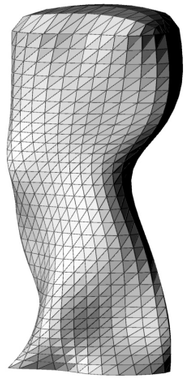
12:00



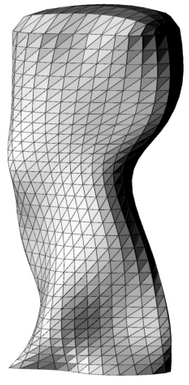
12:15



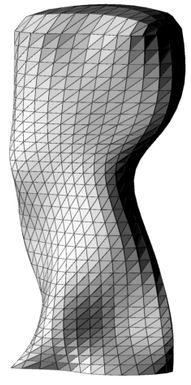
12:30



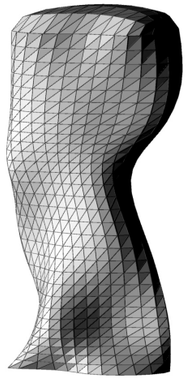
12:45



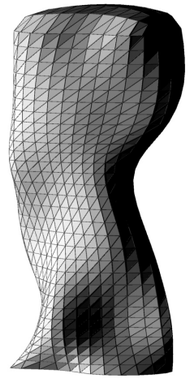
13:00



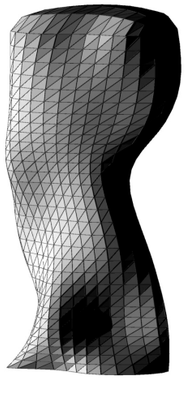
13:15



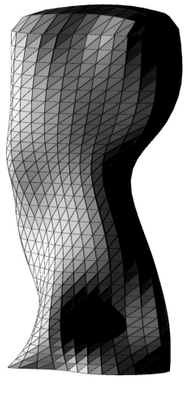
13:30



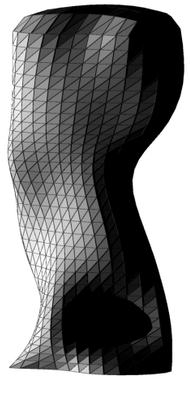
13:45



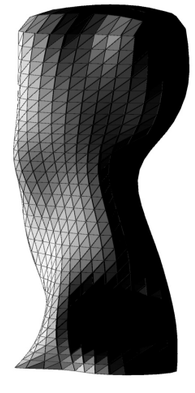
14:00



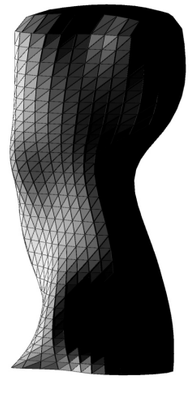
14:15



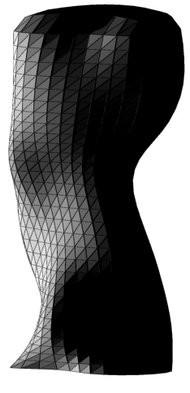
14:30



14:45



15:00



shadows clustering taken on the 21st of December at 51.4879° Latitude -0.178° Longitude, London

sunpath from 9:00 to 16:20|40 minutes frame rate

Wind exposure

It is here presented one of the member of the 30th generation for an experiment where all the fitness parameters, except the one that concerns the wind exposure ,are set to zero. At the and of the procedure the individuals present a morphology whose most relevant trait is to have a highly pronounced **"V-shape"** which is aligned with the direction of the wind(270 degree east). This happens because they try minimize the exposure to wind orienting their surface in order to avoid to experience high positive pressure. The table in *fig2* shows the trend of *wind_expsure* for this experiment which decreases steadily from the 1st generation and stabilises around a value of **0.42** for the individual of the 30th generation. The value of *wind_exposure* indicates the percentage of triangulated faces where the absolute value of pressure is bigger than the 60 percent of the value of the reference pressure. In this way we test the efficiency of the shape at not producing high value of positive or negative pressure which occurs mainly on the roof, lateral and back side of the individual respect to the direction of the wind (*fig1* bouding box represented by black dots). In order to have a term of comparison we tested the morphology of a model which represents the *Gherkin (fig3).* Its value of *wind_exposure* is **0.64** which means that the 64 percent of its surface experiences a level of pressure which bigger than the 60 percent of the value of the reference pressure. The same parameter for our individual is **0.42** which indicates, therefore, a higher efficiency of its morphology respect to wind flow. Although this comparison demonstrates the effectiveness of the procedure, phenomena such as turbulence is only empirically taken into account, using the laws given by the ENV 1991-2-4, while vortex shedding is completely ignored. Although this shortcoming would make the procedure incomplete for an accurate analysis of a form exposed to wind flow, the procedure can be regarded as reliable for the scope of this research.

weights	
w1=0	V_over_FA
w2=0	V_over_F
w3=0	H_centroid
w4=0	curvature
w5=0	solar gain
w6=10	wind exp.
w7=0	FA_R

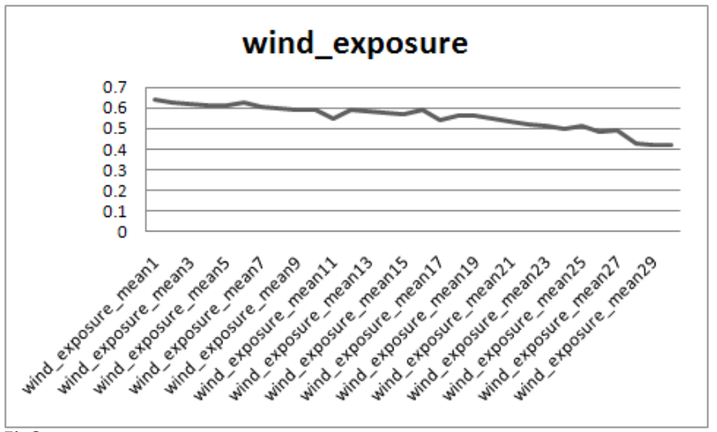


Fig2

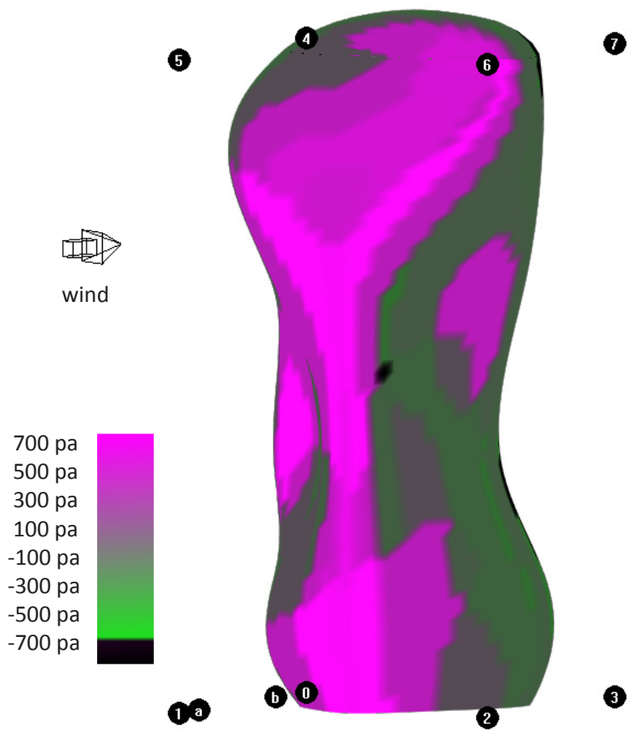
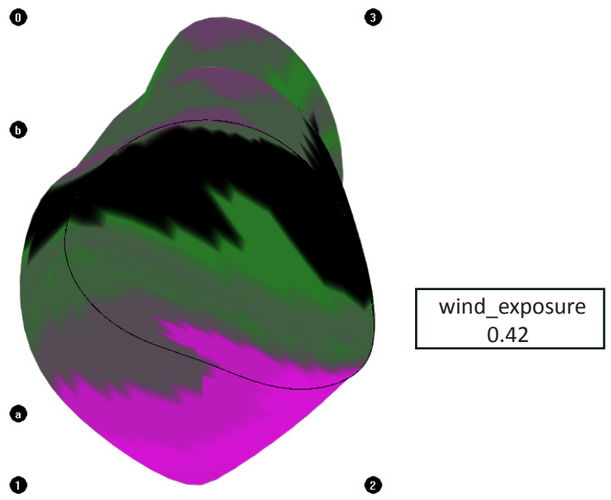


Fig1 perspective view


top view

wind_exposure
0.42

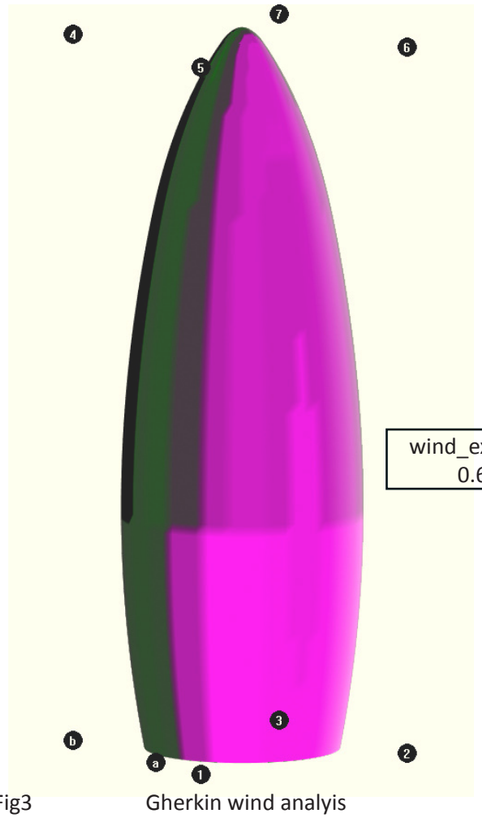


Fig3 Gherkin wind analyis

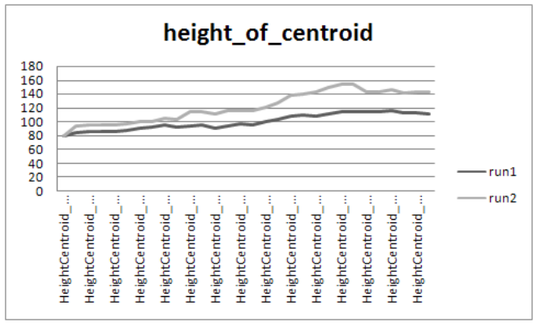
wind

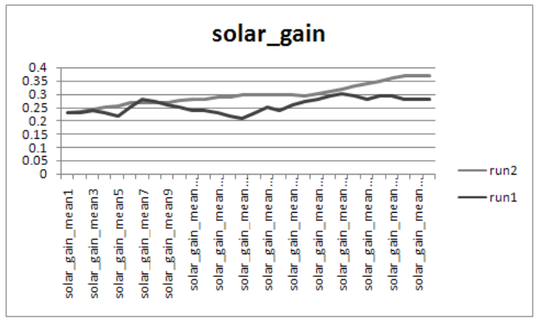
wind_exposure
0.64

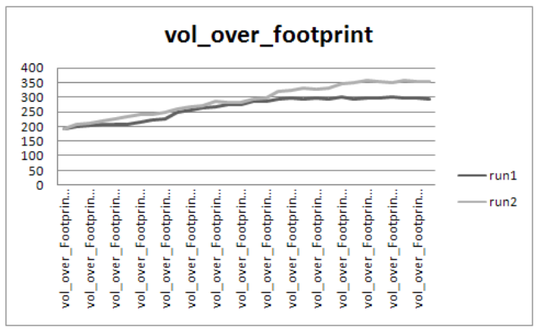

Gherkin render

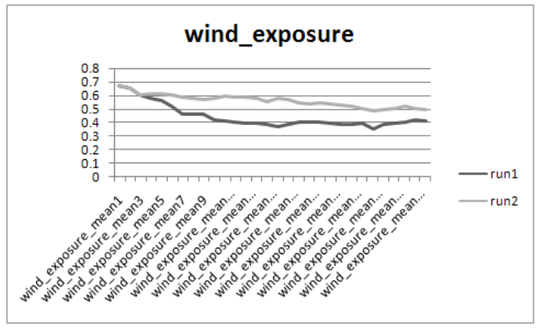
Varying the weights

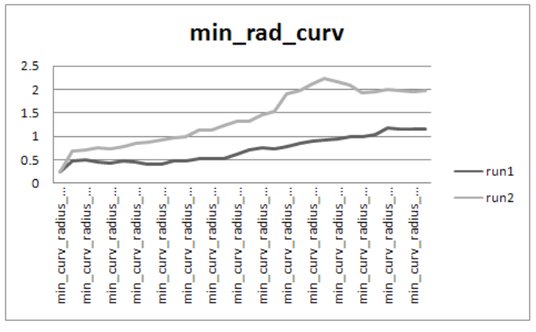
The configurations here presented are obtained by combining all the available fitness parameters. *Fig1* shows a solution given for a set of weights where parameters concerning spatial organization such as “Height_of_Centroid” and “Wind_exposure” dominate the evolution *(run1).* There are some traits that remind of the ones featured by members of the simulation shown in the previous page where only the “wind_exposure” were activated. The sharp appendices at the top aligned with the direction of the wind are the most evident. However, “wind_exposure” influence is here balanced by the other parameters that lead to morphologies presenting a higher position of the volumetric centroid and a smaller “Facade_to_Floors ratio”.
Fig4 shows a configuration obtained starting from an equal set of random numbers (position of points which are the genes of the individuals) respect to the previous experiment but having this time a different set of weights *(run2).* For this simulation spatial organization parameters are weighted to a greater extent, which is recognisable observing the morphology of the individual shown in *fig4.* The most relevant traits for it are a very high position of the volumetric centroid, a small Footprint area and smoothness of its envelope. It is worth saying that these two configurations are obtained stating from the same set of initial genes. However, the individuals are evaluated in two different environment, simulated assigning two different set of weights for the fitness parameters. This demonstrates to what extent the environment can influence their morphology. The tables below show the trend for the fitness parameters for *run1* and *run2.*











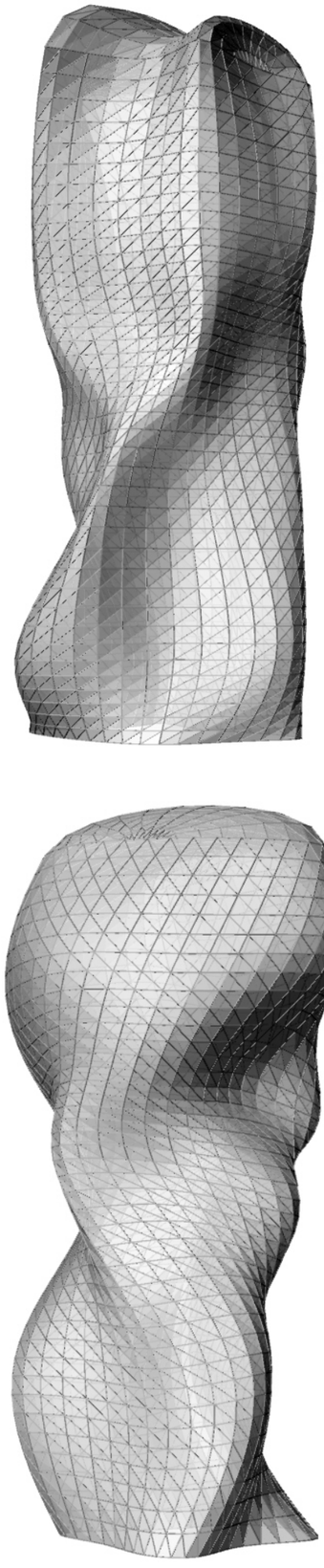
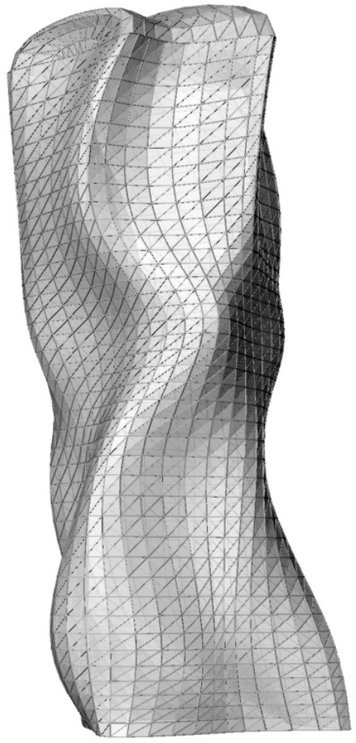


Fig1

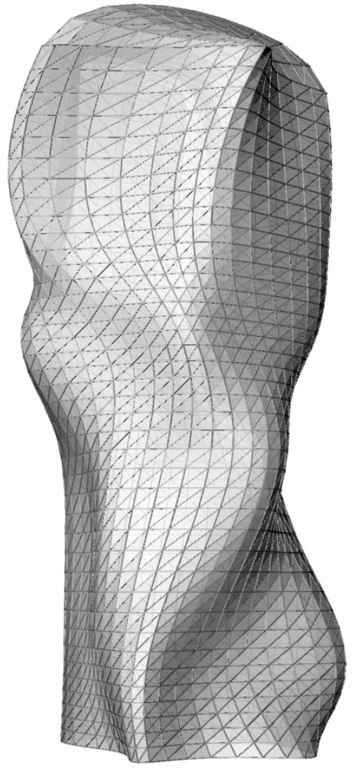
run1	weights
w1=2	V_over_FA
w2=2	V_over_F
w3=10	H_centroid
w4=1	curvature
w5=2	solar gain
w6=10	wind exp.
w7=6	FA_R

Fig2

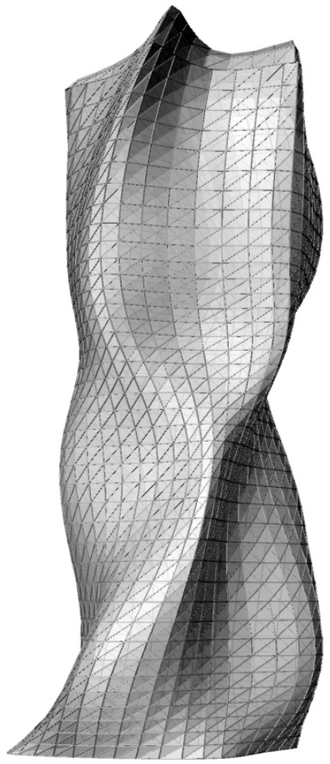
run2	weights
w1=3	V_over_FA
w2=5	V_over_F
w3=10	H_centroid
w4=5	curvature
w5=8	solar gain
w6=4	wind exp.
w7=6	FA_R



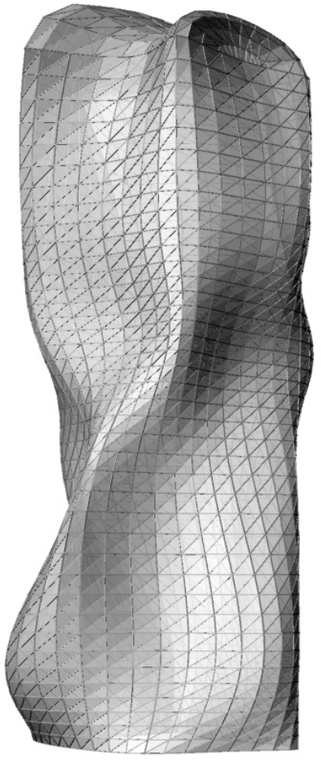
Ga individual|run1 view 1



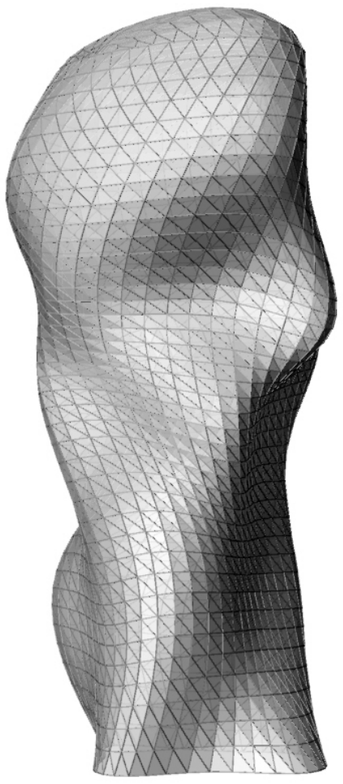
view 2



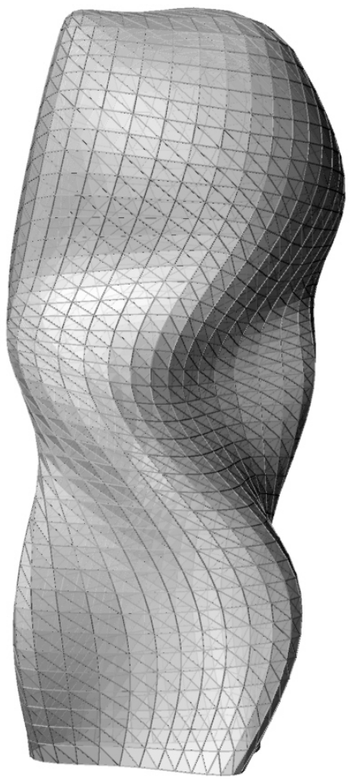
view 3



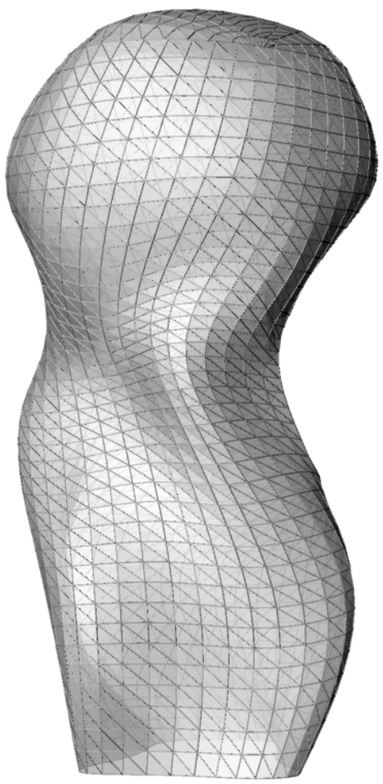
view 4



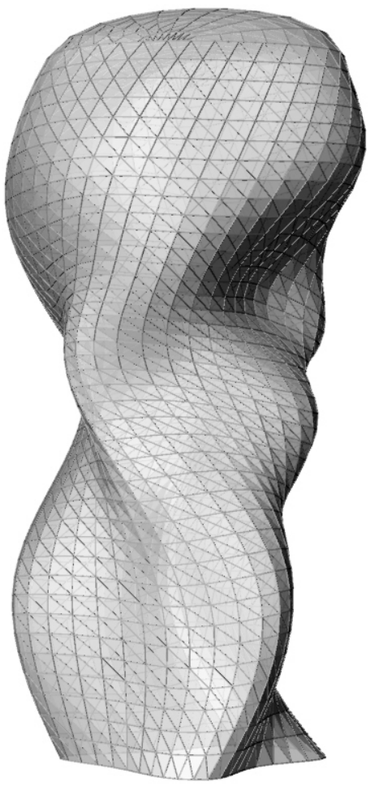
Ga individual|run2 view 1



view 2



view 3



view 4

Artificial Neural Network

Intro

In this chapter are presented some of the experiments that aimed at engaging with another artificial intelligence technique called Artificial Neural Network. The work on Neural Networks builds its foundation on the extensive research carried out by *Christian Derix* at the *UEL*, for further reading see *"Approximating Phenomenological Space"* [14]. The research on this topic proceeded separately but parallel to the one illustrated in the previous chapter. They crossed when at a certain point in the research, the question of how would it be possible to derive and internal subdivision of the generated configurations that respected their external morphology, arose. A distribution of volumes that behaves not only according to an architectural program but also influenced by the traits of the generated spatial configuration.
With regard to the application of artificial neural network that i develop, it has to be said that it is not integrated in the system described in the previous chapter but it is performed post-facto on one or more of the generated configurations.
An other interesting way of applying this type of neural network would be their integration with the evolutionary techniques for the development of an unsupervised mechanism whereby evaluate and select the individuals *("Architecture's New Media, Yehuda Kalay (2004))* [15].

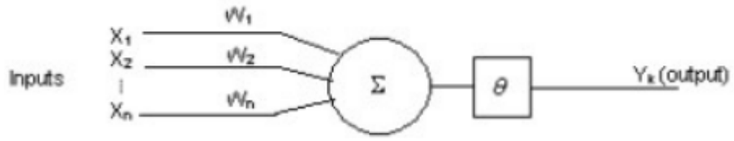


Fig1

SOM: Self Organizing Map

Self-organizing maps (SOMs), invented by Professor Teuvo Kohonen, are a data processing techniques part of the artificial neural network developed by the perception network (Rosenblatt,1962). The training of this type of artificial neural network allows to produce low-dimensional representation of a higher-dimensional input space.
Like all the others neural networks, the way SOMs operates mainly consist in training and mapping. By using this particular type of NN (Neural Network) is possible to codify in the inputs some information and process these data obtaining a map that shows the emergent relations between the inputs. The information that is possible to encode can be any kind of data including geometric description of space [14][2].
The structure of SOMs is made of layers or nodes (neurons) to which is associated a position in the network and a weight vector that has the same dimension of the input vectors. The way the nodes are arranged can be any sort of two-dimensional or three-dimensional grid. In the terminology of the field a "weight" is the thing that is accociated to both nodes or inputs.
A simple example for describing a type of SOM, is having a network made by 25*25 nodes (just in this case but the dimension may vary according to the dimension of the input vector) and an initial set of input vectors (fig2). The picture shows a grid of nodes where for every node there is an associated vector(weights). The input vectors are the one at the side of the grid in red and labelled with numbers from 1 to 6. This example is borrowed from one the application of SOMs developed by *Christian Derix* at CECA.
It is worth mentioning that in this case the weights are represented by the orientation of the vectors associated to the nodes and, therefore, they have two components the x and y cosines. As already said above it is possible to encode in the input any sort of data for the network to map and they do not have to be represented only by geometric vectors. They can be everything that it is useful to visualize the data that we want to encode.

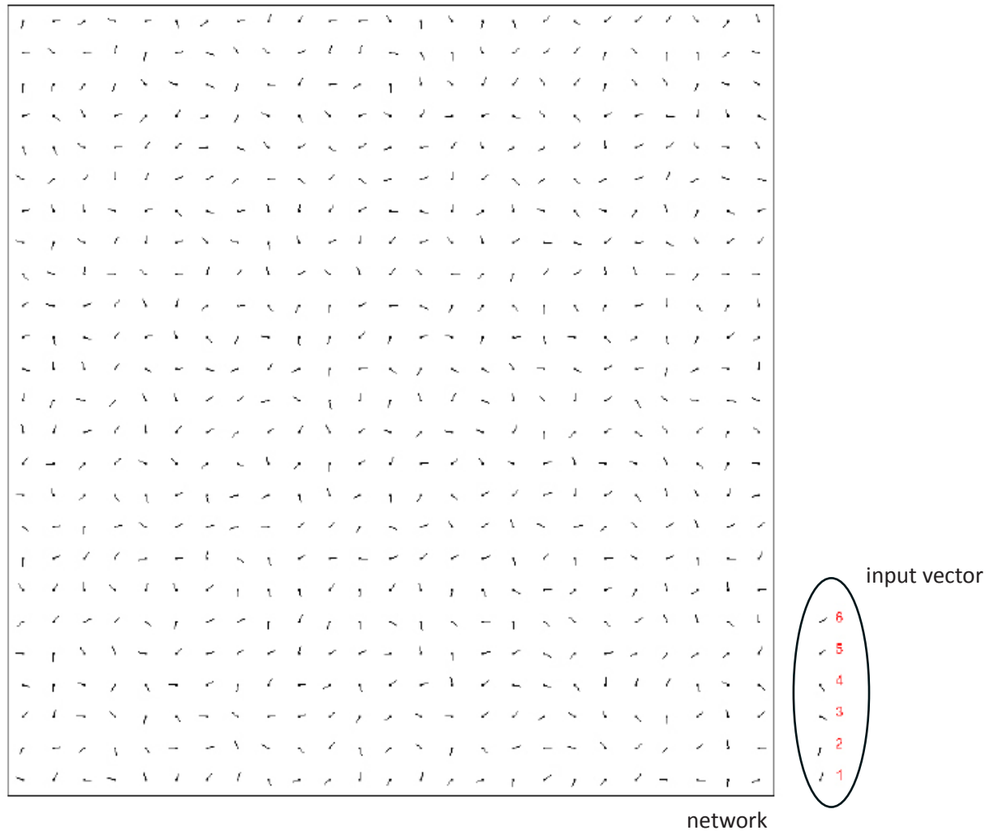


Fig2

Learning Algorithm

What mainly a SOM does, is to self-organize a map in which different parts respond in the same way to a certain sub-space of the input space. In layman terms they can display emergent properties, represented by clusters, of the input space that are not possible to visualize in other way.
The weights for the neurons are usually initialized to random values. The way in which input vectors and network interact is to find the node whose weight vector is the closest to one of the vector of the input space and to locate the position of this node in the map. This node is called Best Matching Unit or simply "the winner" [16].
This can be done with different techniques such dot product or euclidean difference. Those vectors should be normalized as well as their components should all have the same dimension in order to be comparable. With this procedure all the nodes are compared with all the input vectors, one by one, finding each iteration which is the one that is closest to each of them [17].
Once the winner for an input vector has been found, its Euclidean distance to all the others nodes is computed (topological distance in the network). At each iteration there is one winner node to which is associated a certain radius of influence (neighbourhood). Its weight and the weights of all the nodes, whose Euclidean distance from the winner falls in the neighbourhood, will be adjusted to be similar to the correspondent input node *fig3* [17][2]. The degree of adjustment is defined by the so called learning parameters. There are usually two set of learning parameters, one for the winner and one for the others neurons. Both can be represented by a monotonically decreasing coefficient but the one for the winner is constantly higher than the one for the other nodes. This means that when the network starts mapping, the learning parameters have a high value because there is a lot to learn, and iteration after iteration they need to be adjusted to an ever smaller extent. In addition the radius of influence of the winner will also decrease through the iterations [17][3].
To summarize, the magnitude of changes for weights of the nodes in the network decreases with time and with distance from the winner *(fig1)*. "Learn" representing the learner parameters is function of t (time) while N (neighbourhood) is a function of "d" (distance from the winner) and t (time), Wi is the weight of the inputs, Wn the weight for the nodes.

$$Wn(t+1) = Wn(t) + N(d,t) \times Learn(Wi(t) - Wn(t))$$

Fig1

With regard to neighbourhood function, a way for representing it is the Mexican Hat Function (*fig2)*. It is easy to see that neurons close to the winner will be "excited" to adjust their weight towards the input weight. With the increasing of the distance the magnitude of change decrease as well, until it becomes negative (inhibitory feedback). This means that the nodes that are outside the neighbourhood will adjust their weight to be different from the winner at iteration i(i meaning current iteration).
As already seen for Cellular Automata ensuring simultaneity is the key of success. This mean that each iteration (loop through the input vectors and comparing the weights for all the nodes with each of them) the difference between the weights of the input vectors and the ones of the nodes have to be only computed without adjusting the weights of the nodes (in this case the orientation of the geometric vectors). Only after having computed all the differences for the current iteration we can update the network simulating in this way simultaneity. This procedure repeats several times until the map converges distributing the features (weights) of the input vectors into clusters *(fig4).*

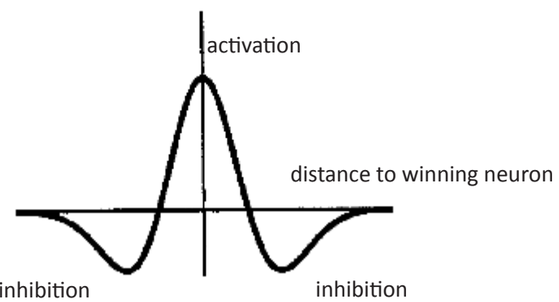


Fig2 mexican hat function

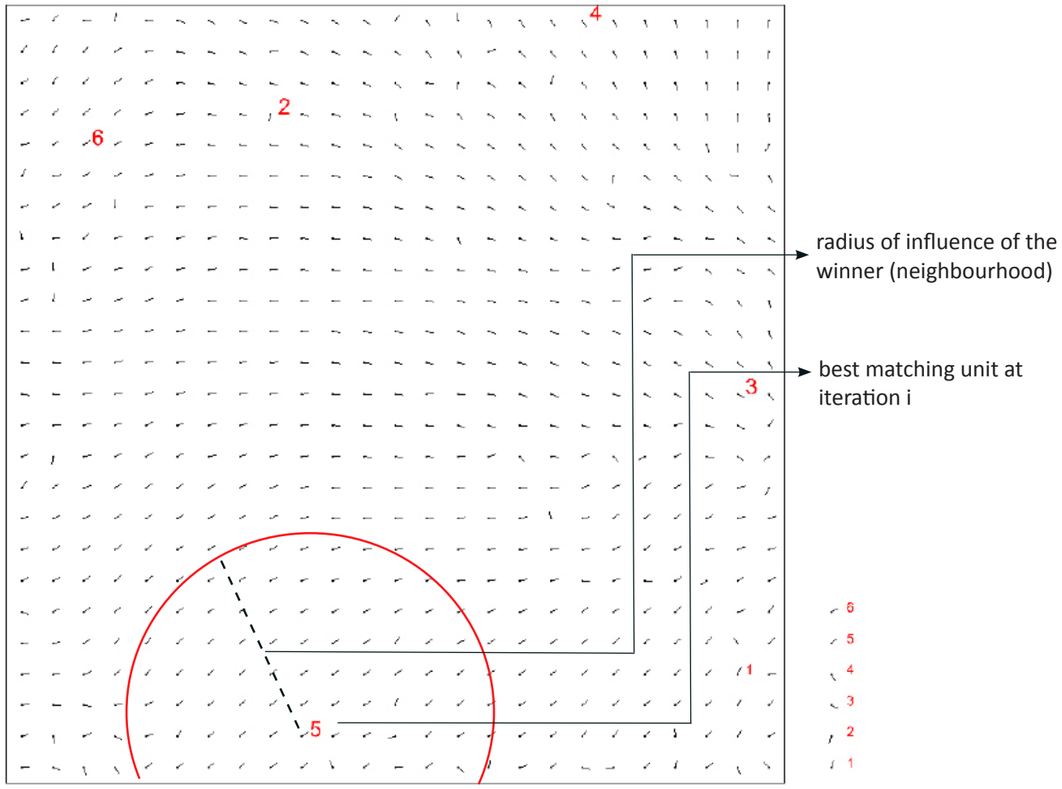


Fig3

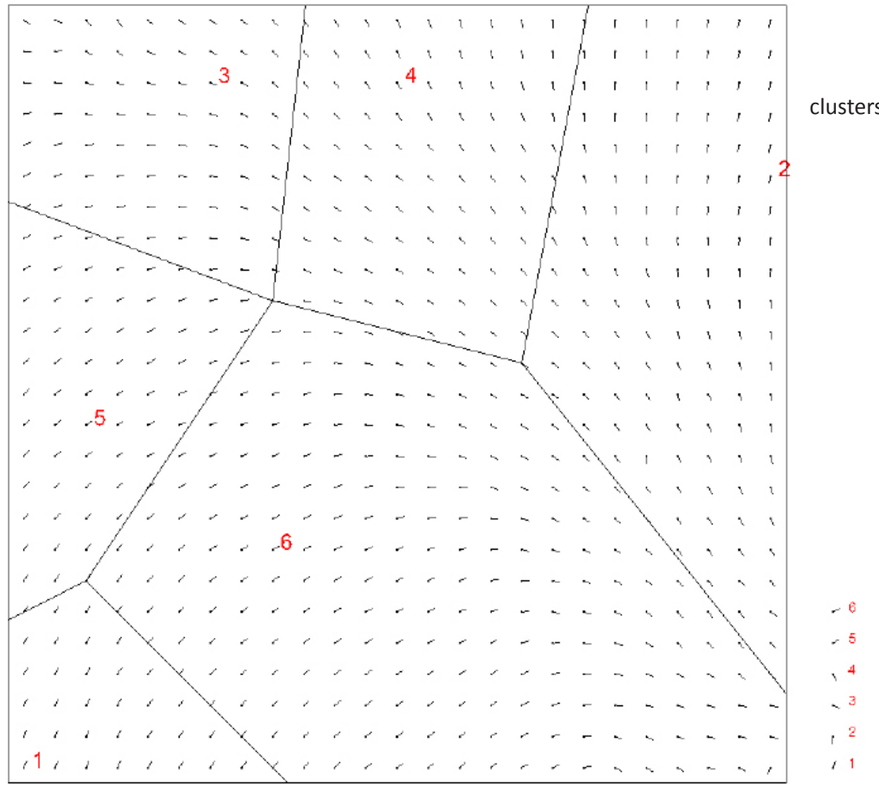


Fig4

Self Organizing Maps -3D Topologies

Along with the already discussed self organizing maps that clusters input vectors into relations there is the possibility of making SOM starting by 3D topology [14][3]. The difference between these two is that the 3D topology can represent a spatial configuration which might be a surface of a volume. It is worth saying that the emergent cluster of the network, after the “training”, is not encoded in it. It is the emergent representation of the relations between the inputs and the topology of the network. With regard to the experiment shown in this paragraph, it can be applied everything said for the 2D topology. This time the weights will be the 3D spatial coordinates of the nodes and input vectors fixed points in the space as can be seen in *fig1.* The 3d grid at the centre represents the 3D SOM and the red circles at the corners are the input vectors. The nodes will continuously adjust their position influenced by the action of the inputs while keeping the topological relations with their neighbours.
Fig1 shows the starting 3d network while *fig2* the emergent cluster where, between the nodes that lie on the lateral surfaces, have been drawn mesh faces. The drawing of rectangular mesh faces between the nodes is possible because their topological relations are mantained. These example shows the possibility of training a neural network for solving a numerous variety of problems.

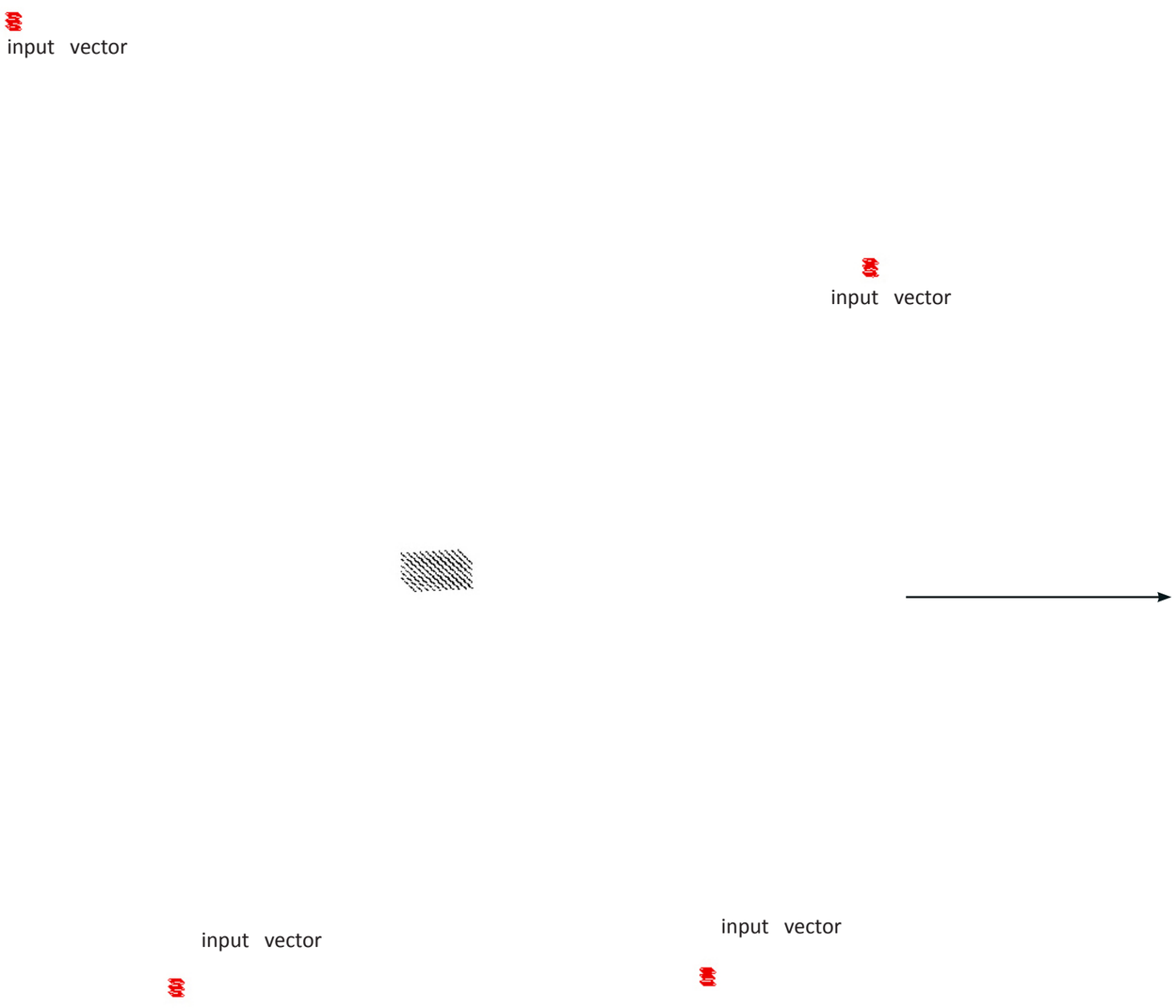


Fig1

3d network, itaration 0

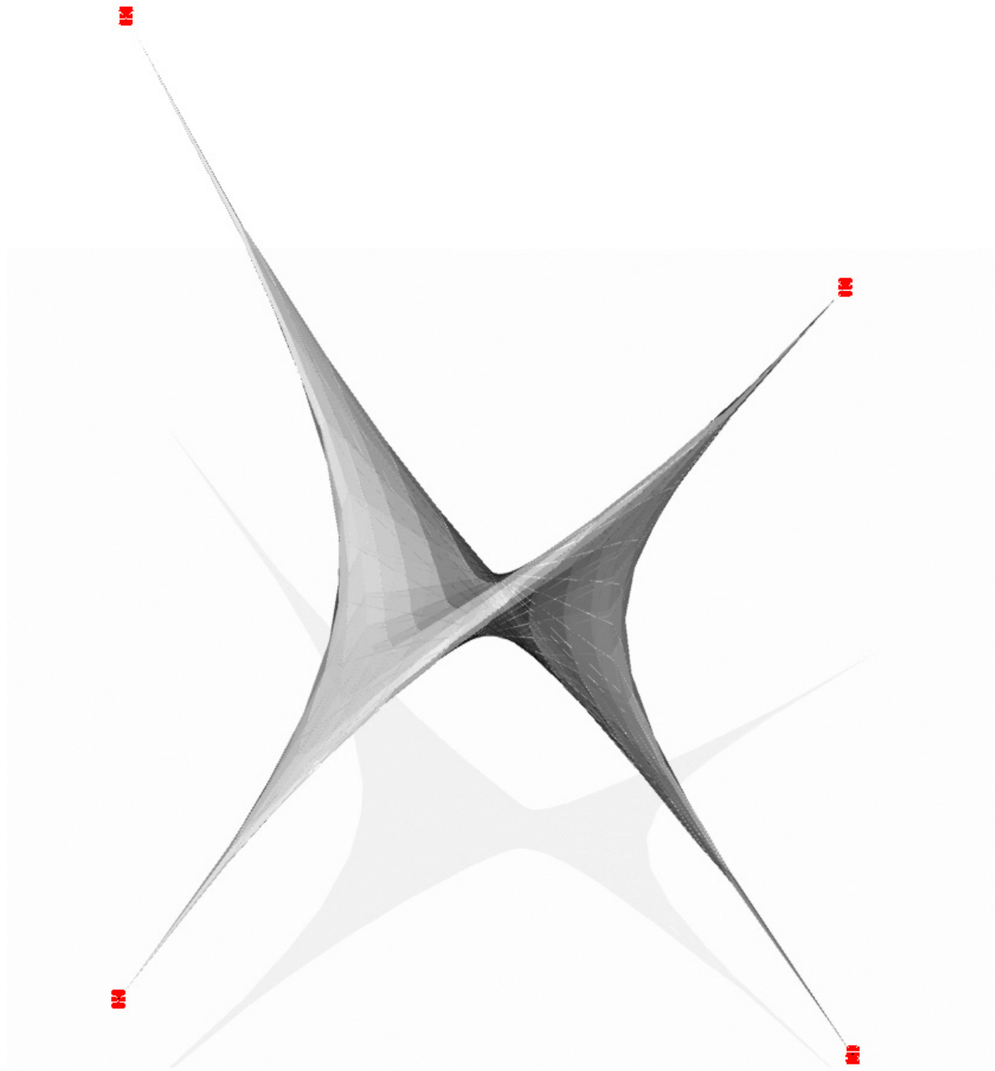


Fig2

3d network, cluster at final iteration

3D SOM- spatial clusters

The research on artificial neural network and Genetic Algorithm meet together in the attempt of deriving an internal sub-division of space that is related to the morphology of the generated configurations. One of the fitness parameters used in the 3rd experiment is related to the solar radiation that the individuals manage to gain on the 21st of December from 9:00 to 16:00. Solar gain, for one of the generated configurations, can be visualised with map shown at the left side of *fig2*. This map is the visualization of a series of arrays that describe vertex by vertex the colours of the mesh. For each vertex of the mesh is associated a vector in which are encoded the RGB value for the neighbouring faces. This description of the map reminds somehow at the input vectors that we had when explaining SOM's in the previous paragraph. This time the input space is set to be the map of solar gain itself which can be represented by an n-dimensional vector. The dimension of the vector varies according to the size of the mesh and the correspondent number of vertices. It is immediate to think that the network, which the input vector would train, can be represented by a 3D grid of nodes describing the volume of one of the individual (*fig2*, black dots at right side). The initial value for the weights of the nodes (in this case vector of RGB colours) is set to be RGB(0,0,0) which represents the black colour.
In order to map the input vectors into clusters that can be related to an architectural program I divided the input space in 4 main sub-spaces according to the required amount of daylight:

- *residential*
- *office*
- *retail*
- *service*

Fig3 and *fig4* shows the clusters produced by the SOM after 62 iterations when the learning parameters are very close to 0 which means that convergence has been achieved.
Although residential and office are the areas that require more daylight the allocation of the first one is preferred to higher position for ensuring good view. The allocation of retail area cannot go above a certain height as shown in *fig3* (right side), where the cluster related to this sub-space, positions itself at the bottom of the building. Service cluster goes through the whole shape and position itself close to the north face of the building for creating a thermal buffer during the winter (*fig4,* right side). Service is the area that should house structural core and service core. *Fig1* shows residentil,office and service cluster for a section taken at the top side of the residential cluster.
The clusters shown in the images are the result of a negotiation between the ability of the network to cluster the input features (solar gain map) into relations and the constraints given by the encoded architectural program.
Although the procedure works properly, it is worth saying that we are at first early steps for the development of this technique. The architectural program that can been encoded is still very simple due to the difficulty of the process. Further investigations needs to be undertaken for deploying the potentiality of such a sophisticated way of reading the space.

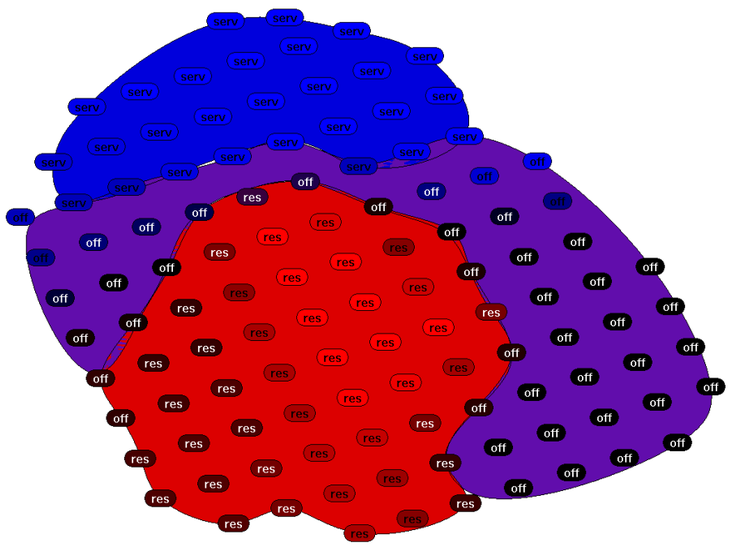


Fig1 section taken at the top side of the residential cluster

solar gain map

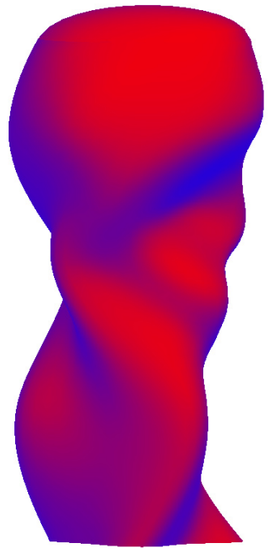
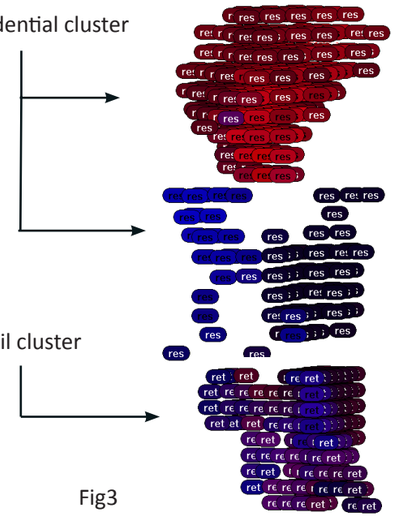


Fig2

initial set of weights for the 3D network

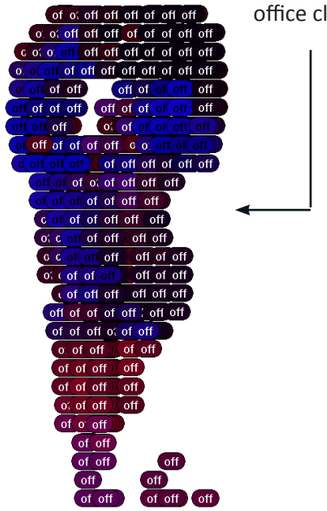
black=RGB(0,0,0)

residential cluster



retail cluster

Fig3



office cluster

services cluster

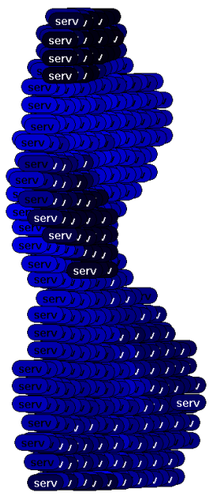


Fig4

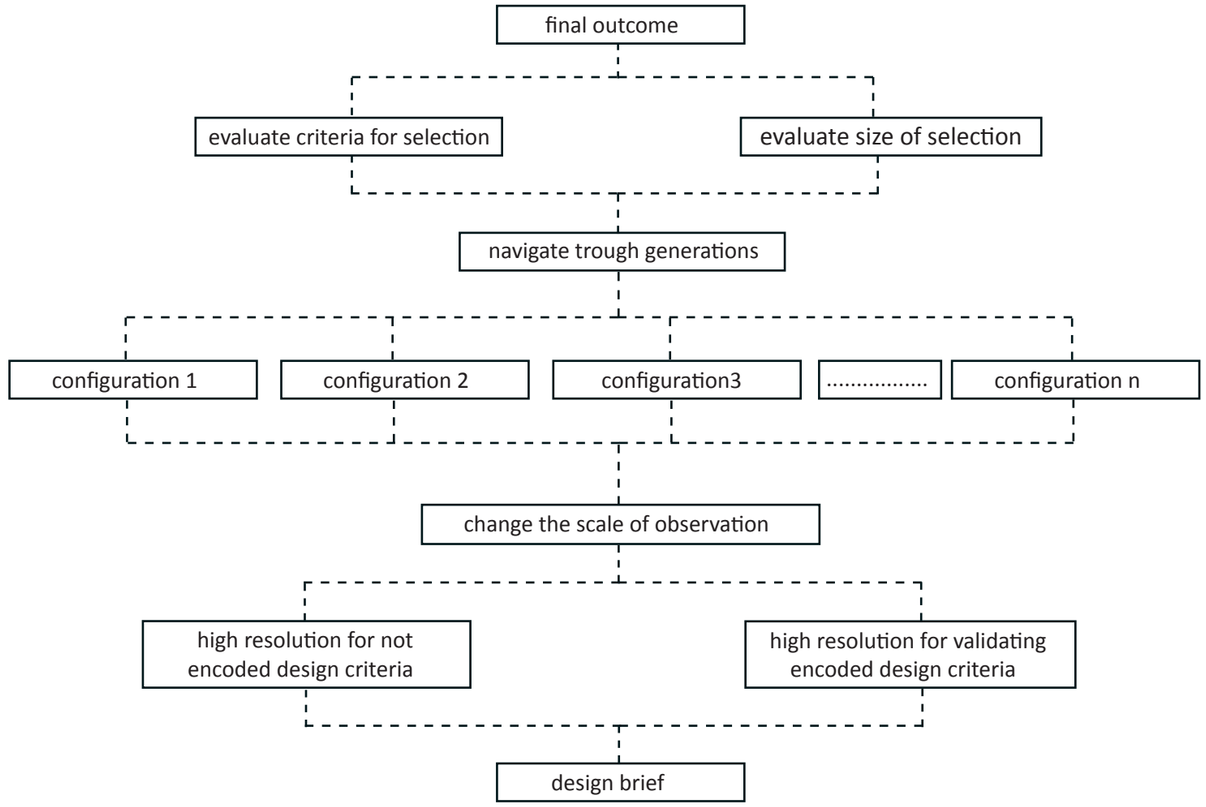
all clusters togheter

Conclusions

In the previous pages are shown some of results from different experiments where the criteria of selection are mainly based on the performance of the individuals. The configuration shown in the images are the most performing ones amongst the most performing generations. Applying this criteria for any other next experiment might seam at first sight a reasonable choice. However it is the case of analysing further more what are the possible options focussing on the repercussions that our proposed methodology has on the design process.
Consideration should be given to the main purpose of this work. It aims at providing a decision support environment, by means of artificial intelligence techniques, for the integration of several performance evaluation tools. This allows the utilization of analysis tools at a conceptual level for generating forms rather than evaluating them “post-facto” [1][6]. For the way the whole procedure has been designed the evolution does not concern one individual at time but an entire set of them, the populations. Evolving populations implies that the final outcome is made by set of solutions rather than one, which something that matches our expectations. The final outcome is, indeed, a set of possible configurations that can be used for having different proposals, sharing same topology and performing behaviour, but having relatively different traits. From this point of view the choice that we made, when presenting the results in previous pages, of taking the most performing individual of the most performing generation seams very restrictive rather than reasonable. Having said that, several questions arise regarding the criteria of selection that one should take into account when facing an entire set of possible configuration that result from the procedure. How can we navigate such a variety of configurations ? What kind of functional/aesthetic sensibility a designer should develop for orienting his choice ?
If order to rationalise this problem we can start making two steps for orienting ourselves :

- **navigate trough generations**
- **changing the scale of observation**

The first step arises from the observation that most of the times there are some individuals that, although do not belong to the final generation, are particularly performing. In the overwhelming majority of cases the final generation does not present the most performing individuals but has the most heterogeneous environment. The first thing that come to mind is to automatically search through the generations for the most performing individual giving a threshold related to its fitness. For instance if we are interested in investigating the performance of the individuals regarding a certain fitness parameter, not necessarily focussing on the total fitness, we can select the most performing one and all the other individuals whose qualities are within a certain distance. In this way we deploy the possibilities offered by such a method and use all the history that has been created generation over generation.
The size of this range, that serves as a filter for choosing different configurations, depends on the available recourses that one has in terms of time and memory. In any case, after doing such a filtering, we end in having a relatively small set of possible configurations that can be further analysed.
If we think at the way we have looked at the results , it is clearly evident that the scale of observation has been set to a very low level. We have monitored their performances on the base of the analysis done over the course of the evolution for generating them. However, in order to proceed with the analysis, we need to change the scale of observation to a higher resolution. This entails the addition of another layer of complexity whereby the chosen individuals are subjected to a detailed evaluation of their characteristics. For doing this, in a reasonable amount of time, we have to make this process automatic writing another piece of code whereby the chosen individuals are selected and analysed using the available commercial performance based evaluation tools. This time we do not work with them for generating morphologies but rather for an accurate evaluation of their behaviours on **three levels**. **The first** one is to make an evaluation, using high resolution tools, for the criteria that have not been taken into account in our procedure such as structural behaviour, lighting, circulation, fire safety etc. **The second** one, is to validate the criteria encoded in our procedure by means of the correspondent high resolution instruments to the ones that we developed over the course of this research. It is worth remembering that simulations, such as the analysis of the exposure to sun and to wind, are mainly based on our cutting edge developed tools and do not take into account higher order phenomena such as turbulence or thermal transmission. As already said above, the development of those tools has been necessary for their utilization at a conceptual level due to the enormous amount of simulations to be performed. **The third** level consist in an aesthetic evaluation of the chosen configurations amongst which there might be several of those that do not match the sensibility of the designer.
In this way we create the logic link between the outcome of our proposed methodology and its possible utilization in an architectural context. It is worth saying that this body of work has never aimed at creating a process that can deliver the final product for a design. The key principle has been to provide an exhaustive exploration and a first rough selection of all possible combinations of variables belonging to the examined topology, which would not be possible to consider with a traditional approach.



Outlook

The project has opened up many directions for future work. It has demonstrated how the abstract representation of spatial configurations can be related to real design constraints, under whose influence, they take shape. The experiments can be considered as the unfolding of the design system. The first is a crude application of an evolutionary algorithm that reveals the core of the system. The second one shows how a spatial configuration, represented by an amorphous surface, can take shape under the action of a simple criteria, equilibrium. Due to the constraints imposed by the structure of the developmental process and the relatively small size of the solution domain, the results delivered by such a system reach the potential of a possible design brief. The third experiment is where the system showed its virtuosity allowing to tackle problems with n-indipendent variable by simply varying an array of numbers, the weights. **Form-finding**, a term that could be easily associated to the patient work of the system which, instructed by design criteria, iteratively searches into the solution domain for deriving a set of optimum spatial layouts. We could say that the generated configurations, over generations of attempts, become expression and virtual representation of the design criteria. However, due to the vastness of the searched space and to the not yet fully understood interrelations between the design criteria, the delivered results have not reached the realm of a design brief.
With regard to the future development of this work there are three main shortcoming that have to be highlighted:

− the generative system that controls the making of the "genome" needs to be further developed. At a conceptual level it is possible to add another layer of complexity that serves as a controller for the management of the genome over the evolution. Inspiring idea come from the newly developed *"Evo-Devo"* evolutionary developmental biology and the computational scheme developed for *Genetic Programming* [18].
The generative system mainly used in this research , although limited, presents the big advantage of being usable at a very general level. It is worth saying that the procedure could also serve for improving an already semi-developed architectural design by exploring possible variation of it, within the size of its solution domain, and under the pressure of its design constraints.

− with regard to performance-evaluation tools, it must be said that time and resources should be invested for the development of their light-weight or "low-resolution" version, in order to be used at the conceptual level in the architectural design process. The tools that I developed in this research, mainly based on vectorial calculus, although not as accurate as Ecotect (sun analysis) or Ansys (CFD) , serve the function they were developed for. Their further improvements will lead to more efficient evaluation and , therefore, development of the generated configurations.

− the absence of any link with the material system. This issue can be tackled by designing a light-weight structural analysis tool based on FEM or Dynamic Relaxation. Embedding it in the algorithm will open the way for introducing the evaluation of intensive quantities such as distribution of stresses or the energy of deformation which can be used as design criteria in order to explore the solution domain while developing a meaningful structural system at the same time.

The repercussion of the proposed approach to design is a radical shift from the way in which design is conceived. Rather than designing forms directly, we design the abstract representation of them, operating with the qualitative measurement of their behaviours under certain design constraints instead that basing the design process on metric measurements [7] [3].

References

1. Branko Kolarevic & Ali M. Malkavi (2005) Performative Architecture • Beyond Instrumentality. Spoon Press
2. Goldberg, David E., 1989, Genetic Algorithms in Search, Optimization & Machine Learning,Addison-Wesley
3. Coates, P. (unpublished) Programming Architecture. London
4. Jörg Krämer and Jan-Oliver Kunze Design Code Professor Finn Geipel Labor für integrative Architectur Techniscule Universität Berlin unpublished thesis 2005 TU Berlin
5. Holland, J. (1998) Emergence: From Chaos to Order. Oxford University Press, Oxford
6. Holland, J. (1992) The General Setting. In Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to biology, Control, and Artificial Intelligence, MIT Press, London: 1-19
7. DeLanda, Manuel, 2002, Deleuze and the use of the Genetic Algorithm in Architecture, Designing for a Digital World, Wiley-Academy, 117-120.
8. Simos Yannas Solar Energy and Housing Design, Volume1 : Principles, Objectives, Guidelines. AA Publications, London
9. Ziona Streliz (2005) Tall Buildings. The British Council for Offices and RIBA Publishing, London
10. Branko Kolarevic (2003) Architecture in the Digital Age. Spoon Press
11. Derix, C. (2008) Evolutionary Architecture. In The Space Craft
12. NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/. Accessed on 10/08/2008
13. A. Jayant, V. Kumar (2008) Prediction of Surface Roughness in CNC Turning Operation using Taguchi Design of Experiments. IE(I) Journal-PR
14. Derix, C. (2007) 'Approximating Phenomenological Space', in Intelligent Computing in Engineering and Architecture: Lecture Notes on Computer Science, Springer, Heidelberg,2007
15. Yehuda Kalay Architecture's New Media Intro, chapter 5, chapter 11
16. "Intro to SOM by Teuvo Kohonen". SOM Toolbox. Accessed on 14/06/2008
17. Haykin, Simon (1999). "9. Self-organizing maps". Neural networks - A comprehensive foundation (2nd edition ed.) . Prentice-Hall
18. P. Coates, T. Broughton and H. Jackson (1999), "Exploring Three - Dimensional Design Worlds using Lindenmayer System and Genetic Programming. In "Evolutionary Design by Computers" by Bentley P (1999). Wiley, 323-341

Essential readings

Cariani, P. (1989) Emergence and Artificial Life. In Artificial Life II, vol X, (ed. Langton, et al), Addison - Wesley, Redwood City, CA, p. 775-797

Dawkins, R.(1988) The Blind Watchmaker. Penguin Books, London

Dawkins, R, (1987) The Evolution of Evolvability. Proceedings Artificial Life, VI, Los Alamos: 201-220

Frazer, J.(1995) An evolutionary Architecture. AA Publications. London

Gary William Flake (1998) The computational Beauty of Nature, MIT press

Leach, N. (2006) Swarm Architecture. In Digital Tectonics (eds. Leach, Turnbull & Williams), Wiley-Academy, London: 70- 77

O'Reilly, U. et al (2003) Evolutionary Computation and Artificial Life in Architecture, In AD: Emergence: Morphogenetic Design Strategies.Willey, London

Ursula W. Goodenough & Robert P. Levine. Genetica. Holt, Rinehart and Winton Publishers, New York

```
Option Explicit

'Script written by <gennaro senatore>
'Script copyrighted by <gennaro senatore>
'Script version 05 September 2008 01:02:41

Public gene_length
Public words
Public  word_length
Public  max_pop
Public maxgens
Public numberof_sections
Public numberof_pointsper_section
Public scale_noise_factor
Public wf

Public max_footprint_area
Public min_footprint_area
Public max_volume
Public min_volume
Public max_height
Public max_facade_area
Public min_facade_area
Public max_volume_centroid_height
Public min_volume_centroid_height
Public min_curv_radius_limit
Public max_curv_limit
Public V_over_Facade_Area_max
Public V_over_Facade_Area_min
Public V_over_Footprint_Area_max
Public V_over_Footprint_Area_min
Public Facade_to_Floors_ratio_max

Public w1,w2,w3,w4,w5,w6,w7,w8
Public min_value
Public min_value_height

min_value_height=30

min_value=30
max_footprint_area=12453
min_footprint_area=2823
max_volume=4135916
min_volume=446594
max_height=315
max_facade_area=124614
min_facade_area=28257
max_volume_centroid_height=219
min_volume_centroid_height=60
min_curv_radius_limit=0.01
max_curv_limit=1/min_curv_radius_limit
'-------------------------------------------------------

V_over_Facade_Area_max=max_volume/min_facade_area
V_over_Facade_Area_min=min_volume/max_facade_area
V_over_Footprint_area_max=max_volume/min_footprint_area
V_over_Footprint_area_min=min_volume/max_footprint_area
Facade_to_Floors_ratio_max=1
'-------------------------------------------------------
numberof_sections=6
numberof_pointsper_section=12
words = numberof_pointsper_section+(1+numberof_pointsper_section)*(numberof_sections-1)

```

```
word_length = 6
gene_length = words*word_length
max_pop = 30
wf=1.2     'width scale factor

''''''''''' weights fit_ratio''''''''''''''''''''''

w1=2                'V_over_FA
w2=2                 'V_over_F
w3=5                 'Heightof_Centroid
w4=0.5               'min_curv_radius
w5=10                'solar_gain
w6=10                'wind_exposure
w7=0                 'FA_R

'''''set mutation rate'''''''''

Public mutation_rate
Public mutation_div
mutation_rate = 0.1
Public operator
Public mutation_sens_factor
mutation_sens_factor=0.5

''''''''''''''''''wind analysis'''''''''''''''
'   Source vectors     S-W   -0.7071,-0.7071,0
'                       W   -1,0,0
Public wind_velocity
Public air_density
Public dynamic_pressure

wind_velocity=30
air_density=1.25
dynamic_pressure=(air_density*wind_velocity^2)/2
''''''''''''''''''''''''''''''''''''''''''''''''''''''



Call Main()
Sub Main()


    Dim population()
    ReDim population(max_pop)
    Dim population_temp()
    ReDim population_temp(max_pop)
    Dim population_state()
    ReDim population_state(max_pop)
    Dim population_volume()
    ReDim population_volume(max_pop)
    Dim population_footprint()
    ReDim population_footprint(max_pop)
    Dim population_vol_centroid()
    ReDim population_vol_centroid(max_pop)
    Dim population_height()
    ReDim population_height(max_pop)
    Dim population_cap()
    ReDim population_cap(max_pop)
    Dim population_min_curv_radius()
    ReDim population_min_curv_radius(max_pop)
    Dim population_facade_area()
    ReDim population_facade_area(max_pop)
    Dim population_solar_gain()
    ReDim population_solar_gain(max_pop)
    Dim population_wind_fitness()
    ReDim population_wind_fitness(max_pop)
    Dim population_floors_tot_area()
    ReDim population_floors_tot_area(max_pop)
    Dim panels_body()
    ReDim panels_body(max_pop)
    Dim panels_cap()
    ReDim panels_cap(max_pop)


    Dim  Vol_over_Facade()
    ReDim Vol_over_Facade(max_pop)
    Dim vol_over_Footprint()
    ReDim vol_over_Footprint(max_pop)
    Dim HeightCentroid()
    ReDim HeightCentroid(max_pop)
    Dim Facade_to_Floors_Ratio()
    ReDim Facade_to_Floors_Ratio(max_pop)

    Dim mum,dad,newmum,newdad
    Dim generation
    Dim sumfitness()
    Dim popfitness()
    ReDim popfitness(max_pop)
    Dim oldpop_fitness()
    ReDim oldpop_fitness(max_pop)
    Dim pop_genome()
    Dim oldpop_genome()
    ReDim pop_genome(max_pop)
    ReDim oldpop_genome(max_pop)
    Dim values()
    ReDim values(words)
    Dim startpoint
    Dim seed
    Dim i,j,k

    Dim Vol_over_Facade_mean()
    Dim vol_over_Footprint_mean()
    Dim HeightCentroid_mean()
    Dim min_curv_radius_mean()
    Dim solar_gain_mean()
    Dim wind_fitness_mean()
    Dim Facade_to_Floors_Ratio_mean()

    '''''''''''''''''''''''''''''''''''''''''''''''''''' solar analysis inputs''''''''
    Dim source_vectors_sun()
    ReDim source_vectors_sun(6)
    Dim source_vectors_sun_mean(2)

    'London

    'Latitude               51 degree      30 minutes       0          seconds
    'Longitude              0   degree      6   minutes       59.76 seconds
    '                       21st december
    '                       time 07:00       0.8844,-0.439,-0.1587
    '                       time 9:00        0.6546,-0.7593,0.0961
    '                       time 10:00       0.4525,-0.8722,0.1859
    '                       time 11:00       0.2305,-0.9425,0.2418
    '                       time 12:00      -0.0072,-0.9655,0.2601
    '                       time 13:00      -0.2443,-0.9396,0.2395
    '                       time 14:00      -0.4649,-0.8666,0.1814
    '                       time 15:00      -0.6537,-0.7514,0.0898
    '                       time 16:00      -0.7981,-0.6018,-0.0292
```

```
'                        time 17:00      -0.8881,-0.4282,-0.1673
'                        time 18:00      -0.9176,-0.2422,-0.3152

'    Source vectors      21st june
'                        time 07:00       0.8882,0.0669,0.4546
'                        time 09:00      0.6539,-0.2564,0.7118
'                        time 08:00       0.7982,-0.1068,0.5928
'                        time 10:00       0.465,-0.3717,0.8035
'                        time 10:30       0.3578,-0.4139,0.8371
'                        time 12:00      0.0073,-0.4707,0.8823
'                        time 13:00      -0.2304,-0.4477,0.864
'                        time 16:00      -0.791,-0.1167,0.6006
'                        time 17:00      -0.8844,-0.0559,0.4633
'                        time 18:00      -0.9176,0.2416,0.3157
'                        time 19:00      -0.8883,0.4276,0.1677
'                        time 20:00      -0.7984,0.6014,0.0294

source_vectors_sun(0)=array(0.6546,-0.7593,0.0961)
source_vectors_sun(1)=array(0.4525,-0.8722,0.1859)
source_vectors_sun(2)=array(0.2305,-0.9425,0.2418)
source_vectors_sun(3)=array(-0.0072,-0.9655,0.2601)
source_vectors_sun(4)=array(-0.2443,-0.9396,0.2395)
source_vectors_sun(5)=array(-0.4649,-0.8666,0.1814)
source_vectors_sun(6)=array(-0.6537,-0.7514,0.0898)


Dim panels_norm_body,panels_norm_cap
Dim panels_heat_body(),panels_heat_cap()
''''''''''''''''''wind analysis inputs''''''''''''''''''''''''
Dim source_vectors_wind()
ReDim source_vectors_wind(0)
Dim source_vectors_wind_mean(0)
Dim index

source_vectors_wind(0)=array(1,0,0)
index=0
Dim panels_pressure_body(),panels_pressure_cap()
Dim cpe_a(7,12)
Dim cpe_b(7,12)
Dim cpe_c(7,12)
make_cpe_a cpe_a
make_cpe_b cpe_b
make_cpe_c cpe_c
Dim mesh_faces_area_body(),mesh_faces_area_cap()
Dim a,b
a=array(0,0,0)
b=array(0,0,0)
Erase a
Erase b

''''''''''''''''''''''' mesh division''''''''''''''''''''''''
Dim num_u_body,num_v_body
Dim num_u_cap,num_v_cap

num_u_body=40
num_v_body=40

num_u_cap=20
num_v_cap=10
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
rhino.Command "selall delete"
rhino.EnableRedraw False
Dim camera(2),target(2)
```

```
Dim cameraheight,camera_x_rnd,camera_y_rnd

''''''''''''''''get max number of generations''''''''''''''''''
seed=rhino.GetReal("seed",654)
maxgens=rhino.GetReal("maxgeneration",5)
' rhino.command "fullscreen"

ReDim sumfitness(maxgens)
ReDim Vol_over_Facade_mean(maxgens)
ReDim vol_over_Footprint_mean(maxgens)
ReDim HeightCentroid_mean(maxgens)
ReDim min_curv_radius_mean(maxgens)
ReDim solar_gain_mean(maxgens)
ReDim wind_fitness_mean(maxgens)
ReDim Facade_to_Floors_Ratio_mean(maxgens)

rnd(-1)
Randomize (seed)
startpoint=array(0,0,0)

'----------- making the gemome ----------

For i=1 To max_pop
    make pop_genome(i)
Next

generation=1
For generation=1 To maxgens

    rhino.AddText "generation"&generation,array(startpoint(0)-300,startpoint(1),

    ReDim population(max_pop)
    ReDim population_temp(max_pop)
    ReDim population_state(max_pop)
    ReDim population_volume(max_pop)
    ReDim population_base_area(max_pop)
    ReDim population_heightofcentroid(max_pop)
    ReDim population_height(max_pop)
    ReDim popfitness(max_pop)
    ReDim population_cap(max_pop)
    ReDim population_min_curv_radius(max_pop)
    ReDim population_facade_area(max_pop)
    ReDim population_solar_gain(max_pop)
    ReDim population_wind_fitness(max_pop)
    ReDim population_floors_tot_area(max_pop)
    ReDim Vol_over_Facade(max_pop)
    ReDim vol_over_Footprint(max_pop)
    ReDim HeightCentroid(max_pop)
    ReDim Facade_to_Floors_Ratio(max_pop)
    ReDim panels_body(max_pop)
    ReDim panels_cap(max_pop)

    sumfitness(generation) = 0

    i=1
    For i=1 To max_pop

        popfitness(i)=0
        population_state(i)=True

        '----------- decoding the genome ----------
        decode pop_genome(i), values
```

```
startpoint(0)=startpoint(0)+1000
rhino.AddText "pop"&i,array(startpoint(0),startpoint(1)-180,_
    _startpoint(2)),20,,0

shapemaking startpoint,values,population_temp(i),_
    _   population_height(i),population_state(i),generation

If population_state(i)=True Then

    population(i)=population_temp(i)(0)

    '----------- evaluation-fitness_function ----------
    first_selection population(i),population_state(i),population_volume(i)_
    _   ,population_footprint(i),population_vol_centroid(i)

    If population_state(i)=True Then

        cameraheight=rnd*8
        camera_x_rnd=rnd*4
        camera_y_rnd=rnd*5
        For j=0 To 90 Step 1
            camera(0)=startpoint(0)-camera_x_rnd*100+800*_
                _   cos(2*j*rhino.Pi/180)
            camera(1)=startpoint(1)-camera_y_rnd*100-1200_
            _   *sin(j*rhino.Pi/180)
            camera(2)=startpoint(2)+500*sin(j*rhino.Pi/180)
            Rhino.ViewCameraTarget ,camera,startpoint
        Next

        capsurf population(i),population_cap(i)
        analysis population(i),population_volume(i),_
            _   population_facade_area(i),population_min_curv_radius(i)

        '''''''' solar analysis''''''''''
        tri_facets  population(i),panels_body(i),panels_norm_body,_
            _   num_u_body,num_v_body,startpoint

        If panels_body(i)<>False Then

            get_floors panels_body(i),startpoint,
                population_floors_tot_area(i)
            Facade_to_Floors_Ratio(i)=population_facade_area(i)(0)_
            _   /population_floors_tot_area(i)

            tri_facets_cap  population_cap(i),panels_cap(i),_
            _   panels_norm_cap,num_u_cap,num_v_cap

            ReDim panels_heat_body(ubound(panels_norm_body))
            ReDim panels_heat_cap(ubound(panels_norm_cap))

            dot_solar panels_body(i),panels_norm_body,panels_heat_body,_
            _   source_vectors_sun,generation
            dot_solar panels_cap(i),panels_norm_cap,panels_heat_cap,_
                _   source_vectors_sun,generation

            get_sun_solar_gain panels_heat_body,panels_heat_cap,_
                _population_solar_gain(i)

            Erase panels_heat_body
            Erase panels_heat_cap

            ''''''''wind analysis''''''''''''''''''''''''''''''''''''''''

            ReDim panels_pressure_body(ubound(panels_norm_body))
            ReDim panels_pressure_cap(ubound(panels_norm_cap))

            bounding_box panels_body(i),
                source_vectors_wind,a,b
            dot_wind panels_body(i),panels_norm_body
                ,panels_pressure_body,_
                _source_vectors_wind,cpe_a,
                cpe_b,cpe_c,a,b,index,generation
            dot_wind panels_cap(i),panels_norm_cap,
                panels_pressure_cap,_
                _source_vectors_wind,cpe_a,
                cpe_b,cpe_c,a,b,index,generation

            get_mesh_faces_area panels_body,panels_
            _   cap,mesh_faces_area_body,_
                _mesh_faces_area_cap_
            get_wind_fitness panels_body(i),panels_cap(i)
            _   ,panels_pressure_body,_
                _panels_pressure_cap,_
                _   population_wind_fitness(i),generation_
            get_wind_force  panels_body,panels_cap,_
                _panels_pressure_body,_
                _panels_pressure_cap,panels_norm_body,_
            _   panels_norm_cap,_
                _mesh_faces_area_body,mesh_faces_area_cap
            _,num_u_body,num_v_cap

            Erase panels_norm_body
            Erase panels_norm_cap
            Erase panels_pressure_body
            Erase panels_pressure_cap

            Erase a
            Erase b

            ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
            'calcualte fitness

            fitness_function popfitness(i),population_volume(i)
                ,population_footprint(i),_
                _population_vol_centroid(i),_
                _population_height(i),_
                _population_facade_area(i),_
                _population_min_curv_radius(i),_
                _population_solar_gain(i),_
                _population_wind_fitness(i),_
                _Facade_to_Floors_Ratio(i)_
                ,i,generation

            Vol_over_Facade(i)=population_volume(i)(0)/population_facade_area(i)(0)
            vol_over_Footprint(i)=population_volume(i)(0)/population_footprint(i)(0)
            HeightCentroid(i)=population_vol_centroid(i)(2)

            rhino.AddText "Fitness="&CStr(popfitness(i)),_
                _array(startpoint(0),startpoint(1)-250,startpoint(2)),20,,0
            rhino.AddText "Facade_to_Floors_Ratio="&CStr(Facade_to_Floors_Ratio(i)),_
                _array(startpoint(0),startpoint(1)-300,startpoint(2)),20,,0
            rhino.AddText "Wind_Fitness="&CStr(population_wind_fitness(i)),_
                _array(startpoint(0),startpoint(1)-300,startpoint(2)),20,,0

        Else
            popfitness(i)=0
```

```
                    rhino.AddText "wrong solution",array(startpoint(0),_
                        _startpoint(1)-200,startpoint(2)),20,,1
                End If

            Else
                popfitness(i)=0
                rhino.AddText "wrong solution",array(startpoint(0),_
                _   startpoint(1)-200,startpoint(2)),20,,1
            End If

        Else
            popfitness(i)=0
            rhino.AddText "wrong solution",array(startpoint(0),_
            _   startpoint(1)-200,startpoint(2)),20,,1
        End If

        sumfitness(generation)=sumfitness(generation)+popfitness(i)

        oldpop_genome(i)=pop_genome(i)
        oldpop_fitness(i)=popfitness(i)


        If (generation<maxgens)  Then

            If isempty(population(i))=False  Then
                If isnull(population(i))=False Then
                    rhino.DeleteObject population(i)
                End If
            End If

            If isempty(population_cap(i))=False  Then
                If isnull(population_cap(i))=False Then
                    rhino.DeleteObject population_cap(i)
                End If
            End If

        End If

    Next

    If generation < maxgens Then
        i=1

        '''''''''''''''''''' set mutation_rate''''''''''''''''''''
        If generation>1 Then

            If sumfitness(generation)>sumfitness(generation-1) Then
                mutation_div=sumfitness(generation)
                operator=1
            Else
                mutation_div=sumfitness(generation-1)
                operator=-1
            End If

        mutation_rate=mutation_rate+operator*_
            _   ((abs(sumfitness(generation)-sumfitness(generation-1)))_
            _/mutation_div)^mutation_sens_factor

            If mutation_rate>1 Then mutation_rate=1
            If mutation_rate<0 Then mutation_rate=0

        End If
```

```
        '---------- natural selection Goldberg roulette ----------
        For i = 1 To max_pop - 1 Step 2

            mum = roulette(sumfitness(generation),oldpop_fitness)
            dad = roulette(sumfitness(generation),oldpop_fitness)

            '---------- crossover ----------

        crossoverSplice oldpop_genome(mum), oldpop_genome(dad)_
        _   , pop_genome(i), pop_genome(i + 1)

            '---------- mutation  --------------

            If Rnd > mutation_rate Then
                mutate pop_genome(i)
                mutate pop_genome(i + 1)
            End If

        Next

    End If

    For j=0 To max_pop*10

        camera(0)=startpoint(0)+300-80*j
        camera(1)=startpoint(1)-1000
        camera(2)=startpoint(2)+300
        target(0)=800
        target(1)=startpoint(1)
        target(2)=startpoint(2)
        Rhino.ViewCameraTarget ,camera,target

    Next

    startpoint(0)=0
    startpoint(1)=startpoint(1)-1300

    Vol_over_Facade_mean(generation)=mean(Vol_over_Facade)
    vol_over_Footprint_mean(generation)=mean(vol_over_Footprint)
    Facade_to_Floors_Ratio_mean(generation)=mean(Facade_to_Floors_Ratio)
    HeightCentroid_mean(generation)=mean(HeightCentroid)
    min_curv_radius_mean(generation)=mean(population_min_curv_radius)
    solar_gain_mean(generation)=mean(population_solar_gain)
    wind_fitness_mean(generation)=mean(population_wind_fitness)

    rhino.Print "gen="&CStr(generation)&"   gen fitness="&CStr(sumfitness(generation))
    rhino.print "mutation_rate="&CStr(mutation_rate)

Next

excel_output sumfitness,Vol_over_Facade_mean,vol_over_Footprint_mean,_
_   Facade_to_Floors_Ratio_mean,_   _HeightCentroid_mean,
    _   min_curv_radius_mean,solar_gain_mean,wind_fitness_mean
_   ,mutation_rate,words,scale_noise_factor

rhino.EnableRedraw True

End Sub

Sub make(ByRef genome)

    Dim j
    genome = ""
```

```
    For j = 1 To gene_length
        If Rnd < 0.5 Then
            genome = genome + "1"
        Else
            genome = genome + "0"
        End If
    Next
End Sub

Sub decode(ByRef genome,ByRef values())
    Dim temp , pos , endpos
    Dim i, j
    Dim stringvalue

    For i = 1 To words
        temp = 0
        endpos = i * word_length

        'work backwards through string
        For j = 0 To word_length-1
            pos = endpos - j

            stringvalue= Mid(genome,pos,1)

            If stringvalue = "1" Then
                temp = temp + 2 ^ j
            End If

        Next
        'set a minimum value for the dimension of the domain
        If temp <min_value  Then temp = min_value
        values(i) = temp

    Next

End Sub

Function random(lower , upper )
    random = Int((upper - lower + 1) * Rnd + lower)
End Function

'------- crossover splice chooses a random split point in the genes and swaps the
'------- two over a b , c d > a d , c b
Sub crossoverSplice(ByRef mum ,ByRef dad,ByRef newmum ,ByRef newdad )

    Dim start

    start = random(1, gene_length - 1)

    newmum = Left(mum, start) + Mid(dad, start + 1)
    newdad = Left(dad, start) + Mid(mum, start + 1)

End Sub

Sub mutate(ByRef genome)
    Dim pos
    Dim stringvalue

    pos = random(1, Len(genome))
    stringvalue=Mid(genome, pos, 1)

    If stringvalue = "0" Then
        genome=left(genome,pos-1)+"1"+Mid(genome, pos+1)
    Else
        genome=left(genome,pos-1)+"0"+Mid(genome, pos+1)

    End If

End Sub

Function roulette(ByRef sumfitness,ByRef oldpop_fitness())
    ' select a single individual via weighted roulette wheel selection

    Dim treshold , partsum , j
    partsum = 0
    j = 0

    treshold = Rnd * sumfitness

    Do
        j = j + 1
        partsum = partsum + oldpop_fitness(j)
    Loop Until ((partsum > treshold) Or (j = max_pop))
    roulette = j


End Function

 Sub shapefinding (ByRef startpoint(),ByRef values(),ByRef individual_temp,height,_
    _    ByRef individual_state,ByVal generation)

    Dim i,j
    Dim arrpoints_coor()
    ReDim arrpoints_coor(numberof_pointsper_section)
    Dim arrpoints()
    Dim sections()
    ReDim sections(numberof_sections-1)
    Dim k:k=1
    Dim previous_height
    Dim check

    check=True
    j=0

    initialise_firstorlast_section startpoint,arrpoints_coor,_
    _    values,k,previous_height,sections,j
    k=1+numberof_pointsper_section

    For j=1 To numberof_sections-2

        initialise  startpoint, arrpoints_coor, values,_
        _    k,previous_height,sections,j
        k=k+1+numberof_pointsper_section

    Next

    initialise_firstorlast_section startpoint,arrpoints_coor,_
    _    values,k,previous_height,sections,j
    k=k+1+numberof_pointsper_section


    For j=0 To ubound(sections)
        If isnull(sections(j))=True Then check=False
        If isempty(sections(j))=True Then check=False
```

```
    Next

    If check=True Then
        individual_temp=Rhino.AddLoftSrf(sections,,,0,1,10)
        height=previous_height

        If generation<maxgens Then
            rhino.DeleteObjects sections
        End If

    Else
        individual_state=False
    End If



End Sub

Sub initialise_firstorlast_section(ByRef startpoint(),_
    _   ByRef arrpoints_coor(),ByRef values()_
    _   ,ByRef k,ByRef previous_height,ByRef sections,ByVal who)


    Dim relative_center
    Dim points
    Dim i
    Dim alfa,radius

    If k=1 Then

        relative_center=array(startpoint(0),startpoint(1),startpoint(2))
        previous_height=relative_center(2)
    Else

        relative_center=array(startpoint(0),startpoint(1),_
        _   startpoint(2)+previous_height+values(k))
        If values(k)<min_value_height Then
            relative_center(2)=relative_center(2)+min_value_height
        End If
        previous_height=relative_center(2)
    End If


    If k=1 Then

        For i=0 To ubound(arrpoints_coor)-1
            alfa=(rhino.Pi*360/(ubound(arrpoints_coor)))/180
            radius=values(k+i)
            arrpoints_coor(i)=array(relative_center(0)+cos(alfa*i)*radius*wf,_
                _   relative_center(1)+sin(alfa*i)*radius*wf,relative_center(2))
        Next


    Else

        For i=0 To ubound(arrpoints_coor)-1
            alfa=(rhino.Pi*360/(ubound(arrpoints_coor)))/180
            radius=values(k+1+i)
            arrpoints_coor(i)=array(relative_center(0)+cos(alfa*i)*radius*wf,_
                _   relative_center(1)+sin(alfa*i)*radius*wf,relative_center(2))
        Next

    End If
```

```


    'points=rhino.AddPointCloud (arrpoints_coor)

    arrpoints_coor(ubound(arrpoints_coor))=arrpoints_coor(0)

    sections(who)=rhino.AddInterpCurve (arrpoints_coor,3,3)

    rhino.faircurve sections(who),1

End Sub

Sub initialise (ByRef startpoint(),ByRef arrpoints_coor(),ByRef values(),ByRef k,_
    _   ByRef previous_height,ByRef sections,ByVal who)


    Dim relative_center
    Dim points
    Dim i
    Dim alfa,radius

    relative_center=array(startpoint(0),startpoint(1),_
        _   startpoint(2)+previous_height+values(k))

    If values(k)<min_value_height Then
        relative_center(2)=relative_center(2)+min_value_height
    End If

    previous_height=relative_center(2)


    For i=0 To ubound(arrpoints_coor)-1
        alfa=(rhino.Pi*360/(ubound(arrpoints_coor)))/180
        radius=values(k+1+i)
        arrpoints_coor(i)=array(relative_center(0)+_
            _   cos(alfa*i)*radius*wf,relative_center(1)_
            _   +sin(alfa*i)*radius*wf,relative_center(2))
    Next

    'points=rhino.AddPointCloud (arrpoints_coor)

    arrpoints_coor(ubound(arrpoints_coor))=arrpoints_coor(0)

    sections(who)=rhino.AddInterpCurve (arrpoints_coor,3,3)

    rhino.faircurve sections(who),1


End Sub

Sub first_selection (ByRef individual,ByRef individual_state,ByRef individual_volume,_
    _   ByRef individual_footprint,ByRef individual_vol_centroid)

    Dim individual_temp
    Dim individual_temp_explode

    Dim vol_centroid
    Dim base_centroid_temp
    Dim vol_centroid_projection
    Dim check1,check2
    Dim i

    'individual_temp=rhino.CopyObject(individual)
```

```
    check1= rhino.CapPlanarHoles(individual)

    vol_centroid=rhino.SurfaceVolumeCentroid(individual)

    If isnull(vol_centroid)=False Then
        individual_volume=rhino.SurfaceVolume(individual)
        vol_centroid_projection=array(vol_centroid(0)(0),vol_centroid(0)(1),0)
        individual_vol_centroid=array(vol_centroid(0)(0),vol_centroid(0)(1),vol_centro

        'rhino.AddPoint vol_centroid_projection

        individual_temp_explode=rhino.ExplodePolysurfaces(individual,True)
        individual_footprint=rhino.SurfaceArea(individual_temp_explode(1))
        check2=rhino.IsPointOnSurface(individual_temp_explode(1),vol_centroid_projecti

        '   For i=0 To ubound(individual_temp_explode)
        '       rhino.DeleteObject individual_temp_explode(i)
        '   Next

    End If

    If check1=False Or isnull(check1)=True Or check2=False Or _
        _   isnull(check2)=True Or isnull(vol_centroid)=True Then
        individual_state=False
    End If

    '   rhino.DeleteObject individual_temp
    '   rhino.SelectObject individual_temp_explode(2)
    If isarray(individual_temp_explode)=True Then

        individual=individual_temp_explode(2)
        rhino.DeleteObject individual_temp_explode(0)
        rhino.DeleteObject individual_temp_explode(1)

        '   rhino.SelectObject individual
    Else
        individual_state=False
    End If

End Sub

Sub capsurf ( ByRef individual_temp,pop_cap)

    Dim startcurve
    Dim curve
    Dim curve_domain

    Dim arrpoints1()
    ReDim arrpoints1(0)
    Dim arrpoints2()
    ReDim arrpoints2(0)

    Dim i,t
    Dim t_step

    Dim idisocurvesu,idisocurveu
    Dim id_isocurvesv1,id_isocurvesv2

    Dim id_isocurvev1
    ReDim id_isocurvev1(0)

    Dim id_isocurvev2()
    ReDim id_isocurvev2(0)
    Dim cap

    '   startcurve=rhino.DuplicateSurfaceBorder(individual_temp)
    startcurve=rhino.DuplicateEdgeCurves(individual_temp)

    curve=startcurve(0)
    '   rhino.SelectObject curve
    curve_domain=rhino.CurveDomain(curve)

    If curve_domain(0)<0 Or curve_domain(1)<(0) Then
        rhino.ReverseCurve curve
        curve_domain=rhino.CurveDomain(curve)
    End If

    t_step=(curve_domain(1)-curve_domain(0))/40

    For t=0 To curve_domain(1)/2 Step t_step

        arrpoints1(i)=rhino.EvaluateCurve(curve ,t)
        arrpoints2(i)=rhino.EvaluateCurve(curve,curve_domain(1)-t)

        '   rhino.AddPoint arrpoints1(i)
        '   rhino.AddPoint arrpoints2(i)
        'rhino.AddLine arrpoints1(i),arrpoints2(i)
        ReDim Preserve arrpoints1(ubound(arrpoints1)+1)
        ReDim Preserve arrpoints2(ubound(arrpoints2)+1)

        i=i+1

    Next

    ReDim Preserve arrpoints1(ubound(arrpoints1)-1)
    ReDim Preserve arrpoints2(ubound(arrpoints2)-1)

    Dim trimcurve1
    Dim trimcurve2

    trimcurve1= Rhino.TrimCurve_
    _   (curve,array(curve_domain(1)-t_step,curve_domain(0)+t_step),False)
    trimcurve2= Rhino.TrimCurve _
    _(curve,array(curve_domain(1)/2-t_step,curve_domain(1)/2+t_step),False)

    rhino.ReverseCurve trimcurve1

    'rhino.DeleteObject curve
    rhino.DeleteObjects startcurve

    Dim surfparams1()
    ReDim surfparams1(ubound(arrpoints1))
    Dim surfparams2()
    ReDim surfparams2(ubound(arrpoints2))

    For i=0 To ubound(arrpoints1)
        surfparams1(i)=rhino.SurfaceClosestPoint(individual_temp,arrpoints1(i))
        surfparams2(i)=rhino.SurfaceClosestPoint(individual_temp,arrpoints2(i))
    Next


    For i=0 To ubound(surfparams1)

        id_isocurvesv1= rhino.ExtractIsoCurve_
        _   (individual_temp,array(0,surfparams1(i)(1)),0)
```

```
        id_isocurvesv2=rhino.ExtractIsoCurve_
        _    (individual_temp,array(0,surfparams2(i)(1)),0)

        id_isocurvev1(i)=id_isocurvesv1(0)
        id_isocurvev2(i)=id_isocurvesv2(0)

        ReDim Preserve id_isocurvev1(ubound(id_isocurvev1)+1)
        ReDim Preserve id_isocurvev2(ubound(id_isocurvev2)+1)

    Next

    ReDim Preserve id_isocurvev1(ubound(id_isocurvev1)-1)
    ReDim Preserve id_isocurvev2(ubound(id_isocurvev2)-1)

    Dim curves1_domain
    Dim curves2_domain
    Dim arrtang1()
    ReDim arrtang1(ubound(id_isocurvev1))
    Dim arrtang2()
    ReDim arrtang2(ubound(id_isocurvev2))

    For i=0 To ubound(id_isocurvev1)

        arrpoints1(i) = rhino.CurveEndPoint(id_isocurvev1(i))
        curves1_domain=rhino.CurveDomain(id_isocurvev1(i))
        arrtang1(i)=rhino.CurveTangent(id_isocurvev1(i),curves1_domain(1))
        '   arrtang1(i)=rhino.VectorUnitize(arrtang1(i))
        arrtang1(i)=rhino.VectorScale(arrtang1(i),0.43)
    Next

    For i=0 To ubound(id_isocurvev2)
        arrpoints2(i)=  rhino.CurveEndPoint(id_isocurvev2(i))
        curves2_domain=rhino.CurveDomain(id_isocurvev2(i))
        arrtang2(i)=rhino.CurveTangent(id_isocurvev2(i),curves2_domain(1))
        arrtang2(i)=rhino.VectorReverse(arrtang2(i))
        '   arrtang2(i)=rhino.VectorUnitize(arrtang2(i))
        arrtang2(i)=rhino.VectorScale(arrtang2(i),0.43)
    Next

    Dim loft_curve()
    ReDim loft_curve(ubound(arrpoints1))

    For i=1 To ubound(arrpoints1)
        loft_curve(i)=Rhino.AddInterpCurve _
        _(array(arrpoints1(i),arrpoints2(i)),3,2,arrtang1(i),arrtang2(i))
    Next


    rhino.DeleteObjects id_isocurvev1
    rhino.DeleteObjects id_isocurvev2

    loft_curve(0)=trimcurve1
    loft_curve(ubound(loft_curve))=trimcurve2

    cap=rhino.AddLoftSrf (loft_curve,,,3,1,10)
    rhino.DeleteObjects loft_curve

    '   get_height cap(0),height

    pop_cap=cap(0)

End Sub
```

```
Sub get_floors (ByVal panels_body,ByVal startpoint(), ByRef individual_floors_tot_area)

    Dim arrcontours
    Dim i
    Dim contours
    Dim mesh_floors_area(),floor_centroid()

    Dim endpoint:endpoint=array(startpoint(0),startpoint(1),startpoint(2)+500)

    Dim vsub(),vortho(),dot(),sum()
    Dim j
    Dim base_point
    Dim vec_z
    Dim sum_dot

    vec_z=array(0,0,1)
    arrContours = Rhino.MeshContourPoints(panels_body,startpoint,endpoint,3)
    ReDim mesh_floors_area(ubound(arrcontours))
    ReDim floor_centroid(ubound(arrcontours))

    For j=0 To ubound(arrcontours)
        ReDim vsub(ubound(arrcontours(j)))
        ReDim vortho(ubound(arrcontours(j)))
        ReDim dot(ubound(arrcontours(j)))
        ReDim sum(ubound(arrcontours(j)))
        sum_dot=0

        base_point=array(startpoint(0),startpoint(1),arrcontours(j)(0)(2))

        For i=0 To ubound(arrcontours(j))
            vsub(i)=rhino.VectorSubtract(arrcontours(j)(i),base_point)
        Next

        For i=0 To ubound(arrcontours(j))

            vortho(i)=Rhino.VectorCrossProduct (vsub(i),vec_z)
        Next

        For i=0 To ubound(arrcontours(j))
            If i<ubound(vsub) Then
                dot(i)=Rhino.VectorDotProduct (vsub(i), vortho(i+1))
            Else
                dot(i)=Rhino.VectorDotProduct (vsub(i), vortho(0))
            End If
        Next

        For i=0 To ubound(arrcontours(j))
            sum_dot=sum_dot+dot(i)

        Next

        mesh_floors_area(j)=sum_dot*0.5
        _floor_centroid(j)=array(base_point(0)+S(0)/
        mesh_floors_area(j),base_point(1)+S(1)/mesh_floors_area(j),S(2))
        individual_floors_tot_area=individual_floors_tot_area_
        _+(mesh_floors_area(j)-mesh_floors_area(j)*20\100)
    Next

End Sub

Sub analysis (ByVal individual,ByRef individual_volume,_
    _    ByRef individual_facade_area,ByRef min_curv_radius)

```

```
    Dim cap_volume
    Dim individual_curv_analysis
    Dim cap_curv_analysis
    Dim cap_area

    individual_facade_area=rhino.SurfaceArea(individual)
    individual_curv_analysis=rhino.SurfaceCurvatureAnalysis(individual)
    min_curv_radius=individual_curv_analysis(3)(0)

End Sub

Sub tri_facets (ByVal id_surf,ByRef panels_body,_
    _    ByRef panels_norm_body(),num_u,num_v,ByVal startpoint)


    Dim domU, domV
    Dim uMin, uMax, vMin, vMax
    Dim uInc, vInc

    domU = Rhino.SurfaceDomain(id_Surf, 0)
    domV = Rhino.SurfaceDomain(id_Surf, 1)

    uMin = domU(0)
    uMax = domU(1)
    vMin = domV(0)
    vMax = domV(1)

    uInc = (uMax-uMin)/num_u
    vInc = (vMax-vMin)/num_v

    Dim i, j
    Dim mesh_points()
    ReDim mesh_points(0)
    Dim mesh_face_vertices()
    Dim counter:counter=0


    For i = uMin To uMax+uinc/2 Step uInc
        For j = vMin To vMax-vInc/2 Step vInc

            mesh_points(counter)= Rhino.EvaluateSurface(id_Surf, Array(i, j))
            ReDim Preserve mesh_points(ubound(mesh_points)+1)
            counter=counter+1

        Next

    Next

    ReDim Preserve mesh_points(ubound(mesh_points)-1)

    Dim header
    counter=-1
    For i=0 To num_u
        For j=0 To num_v-1

            header=i*(num_v)+j
            counter=counter+2
            ReDim Preserve mesh_face_vertices(counter)
            If j=num_v-1 Then
                mesh_face_vertices(counter-1)=_
                _    array(header,header+1,header+num_v,header+num_v)
                mesh_face_vertices(counter)=_
                _    array(header,header-num_v+1,header+1,header+1)
            Else
                mesh_face_vertices(counter-1)=-
                _    array(header,header+num_v+1,header+num_v,header+num_v)
                mesh_face_vertices(counter)=_
                    _    array(header,header+1,header+num_v+1,header+num_v+1)
            End If


        Next
    Next

    If isarray(mesh_points)=True And isarray(mesh_face_vertices)=True Then

        panels_body=    rhino.AddMesh (mesh_points,mesh_face_vertices)
        panels_norm_body = Rhino.MeshVertexNormals(panels_body)
    Else
        panels_body=False
    End If


End Sub

Sub tri_facets_cap (ByVal id_surf_cap,ByRef panels_cap,_
    _    ByRef panels_norm_cap(),num_u,num_v)

    Dim domU, domV
    Dim uMin, uMax, vMin, vMax
    Dim uInc, vInc


    domU = Rhino.SurfaceDomain(id_surf_cap, 0)
    domV = Rhino.SurfaceDomain(id_surf_cap, 1)

    uMin = domU(0)
    uMax = domU(1)
    vMin = domV(0)
    vMax = domV(1)

    uInc = (uMax-uMin)/num_u
    vInc = (vMax-vMin)/num_v

    Dim i, j
    Dim mesh_points_cap()
    ReDim mesh_points_cap(0)
    Dim mesh_face_vertices_cap()
    Dim counter:counter=0


    For i = uMin To uMax+uinc/2 Step uInc
        For j = vMin To vMax+vinc/2 Step vInc

            mesh_points_cap(counter)= Rhino.EvaluateSurface(id_Surf_cap, Array(i, j))
            ReDim Preserve mesh_points_cap(ubound(mesh_points_cap)+1)
            counter=counter+1
        Next
    Next

    ReDim Preserve mesh_points_cap(ubound(mesh_points_cap)-1)

    Dim header
    counter=-1
    For i=0 To num_u
        For j=0 To num_v-1
```

```
        For j=0 To num_v-1

            header=i*(num_v+1)+j
            counter=counter+2
            ReDim Preserve mesh_face_vertices_cap(counter)
            mesh_face_vertices_cap(counter-1)=_
                _   array(header,header+num_v+2,header+num_v+1,header+num_v+1)
            mesh_face_vertices_cap(counter)=_
                _   array(header,header+1,header+num_v+2,header+num_v+2)
        Next
    Next

    panels_cap=rhino.AddMesh (mesh_points_cap,mesh_face_vertices_cap)
    panels_norm_cap = Rhino.MeshVertexNormals(panels_cap)

    For i=0 To ubound(panels_norm_cap)
        panels_norm_cap(i)=rhino.VectorReverse(panels_norm_cap(i))
    Next

End Sub

Sub dot_solar(ByVal panels,ByVal panels_norm(),_
    _ByRef panels_heat(),ByVal source_vectors_sun(),ByVal generation)

    Dim i,j
    Dim rgb_value
    Dim red,green,blue
    Dim incident_angle

    If generation=maxgens Then
        Dim arrcolrs()
        ReDim arrColors( Rhino.MeshVertexCount(panels)-1 )
    End If


    For i = 0 To UBound(panels_norm)

        For j = 0 To ubound(source_vectors_sun)

            incident_angle =rhino.VectorDotProduct_
            _   (panels_norm(i),source_vectors_sun(j))

            If (incident_angle >= 0) Then

                panels_heat(i) = panels_heat(i) + incident_angle
            End If
        Next

        'average panel exposure angle
        average = average / 3

        panels_heat(i)=panels_heat(i)/(ubound(source_vectors_sun)+1)

        If generation=maxgens Then

            red=255 * panels_heat(i)
            green=0
            blue=255 - 255 * panels_heat(i)

            arrcolors(i)=rgb(red,green,blue)
        End If

    Next


    If generation=maxgens Then
        Rhino.MeshVertexColors panels, arrcolors
    End If

End Sub

Sub get_sun_solar_gain(ByVal panels_heat_body(),_
    _ByVal panels_heat_cap(),ByRef sunfitness)

    Dim i
    Dim panel_fitness
    Dim max

    max=ubound(panels_heat_body)+ubound(panels_heat_cap)+2

    For i=0 To ubound(panels_heat_body)
        If panels_heat_body(i)>0.6 Then
            panel_fitness=1'panels_heat_body(i)
            sunfitness=sunfitness+panel_fitness
        End If
    Next

    For i=0 To ubound(panels_heat_cap)
        If panels_heat_cap(i)>0.6 Then
            panel_fitness=1'panels_heat_cap(i)
            sunfitness=sunfitness+panel_fitness
        End If
    Next

    sunfitness=sunfitness/max
    'sunfitness=1-sunfitness


End Sub

Sub make_cpe_a (ByRef cpe_a())

    cpe_a(0,0)=array(90,array(1,0))
    cpe_a(0,1)=array(90,array(1.07,15))
    cpe_a(0,2)=array(90,array(1.10,30))
    cpe_a(0,3)=array(90,array(1.12,45))
    cpe_a(0,4)=array(90,array(0.54,60))
    cpe_a(0,5)=array(90,array(-1.10,75))
    cpe_a(0,6)=array(90,array(-1.30,90))
    cpe_a(0,7)=array(90,array(-0.8,105))
    cpe_a(0,8)=array(90,array(-0.63,120))
    cpe_a(0,9)=array(90,array(-0.50,135))
    cpe_a(0,10)=array(90,array(-0.34,150))
    cpe_a(0,11)=array(90,array(-0.30,165))
    cpe_a(0,12)=array(90,array(-0.34,180))

    cpe_a(1,0)=array(75,array(0.58,0))
    cpe_a(1,1)=array(75,array(0.71,15))
    cpe_a(1,2)=array(75,array(0.85,30))
    cpe_a(1,3)=array(75,array(0.82,45))
    cpe_a(1,4)=array(75,array(0.78,60))
    cpe_a(1,5)=array(75,array(-1.10,75))
    cpe_a(1,6)=array(75,array(-1.21,90))
    cpe_a(1,7)=array(75,array(-0.8,105))
    cpe_a(1,8)=array(75,array(-0.63,120))
    cpe_a(1,9)=array(75,array(-0.50,135))
    cpe_a(1,10)=array(75,array(-0.34,150))
```

```
    cpe_a(1,11)=array(75,array(-0.30,165))
    cpe_a(1,12)=array(75,array(-0.34,180))

    cpe_a(2,0)=array(60,array(0.50,0))
    cpe_a(2,1)=array(60,array(0.63,15))
    cpe_a(2,2)=array(60,array(0.77,30))
    cpe_a(2,3)=array(60,array(0.68,45))
    cpe_a(2,4)=array(60,array(0.59,60))
    cpe_a(2,5)=array(60,array(-1.10,75))
    cpe_a(2,6)=array(60,array(-1.21,90))
    cpe_a(2,7)=array(60,array(-0.8,105))
    cpe_a(2,8)=array(60,array(-0.63,120))
    cpe_a(2,9)=array(60,array(-0.50,135))
    cpe_a(2,10)=array(60,array(-0.34,150))
    cpe_a(2,11)=array(60,array(-0.30,165))
    cpe_a(2,12)=array(60,array(-0.34,180))

    cpe_a(3,0)=array(45,array(0.42,0))
    cpe_a(3,1)=array(45,array(0.53,15))
    cpe_a(3,2)=array(45,array(0.65,30))
    cpe_a(3,3)=array(45,array(0.57,45))
    cpe_a(3,4)=array(45,array(0.50,60))
    cpe_a(3,5)=array(45,array(-1.10,75))
    cpe_a(3,6)=array(45,array(-1.49,90))
    cpe_a(3,7)=array(45,array(-1.15,105))
    cpe_a(3,8)=array(45,array(-1.05,120))
    cpe_a(3,9)=array(45,array(-0.75,135))
    cpe_a(3,10)=array(45,array(-0.52,150))
    cpe_a(3,11)=array(45,array(-0.57,165))
    cpe_a(3,12)=array(45,array(-0.62,180))

    cpe_a(4,0)=array(30,array(-0.60,0))
    cpe_a(4,1)=array(30,array(-0.50,15))
    cpe_a(4,2)=array(30,array(-0.40,30))
    cpe_a(4,3)=array(30,array(-0.61,45))
    cpe_a(4,4)=array(30,array(-0.81,60))
    cpe_a(4,5)=array(30,array(-1.10,75))
    cpe_a(4,6)=array(30,array(-1.54,90))
    cpe_a(4,7)=array(30,array(-1.60,105))
    cpe_a(4,8)=array(30,array(-1.69,120))
    cpe_a(4,9)=array(30,array(-1.43,135))
    cpe_a(4,10)=array(30,array(-1.17,150))
    cpe_a(4,11)=array(30,array(-0.96,165))
    cpe_a(4,12)=array(30,array(-0.76,180))

    cpe_a(5,0)=array(15,array(-0.90,0))
    cpe_a(5,1)=array(15,array(-0.76,15))
    cpe_a(5,2)=array(15,array(-0.63,30))
    cpe_a(5,3)=array(15,array(-1.12,45))
    cpe_a(5,4)=array(15,array(-1.57,60))
    cpe_a(5,5)=array(15,array(-1.53,75))
    cpe_a(5,6)=array(15,array(-1.51,90))
    cpe_a(5,7)=array(15,array(-1.97,105))
    cpe_a(5,8)=array(15,array(-2.44,120))
    cpe_a(5,9)=array(15,array(-2.60,135))
    cpe_a(5,10)=array(15,array(-2.75,150))
    cpe_a(5,11)=array(15,array(-1.92,165))
    cpe_a(5,12)=array(15,array(-1.10,180))

    cpe_a(6,0)=array(5,array(-1.39,0))
    cpe_a(6,1)=array(5,array(-1.57,15))
    cpe_a(6,2)=array(5,array(-1.75,30))
    cpe_a(6,3)=array(5,array(-1.90,45))
    cpe_a(6,4)=array(5,array(-2.05,60))
    cpe_a(6,5)=array(5,array(-1.85,75))
    cpe_a(6,6)=array(5,array(-1.65,90))
    cpe_a(6,7)=array(5,array(-1.87,105))
    cpe_a(6,8)=array(5,array(-2.10,120))
    cpe_a(6,9)=array(5,array(-2.17,135))
    cpe_a(6,10)=array(5,array(-2.24,150))
    cpe_a(6,11)=array(5,array(-1.85,165))
    cpe_a(6,12)=array(5,array(-1.47,180))

    cpe_a(7,0)=array(0,array(-1.43,0))
    cpe_a(7,1)=array(0,array(-1.56,15))
    cpe_a(7,2)=array(0,array(-1.70,30))
    cpe_a(7,3)=array(0,array(-1.85,45))
    cpe_a(7,4)=array(0,array(-2.00,60))
    cpe_a(7,5)=array(0,array(-1.73,75))
    cpe_a(7,6)=array(0,array(-1.47,90))
    cpe_a(7,7)=array(0,array(-1.73,105))
    cpe_a(7,8)=array(0,array(-2.00,120))
    cpe_a(7,9)=array(0,array(-1.85,135))
    cpe_a(7,10)=array(0,array(-1.70,150))
    cpe_a(7,11)=array(0,array(-1.56,165))
    cpe_a(7,12)=array(0,array(-1.43,180))

End Sub

Sub make_cpe_b (ByRef cpe_b())

    cpe_b(0,0)=array(90,array(1,0))
    cpe_b(0,1)=array(90,array(1.07,15))
    cpe_b(0,2)=array(90,array(1.10,30))
    cpe_b(0,3)=array(90,array(1.12,45))
    cpe_b(0,4)=array(90,array(0.54,60))
    cpe_b(0,5)=array(90,array(-0.73,75))
    cpe_b(0,6)=array(90,array(-0.80,90))
    cpe_b(0,7)=array(90,array(-0.73,105))
    cpe_b(0,8)=array(90,array(-0.63,120))
    cpe_b(0,9)=array(90,array(-0.50,135))
    cpe_b(0,10)=array(90,array(-0.34,150))
    cpe_b(0,11)=array(90,array(-0.30,165))
    cpe_b(0,12)=array(90,array(-0.24,180))

    cpe_b(1,0)=array(75,array(0.81,0))
    cpe_b(1,1)=array(75,array(0.82,15))
    cpe_b(1,2)=array(75,array(0.83,30))
    cpe_b(1,3)=array(75,array(0.69,45))
    cpe_b(1,4)=array(75,array(0.55,60))
    cpe_b(1,5)=array(75,array(-0.73,75))
    cpe_b(1,6)=array(75,array(-0.80,90))
    cpe_b(1,7)=array(75,array(-0.73,105))
    cpe_b(1,8)=array(75,array(-0.63,120))
    cpe_b(1,9)=array(75,array(-0.50,135))
    cpe_b(1,10)=array(75,array(-0.34,150))
    cpe_b(1,11)=array(75,array(-0.30,165))
    cpe_b(1,12)=array(75,array(-0.24,180))

    cpe_b(2,0)=array(60,array(0.57,0))
    cpe_b(2,1)=array(60,array(0.63,15))
    cpe_b(2,2)=array(60,array(0.79,30))
    cpe_b(2,3)=array(60,array(0.68,45))
    cpe_b(2,4)=array(60,array(0.47,60))
    cpe_b(2,5)=array(60,array(-0.73,75))
    cpe_b(2,6)=array(60,array(-0.80,90))
```

```
    cpe_b(2,7)=array(60,array(-0.73,105))
    cpe_b(2,8)=array(60,array(-0.63,120))
    cpe_b(2,9)=array(60,array(-0.50,135))
    cpe_b(2,10)=array(60,array(-0.34,150))
    cpe_b(2,11)=array(60,array(-0.30,165))
    cpe_b(2,12)=array(60,array(-0.24,180))

    cpe_b(3,0)=array(45,array(0.50,0))
    cpe_b(3,1)=array(45,array(0.53,15))
    cpe_b(3,2)=array(45,array(0.78,30))
    cpe_b(3,3)=array(45,array(0.57,45))
    cpe_b(3,4)=array(45,array(0.45,60))
    cpe_b(3,5)=array(45,array(-0.73,75))
    cpe_b(3,6)=array(45,array(-0.81,90))
    cpe_b(3,7)=array(45,array(-0.85,105))
    cpe_b(3,8)=array(45,array(-1.29,120))
    cpe_b(3,9)=array(45,array(-0.75,135))
    cpe_b(3,10)=array(45,array(-0.50,150))
    cpe_b(3,11)=array(45,array(-0.54,165))
    cpe_b(3,12)=array(45,array(-0.58,180))

    cpe_b(4,0)=array(30,array(-0.50,0))
    cpe_b(4,1)=array(30,array(-0.50,15))
    cpe_b(4,2)=array(30,array(-0.50,30))
    cpe_b(4,3)=array(30,array(-0.50,45))
    cpe_b(4,4)=array(30,array(-0.50,60))
    cpe_b(4,5)=array(30,array(-0.72,75))
    cpe_b(4,6)=array(30,array(-0.94,90))
    cpe_b(4,7)=array(30,array(-1.60,105))
    cpe_b(4,8)=array(30,array(-2.33,120))
    cpe_b(4,9)=array(30,array(-1.67,135))
    cpe_b(4,10)=array(30,array(-1.02,150))
    cpe_b(4,11)=array(30,array(-0.85,165))
    cpe_b(4,12)=array(30,array(-0.68,180))

    cpe_b(5,0)=array(15,array(-0.83,0))
    cpe_b(5,1)=array(15,array(-0.85,15))
    cpe_b(5,2)=array(15,array(-0.88,30))
    cpe_b(5,3)=array(15,array(-0.87,45))
    cpe_b(5,4)=array(15,array(-0.86,60))
    cpe_b(5,5)=array(15,array(-0.85,75))
    cpe_b(5,6)=array(15,array(-0.84,90))
    cpe_b(5,7)=array(15,array(-1.97,105))
    cpe_b(5,8)=array(15,array(-2.15,120))
    cpe_b(5,9)=array(15,array(-2.22,135))
    cpe_b(5,10)=array(15,array(-2.37,150))
    cpe_b(5,11)=array(15,array(-1.71,165))
    cpe_b(5,12)=array(15,array(-1.05,180))

    cpe_b(6,0)=array(5,array(-1.24,0))
    cpe_b(6,1)=array(5,array(-1.44,15))
    cpe_b(6,2)=array(5,array(-1.64,30))
    cpe_b(6,3)=array(5,array(-1.48,45))
    cpe_b(6,4)=array(5,array(-1.33,60))
    cpe_b(6,5)=array(5,array(-1.08,75))
    cpe_b(6,6)=array(5,array(-0.83,90))
    cpe_b(6,7)=array(5,array(-1.20,105))
    cpe_b(6,8)=array(5,array(-1.57,120))
    cpe_b(6,9)=array(5,array(-1.74,135))
    cpe_b(6,10)=array(5,array(-2.21,150))
    cpe_b(6,11)=array(5,array(-1.85,165))
    cpe_b(6,12)=array(5,array(-1.67,180))

    cpe_b(7,0)=array(0,array(-1.25,0))
    cpe_b(7,1)=array(0,array(-1.50,15))
    cpe_b(7,2)=array(0,array(-1.70,30))
    cpe_b(7,3)=array(0,array(-1.50,45))
    cpe_b(7,4)=array(0,array(-1.24,60))
    cpe_b(7,5)=array(0,array(-1.00,75))
    cpe_b(7,6)=array(0,array(-0.75,90))
    cpe_b(7,7)=array(0,array(-1.00,105))
    cpe_b(7,8)=array(0,array(-1.24,120))
    cpe_b(7,9)=array(0,array(-1.50,135))
    cpe_b(7,10)=array(0,array(-1.70,150))
    cpe_b(7,11)=array(0,array(-1.50,165))
    cpe_b(7,12)=array(0,array(-1.25,180))

End Sub

Sub make_cpe_c (ByRef cpe_c())

    cpe_c(0,0)=array(90,array(0.83,0))
    cpe_c(0,1)=array(90,array(0.68,15))
    cpe_c(0,2)=array(90,array(0.49,30))
    cpe_c(0,3)=array(90,array(0.34,45))
    cpe_c(0,4)=array(90,array(0.26,60))
    cpe_c(0,5)=array(90,array(0.23,75))
    cpe_c(0,6)=array(90,array(0.20,90))
    cpe_c(0,7)=array(90,array(-0.26,105))
    cpe_c(0,8)=array(90,array(-0.29,120))
    cpe_c(0,9)=array(90,array(-0.33,135))
    cpe_c(0,10)=array(90,array(-0.32,150))
    cpe_c(0,11)=array(90,array(-0.28,165))
    cpe_c(0,12)=array(90,array(-0.24,180))

    cpe_c(1,0)=array(75,array(0.81,0))
    cpe_c(1,1)=array(75,array(0.82,15))
    cpe_c(1,2)=array(75,array(0.83,30))
    cpe_c(1,3)=array(75,array(0.68,45))
    cpe_c(1,4)=array(75,array(0.55,60))
    cpe_c(1,5)=array(75,array(0.37,75))
    cpe_c(1,6)=array(75,array(0.20,90))
    cpe_c(1,7)=array(75,array(-0.26,105))
    cpe_c(1,8)=array(75,array(-0.29,120))
    cpe_c(1,9)=array(75,array(-0.33,135))
    cpe_c(1,10)=array(75,array(-0.32,150))
    cpe_c(1,11)=array(75,array(-0.28,165))
    cpe_c(1,12)=array(75,array(-0.24,180))

    cpe_c(2,0)=array(60,array(0.80,0))
    cpe_c(2,1)=array(60,array(0.70,15))
    cpe_c(2,2)=array(60,array(0.62,30))
    cpe_c(2,3)=array(60,array(0.50,45))
    cpe_c(2,4)=array(60,array(0.35,60))
    cpe_c(2,5)=array(60,array(0.27,75))
    cpe_c(2,6)=array(60,array(0.20,90))
    cpe_c(2,7)=array(60,array(-0.26,105))
    cpe_c(2,8)=array(60,array(-0.29,120))
    cpe_c(2,9)=array(60,array(-0.33,135))
    cpe_c(2,10)=array(60,array(-0.32,150))
    cpe_c(2,11)=array(60,array(-0.28,165))
    cpe_c(2,12)=array(60,array(-0.24,180))

    cpe_c(3,0)=array(45,array(0.60,0))
    cpe_c(3,1)=array(45,array(0.57,15))
    cpe_c(3,2)=array(45,array(0.55,30))
```

```
    cpe_c(3,3)=array(45,array(0.47,45))
    cpe_c(3,4)=array(45,array(0.30,60))
    cpe_c(3,5)=array(45,array(-0.1,75))
    cpe_c(3,6)=array(45,array(-0.33,90))
    cpe_c(3,7)=array(45,array(-0.64,105))
    cpe_c(3,8)=array(45,array(-0.96,120))
    cpe_c(3,9)=array(45,array(-0.75,135))
    cpe_c(3,10)=array(45,array(-0.55,150))
    cpe_c(3,11)=array(45,array(-0.48,165))
    cpe_c(3,12)=array(45,array(-0.41,180))

    cpe_c(4,0)=array(30,array(-0.50,0))
    cpe_c(4,1)=array(30,array(-0.50,15))
    cpe_c(4,2)=array(30,array(-0.50,30))
    cpe_c(4,3)=array(30,array(-0.45,45))
    cpe_c(4,4)=array(30,array(-0.40,60))
    cpe_c(4,5)=array(30,array(-0.30,75))
    cpe_c(4,6)=array(30,array(-0.20,90))
    cpe_c(4,7)=array(30,array(-0.70,105))
    cpe_c(4,8)=array(30,array(-1.22,120))
    cpe_c(4,9)=array(30,array(-1.00,135))
    cpe_c(4,10)=array(30,array(-0.79,150))
    cpe_c(4,11)=array(30,array(-0.65,165))
    cpe_c(4,12)=array(30,array(-0.50,180))

    cpe_c(5,0)=array(15,array(-0.81,0))
    cpe_c(5,1)=array(15,array(-0.82,15))
    cpe_c(5,2)=array(15,array(-0.83,30))
    cpe_c(5,3)=array(15,array(-0.70,45))
    cpe_c(5,4)=array(15,array(-0.61,60))
    cpe_c(5,5)=array(15,array(-0.45,75))
    cpe_c(5,6)=array(15,array(-0.27,90))
    cpe_c(5,7)=array(15,array(-0.65,105))
    cpe_c(5,8)=array(15,array(-1.02,120))
    cpe_c(5,9)=array(15,array(-1.01,135))
    cpe_c(5,10)=array(15,array(-1.00,150))
    cpe_c(5,11)=array(15,array(-1.96,165))
    cpe_c(5,12)=array(15,array(-0.92,180))

    cpe_c(6,0)=array(5,array(-1.19,0))
    cpe_c(6,1)=array(5,array(-1.14,15))
    cpe_c(6,2)=array(5,array(-1.09,30))
    cpe_c(6,3)=array(5,array(-0.90,45))
    cpe_c(6,4)=array(5,array(-0.71,60))
    cpe_c(6,5)=array(5,array(-0.50,75))
    cpe_c(6,6)=array(5,array(-0.25,90))
    cpe_c(6,7)=array(5,array(-1.55,105))
    cpe_c(6,8)=array(5,array(-0.77,120))
    cpe_c(6,9)=array(5,array(-0.92,135))
    cpe_c(6,10)=array(5,array(-1.04,150))
    cpe_c(6,11)=array(5,array(-1.08,165))
    cpe_c(6,12)=array(5,array(-1.12,180))

    cpe_c(7,0)=array(0,array(-1.15,0))
    cpe_c(7,1)=array(0,array(-1.09,15))
    cpe_c(7,2)=array(0,array(-1.03,30))
    cpe_c(7,3)=array(0,array(-0.84,45))
    cpe_c(7,4)=array(0,array(-0.64,60))
    cpe_c(7,5)=array(0,array(-0.44,75))
    cpe_c(7,6)=array(0,array(-0.24,90))
    cpe_c(7,7)=array(0,array(-0.44,105))
    cpe_c(7,8)=array(0,array(-0.64,120))
    cpe_c(7,9)=array(0,array(-0.84,135))
    cpe_c(7,10)=array(0,array(-1.03,150))
    cpe_c(7,11)=array(0,array(-1.09,165))
    cpe_c(7,12)=array(0,array(-1.15,180))

End Sub

Sub bounding_box(ByVal panels_body,ByVal source_vectors_wind(),ByRef a,ByRef b)

    Dim  arrBox, arrPoint, intCount

    arrBox = Rhino.BoundingBox(panels_body)

    '   intCount = 0
    '
    '       For Each arrPoint In arrBox
    '
    '       Rhino.AddTextDot CStr(intCount), arrPoint
    '
    '   intCount = intCount + 1

    '   Next

    Dim rear_length,side_length

    If  source_vectors_wind(0)(0)=1 Then
        rear_length=rhino.Distance(arrbox(0),arrbox(3))
        side_length=rhino.Distance(arrbox(0),arrbox(1))

        If side_length>rear_length Then
            a=array((arrbox(1)(0)-rear_length/5),arrbox(1)(1),arrbox(1)(2))
            b=array((arrbox(1)(0)-rear_length),arrbox(1)(1),arrbox(1)(2))

            '   Rhino.AddTextDot "a", a
            '   Rhino.AddTextDot "b", b
        Else
            a=array((arrbox(1)(0)-rear_length/5),arrbox(1)(1),arrbox(1)(2))
            '   Rhino.AddTextDot "a", a

        End If
    End If

    If source_vectors_wind(0)(1)=1 Then
        rear_length=rhino.Distance(arrbox(0),arrbox(1))
        side_length=rhino.Distance(arrbox(1),arrbox(2))

        If side_length>rear_length Then
            a=array(arrbox(2)(0),(arrbox(2)(1)-rear_length/5),arrbox(2)(2))
            b=array(arrbox(2)(0),(arrbox(2)(1)-rear_length),arrbox(2)(2))

            '   Rhino.AddTextDot "a", a
            '   Rhino.AddTextDot "b", b

        Else
            a=array((arrbox(2)(0)),arrbox(2)(1)-rear_length/5,arrbox(2)(2))
            '   Rhino.AddTextDot "a", a

        End If
    End If

End Sub

Sub dot_wind(ByVal panels,ByVal panels_norm()_
    _   ,ByRef panels_pressure(),ByVal source_vectors_wind()_
```

```
    _,ByVal cpe_a,ByVal cpe_b,ByVal cpe_c(),ByVal a_
    _   ,ByVal b,ByVal index,ByVal generation)

    Dim i,j
    Dim rgb_value
    Dim red,green,blue
    Dim incident_angle_beta,incident_angle_teta
    If generation=maxgens-1 Then
        Dim arrcolrs()
        ReDim arrColors( Rhino.MeshVertexCount(panels)-1 )
        Dim panels_pressure_norm
    End If

    Dim cpe_value
    Dim alfa,beta,teta

    Dim panels_norm_teta(2)
    Dim vertices
    Dim check1,check2

    vertices=rhino.MeshVertices(panels)

    For i = 0 To UBound(panels_norm)

        panels_norm_teta(0)=panels_norm(i)(0)
        panels_norm_teta(1)=panels_norm(i)(1)
        panels_norm_teta(2)=0

        For j = 0 To ubound(source_vectors_wind)

            incident_angle_teta =rhino.VectorDotProduct_
            _    (panels_norm_teta,source_vectors_wind(j))
            alfa=rhino.ACos(panels_norm(i)(2))*180/rhino.Pi
            teta=rhino.ACos(incident_angle_teta)*180/rhino.Pi

            If isempty(b)=False Then

                If vertices(i)(index)>=a(index) Then
                    get_cpe  panels_norm(i),cpe_a,alfa,teta,cpe_value
                Else
                    get_cpe  panels_norm(i),cpe_b,alfa,teta,cpe_value
                End If

            Else

                If vertices(i)(index)>=a(index) Then
                    get_cpe  panels_norm(i),cpe_a,alfa,teta,cpe_value
                End If

                If (vertices(i)(index)<a(index) And vertices(i)(index)>b(index)) Ther
                    get_cpe  panels_norm(i),cpe_b,alfa,teta,cpe_value
                End If

                If vertices(i)(index)<=b(index) Then
                    get_cpe  panels_norm(i),cpe_c,alfa,teta,cpe_value
                End If

            End If

            panels_pressure(i) =panels_pressure(i)+cpe_value*dynamic_pressure

        Next

        'average panel exposure angle
        'panels_pressure(i)=panels_pressure(i)/3

        If generation=maxgens-1 Then
            panels_pressure_norm=(panels_pressure(i)+2.75*dynamic_pressure)/_
                _(1.12*dynamic_pressure+2.75*dynamic_pressure)

            If panels_pressure(i)<-0.79*dynamic_pressure Then
                arrcolors(i)=rgb(0,0,0)
            Else
                red=255 * panels_pressure_norm
                green=255-255 * panels_pressure_norm
                blue=255 * panels_pressure_norm
                arrcolors(i)=rgb(red,green,blue)
            End If

        End If

    Next

    If generation=maxgens-1 Then
        Rhino.MeshVertexColors panels, arrcolors
    End If

End Sub

Sub get_cpe (ByVal panels_norm(),ByVal cpe(),ByVal alfa,ByVal teta,ByRef cpe_value)

    Dim i,j
    Dim dif()
    ReDim dif_alfa(7),dif_teta(12)
    Dim index_alfa,index_teta

    If panels_norm(2)>0 Then

        For i=0 To 7
            dif_alfa(i)=array(abs(alfa-cpe(i,0)(0)),i)
        Next

        sort_number dif_alfa
        index_alfa=dif_alfa(0)(1)

    Else
        index_alfa=0
    End If

    For j=0 To 12
        dif_teta(j)=array(abs(teta-cpe(index_alfa,j)(1)(1)),j)
    Next

    sort_number dif_teta
    index_teta=dif_teta(0)(1)

    cpe_value=cpe(index_alfa,index_teta)(1)(0)


End Sub

Sub get_mesh_faces_area (ByVal panels_body,ByVal panels_cap,_
    _   ByRef mesh_faces_area_body(),ByRef mesh_faces_area_cap())

    Dim i,j
    Dim arrfaces
```

```
    Dim arrface(3)
    Dim arrcenters
    Dim arrnormals
    Dim vsub1,vsub2,vsub3
    Dim vortho1,vortho2,vortho3
    Dim dot1,dot2,dot3

    arrfaces = Rhino.MeshFaces(panels_body, False)
    arrcenters = Rhino.MeshFaceCenters(panels_body)
    arrNormals = Rhino.MeshFaceNormals(panels_body)

    i=0
    j=-1

    While i <= UBound(arrFaces)

        arrFace(0) = arrFaces(i)

        arrFace(1) = arrFaces(i+1)

        arrFace(2) = arrFaces(i+2)

        arrFace(3) = arrFaces(i)

        'arrvertices(0)=array(0,1,2,2)
        'Rhino.AddMesh arrFace,arrvertices

        j=j+1
        i=i + 3

        vsub1=Rhino.VectorSubtract (arrface(0), arrcenters(j))
        vsub2=Rhino.VectorSubtract (arrface(1), arrcenters(j))
        vsub3=Rhino.VectorSubtract (arrface(2), arrcenters(j))

        vortho1=Rhino.VectorCrossProduct (vsub1,arrnormals(j))
        vortho2=Rhino.VectorCrossProduct (vsub2,arrnormals(j))
        vortho3=Rhino.VectorCrossProduct (vsub3,arrnormals(j))

        dot1=Rhino.VectorDotProduct (vsub1, vortho2)
        dot2=Rhino.VectorDotProduct (vsub2, vortho3)
        dot3=Rhino.VectorDotProduct (vsub3, vortho1)

        ReDim Preserve mesh_faces_area_body(j)

        mesh_faces_area_body(j)=(dot1+dot2+dot3)*0.5

    Wend

    Erase arrfaces
    Erase arrcenters
    Erase arrnormals

    arrfaces   = Rhino.MeshFaces(panels_cap, False)
    arrcenters = Rhino.MeshFaceCenters(panels_cap)
    arrNormals = Rhino.MeshFaceNormals(panels_cap)

    i=0
    j=-1

    While i <= UBound(arrFaces)

        arrFace(0) = arrFaces(i)

        arrFace(1) = arrFaces(i+1)

        arrFace(2) = arrFaces(i+2)

        arrFace(3) = arrFaces(i)

        'arrvertices(0)=array(0,1,2,2)
        'Rhino.AddMesh arrFace,arrvertices

        j=j+1
        i=i + 3

        vsub1=Rhino.VectorSubtract (arrface(0), arrcenters(j))
        vsub2=Rhino.VectorSubtract (arrface(1), arrcenters(j))
        vsub3=Rhino.VectorSubtract (arrface(2), arrcenters(j))

        vortho1=Rhino.VectorCrossProduct (vsub1,arrnormals(j))
        vortho2=Rhino.VectorCrossProduct (vsub2,arrnormals(j))
        vortho3=Rhino.VectorCrossProduct (vsub3,arrnormals(j))

        dot1=Rhino.VectorDotProduct (vsub1, vortho2)
        dot2=Rhino.VectorDotProduct (vsub2, vortho3)
        dot3=Rhino.VectorDotProduct (vsub3, vortho1)

        ReDim Preserve mesh_faces_area_cap(j)

        mesh_faces_area_cap(j)=(dot1+dot2+dot3)*0.5

    Wend
End Sub

Sub get_wind_fitness(ByVal panels_body,ByVal panels_cap,ByVal panels_pressure_body(),_
    _ByVal panels_pressure_cap(),ByRef individual_wind_fitness,ByVal generation)

    Dim i,j,k,l
    Dim max

    max=ubound(panels_pressure_body)+ubound(panels_pressure_cap)

    For i=0 To ubound(panels_pressure_body)
        If (abs(panels_pressure_body(i))>0.6*dynamic_pressure)  Then
            individual_wind_fitness=individual_wind_fitness+1
        End If
    Next

    For i=0 To ubound(panels_pressure_cap)
        If (abs(panels_pressure_body(i))>0.6*dynamic_pressure)  Then
            individual_wind_fitness=individual_wind_fitness+1
        End If
    Next

    individual_wind_fitness=individual_wind_fitness/max
    'individual_wind_fitness=1-individual_wind_fitness

    If generation>3 And generation<int(maxgens/2) Then
        rhino.DeleteObject panels_body
        rhino.DeleteObject panels_cap
    End If

End Sub

Sub get_wind_force (ByVal panels_body,ByVal panels_cap,ByVal panels_pressure_body(),_
    _   ByVal panels_pressure_cap(),ByVal panels_norm_body()_
```

```
    _    ,ByVal panels_norm_cap(),ByRef mesh_faces_area_body(),
    _    ByRef mesh_faces_area_cap(),ByVal num_u_body,ByVal num_v_cap)


    Dim i,j,k,l
    Dim arrvertices_body,arrvertices_cap
    Dim force_body(2),force_cap(2),force_wind_tot(2)
    arrVertices_body = Rhino.MeshVertices(panels_body)
    arrVertices_cap = Rhino.MeshVertices(panels_cap)

    k=0

    For i=0 To ubound(panels_norm_body)-num_u_body
        panels_norm_body(i)=rhino.VectorReverse(panels_norm_body(i))

        For j=0 To 2
            force_body(j)=force_body(j)+(panels_norm_body(i)(j)*panels_pressure_body(i
            _*(mesh_faces_area_body(l)+mesh_faces_area_body(l+1))
        Next
    Next

    l=0

    For i=0 To ubound(panels_norm_cap)-(num_v_cap+1)
        panels_norm_cap(i)=rhino.VectorReverse(panels_norm_cap(i))
        If ((i) Mod (num_v_cap+1))<>0 Then
            For j=0 To 2
                force_cap(j)=force_cap(j)+(panels_norm_cap(i)(j)*panels_pressure_cap(i
                _*(mesh_faces_area_cap(l)+mesh_faces_area_cap(l+1))
            Next
        End If
    Next

    For i=0 To 2
        force_wind_tot(i)=force_body(i)+force_cap(i)
    Next

    '   For i=0 To 2
    '       rhino.Print "force_wind_tot"&CStr(i)&"   "&CStr(force_wind_tot(i))
    '   Next

End Sub

Sub fitness_function (ByRef individual_fitness_
    _,ByVal individual_volume,ByVal individual_footprint_
    _,ByVal individual_vol_centroid,_
    _ ByRef individual_height,ByVal individual_facade_area,_
    _ByVal individual_min_curv_radius,ByVal individual_solar_gain,_
    _ ByVal individual_wind_fitness,ByVal Facade_to_Floors_Ratio,_
        _ByVal who,ByVal generation)

    Dim V_over_FA,V_over_FT
    Dim V_over_FA_norm,V_over_FT_norm
    Dim vol_norm
    Dim footprint_norm
    Dim facade_area_norm
    Dim centroid_height_norm,centroid_height
    Dim height_norm
    Dim max_curv_norm,max_curv
    Dim facade_to_floors_ratio_norm
    Dim tot

    Dim f1,f2,f3,f4,f5,f6,f7,f8
```

```
    'norm height_centroid
    centroid_height_norm=(individual_vol_centroid(2)-min_volume_centroid_height)/_
        _(max_volume_centroid_height-min_volume_centroid_height)
    'norm curvature
    max_curv=1/individual_min_curv_radius
    max_curv_norm=max_curv/max_curv_limit
    'norm FA_R
    facade_to_floors_ratio_norm=facade_to_floors_ratio/facade_to_floors_ratio_max
    'norm height
    height_norm=individual_height/max_height


    '-------------------------------------------------------------------


    V_over_FA=individual_volume(0)/individual_facade_area(0)
    V_over_FT=individual_volume(0)/individual_footprint(0)
    V_over_FA_norm=(V_over_FA-V_over_facade_Area_min)/_
          _    (V_over_Facade_Area_max-V_over_Facade_Area_min)_
    _    V_over_FT_norm=(V_over_FT-V_over_Footprint_area_min)_
    _    /(V_over_Footprint_area_max-V_over_Footprint_area_min)

    f1=(V_over_FA_norm)^0.5
    f2=(V_over_FT_norm)^0.5

    ''''''Height of Centroid '''''
    f3=centroid_height_norm^2
    ''''' max_curv ''''''
    If abs(max_curv_norm)<1 Then
        f4=(1-abs(max_curv_norm))^2
    Else
        f4=0
    End If

    ''''' solar_gain''''''
    f5=(individual_solar_gain)^2
    ''''' wind_fitness'''''''''
    f6=(1-individual_wind_fitness)^2

    '''' Facade_to_Floors_ratio
    If facade_to_floors_ratio>1 Then
        f7=0
    Else
        f7=(1-facade_to_floors_ratio_norm)^2
    End If

    ''' percentage fitness'''''
    tot=w1+w2+w3+w4+w5+w6+w7
    individual_fitness=((w1*f1+w2*f2+w3*f3+w4*f4+w5*f5+w6*f6+w7*f7)/tot)*100

End Sub

Sub sort_number(ByRef alfa())

    Dim swap
    Dim i, j
    Dim temp

    Do
        swap = False
        For i = 0 To UBound(alfa) - 1
            If (alfa(i + 1)(0) < alfa(i)(0)) Then
                temp = alfa(i)
```

```
                alfa(i) = alfa(i + 1)
                alfa(i + 1) = temp
                swap = True
            End If
        Next
    Loop While (swap)

End Sub

Sub get_height(ByRef cap,height)

    Dim surf_domainu
    Dim surf_domainv
    Dim points()
    ReDim points(0)
    Dim ustep
    Dim u
    Dim i

    surf_domainu=rhino.SurfaceDomain(cap,0)
    surf_domainv=rhino.SurfaceDomain(cap,1)

    ustep=(surf_domainu(1)-surf_domainu(0))/10
    i=0

    For u=surf_domainu(0) To surf_domainu(1)-ustep Step ustep
        points(i)=rhino.EvaluateSurface(cap,array(u,surf_domainv(1)/2))
        i=i+1
        ReDim Preserve points(ubound(points)+1)
    Next

    ReDim Preserve points(ubound(points)-1)
    'rhino.AddPointCloud points

    sort_points_z points

    height=points(ubound(points))(2)

End Sub

Sub sort_points_z(ByRef points())

    Dim swap
    Dim i, j
    Dim temp

    Do
        swap = False
        For i = 0 To UBound(points) - 1
            If (points(i + 1)(2) < points(i)(2)) Then
                temp = points(i)
                points(i) = points(i + 1)
                points(i + 1) = temp
                swap = True
            End If
        Next
    Loop While (swap)

End Sub

Function mean (ByRef  numbers())

    Dim i
    Dim sum
    Dim division

    For i=0 To ubound(numbers)

        If isempty(numbers(i))=False Then
            sum=sum+numbers(i)
            division=division+1
        End If
    Next


    mean=sum/division

End Function

Sub excel_output (ByVal sumfitness(),ByVal Vol_over_Facade_mean(),
_    ByVal vol_over_Footprint_mean(),_
_    _ByVal Facade_to_Floors_Ratio_mean(),ByVal HeightCentroid_mean(),
ByVal min_curv_radius_mean()_
__   ,ByVal solar_gain_mean(),ByVal wind_fitness_mean(),_
ByVal mutation_rate,ByVal words,ByVal scale_noise_factor )

    Dim Excel
    Dim ExcelSheet
    Dim ExcelWorkbook
    'Launch Excel
    Set Excel = createobject("Excel.Application")

    ' Create a new workbook and find the active sheet.
    Set ExcelWorkbook = Excel.Workbooks.Add
    Set ExcelSheet = Excel.ActiveSheet

    Dim i

    For i=1 To ubound(sumfitness)

        ExcelSheet.Cells(i, 15).Value ="generation fitness" & CStr(i)
        ExcelSheet.Cells(i, 15+1).Value = sumfitness(i)

        ExcelSheet.Cells(i, 15+2).Value ="Vol_over_Facade_mean" & CStr(i)
        ExcelSheet.Cells(i, 15+3).Value = Vol_over_Facade_mean(i)

        ExcelSheet.Cells(i, 15+4).Value ="vol_over_Footprint_mean" & CStr(i)
        ExcelSheet.Cells(i, 15+5).Value = vol_over_Footprint_mean(i)

        ExcelSheet.Cells(i, 15+6).Value ="HeightCentroid_mean" & CStr(i)
        ExcelSheet.Cells(i, 15+7).Value = HeightCentroid_mean(i)

        ExcelSheet.Cells(i, 15+8).Value ="min_curv_radius_mean" & CStr(i)
        ExcelSheet.Cells(i, 15+9).Value =min_curv_radius_mean(i)

        ExcelSheet.Cells(i, 15+10).Value ="solar_gain_mean" & CStr(i)
        ExcelSheet.Cells(i, 15+11).Value =solar_gain_mean(i)

        ExcelSheet.Cells(i, 15+12).Value = "wind_exposure_mean" & CStr(i)
        ExcelSheet.Cells(i, 15+13).Value =wind_fitness_mean(i)

        ExcelSheet.Cells(i, 15+14).Value ="Facade_to_Floors_Ratio_mean" & CStr(i)
        ExcelSheet.Cells(i, 15+15).Value =Facade_to_Floors_Ratio_mean(i)
    Next
```

```
    ExcelSheet.Cells(ubound(sumfitness)+1, 15).Value = "mutation_rate"
    ExcelSheet.Cells(ubound(sumfitness)+1, 15+1).Value = CStr(mutation_rate)
    ExcelSheet.Cells(ubound(sumfitness)+2, 15).Value = "words"
    ExcelSheet.Cells(ubound(sumfitness)+2, 15+1).Value = CStr(words)
    ExcelSheet.Cells(ubound(sumfitness)+3, 15).Value = "scale_noise_factor"
    ExcelSheet.Cells(ubound(sumfitness)+3, 15+1).Value = CStr(scale_noise_factor)
    ExcelSheet.Cells(ubound(sumfitness)+4, 15).Value = "w1 V_over_FA"
    ExcelSheet.Cells(ubound(sumfitness)+4, 15+1).Value = CStr(w1)
    ExcelSheet.Cells(ubound(sumfitness)+5, 15).Value = "w2 V_over_F"
    ExcelSheet.Cells(ubound(sumfitness)+5, 15+1).Value = CStr(w2)
    ExcelSheet.Cells(ubound(sumfitness)+6, 15).Value = "w3 Height_Centroid"
    ExcelSheet.Cells(ubound(sumfitness)+6, 15+1).Value = CStr(w3)
    ExcelSheet.Cells(ubound(sumfitness)+7, 15).Value = "w4 min_curv_radius"
    ExcelSheet.Cells(ubound(sumfitness)+7, 15+1).Value = CStr(w4)
    ExcelSheet.Cells(ubound(sumfitness)+8, 15).Value = "w5 solar_gain"
    ExcelSheet.Cells(ubound(sumfitness)+8, 15+1).Value = CStr(w5)
    ExcelSheet.Cells(ubound(sumfitness)+9, 15).Value = "w6 wind_exposure"
    ExcelSheet.Cells(ubound(sumfitness)+9, 15+1).Value = CStr(w6)
    ExcelSheet.Cells(ubound(sumfitness)+10, 15).Value = "w7 FA_R"
    ExcelSheet.Cells(ubound(sumfitness)+10, 15+1).Value = CStr(w7)



    ExcelWorkbook.SaveAs "C:\Documents and Settings\gennaro\Desktop\EmTech\dissertation\ga\results\_
    "_norm_fit_values alfa_radius\GA_info_"& "pop="&CStr(max_pop)&"_gen="&CStr(maxgens)&"_w1="&CStr(w1)_
    _    &"_w2="&CStr(w2)&"_w3="&CStr(w3)&"_w4="&CStr(w4)&"_w5="&CStr(w5)&"_w6="&CStr(w6)&"_w7="&CStr(w7)_
    _    &"_pp="&CStr(numberof_pointsper_section)&"_msf="&CStr(mutation_sens_factor)&"_wf="&CStr(wf)&".xls"
    Excel.Application.Quit

End Sub
```