Self-adaptation to High Density Based on Sunlight Analysis

H. Hua
School of Architecture, Southeast University, Nanjing, P. R. China
www.generativebox.com
e-mail: whitegreen@163.com

B. Li, PhD.
CAAD, ETH, Zurich, Switzerland

Prof. H. Zhang, PhD.
School of Architecture, Southeast University, Nanjing, P. R. China

Abstract

The modernism architects began to improve the housing conditions by sunlight analysis in 1920s. Nowadays, generative design employs various analyses as criteria for form-generation. This research investigates the optimal organisation of cubic houses in 3-dimensional array for high density. A generative tool is developed based on the integration of iterated algorithm and sunlight analysis.

Both brute-force strategy and advantage search algorithms such as genetic algorithm are not available for this optimisation task. A “make it better” scheme based on iterated operation is applied to solve this problem to some extent. It is interesting the algorithm gets into the 3-dimentional matrix deeply to observe the interactions between the units of the matrix. In fact, these interactions are based on the dynamical sunlight which produces shades in the matrix. The iterated processes add or eliminate the units according to sunlight analysis, leading to local transitions in 3-dimentional array.

The local behaviours shape the light environment in the matrix and the changing environment drives local transformations during the iterated processes. The accumulation of local transformations leads to the self-adaptation of the whole system, towards extremely high density in 3-dimensional context.

The system articulates complex form of high density. It is always unpredictable or unexpected, however, it innovates upon the design process. The generative tool helps the architects manipulate the complexity by studying local behaviours and setting up the transition rules of dynamical system.

Keywords: self-adaptation, iterated process, sunlight analysis, high density

1. Form of High Density

More sunlight and fresh air are brought into buildings to improve living conditions by Modernism architects since 1920s. The raw strategy is inevitably responsible for the "simple" style of their early works. The awesome "Radiant City" of Le Corbusier came from the requirements of abundance of sunlight, air recirculation, traffic efficiency as well as green spaces, upon which the individual freedoms depend. Furthermore, Walter Gropius applied sunlight analysis on residence planning and proved that "ordered" array of housing is more competent than complicated traditional blocks in sunshine estimation.

In fact, it is just an accident that the efficiency leads to simplicity. If "efficiency" problems in architecture domain are interpreted as optimisation tasks based on mathematical models, we will not get into the "function-form" issue, but develop variations of techniques then appreciate what the algorithms themselves will bring to us. In addition, this is more interesting considering the notion of "Adaptation Builds Complexity" by *John Holland* who developed new optimization methods now known as genetic algorithms in 1970s. His achievements suggest a sophisticated relation between optimisation process and form generation. And it is natural that unexpected and complex articulations emerge from the execution of the computer program, through which the nature of the problem may be exposed.

2. 3-Dinmensional Matrix

Employing unique search techniques, this design research takes an investigation to the organisation of high density housing, taking sunshine duration as single criterion. Other architectural factors such as structure and function are not put into the main body of the generative design, while they will be reconsidered in the phase of further design below. First, we set up a model of 3-dimensional array of cubic houses. Each unit gets a state of "O" or "V" which presents a unit space occupied by house and a void (unoccupied) unit space respectively. There must be a unique or series of solutions getting a highest density in the 3-dimensional array in which all units satisfy sunshine duration, however, the right way to catch the solutions is not always available due to the complexity of the subject as can be seen below.

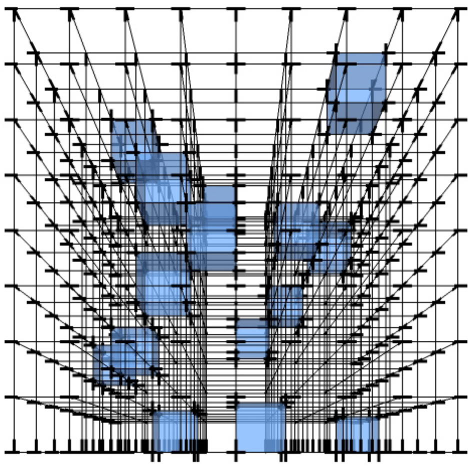

Figure 1. 3-dimensional array of cubic houses

Theoretically, the answer can be found by brute-force search strategy which checks the candidate layouts one by one. It seems that it is an *$O(2^n)$* problem as the number of units grows, however, it is actually more complex for the sunlight analysis in each iteration becomes more complex as the 3-dimentional matrix gets larger. Once the matrix comes to 4×4×4, the sum of the subjects reaches to 2^{64}, or 1.844×10^{19}, making the search task an impossible mission. Genetic algorithms are born to conquer this kind of problem. Whereas, the genetic representation of the problem based on 3-dimenional array and sunlight estimation is so difficult. If the candidates are generated without any constraints, then most candidates are illegal (not all the units of them satisfy sunshine duration criterion). On the other hand, the candidates could be tested in initialisation. In spite of that, most percentages of new subjects produced by the crossover operations are illegal and subsequently the whole scheme fails. At last, we come back to analyse the mechanism of 3-dimensional array in sunlight context and try to develop a valid way to find a high density solution, not the highest.

3. Strategy and Algorithms for High Density

3.1 Sunlight Analysis

This research commences with tracing the path of sun. By conventions in architecture, it is presumed that:

1. The path of sun (relative to the earth) is a circle (not helix or ellipse) during a particular day.
2. The plane of sun path is constant relative to the earth during a particular day.

They are not true, however, it doesn't matter because the difference between computed value and true value is limited. Taking the time as parameter, the path of sun could be described by a formula which is acquired by geometry techniques (see Figure. 2) on the presumptions.

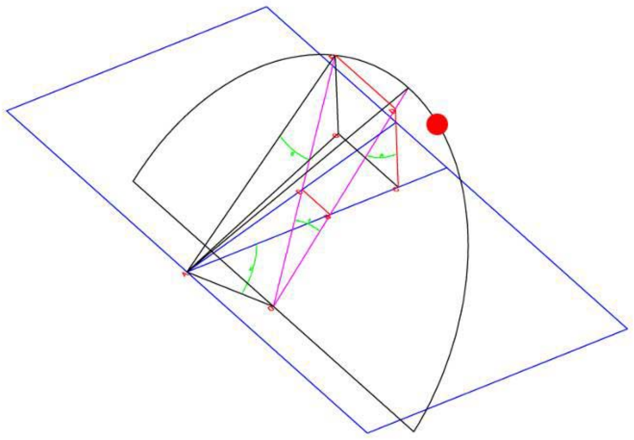
Figure 2. Geometry analysis of sun path

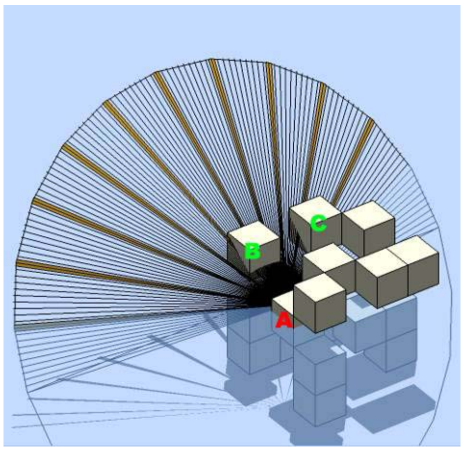

Figure 3. Shade analysis

It is clear that the sunshine is essential to houses. On the other hand, it is a real problem that how to evaluate the quality of sunshine to a particular room in a house. In areas where the low temperature is uncomfortable in winter, sunshine is important to warm the room. The architectural code of P. R. China makes a compromise between feasibility of the code and the accuracy of sunshine assessment. *Code of* *urban residential areas planning & design* orders that the sunshine duration of housing must be more than 2 hours on the day of "*Dahan*" (in big cities such as Nanjing). "*Dahan*" is around January 20th, defined as the coldest day in traditional lunar calendar established around 104 B.C. based on the climates and related farming activities. It is reasonable that "*Dahan*" is approximately a month later than "*Dongzhi*" on which the sunshine duration is shortest in daytime.

An algorithm to calculate the sunshine duration for any unit in the matrix is essential. A unit is likely to be shaded by several other units (see Figure 3). The algorithm searches all the units to check whether it produces shade on a target unit during the whole daytime. There is a set of shaded times because more than one units could project shades on a target. Superposing these data, the sunshine duration of the target comes out.

This method is not the most efficient way to calculate shades. In China, Commercial software has been developed for sunlight analysis whose aim task is checking whether the architectural layouts meet the code. Advance algorithms are employed in the *Sunshine 2.0* by CAAD lab of Tsinghua University which leaded by Wang Gu [4], a domestic pioneer who began to step into form-generation from analysis-oriented research. Actually, it is more powerful if advanced analysis software is embedded in the generative process, though this is not realized in our research yet.

3.2 Iterated Algorithm

The analysis does not only provide a way to calculate the finesses of subjects, but also reveals a clue for high density optimisation. It indicates that the amount of illegal units (whose sunshine duration under the criterion) in a 3-dimentional matrix is related to interactions between them. Here, the interactions are established by shades, or dynamical sunlight, which define a unit as shading one or shaded one. Available is the data describing a particular unit produces how much shades to others and who make how much shade to it. So, two special attributions are attached to each unit: "Shade" and Sunshine Duration. One's "Shade" is defined by the sum of all others' shading durations produced by it. Usually, there are both "killers" whose "Shade" is extreme high and "victims" whose Sunshine duration is very low in a matrix. So that's the key problem how to eliminate all killers and save as many victims as possible.

The "make it better" strategy is introduced to solve the complex problem. It aims to improve both the potential and current density of a subject gradually based on iterated operations. Generally, the algorithm acts like this in every iteration:

1. Find and delete (make it unoccupied) the one whose shade is greatest.
2. Delete an illegal one.
3. Create a new one (make it occupied) where sunshine duration is high enough.

The relations (shading and shaded) between the units in the matrix make up the dynamical environment which drives local transitions during the iterated processes described above. On the other hand, the local behaviour reshapes the environment built by units. These iterated processes outline the main idea of "make it better" scheme, though it causes too many bugs in execution. It is improved as below:

1. Get a set in which all elements:
   (1) Unoccupied (by house)
   (2) Legal (sunshine duration is higher than the criterion)
   (3) Shade of which are lower than a constant LIM1

   Then occupy the one whose Shade is lowest in the set.

2. Get a set in which all elements:
   (1) Occupied
   (2) Illegal

   Then eliminate (make it unoccupied) the one whose Shade is highest.

3. A copy of step 1

4. Get a set in which all elements:
   (1) Occupied
   (2) Shade of which are high than a constant LIM2

   Then eliminate the one whose shade is highest.

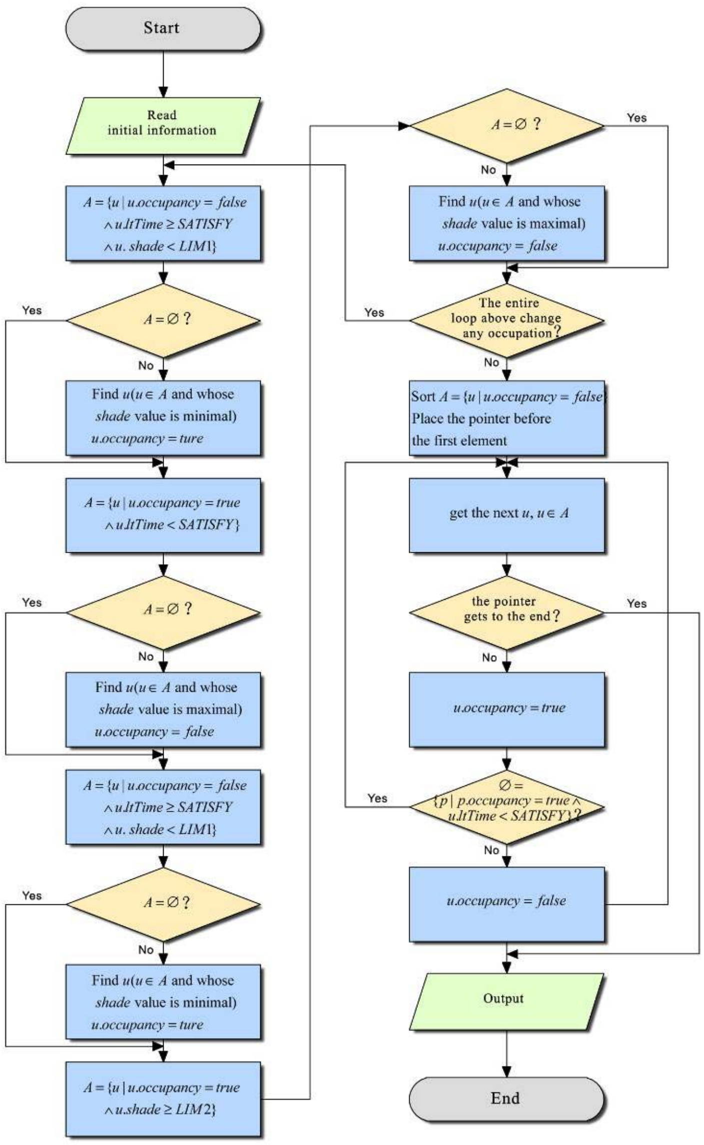
Start
Read initial information
A = {u | u.occupancy = false ∧ u.ltTime ≥ SATISFY ∧ u. shade < LIM1}
A = ∅ ?
Yes
No
Find u(u ∈ A and whose shade value is minimal) u.occupancy = ture
A = {u | u.occupancy = true ∧ u.ltTime < SATISFY}
A = ∅ ?
Yes
No
Find u(u ∈ A and whose shade value is maximal) u.occupancy = false
A = {u | u.occupancy = false ∧ u.ltTime ≥ SATISFY ∧ u. shade < LIM1}
A = ∅ ?
Yes
No
Find u(u ∈ A and whose shade value is minimal) u.occupancy = ture
A = {u | u.occupancy = true ∧ u.shade ≥ LIM2}
A = ∅ ?
Yes
No
Find u(u ∈ A and whose shade value is maximal) u.occupancy = false
The entire loop above change any occupation?
Yes
No
Sort A = {u | u.occupancy = false} Place the pointer before the first element
get the next u, u ∈ A
the pointer gets to the end?
Yes
No
u.occupancy = true
∅ = {p | p.occupancy = true ∧ u.ltTime < SATISFY}?
Yes
No
u.occupancy = false
Output
End

Figure 4. Flowchart

What is worth mentioning is that the algorithm consisting of the four steps is full of varieties both in framework and in details. Anyway, it avoids these situations:

1. The program gets its end without substantial improvement.
2. The program just involves a small number of units in the whole procedure.
3. The program is too difficult to get its end.

The program suggests this algorithm works well. It does make the matrix a more potential one and a higher density one during iterations. At least, it is capable of adding units to the initial array and making sure all the units are legal after execution. Paradoxically, the weakness of this algorithm is clear. First, it is very difficult to prove that this procedure will generate the highest one from randomly initialisation. In fact it has not this capacity according to our experiments. Then, the performance of the program is very sensitive to the constant of LIM1 and LIM2 which are very difficult to get optimal values. Additionally, the initialisation plays an important role, however, it is clueless which one is best considering the total number of initialisations grows at *$O(2^n)$* (n denotes the sum of units in the matrix).

Besides these disadvantages, we found there are some unoccupied units in certain solutions are legal and they will not “kill” others if occupied. In other words, the algorithm fails to explore the full potential of the matrix. The first step of the iterated algorithm above is responsible for the failure explicitly. The four steps are designed to make a compromise between the robust of the program and the high fitness of the final state, it is better to design another algorithm for additional improvement than modify the framework of the exiting algorithm. Applied to the solution of main process, the new algorithm checks the legal unoccupied units one by one to see if they will make others illegal if be occupied. The two algorithms make up the framework of our final program.

The program drives the matrix to high density successfully. While, the matrix can also be regarded as an automated machine which modifies its state iteratively according to inner interactions. Self-adaptation is achieved by its iterated actions which aim to improve the micro-environment made up by its own units.

3.3 Performance of the Program

A user interface is developed to allow designers to set the parameters in algorithms and observe the performance of the program. It also helps the programmer to get a graphic understanding of the algorithms and make further improvements. The 3-dimensional array is presented by plans (Figure 5) rather than 3-d view, for algorithms dealing with 3-d view are complex and use too much proportion of CPU. Figure 5 shows a 6×6×6 matrix with six plans (the left top one presents the first floor). Users can set the initial density of the array before running the search program. This parameter is necessary for the initialisation based on random functions and will bring distinguishing performances with different values. The experiments indicate that extreme low/high values are not appropriate. The two important constants of LIM1 and LIM2 can also be modified if the users plan to try alternative values and see what will happen.

In our experiment, running the program on a 4×4×4 matrix for 20 times emerges four high density ones which contains 24, 24, 26 and 25 units (Figure 6). It is not clear that if there is a pattern in the high density ones for this matrix is too small. If the array grows into some extent like 20×20×20, the pattern will be easily recognised if it exits. Unfortunately, the program written in Actionscript 2.0 loses its capacity of manipulating a matrix large than 7×7×7 on an ordinary personal computer. Suppose high efficiency languages like C or Java run 10 times faster than Actionscript, a 15×15×15 matrix is be likely be solved by this program, for the complex of this algorithm is *$O(n)$* (n denotes the sum of units in the matrix).

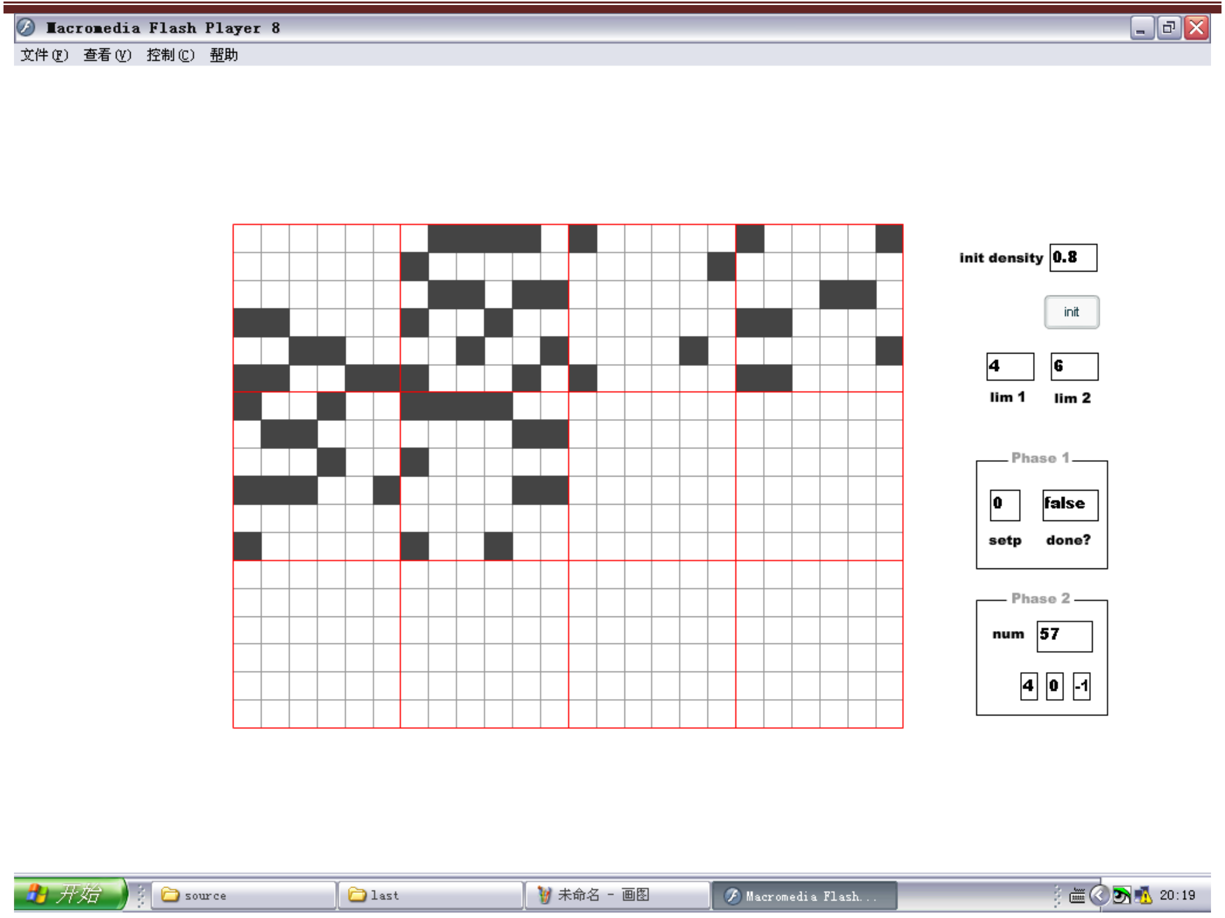

Figure 5. Interface of the program

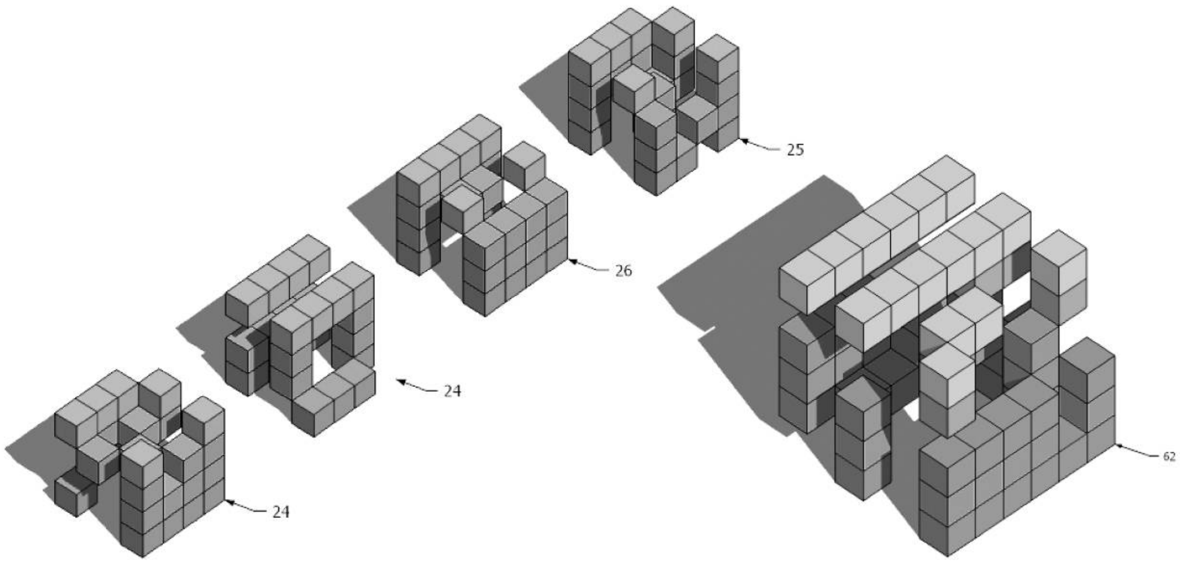

Figure 6. Solutions from 4×4×4 matrix *Figure 7. Solution from 6×6×6 matrix*

It is tried for a great number of times to get a high density one in a 6×6×6 matrix (Figure 7). The number of 42 seems to be the limit that allows 28.7% of the units occupied by houses. This solution is employed by our further design.

Further Design

The outline of layout is derived from the generative tool and it needs further design to become a true residential plan.

Initially, the dimension of the 3-dimentional array should be defined. In this project, the unit is set to be six metres in width, depth and height, and the whole matrix is as large as a cube of 36×36×36 metres. The height of six metres is inappropriate for small house, so most units are divided into two layers with three metres in height. And the whole unit could either be used by single family or provides two separate spaces for different inhabitants. Furthermore, adjacent units could be connected and make up a large space for big families or public facilities.

The transport inside of the matrix is anther substantial issue. Two cores with elevators and stairs are built into the matrix (Figure 8). The north one just occupies the exiting units and does not add new volumes. The south one adds several units but makes no shade to others. Aisles and bridges are also added to make horizontal connections.

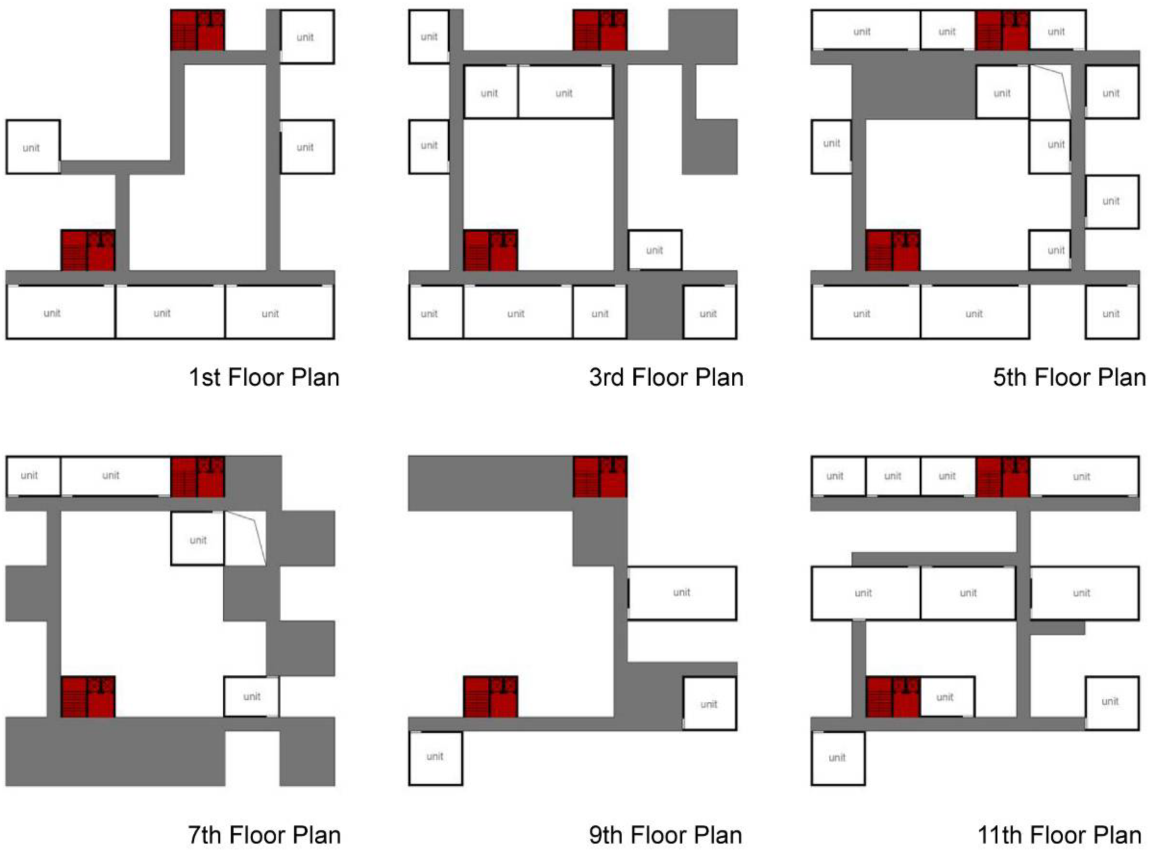

Figure 8. Plans

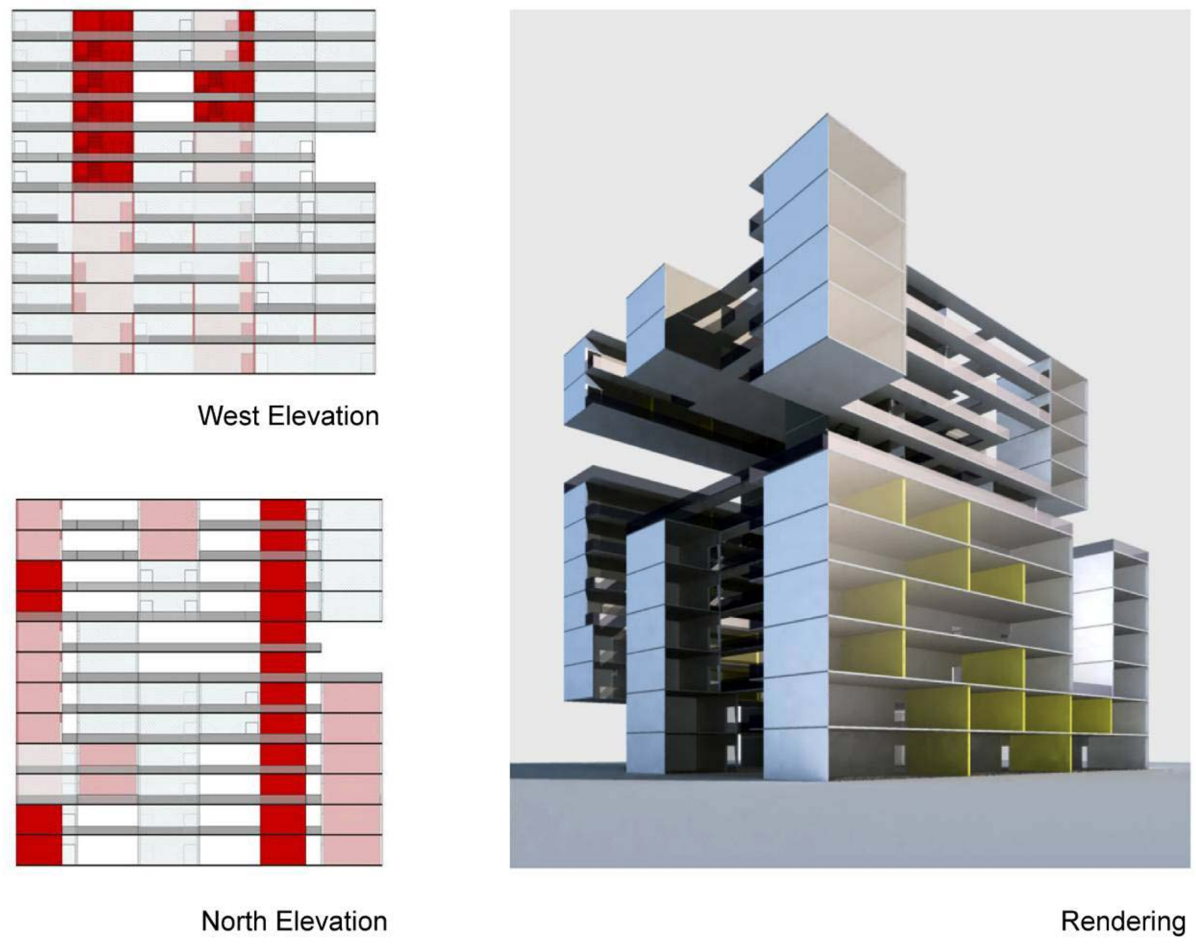

Figure 9. Elevations & rendering

The complex form is a precise solution for high density and sunlight criterion. In architectural viewpoint, the form makes each unit unique and build up various micro-environments. These features suggest that the optimisation of architecture may lead to innovative and diverse environments for inhabitants.

4. Conclusion

From the analysis of sunshine to architectural design, this research design tries to find an optimal form according to simple criterion. The core of the whole process is the automated iterated process which improves the matrix gradually. Though, the iterated function made up of four steps are not full-developed, it succeeds in generating complex and excellent prototypes for further design.

Analysis is always applied to make diagrams regarded as the sources of architectural design. As is shown above, design computation provides a new way how to employ analysis to drive design processes. In computer science, analysis on a special domain usually contributes to algorithms for fitness calculation but lose opportunities to develop innovative schemes dealing with the subjects' inner structures which determine its fitness. In this research, the integration of iterative process and sunlight analysis introduces a new method to improve the structure of the matrix for higher density and better sunshine quality. It seems that analysis processes are more powerful if it interacts with the volatile subject.

Great efforts were made for form-finding in architectural design and a variety of strategies with emphasis on different aspects have been developed since Modernism. Different from them, algorithm based form-generation is able to build strict connections between analysis and form. Simultaneously, these connections do not means simplicity, but are capable of creating unlimited structures and forms. Compared with the evolution mechanism of nature organisms, recent algorithms are too simply and stiff to generate complex systems with multiple hierarchies, however, they are stimulating the designers to step into systematic processes.

References

[1] Michael Hensel and Achim Menges, "*Differentiation and Performance Architectures and Modulated Environments*", *Techniques and Technologies in Morphogenetic Design*, Wiley-Academy, London, 2006

[2] Biao Li, *"A Generative Tool Base on Multi-Agent System: Algorithm of 'HighFAR' and Its Computer Programming", CAADRIA 2008*, Chiang Mai, 2008

[3] John Frazer, *"An Evolutionary Architecture",* John Frazer and the Architectural Association, London, 1995

[4] Gu Wang (王诂), *"建筑日照计算的新概念"*, 建筑学报, 北京, 2001-02

[5] Ilya Prigogine, *"From Being to Becoming: Time and Complexity in the Physical Sciences"*, W.H.Freeman & Co Ltd, San Francisco, 1981