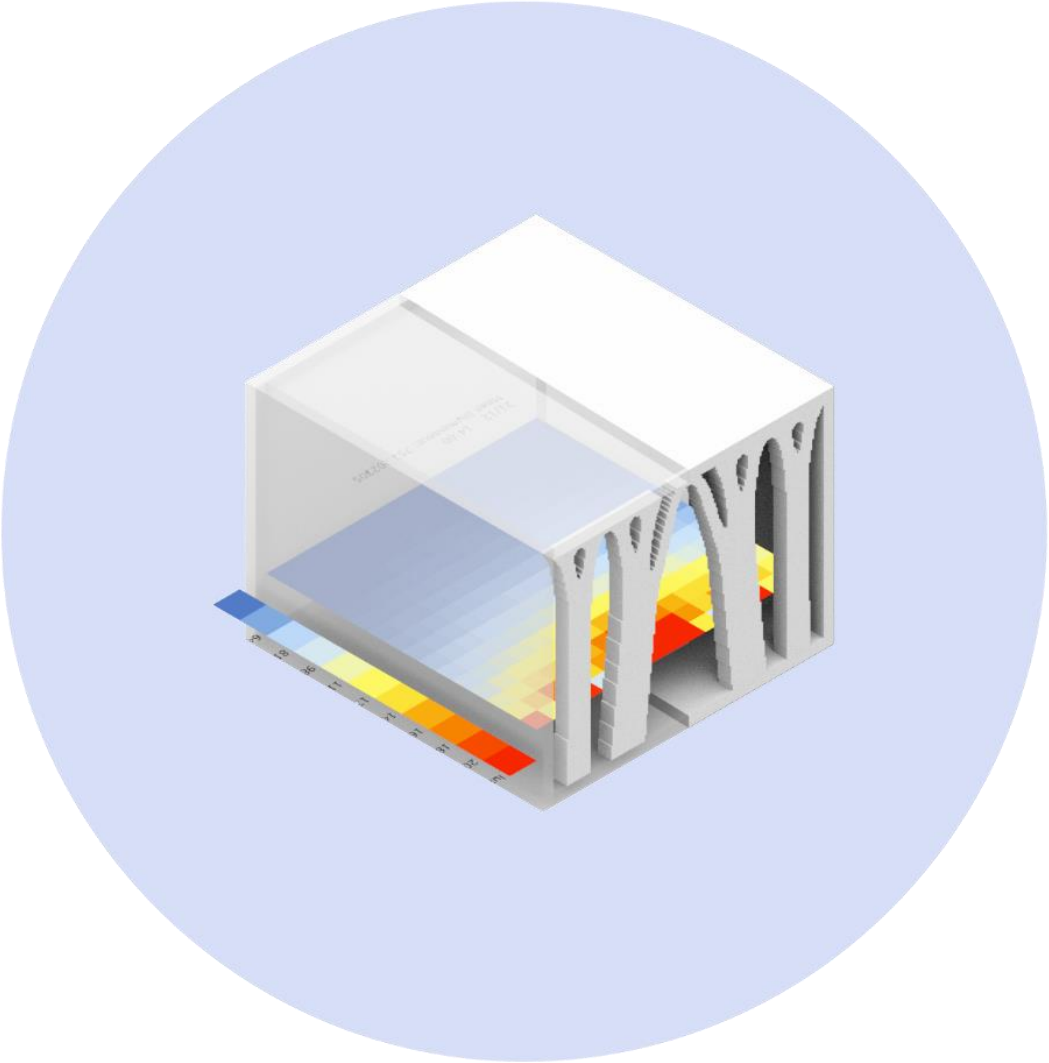


National Technical University of Athens

Master Thesis

in

M.Sc Computational Engineering in Solids School Of Chemical Engineers

Subject:

Combining Structural Topology Optimization with Daylight Analysis: A Designing Methodology development through programming procedures and parametric generative designing tools.

By

Dimitrios Gonidakis || 52117004

Supervisor:
Professor Nikolaos Lagaros

Athens 2020

||Acknowledgments||

Before the navigation over this thesis contents, I would like to thank some people whom their help have been valuable in order to complete this present thesis. First of all I would like to thank my supervisor Professor Nikolaos Lagaros because he believed in me and my potential. He entrusted me and his presence was constant, whenever I needed guidance. Also, I would like to thank Rector Andreas Boundouvis because as a headmaster of the computational engineering master program he supported me and my academic Architectural background, giving me motivation to continue my efforts . Additionally, a special thanks to Professor Benjamin Dillenburger who warmly accepted me uninvited and offer me his advises over my academic concerns.

Apart from the academic circle of professors, I would like to thank some people of my personal circle: my classmate Panagiotis Trifa, who his assistance during our studies in this master of science program was crucial to my “computational growth”, Professor Petros Karatsareas who kept me inspired to presume my academic dreams and Alexandros Theodoridis, Aliki Kopaneli, Korina Antoniadou and Ilias Sifakis who were the best pair of ears during this journey of writing my master thesis.

A warm thank you is the least I could express.

Dimitris Gonidakis

||Contents||

||Abbreviations||

AM: Additive Manufacturing

BESO: Bi-directional Evolutionary Structural Optimization

CBDM: Climate-Based Daylight modeling method

cDA: continuous Daylight Autonomy

DA: Daylight Autonomy

DC: Daylight Coefficient

DF: Daylight Factor

GH: GrassHopper

HB: HoneyBee

LSM: Level-Set Method

sDA: spatial Daylight Autonomy

SIMP: Solid Isotropic Material with Penalization

TO: Topology Optimization

UDI: Useful Daylight Illuminance

||List of figures||

||Prologue||
Pre-Thoughts

Since the beginning of this research for the formulation of the present thesis, have been posed a-priori some guidelines. These compose the basic pillars of the research and writing of this thesis. The key pillars are the following:

Writing Code_ It is crucial a thesis on a Master of Science program which is related with computational engineering to take advantage the potential of writing source code. In other words writing source code gives a deeper knowledge on a subject. However, computational software is used to boost and enforce the research. Also, there is no doubt that programming provides a more active relation with the researcher than the passive use of software.

Space Designing_ It was a major need to connect this thesis with architectural design terms. Both architectural design/concept and computational engineering could be underpinning each other. Structural topology optimization it is a powerful and versatile tool which could cooperate perfectly with designer's needs. Consequently, it is a fact that it would be the baseline of this thesis.

Keep it simple_ The motto "keep it simple" is essential over a sector (computational engineering), where from its nature is multidisciplinary and has unlimited possibilities. In other words, the limits over a master thesis should be solid, focused and not to deviate in order to solve each sub-case of every issue.

Future work_ The most important key pillar is the potential of evolving the present work. It has been obvious, since the beginning, that this thesis should be a baseline work for future research. Hence, this research would be a study over the subject that it would be selected.

Let's try to answer

Given the foresaid, a question has been raised: "How structural topology optimization, could be cooperate with space design parameters?" The question itself with the use of "how" indicates the way/s or better posed the methodology/ies, with which this question could be answered.

In order to response, it would be crucial to introduce and address the "space design parameters" term. One of the most important parameter in architectural design is window/void design. Natural light has concerned lots of architects since the existence of architecture. The psychology

and the comfort of the end-user of a space it is a basic parameter that most designers/architects take into consideration. Thus, natural light availability and end-user's visual comfort were selected to represent these parameters. Nevertheless, the need to "keep it simple", led the focus and evolution of the present research to the origins of these subjects, which is daylight availability. This parameter has been decided to be the one that will be connected with structural topology optimization.

To sum up, the present work studies ways which structural topology optimization could get connected with daylight analysis and propose a set of methodologies addressed to engineers and designers. The architectural background combined with Computational engineering offers a variation of tools that has been helpful in order to achieve this connection.

Research Tools

As it has been already foresaid the variety of academic and designing computational tools has been used to form the outcome of this thesis:

Literature

First of all the major tool has been the research over the literature and bibliography around structural topology optimization and daylight analysis. It is crucial a project to use (or reject) academic solid references, which helps to evolve the ratiocination of an academic project. Elsevier, ScienceDirect, Academia.Edu, the libraries of National Technical University of Athens and Technical Chamber of Greece were the major academic source of literature review. As it has been already mentioned a variety of designing computational tools has been used such as:

MATLAB

MATLAB is a powerful tool when it comes to programming linear algebra. Is quite fast and offers versatility to the programmer. In addition, the majority of academic issues around structural topology optimization are related with MATLAB. Hence, it has been the base of all the computational source codes that have been developed and appear over the appendices.

Python 3

Even though, MATLAB is capable to face basic issues of structural topology optimization and daylight analysis, it lacks of potential to get connected with any existing designing software. On the other hand, python is a program language that has a vast application over various sectors.

Lots of designing software is cooperating really good with python. Also, Python has a big library of tools that could save time from the designer/engineer.

Grasshopper

Grasshopper launched in 2007 (official grasshopper website) and is a standard plug-in of Rhino 6 designing software, which gives the opportunity to introduce parametric design in order to produce geometries. It combines attributes of designing and programming and gives the control to the end-user about the outcome. Grasshopper comes with a variety of plug-ins, which could make various analyses over different issues. Also, it can be connected with various program languages one of them is python.

Ladybug tools

Last but not least, there are the Ladybug tools (Ladybug, Honeybee). Ladybug tools are open-source python libraries, which are capable to connect various software which are related with environmental design. Also, they can be connected with grasshopper as a plug-in where could model sun and daylight analysis using the Radiance, Daysim and Energy-Plus software. Especially, honeybee can offer a fast way to produce a daylight analysis over a basic geometry.

Structure of the thesis

After tool's demonstration a thesis overview is following. Chapter 1 contains literature overview for structural topology optimization and daylight analysis. In Chapter 2, there is an extensive analysis of Bi-directional Evolutionary Structural Optimization structural topology optimization algorithm. Cases of gravitational load and distributed load are included. Afterwards, Daylight coefficients method is provided in Chapter 3, with code development provided in appendix. Over Chapter 4 there is a detailed description of daylight analysis on Grasshopper's Honeybee. Additionally, the comparison between GH design method with Daylight Coefficient method. In Chapter 5, a design framework and study case is provided. Last but not least conclusions of thesis analysis are provided in Chapter 6.

||Chapter 01||
Literature Review

Abstract

In this chapter it will be demonstrated a brief review over history and issues of topology optimization and daylight analysis. The scope of the introduction is to familiarize the reader about the definitions regarding of these two subjects, as well as the concept of this thesis. Topology Optimization research was focused on the three benchmark methods: Solid Isotropic Material with Penalization, Bi-direction Evolutionary Structural Optimization, Level-Set Methods, while Daylight Analysis narrowed at Sky Model evolution, Daylight coefficients and Climate Based Daylight Models. The research over the literature, about these sectors was useful on understanding their nature and taking decisions about the methods that would be used on this thesis.

Keywords: Topology optimization, timeline, daylight analysis, SIMP, BESO, Level-Set, Daylight Coefficients, Sky model, Climate Based Daylight Model.

1.1 Topology Optimization (TO)

1.1.1 Definitions and Timeline

There is no doubt that over the past decades topology optimization in on an ongoing evolution due to the benefits that can offer. Its application is being introduced more and more into industrial products and that has turn into a popular subject in academic community, so as to explore the full potential of topology optimization (T.O). This success is accompanied with two other developments the first is about industrial production and the second about material science. Furthermore, additive manufacturing has reached to a point that is accessible, cheaper and more accurate that used to be. So, it is more than capable to support the complexity of T.O designs and produce the accurately.

Moreover, the study and development of new composite materials, as well as the success to study their properties with precision, lead to apply designs to various industries such as to aeronautics. The term topology optimization is a way to resolve a design problem by finding the optimal density distribution of material in a fixed domain (Bendsoe and Sigmund, Topology Optimization: Theory Methods and Applications 2003). So , there is no doubt that there is an upward tendency is study and application of Topology Optimization, because of the idea of

producing something that has its maximum potential but with the minimum material/cost/energy consumption.

It is too difficult to catalogue and mention every book and paper referred to topology optimization till today. Nevertheless are some major publications that helped topology optimization's evolution and this is the reason are worth-mentioned (Rozvany, A critical review of established methods of structural topology optimization 2009). The first attempt, which is recorded, travels back to the beginnings of 20th century. Australian inventor Michell AGM (Michell 1904) published his study on the criteria/ stain constraints under a truss structure attains the minimum material to be constructed ("limit of economy of material"). His study was the limestone of the following studies about T.O. The basic problem was that the complexity of the subject was needing a computational power that became available and accessible, after a few decades. From the 70's, new types of computer offered an access to a vast numerical investigation on the subject of T.O, with their advanced computing capabilities and numerical simulation methods.

At the same exact timeline, the work of Michell's study was brought again on the spotlight, the main figure who evolved his work was Rozvany G. (1972) and his partners. Their work focused to Michell's theory to grillage (beam systems). Taking into consideration these applications, Prager and Rozvany (1977) developed the "optimal layout theory", which is considered the first general theory of T.O. The first applications were on grid-type structures and the calculation of their exact analytical solutions. The major problem, till then, was that the a topology optimization's problem could be formulated as a binary design, where the structure consists either solid material or void. Consequently, this formulation derived to "ill-posed" solutions, which could not provide admissible designs with refined geometrical details in the continuum framework. (Chwng and Olhoff 1981, Kohn and Strang 1986)

The solution came with the landmark paper of Bendsoe and Kikuchi (1988), who proposed the application of homogenization method of topology optimization consists of solving a class of shape optimization problem where the topology is made from an infinite number of micro scale voids which produces a porous structure, which lead to stable computational schemes (well-posed problem formulation). Since then numerical topology have been flourished and various methods have been starting to appear. Organizing them by chronological order of their appearance they are:

- density based methods
- evolutionary procedures
- bubble methods
- topological derivative
- level-set methods
- phase field method

Regarding on their update mechanism, all these approaches could be generally categorized into two basic groups, either density or boundary variation.

1.1.2 Benchmark Topology optimization methods

Even though, a variety of methods has appeared over the last thirty years, the most popular among them are still S.I.M.P, E.S.O/B.E.S.O and Level-Set method, which worth a further analysis.

1.1.2.1 Solid Isotropic Material with Penalization (SIMP)

SIMP is the oldest and one of the most implemented methodologies to crate structural topology optimization analysis. It is a density-based approach, where the distribution of material's density is connected with the material distribution scheme. By discretizing the design domain, a power-law interpolation function between solid/void is used in order to calculate the mechanical properties (like stiffness tensor). Consequently, this power-law eliminates unquestioningly intermediate density values by penalizing them. The outcome of this procedure is a structure with solid and void configuration. Although, this method ends with some major drawbacks, since its invention SIMP has been through lots of modification (modified interpolation scheme, RAMP etc.) in order to mitigate its inherited drawbacks. (Bendsoe and Sigmund 2003, Zhou and Rozvany 1991)

1.1.2.2 Bi-directional Evolutionary Structural Optimization (BESO)

BESO (Querin, Steven και Xie 1998) also referred in literature as ESO/BESO method, is the evolution of ESO method (Xie and Steven 1993). The optimization of the structure in this method is related with addition/removal of elements. These are the design variables, which BESO uses to "search" the stiffest design in a given design domain (Huang και Xie, Convergent and mesh-independent solutions for bi-directional evolutionary structural optimization method 2007)One of the major drawbacks of BESO used to be its unstable nature on the basic structural topology optimization problem. Sensitivity analysis helped to overcome it, though. A variety of designing

microstructures for materials to structural topology optimization problems have been successfully resolved with the application of BESO (Radman 2013)

1.1.2.3 Level-Set Method (LSM)

Level-Set Method (Osher and Sethian 1988) is the youngest among the benchmark methods in terms of academic interest, yet stills very popular. It uses the level set function concept in order to track the motion of the structural boundaries in combination with a speed function. This method is characterized as an evolutionary one. The main concept is the addition or removal of material in regions of high and low stress respectively. In addition, a threshold (percentages of the maximum initial stress) defines the rate of elimination or addition of material. The main difference of the other method is that is by its nature a non-mesh dependent method which makes popular in 3d printing projects, because it eliminates the post-processing procedure in order a 3d-optimized topology need to pass with the mess dependent methods.

1.2 Daylight Analysis

1.2.1 Definitions and Timeline

Daylight analysis counts approximately the same years as topology optimization. It started with the measurements of outdoor illumination around the 1895 (Walsh 1951). The need of calculating illumination is connected to the building performance and the user's comfort. Hence, after the invention of photometer, by the physicist Trotter, a variety of methods have been developed since then, in order to achieve a relatively accurate way to model the natural light. (Kota and Haberl 2009)

In the beginning graph type methods were developed. Among the most noticeable are Waldram diagrams (which still useful in bioclimatic design for small scales), Pliejel's pepper-pot diagrams, Building Research Establishment (BRE) daylight protractors and Graphic Daylight Design Method (GDDM) method (Fuller 1985). Afterwards, calculation methods were added to the empirical methods of daylight calculation. Gradually, with the computational tool advancement, daylight analysis became more and more accurate. Consequently, static (Daylight Factor, DF) and dynamic (Daylight Autonomy, DA) indicators develop, which evaluate the daylight performance of a space. As a result, software has been developed which could perform daylight analysis like RADIANCE (Reinhart and Herkel 2000) and DAYSIM (Ward and Rubinstein 1988).

1.2.2 Sky Models

Daylight Analysis calculations has been more accurate with the help of the models of sky, which is a combination of computation methods, which have been validated with numerical experiments. There is doubt that, the amount of light which enters inside an interior space is a result of its "openings" and the distribution of sky luminance. The life of sky models starts around 1921 by Kimball and Hand (1921), (1922), (H. Kimball 1923). They studied the sky luminance in Chicago and Washington by performing extensive measurements for three years. They ended by dividing their measurements into two general categories overcast and clear sky. After some years, G. I. Pokrovsky (1929), taking into consideration Rayleigh scattering, proposed a new formula to calculate the luminance distribution of a clear cloudless. In 1942 Moon and Spencer (1942) demonstrated an empirical formula to represent the luminance distribution under the average overcast sky.

In the beginning of 50s McDermott and Gordon-Smith (1951) proposed a method to calculate the luminance distribution of fully overcast sky. Later in 1955, the Moon-Spenser's model was adopted by The International Commission on Illumination (CIE 1955) as the standard for computing the overcast sky luminance distribution (Hopkinson, Petherbridge and Longmore 1966). While the same year, Kittler (1967) developed a method for luminance distribution calculation of the clear blue sky. CIE adopted (CIE 1973) Kittler's model, as standard for computational model for luminance distribution of the clear blue sky with sun. Although, the limestone of sky models published by Perez and his partners (Perez, Seals and Michalsky 1993), where all sky conditions were modeled from overcast to clear, through partly cloudy.

1.2.3 Daylight Calculation Models

As it has been foresaid, daylight analysis counts approximately 100 years of life. Among these years a variety of methods have been invented. Many of them have been translated into software. Nevertheless, are there two benchmark models, which affect the timeline and evolution of the daylight analysis methods. The Daylight Coefficient method by Tregenza and Waters (1983) and Climate-Based Daylight Modeling method by Mardaljevic (2000) are the two methods which affected the research directions of the present thesis.

1.2.3.1 Daylight Coefficient method (DC)

Daylight Factor and Lumen Method

Before proceed to the description of DC method, it is essential to explain the concept of Daylight Factor (DF) and Lumen Method (LM). Both are analytical expressions of daylight calculations and are the harbingers of the contemporary methods.

DF It was introduced by the English physicist Trotter and elaborated by Hopkinson. The definition of DF is the ration (%) between the illumination of an interior horizontal plane and the corresponding exterior illumination. Hopkinson, enriched this definition saying that: "it is possible to define the DF as a summation of three individual contributions, that take account the sky component (SC), the external reflected component (ERC) and the internal reflected component (IRC)." (Gherri 2015).

LM in another analytical expression of daylight analysis, which is based on empirical methods. H.G. Fruhling introduced LM, in 1928 but with some simplifications, which did not take account the ERC and IRC (Kota and Haberl 2009). Dresler . (1952) extended the LM by adding the reflected components

This is a simpler posing of the split flux* method.

DC Method: Overview

DC method is an analytical method which was based on numerical calculation of the three foresaid components of DF. According to its inspirators the DF is simply a weighted integral of the daylight coefficients (Tregenza and Wilson 2011). It is based on the inspired idea of sky dome division into equal 145 patches, with a conical aperture of 10º15', in order to cover the 68% of the celestial dome. Additionally, this subdivision assisted the calculation of the illumination of a point as a summation of the contribution of each sky's segment. Hence, the total illumination could be measured with precision as a linear super-imposition of the contribution of each sky segment filtered on a point, described by the relationship which will be explained extensively in the chapter 4:

$$E(x) = \sum_{n=1}^{N} DC(x)_n * L_n * \Delta S_n$$

One of the major advantages is the precision of illumination measurement under any sky condition. On the other hand one of DC's major disadvantage is its weakness to express simultaneously changes of the sky illumination levels because of the weather change. That is the reason that in literature it could find it as a Static Model. Last but not least, according to Reinhart DC is the baseline to the dynamic approaches research.

1.2.3.2 Climate-Based Daylight modeling method (CBDM)

Climate-Based Daylight Modelling (CBDM) was introduced by Mardaljevic in 2000. It is formed in order to "predicting" the continuous variations of quantity and quality of light radiation. By taking into consideration weather data sets of a region it can analyze with precision the illumination from the sky and the sun. It is based on the update of DC with a Standard Daylight Coefficient scheme (SDC), which includes sky diffuse contributions, diffuse ground contributions, solar contributions and indirect solar contributions. (Gherri 2015)

With the development of CBDM method new metrics were introduced in order to describe their impact inside of a space:

Useful Daylight Illuminance (UDI)

UDI is a parameter that indicates the attainment of levels of natural illumination considered useful. Additionally, UDI can be associated with the detention of the excessive levels of natural light associated with forms of visual discomfort, irritating glare and overheating. In other words, UDI is defined as the set of the illuminations that lie within the range 100 to 2500 lux in fact it has been established that daily illuminations that fall within 100-2500lux interval can generically be defined useful. (Nabil and Mardaljevic 2005)

Daylight Autonomy (DA)

DA is a measurement of the annual percentage frequency at which a pre-determined minimum level of illumination (usually 500 lux) can be maintained on the working plane thanks to only natural light. (Rogers 2006)

continuous Daylight Autonomy (cDA)

cDA assigns partial credits thereby introducing illumination thresholds lower than previous ones, useful to analyze specific visual tasks, which do not necessarily require the attainment the 500 lux foreseen traditionally by DA. (Gherri 2015)

spatial Daylight Autonomy (sDA)

sDA is the percentage of a work area or the plane of calculation on which fall 300 lux for at least 50% of the year, basing the calculation on a working day of 10 hours. (Heschong, et al. 2012)

1.3 Summary

According to the research over the literature, is obvious that the issue of TO is a very popular one. The basic difficulty since the beginning has been which one TO method is the “best” or the “most suitable” to cover thesis’ needs. There is no a solid answer to this issue. As a result, the main axis of selecting a topology optimization method and integrated with daylight analysis disciplines, and its potential for further research like the constructability (additive manufacturing procedures AM).

Given that SIMP is the most mature method, there is a vast investigation over it. Almost every issue that is new it multidisciplinary topology optimization, it has been approached with SIMP method. On the other hand BESO and LSM are relatively “younger” than SIMP, which gives space for more experimentation and learning through an academic procedure like this thesis. Both have given samples of excellent compatibility with additive manufacturing. However, BESO has been developed for structural designs and its solid/void design variable helps its association with daylight design. This association will be demonstrated over the chapter 5.

On the other hand daylight analysis also is a popular subject for engineers and designers. It is an issue with great complexity, requiring a good knowledge of fluid mechanics, as well as geometry. Dynamic models are more accurate and more related to reality, although they are extremely complicated. There is plethora of software which could make the calculations under the prism of a dynamic model. However, the scope of this thesis is to produce computational tools. Thus, the static model of DC is more accessible (yet not less complicated) and it could be the baseline to comprehend the computational procedure for the dynamic model too.

||Chapter 02||
Bi-Directional Evolutionary Structural Optimization in 2D

Abstract

In this Chapter an extensive analysis of Bi-Directional Evolutionary Structural Optimization (BESO) algorithm will be introduced. The research over this method was addressed to issues, which are associated with relatively realistic conditions. More specific, inputs such as real material indices (concrete) and loading cases (gravitational load, distributed load) were majorly studied. Hence, modification of the source code was made to comply these research. Its results were visualized and presented also in this chapter.

Keywords: BESO, concrete, loading cases, code modifications

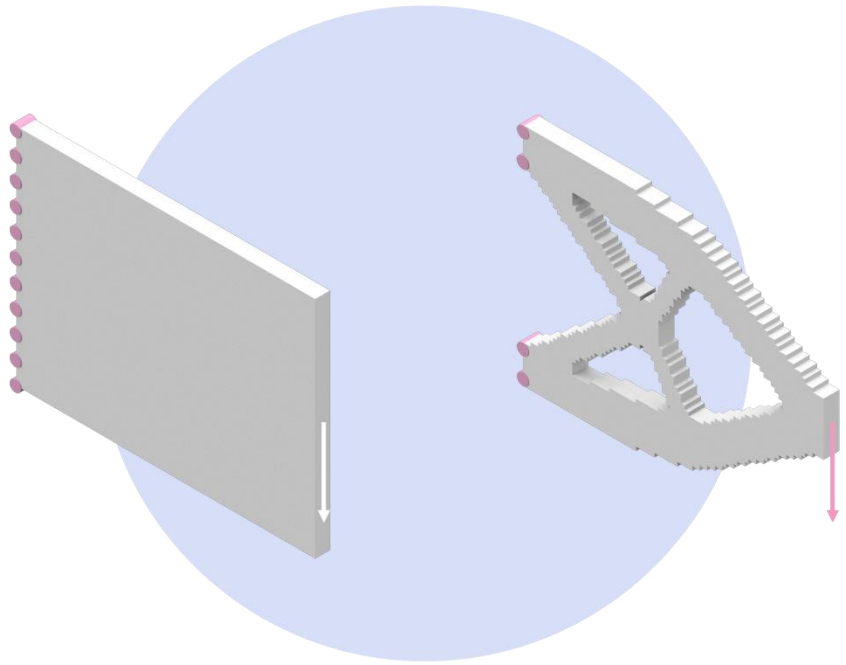


Figure 1
Cantilever Beam Benchmark example from soft-kill BESO provided by Huang and Xie. On the left is the design domain while on the right is the optimized one. (Inputs of optimization) mesh size: 80x60, Young's modulus: 27000 GPa, Poison ratio: 0.2 Load: -30KN, volume fraction: 0.45, evolution rate: 0.01(outputs of optimization) iterations: 91, C=0.003 Nm.

Bi-Directional Evolutionary Structural Optimization (BESO), by its present form, was developed (Huang and Xie 2007), to overcome the deficiencies of ESO method (solution existence, checkerboard, mesh dependency, local optimum etc.). The main concept of this method is that, at every

iteration, allows the simultaneous removal and addition of material. Two unrelated parameters, (RR) Rejection Ration and (IR) Inclusion Ratio, determine the amount of elements, which will be removed or added respectively. (Xia, et al. 2016)

The major goal in structural topology optimization is the search of the structure with the "best" stiffness matrix, over a design domain. As, it was mentioned, (hard-kill) BESO is based on the adding/removing element procedure, where the element is defined as a design variable. In this thesis, the last update (that is the generalize form of BESO) it will be used, which is called soft-kill BESO. Hence, the basic scheme of optimization problem with volume constraint is stated as:

$$\text{Min: } C = \frac{1}{2} f^T u \tag{2.01}$$

$$\text{Subject to: } V^* - \sum_{i=1}^{N} V_i x_i = 0 \tag{2.02}$$

$$x_i = x_{min} \text{ or } 1 \tag{2.03}$$

Where (2.1) express strain's energy compliance and V_i, V^* are the volume of an individual element and the prescribed volume of the design domain respectively. The binary design variable x_i is the relative density of the ith element, where x_{min} is a small value which defines the void element. Consequently, no element is completely removed. This is the major difference between the soft-kill and the hard-kill BESO, i.e. x_{min}= 0 in hard-kill while x_{min}=0,001 in soft-kill.

2.1 Material Interpolation Scheme

The material interpolation scheme with penalization was introduced in the SIMP method (Bendsøe 1989), (Bendsoe and Sigmund 2003), (Rietz 2001), (Zhou and Rozvany 1991). In order to achieve a nearly solid-void design, element density is a function interpolation of Young's modulus of the intermediate material:

$$E(x_i) = E_0 x_i^p \tag{2.04}$$

where E_0 it is the solid material Young;s modulus and p is the penalization exponent. By assuming the independence of Poisson's ration among the design variable, the global stiffness matrix K_G is expressed by the local (elemental) stiffness matrices and the variables x_i:

$$K_G = \sum_{i=1}^{n} x_i^p K_i^0 \tag{2.05}$$

where the K on the second part express the stiffness matrix of the solid element.

2.2 Sensitivity Analysis

By considering the mean compliance C (eq. 2.1) as the objective function and the design variable x_i continuously changes from x_{min} to 1, the sensitivity of the objective function in respect to the changing design variable is:

$$\frac{dC}{dx_i} = \frac{1}{2}\frac{df^T}{dx_i}u + \frac{1}{2}f^T\frac{du}{dx_i} \tag{2.06}$$

For the definition of the sensitivity of the displacement vector, adjoint method is going to be applied. Nevertheless, will be added an extra term with a Lagrangian multiplier to the objective function without affecting the equilibrium of a static structure:

$$C = \frac{1}{2}f^T u + \lambda^T(f - Ku) \tag{2.07}$$

Therefore, the sensitivity in respect to the change in the design variable can be formed as:

$$\frac{dC}{dx_i} = \frac{1}{2}\frac{df^T}{dx_i}u + \frac{1}{2}f^T\frac{du}{dx_i} + \frac{d\lambda^T}{dx_i}(f - Ku) + \lambda^T\left(\frac{df}{dx_i} - \frac{dK}{dx_i}u - K\frac{du}{dx_i}\right) \tag{2.08}$$

Due to the equilibrium

$$\frac{d\lambda^T}{dx_i}(f - Ku) = 0 \tag{2.09}$$

Because element's variation does not affect the applies load vector

$$\frac{df}{dx_i} = 0 \tag{2.10}$$

The modified objective function summarized into:

$$\frac{dC}{dx_i} = \left(\frac{1}{2}f^T - \lambda^T K\right)\frac{du}{dx_i} - \lambda^T\frac{dK}{dx_i}u \tag{2.11}$$

Given that (f-Ku) due to the static equilibrium the λ (Lagrangian multiplier) has no constraint so can be chosen freely. In order to eliminate the term of displacement derivative from (2.11), λ has to verify the below relation:

$$\frac{1}{2}f^{T} - \lambda^{T}K = 0 \tag{2.12}$$

Therefore, emerges that λ in order to satisfy (2.12) and the static equilibrium has to be:

$$\lambda = \frac{1}{2}u \tag{2.13}$$

Consequently, the eq (2.11) is formed as:

$$\frac{dC}{dx_i} = -\frac{1}{2}u^{T}\frac{dK}{dx_i}u \tag{2.14}$$

The last step is to introduce the interpolation scheme of (2.5) into the (2.14), as a result the basis for sensitivity number it is formed:

$$\frac{dC}{dx_i} = -\frac{1}{2}px_i^{p-1}u_i^{T}K_i^{0}u_i \tag{2.15}$$

Considering that in BESO, the design variables character is discrete, only two bound materials are allowed. So the sensitivity number follows the relation below:

$$\alpha_i = -\frac{1}{p}\frac{dC}{dx_i} = \begin{cases} \frac{1}{2}u_i^{T}K_i^{0}u_i\text{ , when } x_i = 1 \\ \frac{x_{min}^{p-1}}{2}u_i^{T}K_i^{0}u_i\text{, when } x_i = x_{min} \end{cases} \tag{2.16}$$

When the penalty exponent p tends to infinity, then (2.16) ends:

$$\alpha_i == \begin{cases} \frac{1}{2}u_i^{T}K_i^{0}u_i\text{ , when } x_i = 1 \\ 0\text{ , when } x_i = x_{min} \end{cases} \tag{2.17}$$

Which indicates that the soft-kill method is more generic than the hard-kill one.

2.3 Stabilizing the Evolutionary Process

Because of the nature of the sensitivity numbers, which are based on the discretization of design variables (solid/void), both objective function and topology were difficult to converge. Therefore, this is one the main problem in ESO/BESO methods was that large oscillations were observed in the evolution history of the objective function. Huang and Xie (2007) have found that averaging the sensitivity number with its historical information is an effective way to solve this problem. The simple averaging scheme is given as:

$$\alpha_i = \frac{\alpha_i^k + \alpha_i^{k-1}}{2} \tag{2.18}$$

where k is the current iteration number. Then let αk i = αi which will be used for the next iteration. Thus, the updated sensitivity number includes the whole history of the sensitivity information in the previous iterations

2.4 Element Removal/Addition and Convergence Criterion

Target volume for the next iteration (V_{k+1}) is essential to be defined before elements addition or removal from the design domain. Considering that the volume constraint (V*) could be smaller or greater than the initial volume in each iteration which defines the increment or removal over the target volume. Volume's evolution can be expressed as:

$$V_{k+1} = V_k(1 \pm ER), \qquad (k = 1,2,3 \dots) \tag{2.19}$$

ER is the evolutionary volume ratio, which becomes equal to zero once the volume constraint is satisfied.

In BESO, the sensitivity number of elements works as an indicator to be sorted (from the highest to the lowest). In order to achieve the desire volume of the final design in each iteration, there addition or removal of element/s according to the follow relations:

$$\alpha_i \leq \alpha_{del}^{th} \tag{2.20a}$$

$$\alpha_i > \alpha_{add}^{th} \tag{2.20b}$$

The second part refers to threshold sensitivity numbers. Depending on the volume needs of the iteration, BESO uses either 2.19a or 2.19b. So as to avoid extremely large quantities of elements

to be added or removed and the method to lose its integrity a volume addition ration (AR) must be defined.

$$\mathrm{AR} = \frac{\text{number of added elements}}{\text{total elements}} \tag{2.21}$$

If AR>AR_{max} then the sensitivity threshold must be calculated. AR_{max} is introduced to ensure that not too many elements are added in a single iteration normally is around 1%.

Last but not least, the converge criterion, which its compliance completes the evolution of the design, is satisfied when:

$$\mathrm{error} = \frac{\left|\sum_{i=1}^{N} C_{k-i+1} - \sum_{i=1}^{N} C_{k-N+1}\right|}{\sum_{i=1}^{N} C_{k-i+1}} \leq \tau \tag{2.22}$$

Where k is the current iteration number, τ convergence tolerance criterion and N is a selected integer number. Normally, N=5 which implies that the change in the mean compliance over the last 10 iterations is acceptably small.

2.5 BESO Algorithm and Flowchart

The steps of BESO method can organized into the following a step algorithm and a Flowchart. (Aremu, et al. 2010)

1| Discretize the design domain with finite element mesh and introduce the constraints and other design variables (ER, V* and AR)

2| Run the finite element analysis and calculate the sensitivity number of each element

3| Filtering the sensitivity numbers.

4| Set target Volume for the next iteration

5a| if the target volume is greater to the subjected one, check the removal threshold and remove the elements with sensitivity number smaller than the threshold.

6b| if the target volume is smaller to the subjected one, check the addition threshold and remove the elements with sensitivity number greater than the threshold.

7| Check the compliance convergence if the error is greater than the tolerance repeat the procedure from step 2

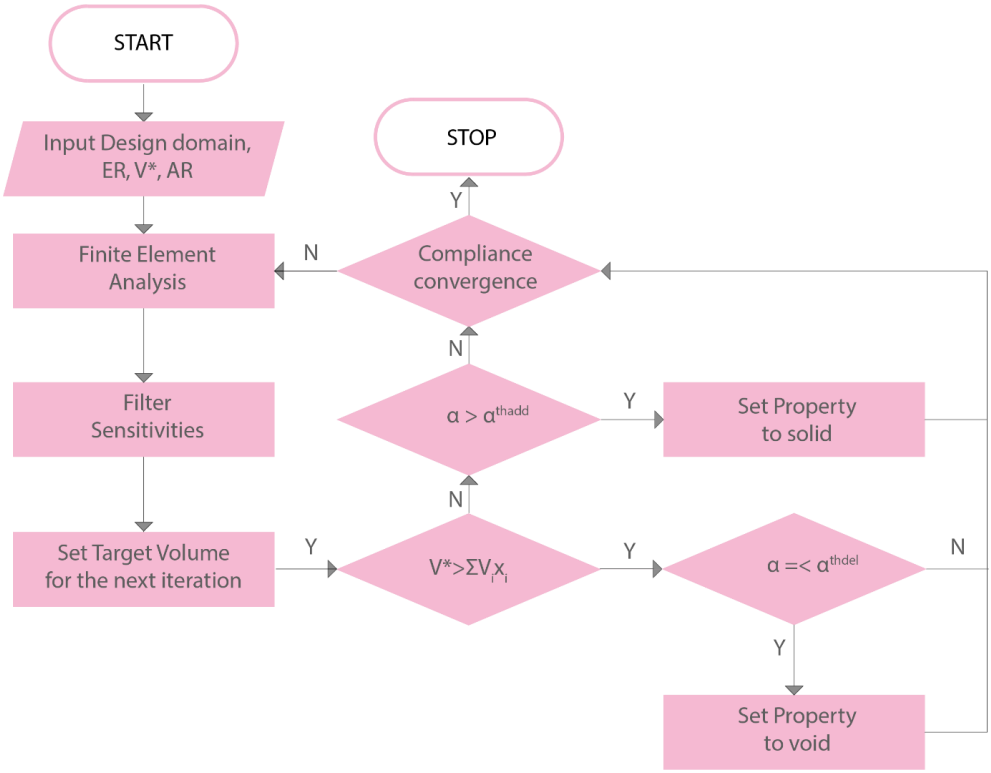


Figure 2
Flowchart of Bi-directional Structural Optimization Method

2.6 Design Variables

Before, proceed to the study case inputs it is crucial to present the main design variables, which will affect the outcome of optimized design product. These are the support of the structure and the loading conditions. Both are going to affect the structural topology optimization procedure and produce the desired outcome, which is expected to be close to some relatively realistic conditions.

2.6.1 Gravity Load Design variable

One of the main goals of this thesis is the design product to be self-support, so whole design process is based on self-weight structural analysis. Over the above sections, the function of BESO was analyzed extensively. In this section, some methods will be demonstrated, which are capable to integrate gravity-loads into both topology optimization and FE analysis methods.

2.6.1.1 Gravity load Discretization

The generalization of the static equation which is used to displacement model of finite elements analysis is:

$$K\vec{u} = \vec{F} \tag{2.23a}$$

F expresses the force vector and it can be divided into surface load vector and body load vector:

$$K\vec{u} = \overrightarrow{R_s} + \overrightarrow{R_b} \tag{2.23b}$$

Weight forces are regarded as body forces and affect the whole body and its structural behavior in the continuum. In terms of finites elements, when discretize weight-loads, they are proportionally distributed on each element's nodes. Therefore, each element contribution on the total body weight load is expressed by:

$$\vec{R}_b^e = \iiint_{V^e} N^T \rho \vec{g} dV = \rho \vec{g} \iiint_{V^e} N^T dV \tag{2.24}$$

Where N is the element's shape function and ρg are the constant values which describe material's density and gravitational acceleration. Given that, the elemental gravity force can be calculated, the sum of every sub-load it will calculate the total one:

$$\vec{R}_b = \sum_{i=1}^{n} \vec{R}_b^i \tag{2.25}$$

According to the above statements, there is direct impact between gravitational load and the element's nodes and their coordinates. BESO method uses Finite Element Method in order to reach to an optimum result. So, inherits the mesh dependency nature of the structural analysis that uses. Hence, the quality and structure of the mesh/grid are key ingredients to a successful outcome. When it comes to examples and applications in 2-D cases, a structured rectangular grid it is used. This grid consists of four-node linear quadrilateral elements, which are a special category of an iso-parametric four-node element. Consequently, the whole approach it will be the generic one.

2.6.1.2 Sensitivity analysis for Gravity-Load

Generally topology optimization examples and studies are dedicated to fixed loads. However, gravity load is a significantly important parameter in various engineering problems, especial in civil engineering. There are plenty of matters which emerge with the use of gravity load into the evolutionary method. Huang and Xie (2011) studied the case of gravity load into BESO method.

The elemental load of a four-node quad element, which is aligned with the global direction, can be obtained:

$$f_i = V_i \rho_i g \begin{bmatrix} 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \end{bmatrix}^T \tag{2.26}$$

g is the gravity acceleration. The elemental variation affects only the self-weight loads. Hence, the external forces variation of (2.10) is updated:

$$\frac{df}{dx_i} = V_i \rho^0 g f^{-T} u_i \tag{2.27}$$

465 Consequently, the variation of mean compliance of 2.15 is updated as well:

$$\frac{dC}{dx_i} = V_i \rho^0 g f^{-T} u_i - \frac{1+p}{2[1+p(1-x_i)]^2} u_i^T K_i^0 u_i \tag{2.28}$$

And sensitivity number of 2.16 ends with the below form

$$\alpha_i = -\frac{1}{p+1}\frac{dC}{dx_i} = \begin{cases} -\frac{V_i \rho^0 g}{p+1} f^{-T} u_i + \frac{1}{2} u_i^T K_i^0 u_i, & x_i = 1 \\ -\frac{V_i \rho^0 g}{p+1} f^{-T} u_i + \frac{1}{2[1+p(1-x_{min})]^2} u_i^T K_i^0 u_i, & x_i = x_{min} \end{cases} \tag{2.29}$$

Huang and Xie (2011) were experimenting with both soft kill and hard kill method with gravitational load implementation, where concluded that both work relatively the same so the sensitivity number can be translated as in the relation (2.17). Indeed by using the gravitational load with hard kill sensitivity scheme the arch problem were more stable and easy during the computational procedure. In Appendix A1 there is the addition on the soft-kill BESO by Huang and Xie with gravitational load.

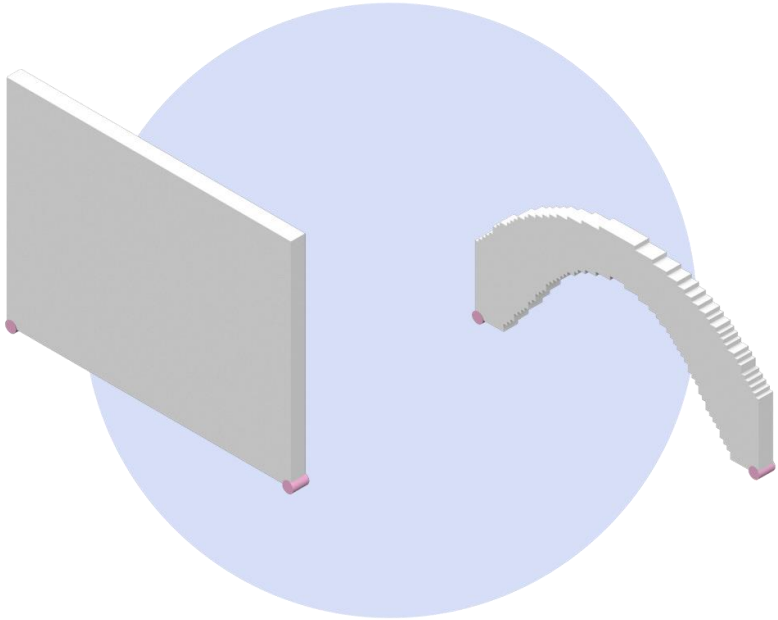


Figure 3
Application of BESO Algorithm using only gravitational load as a nodal design variable. Same material variables and BESO tuning has been used as the benchmark cantilever beam approach.

2.6.2 Distributed Load Design variable

The equal distributed load it could be translated as a nodal across the domain boundary like that:

$$f_i = N * F \left[0\ 0\ \ 0\ \ \frac{1}{n}\ \ 0\ \ \frac{1}{n}\ \ 0\ \ 0 \right]^T \tag{2.30}$$

Where I the boundary element, n the number of the boundary elements receiving the F force and N the dimension of the boundary. In Appendix A, there is the modified MATLAB function, which has been used in order to define the distributed load.

2.6.3 Structure's support

The support of the structure in a topology optimization case it is expressed in the form of constraints. In the 2d case, which is examined the nodal degrees of freedom are movements over the direction x and y respectively. In order to discretize a wall that is founded in the ground, the restriction of the movements on the discretized border which is related with the founded border of the structure is an accepted way to model it.

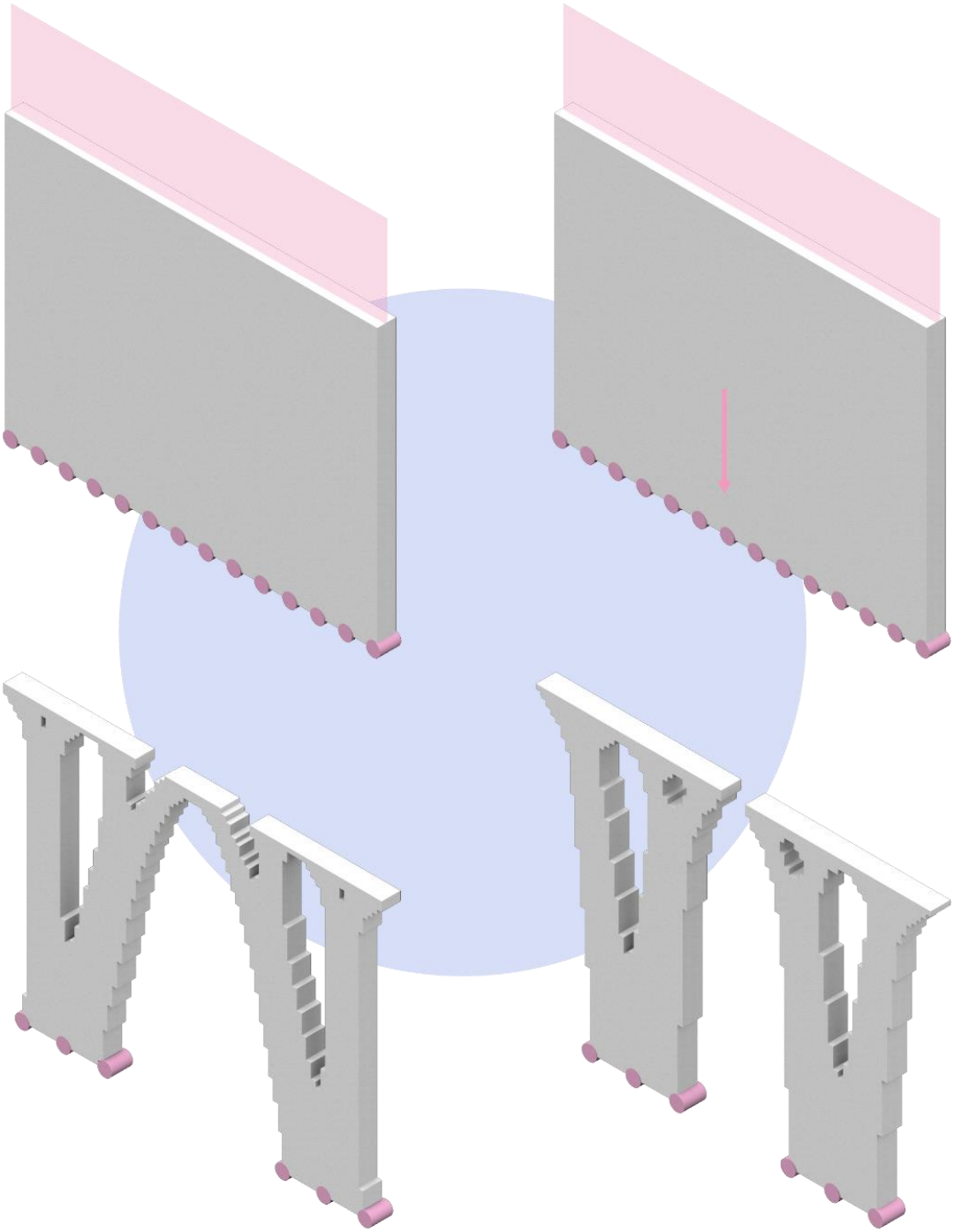


Figure 4
study cases

||Chapter 03||
Daylight Coefficients

Abstract

This chapter is based on the benchmark paper, which Tregenza and Waters published in 1983, entitled as “Daylight Coefficients”. The definition of the term, as well as the analysis about the mathematical model which express it, will be the main issue. Additionally, will be explained the transition from the natural integral model to discrete, in order to be computed through reiterative procedure. Last but not least a code was developed (appendix c). Scope of a source code development was not the highly accuracy of the results. It was obvious that there were plenty of commercial or open source programs that could offer a better result. However, by understanding the computational procedure of daylight coefficients was assisted to enrich the knowledge and research over the subject of daylight analysis.

Keywords: Daylight Coefficients, computational code

3.1 Definition

The quantity of daylight*, falling on a room’s surface is a result of two factors: sky’s luminance and the geometry and materials of its surrounding. These factors are independent, mostly for practical reasons. However, the polar character of the sky vault, as well as the state of clearness of the sky make its luminance varies independently from one angle to another. Needless to say the interior does not follow every variation of sky changes, but only those which can “watch” through its windows. Also, sometimes it is possible that the surroundings could affect more the illumination of the interior than the sky.

As it is foresaid, the illumination of a point in the interior of a room is a contribution of the sky, the exterior reflectance and the interior reflectance components. By introducing the concept of superimposition principle, it is possible to examine separately the contribution of these three components and then by adding them to calculate the total illumination of a room.

3.2 Sky component

The following relation demonstrates the contribution to the illuminance at a point from a small patch of sky. More material regarding sky division and patch definition where explained on the appendix B2 (P. Tregenza 1987):

$$\Delta E = D_{\theta\varphi} L_{\theta\varphi} \Delta S_{\theta\varphi} \tag{3.01}$$

where $L_{\theta\phi}$ and $S_{\theta\phi}$ are the luminance and the angular size of the sky element at azimuth ϕ and altitude θ respectively. $D_{\theta\phi}$ is a quantity which depends on room's geometry and its surfaces' reflectance, windows' transmittance and surrounding. This quantity is called Daylight Coefficient and is indicator of the sensitivity of internal illuminance to the changes of a sky element.

The total daylight illuminance at a point is obtained by the double integration of the (3.01):

$$E = \int_0^{2\pi} \int_0^{\pi/2} D_{\theta\varphi} L_{\theta\varphi} \cos\theta \, d\theta d\varphi \tag{3.02}$$

DC for a horizontal and vertical unobstructed surface can be calculated from 3.03a and 3.03b respectively:

$$D_{\theta\varphi} = \sin\theta \tag{3.03a}$$

$$D_{\theta\varphi} = \begin{cases} \cos\theta \cos(\varphi - \nu), 0 \leq \theta \leq \frac{\pi}{2} \text{ and } -\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2} \\ 0 \end{cases} \tag{3.03β}$$

3.3 Single point Daylight Coefficient

Let's consider a typical rectangular room with a window on one of its walls and a point at the center of its volume. The DC of this point is a result of direct and inter-reflected light. For direct light:

$$D_{\theta\varphi_o} = \sin\theta \, T_\omega \tag{3.04}$$

3.04 is the DC from the sky and external reflected for a sky patch with θ, ϕ altitude and azimuth, where T_ω is the glass transmittance at incident angle ω.

For inter-reflected light, the concept of split-flux principle is used, which allows to examine separately:

- The ground reflected light
- The upper surfaces of the room reflected light
- The light which is enter from the window and is reflected by the lower surfaces of the room

$$E_{gsky} = \frac{L \sin\theta \, R_g}{\pi} \tag{3.05a}$$

$$F_{wind-g} = \frac{L \sin\theta \, R_g W}{2} \tag{3.05b}$$

The first relation expresses the luminance due to sky element of unit solid angle, where L is the luminance of the sky element and R_g is the mean ground reflectance. The second one is the flux, which ground will receive from the window, where W is the window area. By the same way they can be calculated the fluxes received and reflected from upper room surfaces:

$$L * D_{up\_received} = \frac{L \sin\theta \, R_g WT}{2} \tag{3.06a}$$

$$L * D_{up-reflected} = \frac{L \sin\theta \, R_g R_{cw} WT}{2} \tag{3.06b}$$

Where T is the mean glass transmittance and R_{cw} is the reflectance of upper walls and ceiling. So, the inter-reflected mean luminance of the interior due to ground is:

$$E_{mean\_g} = \frac{L \sin\theta \, R_g R_{cw} WT}{2A(1-R)} \tag{3.07}$$

Where A and R are area and mean reflectivity of the room respectively. Thus the DC from the ground reflection is:

$$D_{\theta\varphi_g} = \frac{\sin\theta \, R_g R_{cw} WT}{2A(1-R)} \tag{3.08}$$

Following a similar derivation, the DC of the inter-reflected light which enters in the room through the window and is reflected by the lower parts of the room is:

$$D_{\theta\varphi_{low}} = \frac{\cos\theta \, R_{fw} WT_{\omega}}{2A(1-R)} \tag{3.08}$$

$$\cos\omega = \cos\theta \cos(\varphi - \nu)$$

By adding these three daylight coefficients, point DC could be calculated:

$$D_{\theta\varphi} = D_{\theta\varphi_0} + D_{\theta\varphi_g} + D_{\theta\varphi_{low}} \tag{3.09}$$

3.4 Finite Element implementation

The foresaid analysis is addressed to calculate the illuminance of on point inside the room. Nevertheless, the implementation of finite elements can discretize the procedure and make it computationally friendly. The room now is being divided into m finite elements and the sky into n finite patches of equal size respectively. With the assumption that the interior distances are really small compared with the exterior, the illuminance at the center of each element can be calculated. Given the relation from (3.02) the illuminance of m point is the double integral which could be expressed as the summation of:

$$E_m = \sum_{i=1}^{n} D_i S_i L_i \tag{3.10}$$

In terms of linear algebra D is a vector, which express the daylight coefficients for the m point. By calculating (3.10) for every point, the total illuminance of the room can be calculated.

$$E_j = \sum_{j=1}^{m} \sum_{i=1}^{n} D_i S_i L_i \tag{3.11}$$

The equation (3.11) easily can be expressed in matrix form as:

$$\begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_m \end{bmatrix} = \begin{bmatrix} D_{11} & D_{12} & \dots & D_{1n} \\ D_{21} & D_{22} & \cdots & D_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ D_{m1} & D_{m2} & \dots & D_{mn} \end{bmatrix} \begin{bmatrix} S_1 L_1 \\ S_2 L_2 \\ \vdots \\ S_n L_n \end{bmatrix} \tag{3.12}$$

So, the illuminance it can be expressed as a vector [E] of the product of the daylight coefficient matrix [D] with sky's patches luminance and solid angle vector [S*L]. The matrix [D] it is a product of three sub-matrices which are related with the three foresaid inter-reflection categories. More specific:

$$[D] = [R_3][R_2][R_1] \tag{3.13}$$

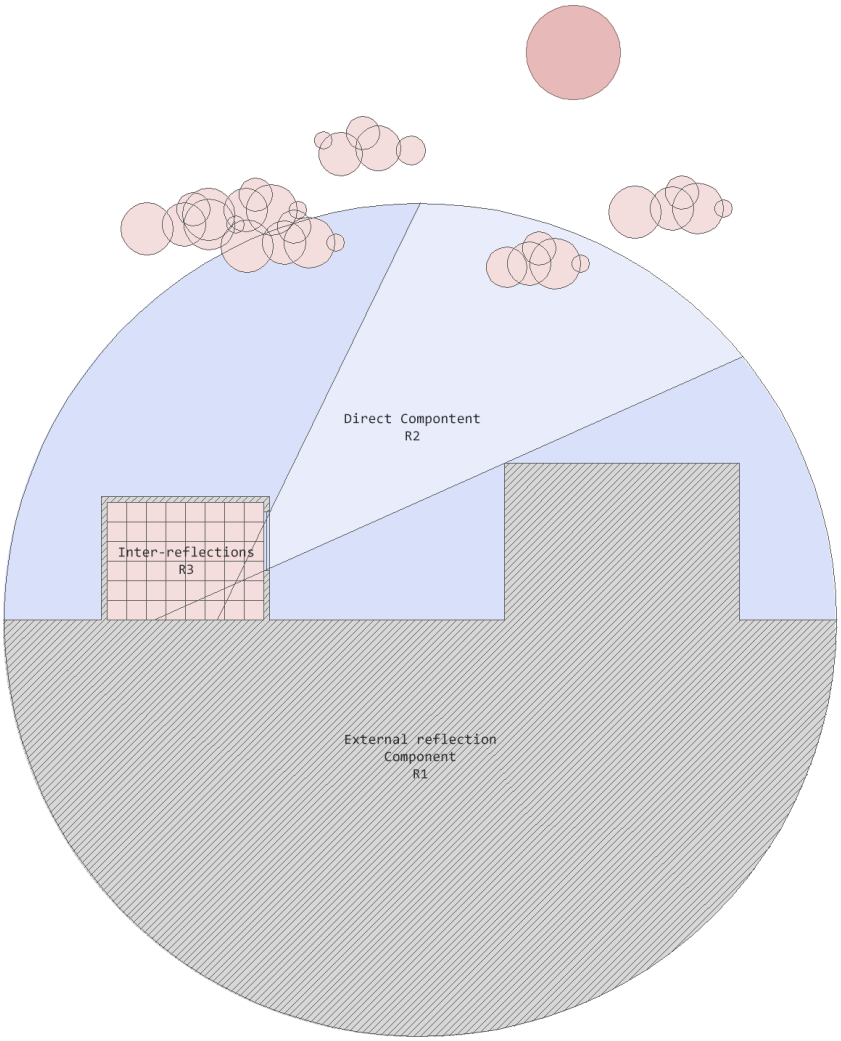


Figure 5
Illustration of daylight coefficient components

3.4.1 Externally reflected components

The matrix R_1 expresses the externally reflected components of daylight coefficients. Exterior surfaces could either diffuse or obstruct the light coming from a sky element. This could be expressed as a matrix which its rows are reinforce or weaken the light quantity related to each sky patch. In other words R_1 is a n x n, where n is the number of sky elements, transformation matrix where the natural skylight changes in respect with the unique exterior surrounding of each place. A special case, and the most simple, is where there is no obstruction in any zone then R_1 is the unity matrix.

$$[R_1] = I \tag{3.14}$$

3.3.2 Direct component

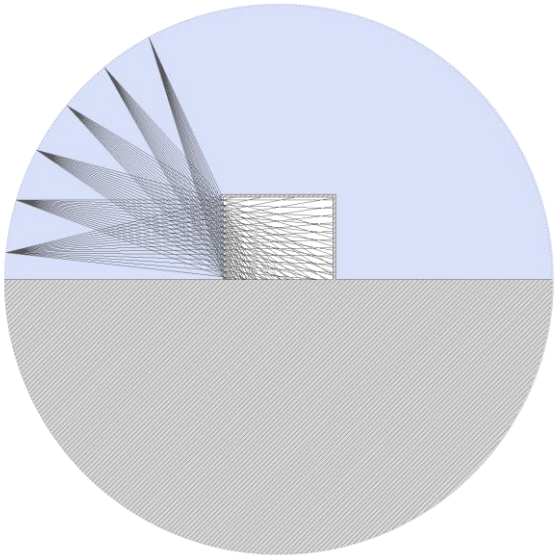


Figure 6
Illustration of R2 Direct Component

R_2 matrix represents the relation of room with the sky elements, so its size is m x n, where m is the number of the room elements and n the one for sky patches. An element r_2, is non-zero in case of the two lines which connect the center of an element with this of the window and the sky element's centers respectively are collinear. In case of a non-zero element which is on a horizontal plane r_2 is:

$$r_{2ij} = \sin\theta \, T_\omega \tag{3.14}$$

where θ is the altitude of the sky element and Tω is the window transmittance. In case of a vertical surface point with azimuth v (normal vector direction) the (3.14) is expressed as:

$$r_{2ij} = \cos\theta \cos(\varphi - \nu) \, T_\omega \tag{3.15}$$

where θ and ϕ are the altitude and azimuth of the sky element respectively.

Note that in case which sky patches are relatively very large compared with the window area, a significant amount of the sky may be invisible, which means that the equations must incorporate appropriate factors.

3.3.3 Internal reflected components

The illuminance after inter-reflection is a result of the initial illuminance E0i adding the inter-reflected one. More specific:

$$E_i = E_1\alpha_{i1} + E_2\alpha_{i2} + \cdots + E0_\iota + \cdots + E_m\alpha_{im} \tag{3.17}$$

where α_{ij} is the ration of illuminance that receives i surface element from the reflected light from j surface element. The above relation could be expressed also in a matrix form like:

$$\begin{bmatrix} E0_1 \\ E0_2 \\ \vdots \\ E0_m \end{bmatrix} = \begin{bmatrix} 1-\alpha_{11} & -\alpha_{12} & \dots & -\alpha_{1m} \\ -\alpha_{21} & 1-\alpha_{22} & \cdots & -\alpha_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ -\alpha_{m1} & -\alpha_{m2} & \dots & 1-\alpha_{mm} \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_m \end{bmatrix} \tag{3.18}$$

The quantity α_{ij} could be analyzed as:

$$\alpha_{ij} = \rho_i f_{ij} \tag{3.18a}$$

where ρ is the reflectance and f is the form factor (view factor) that is a unit-less geometric quantity. For more information about form factors there is a section on the (appendix B3) (Tzempelikos 2001), (Alshaibani 1996). The relation (3.18) explains that the initial illuminance is the product of a matrix A, which is defined by the geometric relation and material's reflection properties of the sub-surfaces of the interior, with the end illuminance. By inverting this matrix the following relation occurs:

$$[E] = [A]^{-1}[E0] \tag{3.19}$$

Given that the initial illuminance is a product direct and external reflected components the reversed A matrix is the R_3 matrix.

$$[R_3] = [A]^{-1} \tag{3.20}$$

3.3.4 Sky luminance

The sky luminance is multi-variant parameter, because of the light's nature. It passes through various transformations till it reaches the ground of the earth. In equation 3.12 a vector S*L it is formed which is the product of a patch's solid angle with its own brightness. This vector can be translated into a matrix dot product:

$$\begin{bmatrix} S_1L_1 \\ S_2L_2 \\ \vdots \\ S_nL_n \end{bmatrix} = \begin{bmatrix} S_1 & 0 & \dots & 0 \\ 0 & S_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & S_n \end{bmatrix} \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_n \end{bmatrix} \tag{3.21}$$

The L vector it could be further sub-divided into other matrix-vector products which are related with the sky condition:

$$[L] = [T_2][T_1][L_0] \tag{3.22}$$

where T1 is a transition matrix which gives the clear sky luminance distribution, while the T2 is the matrix which defines the change of distribution for example because of the clouds.

3.4 Window design

As it has been done for the topology optimization, it is essential to define the design variable for daylight analysis. This is definitely the window design. The amount of area, geometry and material of a window could significantly affect the daylight availability of an interior space. In this study the amount of the area is what will be taken as a design variable. According to Architect's Data Book (Neufert 2007), there are some directions for window design, driven from empirical studies. Some minimums are provided regarding the size of window over a room. More specific, among all of these a minimum of 17% of the room's area is needed as window area and for spaces with height greater than 3.5 meters a minimum of 30% of the interior opposite to the window wall.

||Chapter 04||
BESO and Daylight Analysis:
Grasshopper Formulation and code comparison

Abstract

In this chapter will be presented the connection between structural topology optimization with daylight analysis in Grasshopper (GH) with the help of Honeybee. This will be accomplished with the combination of two of their benchmark projects respectively cantilever beam (as loading and support parameters) and daylight coefficients. The scope of this connection is to focus only on the design process. In other words, no realistic inputs were introduced. An illustration of the designing method's on GH will be the main theme of this chapter. Also, will be presented a comparison between the parametric procedure over GH and the code over various standard CIE cloudy skies.

Keywords: daylight analysis, structural topology optimization, connection, benchmark, grasshopper, honeybee, illustration, design method

4.1 Rhino's Grasshopper

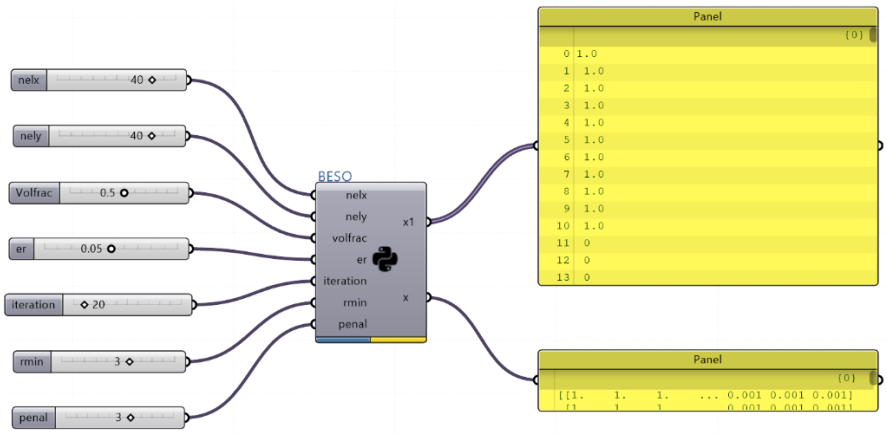
As it is foresaid, grasshopper is rhino's plug-in, which is an useful tool for parametric design. Grasshopper works with operational buttons/ toggles, which receive inputs either mathematical (equations etc.) either geometrical (connecting rhino's designed geometries with toggles). However, it seems extremely difficult to provide a detailed writing description. Consequently, an illustrated algorithm of the steps which have been followed, accompanied with some description, explain the procedure clearer.

4.2 Structural Topology Optimization

STEP 1. Link GH with BESO

Introduce the python code in grasshopper. The basic GHpython plugin is unable to process the extra python libraries such as numpy. Hence, for this task it used CH_Cpython plugin, created by Mahmoud Abdel Rahman. In the **CH_Cpython** toggle, is copied a simplified version of BESO algorithm where all the function of appendix A2 are merge into one script. The toggle receives the same inputs as BESO: mesh size, volume fraction, filtering radius, iterations number (to avoid the while loop), penalization number and evolution rate. The outputs are a numpy 2D array and a list x1:

Figure 7
CH_Ppython toggle with BESO script



STEP 2 Plane and Grid Construction

Construct the plane (a) and form the grid where the design variables will be set. In order to achieve that it is necessary to design a point in rhino environment and connect it with a GH point toggle. Then use this point as the plane's origin (center) and two vectors for its directions. Then connect the plane with the rectangular grid creation (b). The size of the grid cells should be coincide with the grid set over the topology optimization:

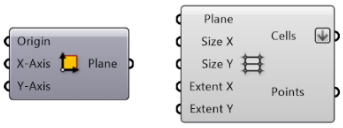
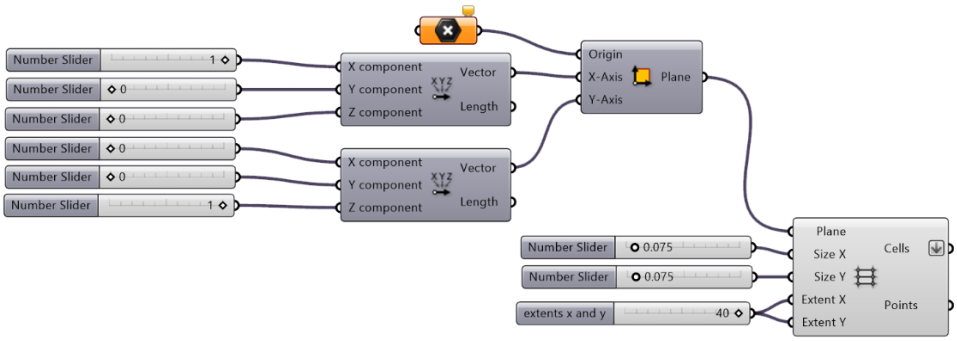


Figure 8
from left: to right (a)plane construction and (b) rectangle grid (recgrid), on the bottom the GH connection scheme of plane and grid toggles.



After this configuration on the rhino screen it will be appeared a generic plane in a form of rectangular grid and inside (or out of its borders depending of its size) the desired grid, as it is illustrated on the following figure 9:

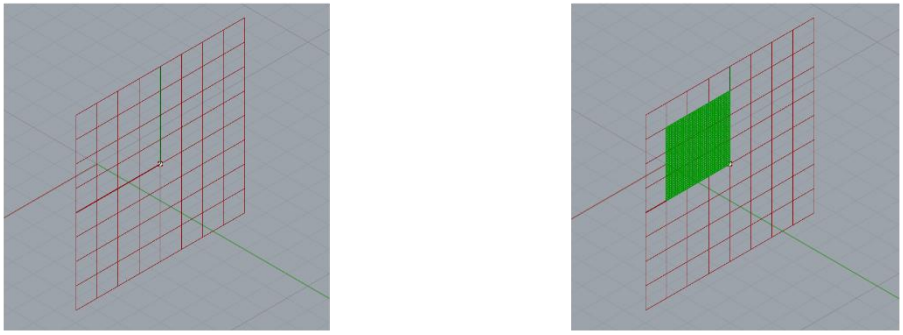


Figure 9
plane and grid visualization from grasshopper to rhino's environment

STEP 3 Pattern Design

If the topology optimization procedure with BESO algorithm it is approached with absolute designing criteria, it is nothing more than a pattern with solid and void rectangles. So the third step is to help GH read the x1 output as a pattern list. Firstly, it should be drawn one rectangle curve and a surface corresponding to that curve. In figure 8, the rectangle A and the surface B, should be on the same place in order to succeed this configuration. Then they must be set as a GH curve and geometry respectively.

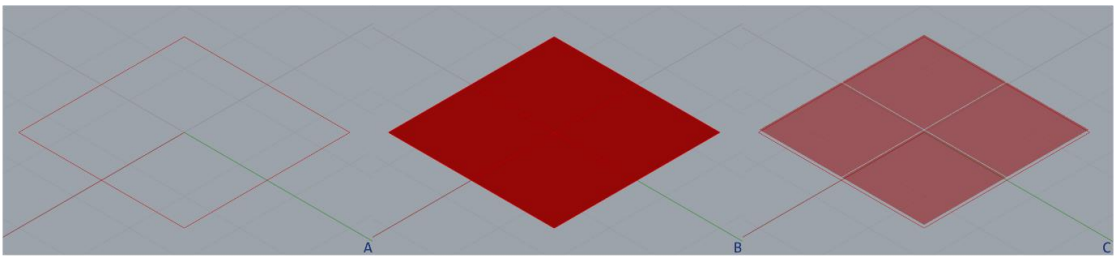


Figure 10
pattern source geometries

Afterwards, this geometries will be seperatly merged with two empty ones (representing the void). with the GH merge toggle (c) in a data tree*. The first merge toggle it will receive the curve (defining the region) and the second one the surface (definining the solid/void mode), this will help use as many time as the pattern requires. In the meanwhile, repeat data toggle (d) gothers the results of the BESO list of the step 1 and stores it as the data to be repeated and uses for its length, the one of the grid cell's list (e). The way to connect the above sub-steps is

with two list item (f) toggles coresponding to the merges. So, the will correspond its data tree to 0 / 1 list value.

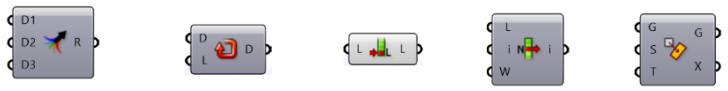


Figure 11
from the left to right (c) merge, (d) repeat data, (e) list length, (f) list item, (g) rectangle mapping

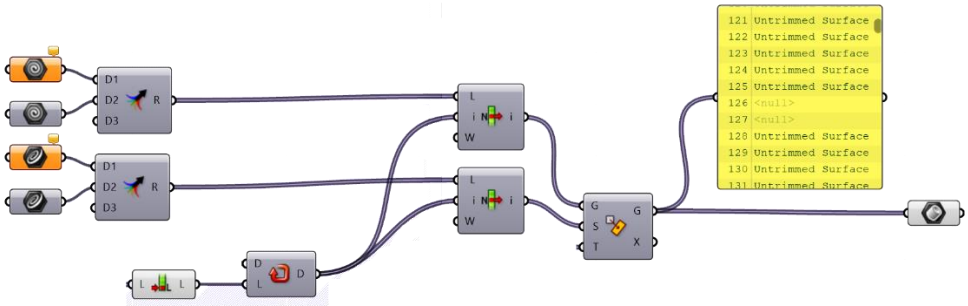


Figure 12
Pattern configuration. In D on repeat data it is expected a 0/1 list

Last but not least, this identification should be shown over the grid. Rectangle mapping toggle (g) it will provide this outcome, by receiving the geometry merge indexes and the curve source indexes by the corresponding list item (f) toggles and by using the grid cells' as its target. Eventually the end product it is shown in figure 11. This geometry is sensitive in every change of topology optimization's list, and will follow its changes.

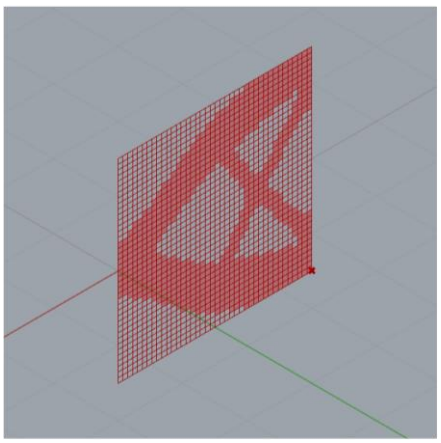


Figure 13
Procedure outcome: Cantilever Beam pattern

The final sub-step is to translate this pattern into a form which will be useful for the daylight analysis, so brep* it is used in order to achieve that.

4.3 Daylight Analysis

Daylight analysis is available in GH through honeybee, which is one of ladybug tool created by Mostapha Sadeghipour Roudsari and Chris Mackey. It is an easy way use the attributes of radiance, Daysim and energy-plus all in GH platform. In the current case for this project analysis the grid-based daylight is used.

The most useful honeybee toggles were honeybee surfaces, standard CIE sky component, rad parameters, grid base analysis and run daylight simulation toggle:

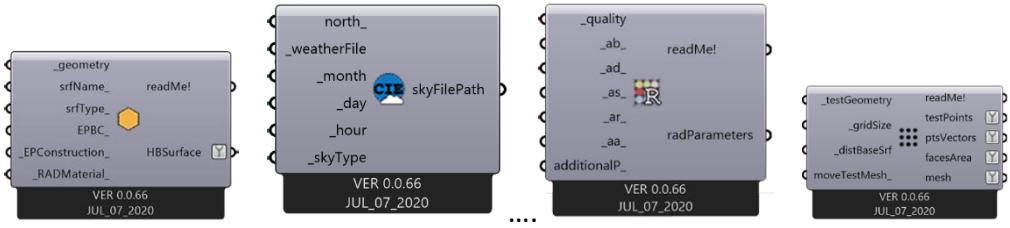


Figure 14
Above. From left to right the honeybee surface (a) cie standard sky (b) rad parameters (c) test point (d). Below grid based daylight analysis (e), daylight analysis simulation (f)

STEP 1 Honey-bee surfaces

The first step is to match every room geometry with Honebee surfaces. This kind of toggle gives indentities to a surface that is essential for a daylight simulation to be run. In the case of a simple room six surfaces they are going to be needed. One for floor, one for roof, three for exterior walls and one for the façade with the structural TO pattern. Consequnetly, six honeybee surfaces toggles will need and six coressponding brep geometries. The Façade's has already been created from the stuctural TO procedure:

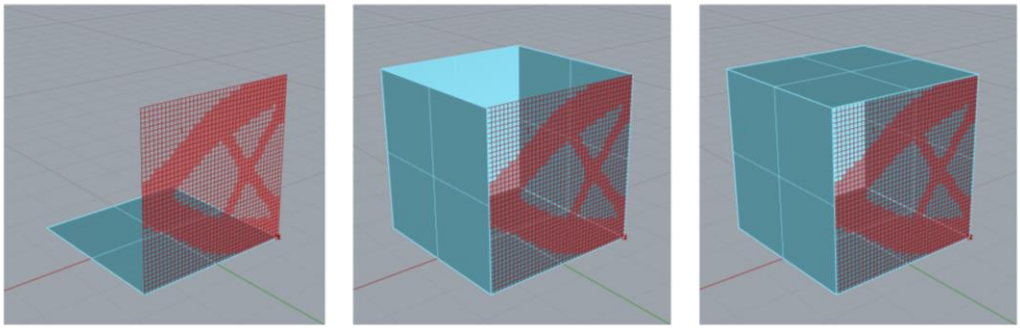
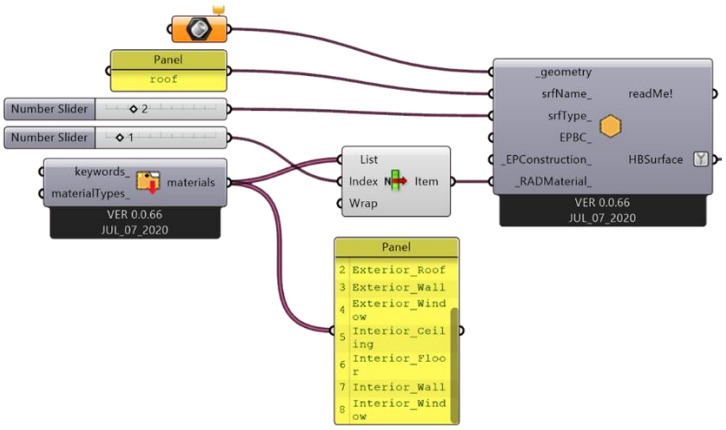


Figure 15
Room's components designed (above) and set as honeybee components (below)



All these surfaces will be merged into one brep that it will be used as a honeybee object for the daylight analysis.

STEP 2 CIE Sky Definition and rad parameters

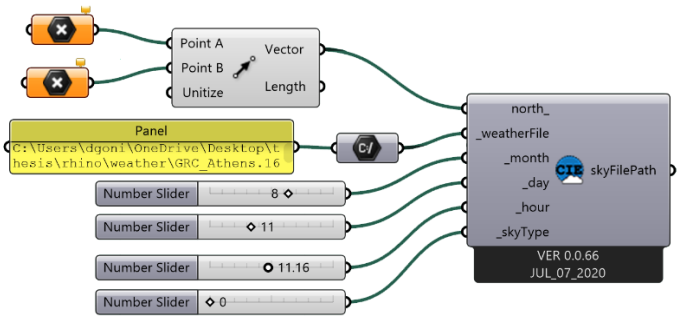


Figure 16
Sky type component

The sky element (b toggle) is essential to run a daylight analysis. The CIE overcast standard sky it will be used like it has been used for daylight coefficient code. In Honeybee a weather file

*.epw is needed, a vector which defines the true north and sliders for the month, day, hour and sky type.

On the other hand rad parameters (radiance parameters) will be used to define ambient characteristics. _ad_ is for ambient divisions for Monte Carlo method and _ar_ the ambient resolution.

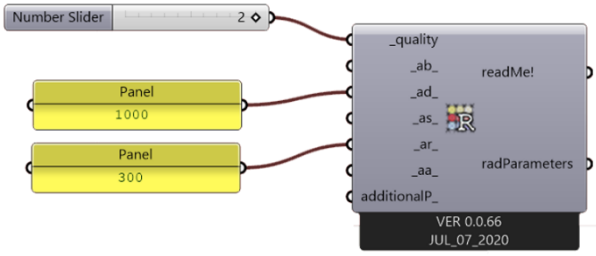


Figure 17
Rad (Radiance program) parameters

STEP 3 Test Points, Grid-Based Analysis and Daylight Simulation

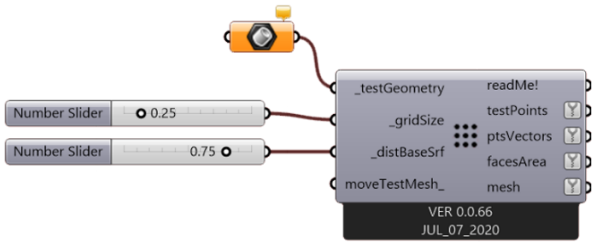


Figure 18
Grid Testing Point definition component

A plane, which an offset grid test points will be used to simulate the daylight analysis. By assembling all the elements and connecting them with the grid-based analysis toggle and daylight simulation toggle, the radiance program it will run and produce files with the results, which can be later used for simulation purposes

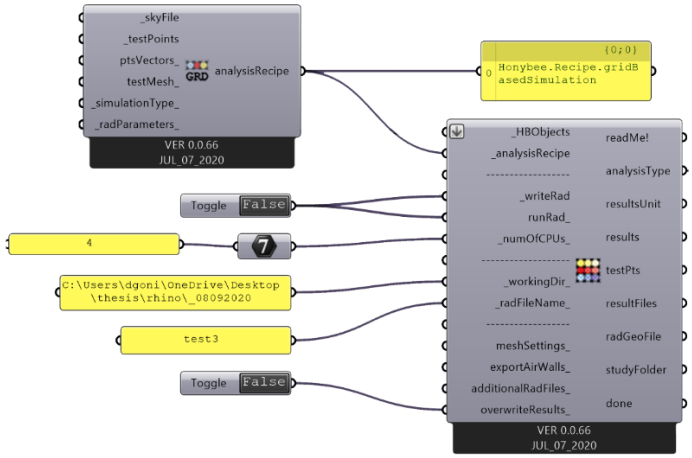


Figure 19
Grid-Based Analysis and Daylight Simulation components

4.4 Code Inputs

Once the design method, in Grasshopper it is formed, a validation of the written code of daylight coefficients seems to be needed. In order to procced it, a study case is mandatory, where specific inputs will be provided. First of all should be defined the finite element discretization over the room. The elemental size that has been chosen is 0.1m x 0.1m. This in terms of structural topology optimization creates a coarse mesh, but in terms of window design is the minimum window that can be constructed. Additionally, it has been found computationally heavy to compute at the same time in finer meshed both topology optimization and daylight coefficient algorithms. A source code is provided in appendix A1 and B1.

Also no realistic material inputs have been provided in order to focus on the design methods. The study case it will need to define the geometry and variables like hour, day, month. Hence, the following list it summarize the basic variables of study case:

- Room geometry: 3 x 3 x 3 m.
- Rooms division elemental size: 0.1 x 0.1 m
- Material: the design material provided be BESO algorithm (Huang and Xie)
- Façade: an optimized geometry with cantilever beam's constraint is used with 0.45 target volume. The reason is to check code's behavior with a non-conventional geometry. Figure 11
- Hour: three samples of hours: h = 10:00, 12:00, 14:00.
- Day: the 21st is chosen as a sample.

- Month: all months taken into consideration.
- Location: Athens
- Luminosity test point grid: 0.1 x 0.1 m (900 testing points) over the floor level.

4.4.1 CIE Standard Skies Formula

The following part describes the steps of calculating the relative luminance distribution. The position of both sky elements (Appendix B2) and sun are the main variables of sky illuminance distribution (P. Tregenza 2004), (Darula and Kittler 2002). Additionally, there are some indicators a, b, c, d, e which has been introduced so as to describe the atmospheric conditions, which affect sky's luminosity. The following ration gives the luminosity level of each sky's element:

$$L_i = \frac{La_i}{Lz} = \frac{f(x)\varphi(z)}{f(g_z)\varphi(0)} \tag{4.01}$$

In order to calculate Lα of each sky elements, f and ϕ are provided:

$$f(z) = 1 + c * \left[\exp(dx) - \exp\left(d\frac{\pi}{2}\right)\right] + e\cos^2 x \tag{4.02}$$

$$\varphi(z) = \begin{cases} 1 + a\exp\left(\frac{b}{\cos z}\right), \text{when } 0 \leq z < \frac{\pi}{2} \\ 1, \text{when } z = \frac{\pi}{2} \end{cases} \tag{4.03}$$

Where g_s is the altitude of the sun, z is the angular distance between element's center and zenith (π/2 – g), zs is the angular distance between sun's center and zenith (π/2 – gs) and x is the shortest angular distance between element's and sun's centers respectively. x is given by the following relation:

$$x = \arccos(\cos zs \cos z + \sin zs \sin z \cos |a - as|) \tag{4.04}$$

Where α and αs are the azimuth angles of the sky element and the sun. Hence, by defining sun's position, sky's geometry and condition (a, b, c, d, e), it is possible to model the sky condition for the study case.

4.5 GH simulation

The same procedure it has been followed in GH with the same inputs as the code. The sky type, which has been chosen, is “cloudy” and Athens weather data has been provide (epw). The results of the simulation has been visualized in the following figure:

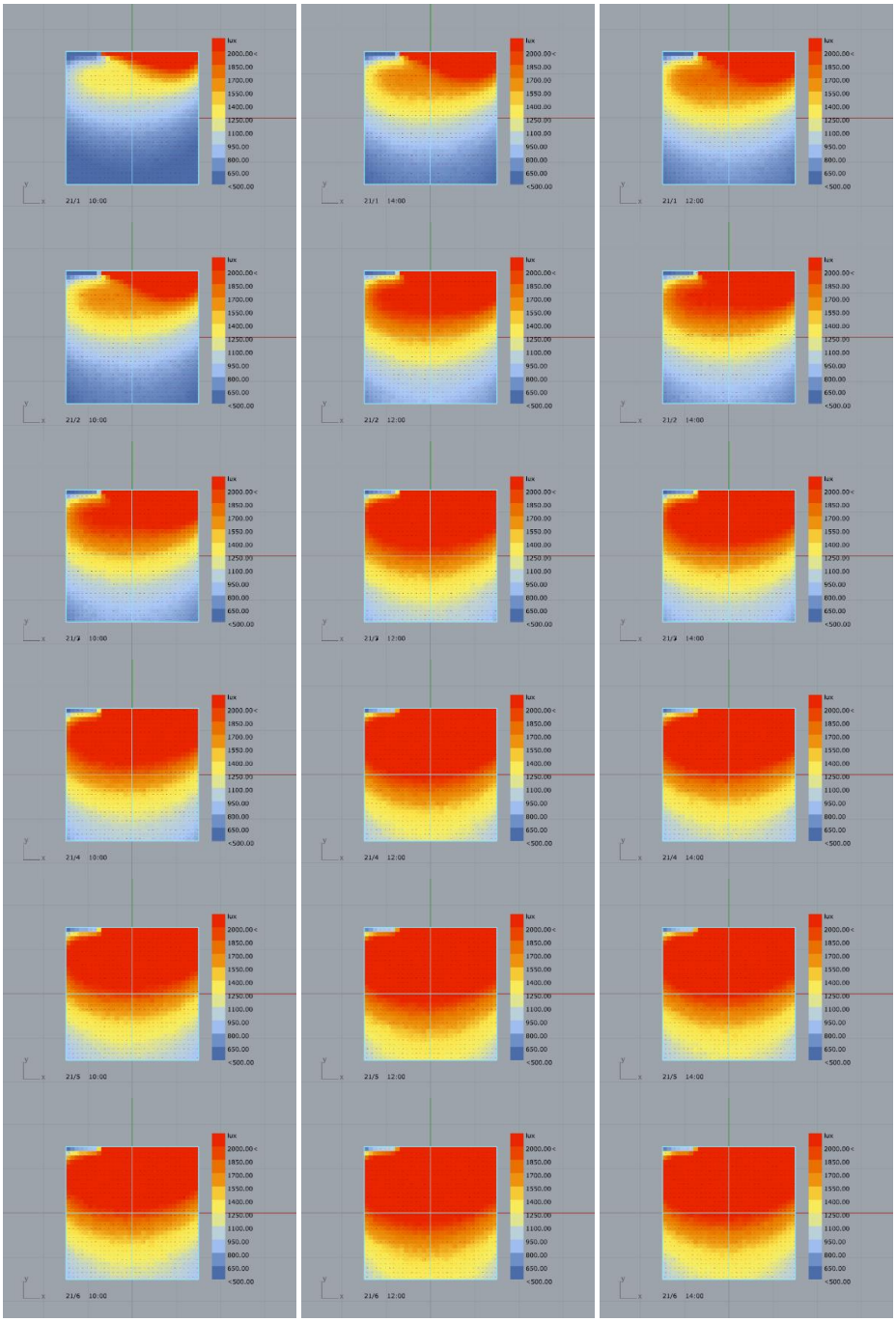


Figure 20
From top to bottom January’s, February’s, March’s, April’s, May’s and June’s results for 10:00 | 12:00 | 14:00 Athens zone time

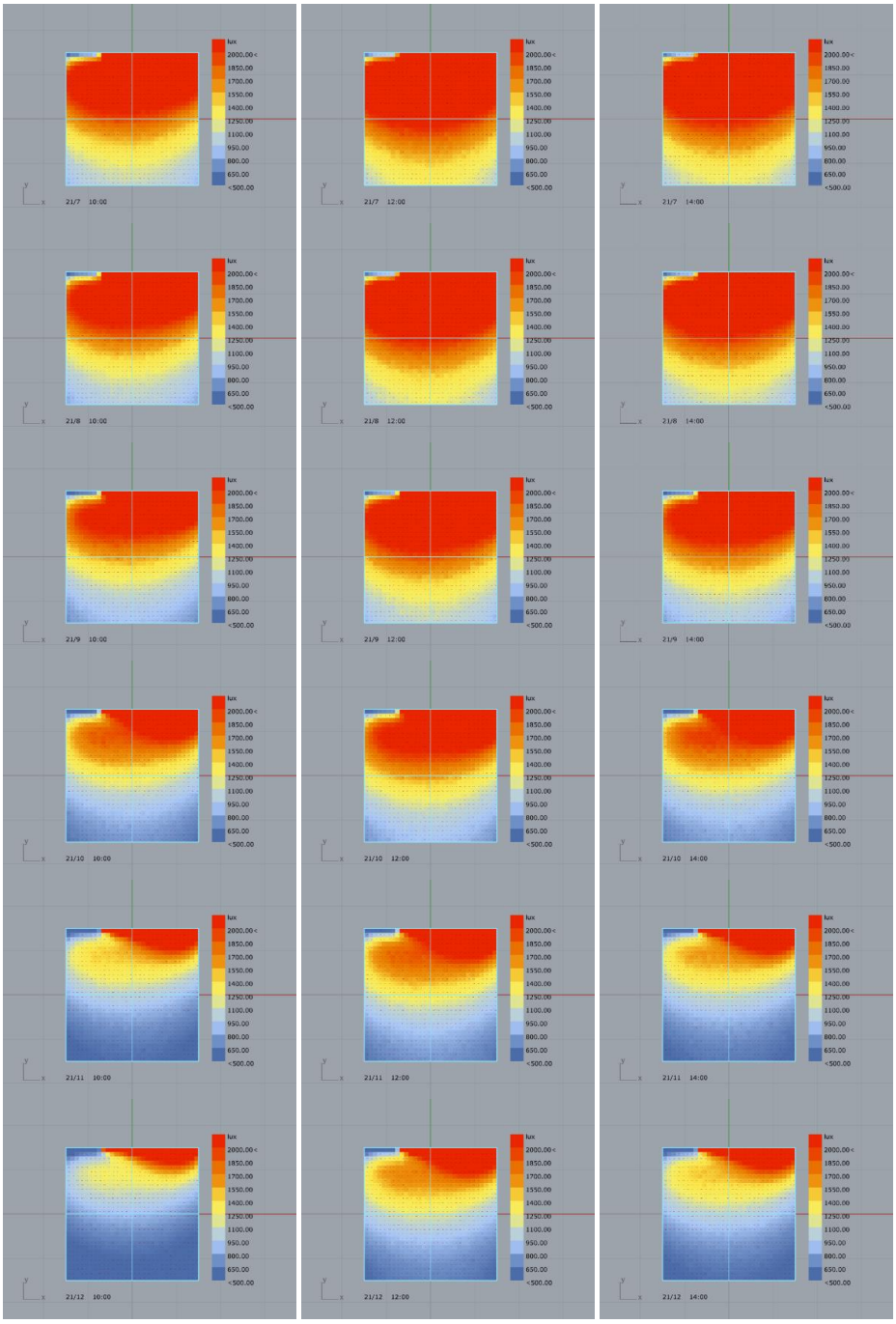


Figure 21
From top to bottom July's, August's, September's, October's, November's and December's results for 10:00 | 12:00 | 14:00 Athens zone time.

4.6 Comparison

The mean luminance under of a variety of different CIE cloudy standard skies, in three different sky geometries, in respect with the honeybee model has been summarized over the figures 17, 18, 19. The sky types that where most related to HB sky have been type 2, 4, 6, 7. Type 2 and Type 4 standard skies are referred as overcast and Type 6 and Type 7 as Partly cloudy.

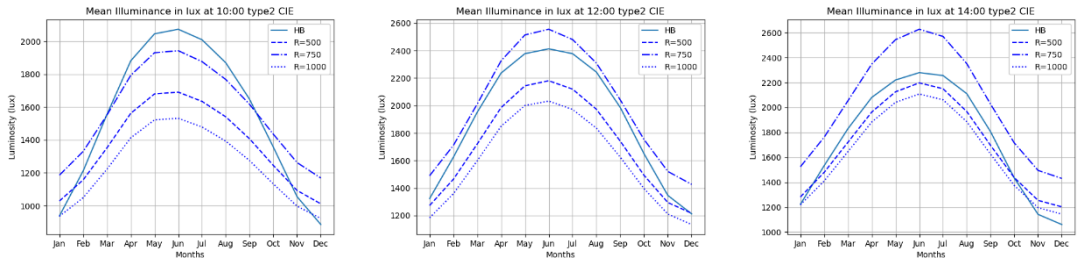


Figure 22
Type 2 of CIE Standard General Skies for 10:00 | 12:00| 14:00 o' clock

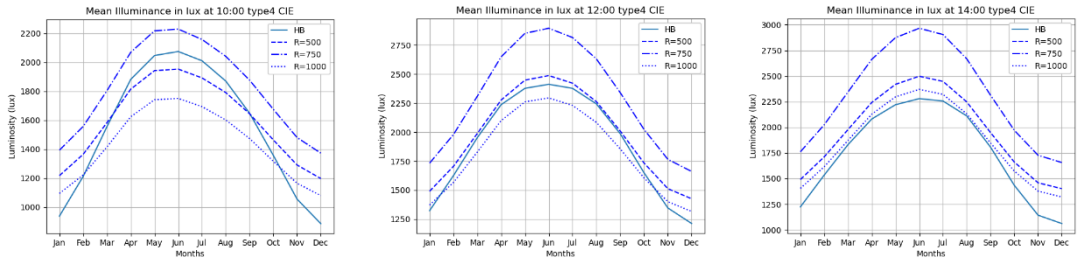


Figure 23
Type 4 of CIE Standard General Skies for 10:00 | 12:00| 14:00 o' clock

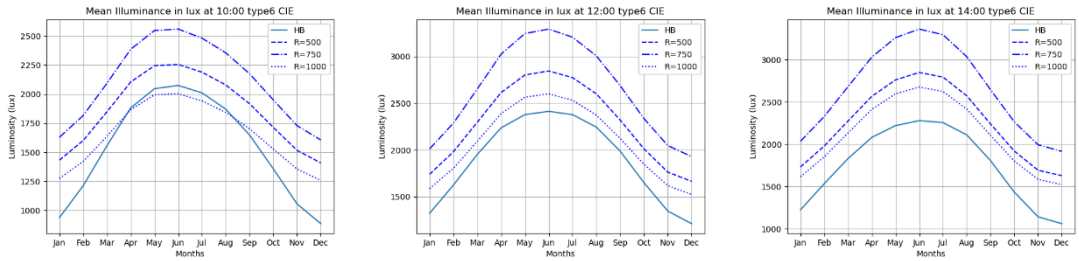


Figure 24
Type 6 of CIE Standard General Skies for 10:00 | 12:00| 14:00 o' clock

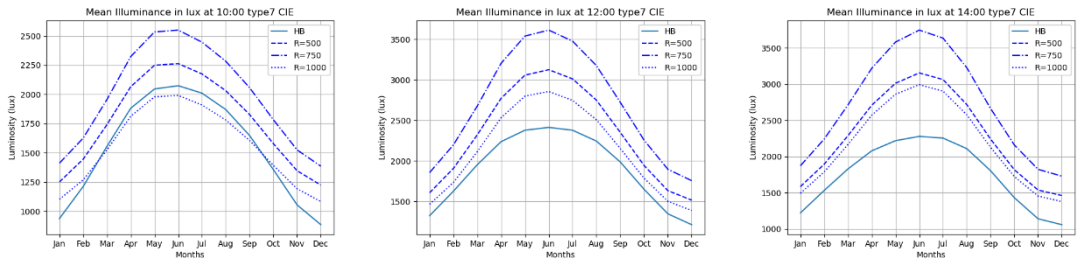


Figure 25
Type 7 of CIE Standard General Skies for 10:00 | 12:00| 14:00 o' clock

According to the graphs Type 4 seems to be numerically closer to HB sky for R=500, 1000 of the sky dome and for the 12:00 and 14:00. At 10:00, R=500 seems to function better, even though there is a great difference in months with low sun's altitude (January, December). On the other

hand, Type 7 for R = 1000 gives results close to the HB sky. It is obvious that this issue need further examination over more sky geometries and sky types. Additionally form factor calculation needs improvement in order to succeed a more accurate outcome.

Another issue in this comparison it has been that HB does not provide arithmetical information. The lack of the reflectivity number of the surfaces, the sky indices of “cloudy sky” has not been helpful over this comparison. Nevertheless, the goal has not been to develop the most accurate daylight coefficient method but to point the fact that there is a major advantage of control by computing this method instead of using the HB. Moreover, GH needs approximately ten minutes in order to run a single simulation, while the code with the appropriate setups only 35 seconds. There is no doubt, that the code of DC need more elaboration, thus the study case it will use the GB and HB to crosscheck different room geometry and finer façade mesh.

||Chapter 05||
Design Strategies

Abstract

In this chapter a more solid and relatively realistic case study will be presented to examine the dual issue of optimization. The main direction is the way, which daylight analysis and structural topology optimization could cooperate in order to organize a design strategy. More specific, structural topology optimization is converted into a space design parameter, not related to one single target volume, but over a set of these. Finding the minimum of the target volumes' subset, which satisfies both topology optimization and daylight analysis, will provide the optimum solution. Grasshopper's Honeybee was used in order to subtract results from a validate method over the scientific community.

Keywords: dual optimization, honeybee, grasshopper, set of target volume

5.1 The issue of duality

The procedure of structural topology optimization as it known till today is a static procedure, while daylight is not. However, it can be searched this geometry of the domain which could satisfy the majority of the need for natural light over a year. In order to combine the structural topology optimization with daylight analysis it was mandatory to infuse the one into another. Openings or voids are the words that are often referred to windows in architectural. This association gave the idea to connect the window design which is the basic parameter of daylight availability inside a building, with the void design variable of structural topology optimization. (Marler and Arora 2004)

The major issue over this connection was the computational time. The more fine and detailed the elements of a TO and Daylight Analysis become the more computational time it takes. In order to, reduce calculations, as well as time, the following algorithm introduced. A range of target volumes was set according to window design and structural topology optimization design needs. In the meantime, a lower and upper boundary for mean illuminance was defined according to the use of space (house, office etc.). BESO provided the geometries of these target volumes and the Daylight Analysis evaluated them by keeping some and eliminating others. The minimum value of this subset is the desired one. There is no doubt that the set of the new target volume's should be researched further over various issues of daylight analysis (visual comfort etc.).

The following types trying to mathematically describe the procedure:

$$\text{Min: } V_j^* = V_{j-1}^* - st, \qquad st = 0.5 \;\; V^* \in [0.7, 0.35] \tag{5.01}$$

$$\text{Min: } C = \frac{1}{2} f^T u \tag{5.02a}$$

$$\text{Subject to: } V_j^* - \sum_{i=1}^{N} V_i x_i = 0 \tag{5.02b}$$

$$x_i = x_{min} \text{ or } 1 \tag{5.02c}$$

$$\begin{cases} E_{max} \geq E_j \geq E_{min}, & \text{select } V_j^* \\ \text{otherwise reject } V_j^* \end{cases} \tag{5.03}$$

5.2 Algorithm steps and Flow Chart

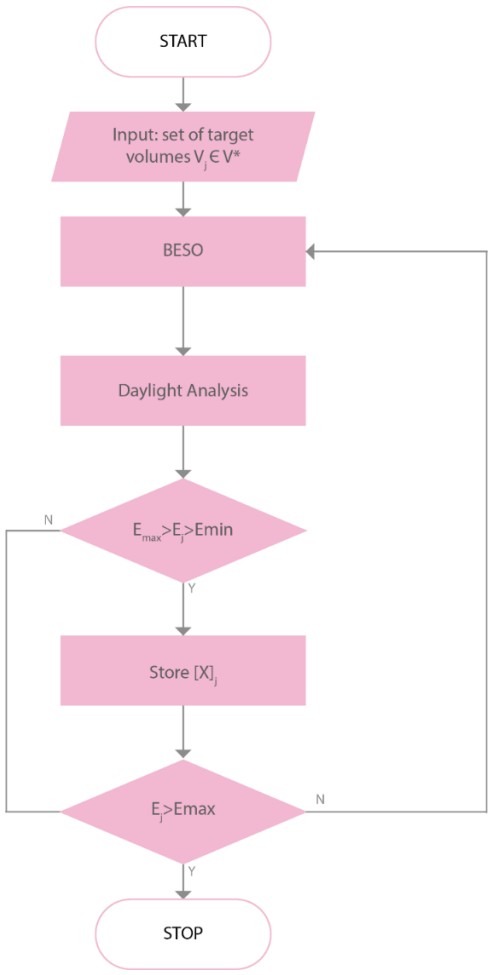


Figure 26
Flow Chart BESO and Daylight Analysis

1| Define the set of target volumes

2| Run Structural Topology Optimization for the target volume

3| Run daylight analysis for the target volume

4| Check mean illuminance. If $E_{max} > E_{mean} > E_{min}$ then save x design variable matrix.

5| If $E_{mean} < E_{max,}$ repeat step 2, else end procedure.

5.3 Case study

In this section a combination (with numerical inputs) of the main ingredients of this thesis will be presented in steps.

1| First of all, define room's geometry. A space of 4m x 4m x 3m was the case study, only one of the four wall was the window wall.

2| Specify the set of target volumes. The maximum target volume was 0.70, which is the minimum needs of windows, and the minimum was 0.45, which is the target volume where the optimized result retains its continuity. The step of target volume set was 0.05, so:

$$V^* = [0.45, 0.50, 0.55, 0.60, 0.65, 0.70] \tag{5.04}$$

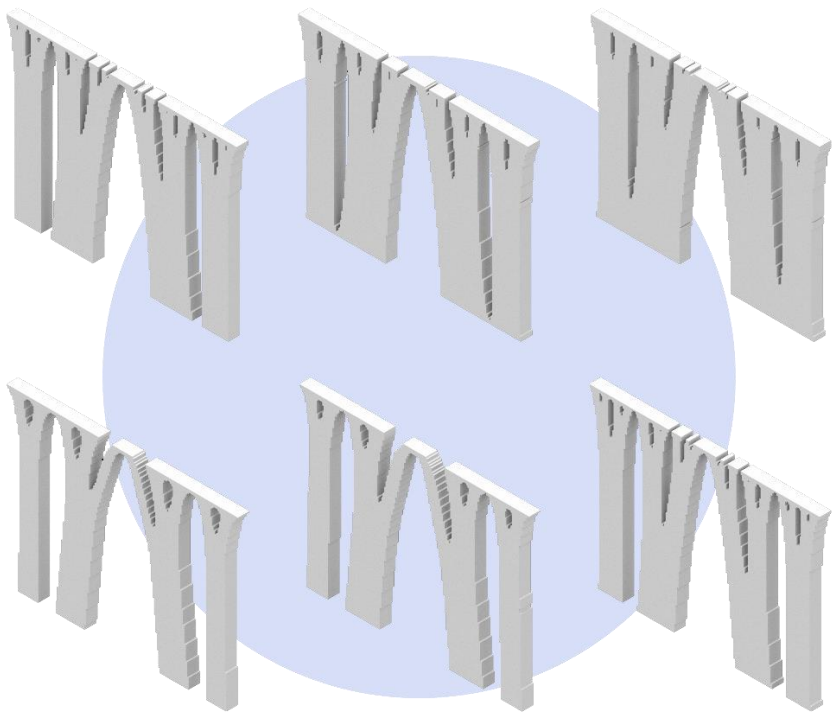


Figure 27
from left to right and top to bottom target volumes: 0.70, 0.65, 0.60, 0.55, 0.50, 0.45

3| Run BESO for each target volume. The inputs of the algorithm were nelx = 160, nely = 120, penalization 3., filter radius 3. for target volumes equal-greater than 0.60 or else 6. Material inputs were related with concrete, thickness of the wall is 0.25 m and the loading case was distributed over the roof.

4| Set boundaries of mean illuminance. The maximum acceptable is 2000 lux, while the minimum for spaces for simple visual tasks is 500 lux. (Futrell, Ozelkanb and Brentrupc 2015)

5| Over a range of target volumes, where structural TO compliance is converged, run daylight analysis over the 21st of December (smallest day) and 21st of June (largest day).

6| Select the minimum over the set of target volumes, which satisfies the illuminance constraint.

Over the following figures there is the simulation both the optimized wall and illumination levels. For the topology optimization it was used the MATLAB code and for daylight analysis was used honeybee over cloudy sky. After this procedure the accepted target volumes are 0.45, 0.50 and 0.55. Consequently the minimum volume which maximize the daylight in the current case study is 0.45. Nevertheless, it would be interesting to examine these three designs over various indices of daylight analysis.

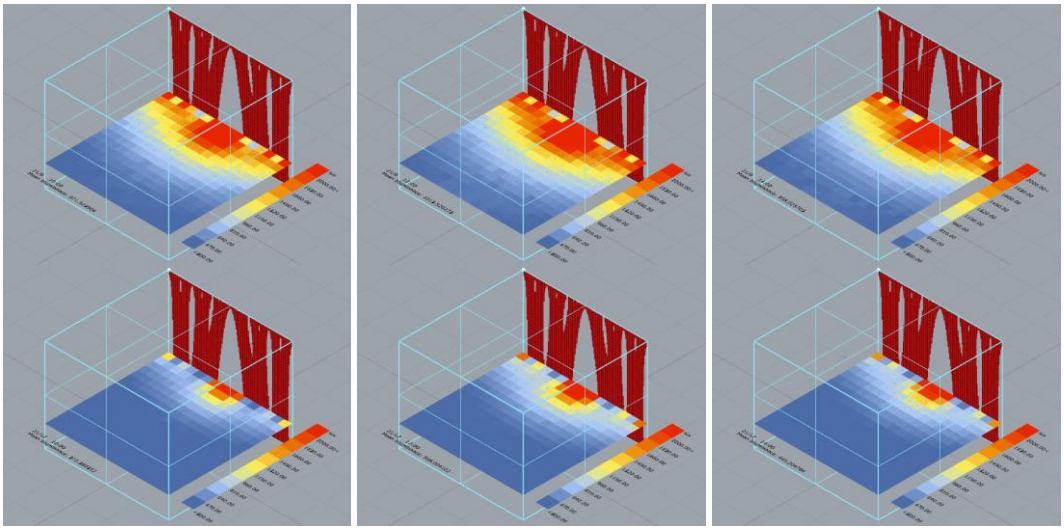


Figure 28
from left to right and top to bottom HB grid-based daylight analysis for 10:00 | 12:00 | 14:00 of 21st of July and December. TARGET VOLUME 0.70

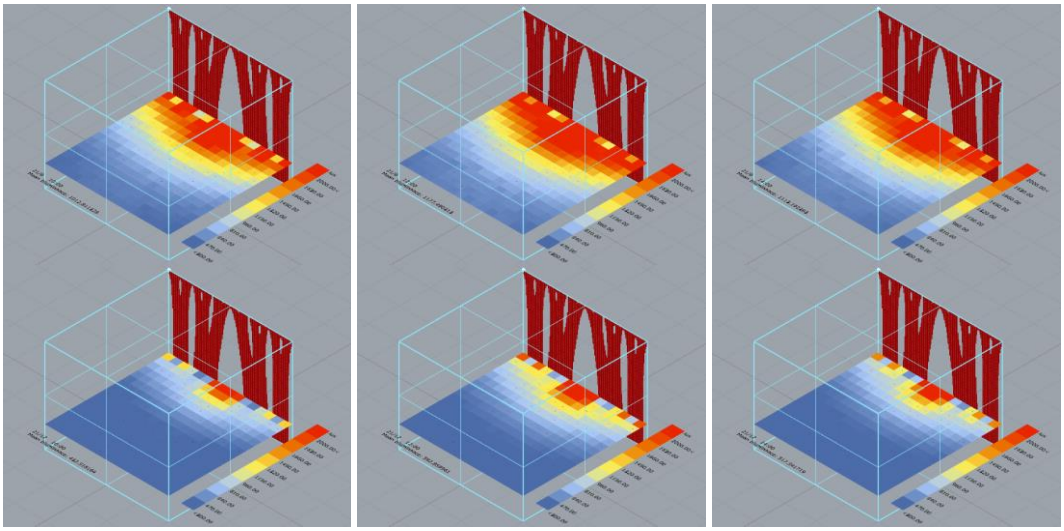


Figure 29
from left to right and top to bottom HB grid-based daylight analysis for 10:00 | 12:00 | 14:00 of 21st of July and December. TARGET VOLUME 0.65

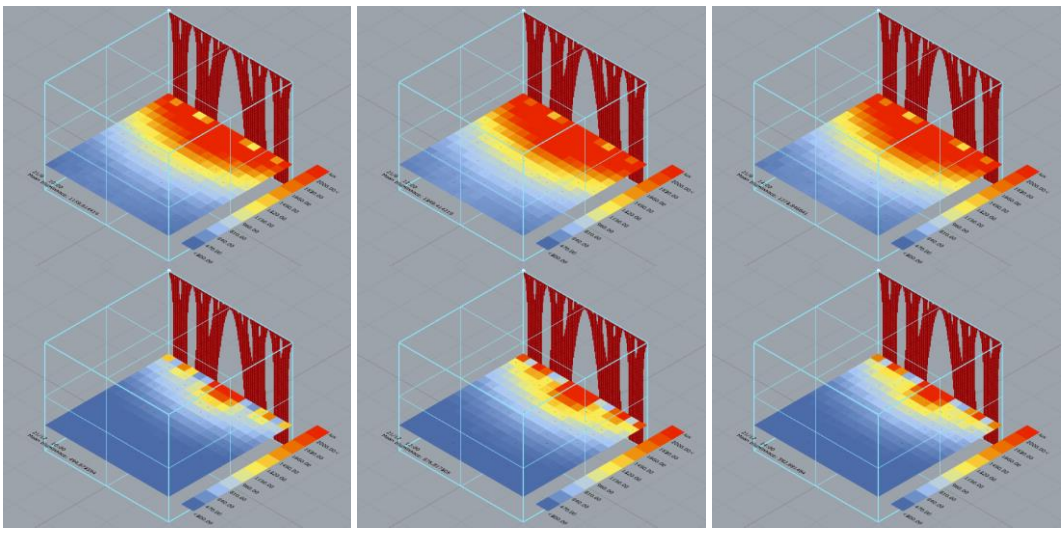


Figure 30
from left to right and top to bottom HB grid-based daylight analysis for 10:00 | 12:00 | 14:00 of 21st of July and December. TARGET VOLUME 0.60

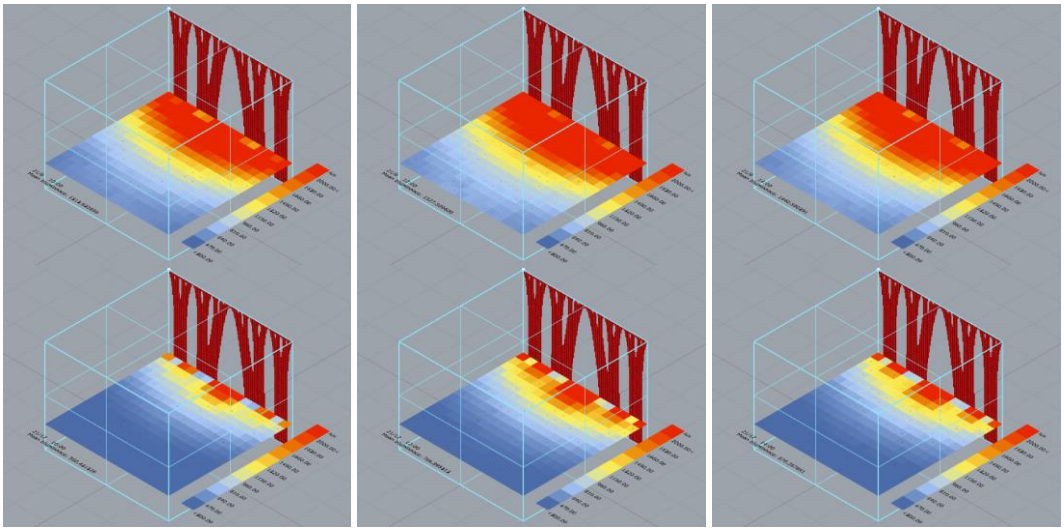


Figure 31
from left to right and top to bottom HB grid-based daylight analysis for 10:00 | 12:00 | 14:00 of 21st of July and December. TARGET VOLUME 0.55

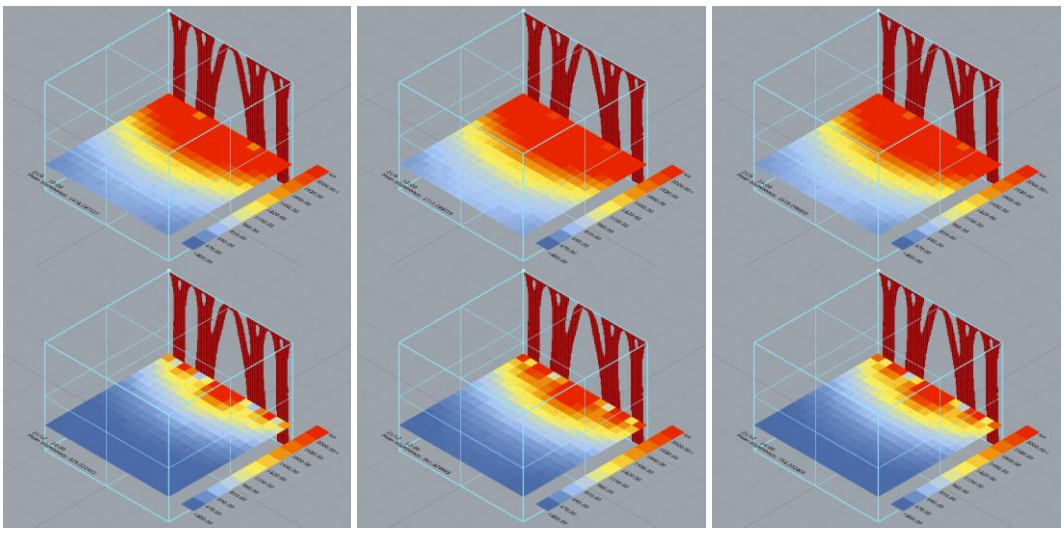


Figure 32
from left to right and top to bottom HB grid-based daylight analysis for 10:00 | 12:00 | 14:00 of 21st of July and December. TARGET VOLUME 0.50

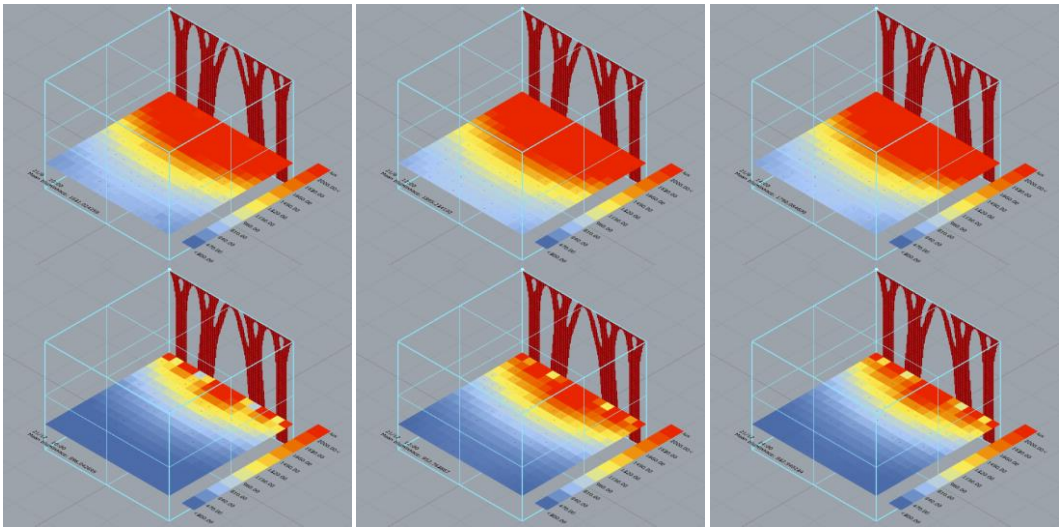


Figure 33

from left to right and top to bottom HB grid-based daylight analysis for 10:00 | 12:00 | 14:00 of 21st of July and December. TARGET VOLUME 0.45

||Chapter 06||
Conclusions

The flashpoint of this thesis was whether structural topology optimization could cooperate with qualitative space design parameters. After the research, computation and writing procedure it is clear that it is possible to connect both structural and daylight analysis. Firstly, a break down and extensive research over these two subjects was the first step. An element based design variable structural topology optimization algorithm was chosen for strategic reason. It seemed to be easier to connect it with the daylight analysis in coding level.

Then the connection occurred through the design variables, because the minimization of strain energy by reducing the volume of a design domain could cooperate with the maximization of daylight in a room. At the begging the goal was to make daylight analysis a parameter of structural topology optimization. This would have been succeeded by introducing daylight as a constraint. However, this goal was modified and evolved during the research over these two turned into a dual optimization matter.

So, the connection over these two subjects was updated in two main procedures. The one was this of the search of the minimum volume over a target volumes' set, which minimize the strain energy. On the other hand the second maximized the daylight under a set of constraints of minimum and maximum mean daylight levels. The combination of both of them was the main headline of this thesis framework. The design methodology which has been proposed successfully analyzed it. It is clear that a further research over distinctive study cases has to be made in order to check the upper limit of mean illuminance. But the results demonstrated a logical pattern over the minimization/maximization procedure of solid volume/ windows.

Last but not least, the manufacturability of the geometry even though it was not as an autonomous chapter and goal of this thesis is always present by the choices that were taken over the study case (material, support an d loading cases). To sum up, the chapter structure of this thesis reflects the reasoning which was followed in order to organize and actualize this design method. The results of this method were quit promising and it could be the basis to evolve it through the upcoming future thoughts, which also was one of the major if not the most important goal, to set the basis for future academic work.

6.1 Future thoughts

During the thesis' writing a list of thoughts, concerns and remarks were immerged, hence there catalogued and enlisted for future academic work over this subject.

- The basic connection between structural topology optimization has been completed successfully. The duality matter was resolved successfully in a basic yet efficient form. The next step is to incorporate daylight analysis with a more organic way into the structural topology optimization procedure.
- The realistic conditions of the materials and loading gave results which are 3d-printer friendly. Also, the loading case of distributed load over one edge offers a geometry with acceptable geometry's angle. However, manufacturability should be considered and studied further.
- Void control, related with the filtering scheme, in topology optimization could collaborate well with daylight analysis.
- The sensitivity Analysis of structural topology optimization could be linked with the daylight analysis in order to avoid regions of overexposure on the sun/light.
- Changing the type of topology optimization it would be challenging. Also a study over the lattice structure it would be highly related to shading needs of a space.
- The change of 2d domain into 3d domain will evolve this procedure.
- The calculation of form factors could be improved with more sophisticated new methods which are computationally more efficient than the direct.
- The daylight analysis method could be evolved from static to dynamic. Hence it will be more accurate with the realistic conditions.
- Visual comfort indices according to the hours of occupancy of the place (dynamic indices) it will evolve the present design procedure and make this research extremely useful.
- Although, writing code has not been as much efficient as the grasshopper, in terms of results, it has been the baseline of the whole computational procedure comprehension.
- Grasshopper procedure is closer to a generative design mentality, where the designer takes the final decision, while the code procedure is more deterministic.
- By updating the form factors calculation by more contemporary and sophisticated methods, it would be possible to make a stronger computational tool natural light distribution inside of a room.

- Enrich the design method with other type of analysis, such as: thermal analysis, energy analysis.

||Appendices||

A1. Matlab and a simple Python3 Code for 2D BESO.

```
%%%%  MODIFIED SOFT-KILL BESO CODE IN ORDER TO CORRESPOND OVER  %%%%
%%%%   GRAVITATIONAL LOAD AND STRUCTURES DESIGNED INTO METERS   %%%%
%%%%        THE MODIFICATION PRODUCED  BY DIMITRIS GONIDAKIS        %%%%
%%%%           ORIGINAL SOURCE CODE  BY X. HUANG and Y.M. XIE          %%%%
%%%%            THIS CODE IS FOR ACADEMIC SCOPE IT DOES NOT            %%%%
%%%%                    TESTED ON REAL CONSTRUCTIONS                   %%%%

el_x = 0.5;
el_y = 0.5;
height = 3;
width = 2;
nelx= width/el_x;
nely= height/el_y;
volfrac = 0.45;
er = 0.01;
rmin = 5.;

% INITIALIZE
x(1:nely,1:nelx) = 1.;
vol=1.;
i = 0;
change = 1.;
penal = 3.;
[KE] = lk;

% START iTH ITERATION
while change>0.0001
  i = i + 1;
  vol = max(vol*(1-er),volfrac);
  if i >1;
    olddc = dc;
  end
  % FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal, width, height);
  % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  c(i) = 0.;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2;2*n2+1;2*n2+2;2*n1+1;2*n1+2],1);
      if (x(ely,elx) == 1);
        dc(ely,elx) =0.5*Ue'*KE*Ue;
      else
        dc(ely,elx) =0%
      end
      c(i) = c(i) +x(ely,elx)^(penal)*0.5*Ue'*KE*Ue;% strain energy compliance
    end
  end
```

```
% FILTERING OF SENSITIVITIES
[dc] = check(nelx,nely,rmin,dc);
% STABLIZATION OF EVOLUTIONARY PROCESS
if i > 1;
   dc = (dc+olddc)/2.;
end
% BESO DESIGN UPDATE
[x] = ADDDEL(nelx,nely,vol,dc,x);

% PRINT RESULTS
if i>10;
change=abs(sum(c(i-9:i-5))-sum(c(i-4:i)))/sum(c(i-4:i));
end
disp([' It.: ' sprintf('%4i',i) ' Obj.: ' sprintf('%10.4f',c(i))...
' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ' ch.: ' sprintf('%6.3f',change )])

% PLOT DENSITIES
colormap(gray); imagesc(-x); axis equal; axis tight;
axis off;
pause(1e-6);

end

%%%%%%%%%% MESH-INDEPENDENCY FILTER%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dcf]=check(nelx,nely,rmin,dc)
dcf=zeros(nely,nelx);
for i = 1:nelx
   for j = 1:nely
      sum=0.0;
      for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
         for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
            fac = rmin-sqrt((i-k)^2+(j-l)^2);
            sum = sum+max(0,fac);
            dcf(j,i) = dcf(j,i) + max(0,fac)*dc(l,k);
         end
      end
      dcf(j,i) = dcf(j,i)/sum;
   end
end
end

%%%%%%%%%% OPTIMALITY CRITERIA UPDATE%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x]=ADDDEL(nelx,nely,volfra,dc,x)
l1 = min(min(dc)); l2 = max(max(dc));
while ((l2-l1)/l2 > 1.0e-5)
th = (l1+l2)/2.0;
x = max(0.001,sign(dc-th));
if sum(sum(x))-volfra*(nelx*nely) > 0;
l1 = th;
else
l2 = th;
```

```
end
end

%%%%%%%%%% ELEMENT STIFFNESS MATRIX%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk
E = 1. ;
E = 27085 * 10^9;
nu = 0.21;
k=[ 1/2-nu/6 1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
-1/4+nu/12 -1/8-nu/8 nu/6 1/8-3*nu/8];
KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
```

The function of Fe analysis contains the cases of both gravity and distributed load. In order to achieve stability by using both at the same time it has been noticed that: the mesh, must not be coarse, penalization should not be greater than 3, volume fraction and sensitivity radius should be revised wisely in respect of the mesh quality.

```
%%%%%%% FE-ANALYSIS WITH GRAVITATIONAL LOADS $$$%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal, width, height)
[KE] = lk;
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1));
F = zeros(2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end
% % DEFINE GRAVITY LOAD
d = 2240; % concrete density
g = 10; % gravitational accelaration
th = 0.25; % geometry's thickness
Vel = width*height*th/(nelx*nely); % elemental volume
Fg=d*g*Vel; % elemental gravitational load

% % NODES DEFINITION
W=[1 (nely+1) (nely+1)*(nelx+1)-nely (nely+1)*(nelx+1)];
w1=[2:nely];
w2=[(nely+2):(nely+1):(nely+1)*(nelx)];
w3=[(2*(nely+1)):(nely+1):(nely+1)*(nelx)];
```

```
w4=[((nely+1)*(nelx)+2):((nely+1)*(nelx+1)-1)];
w5=[1:(nely+1)*(nelx+1)];

W1=union(w1,w2);
W2=union(w3,w4);
W3=union(W1,W2);
W4=setdiff(w5,W3);
W5=setdiff(W4,W);

% % GRAVITATIONAL LOAD ASSIGNMENT OVER THE NODES
F(2*W')=-Fg/4; % corner nodes
F(2*W3')=-Fg/2; % border nodes
F(2*W5')=-Fg; % internal nodes

% % SUPPORT DEFINITION FOR THE ARCH PROBLEM
% sup = [(nely+1) (nely+1)*(nelx+1)];
% fixeddofs=[2*sup-1  2*sup ];
% fixeddofs = sort(fixeddofs);

% % DEFINE DISTRUBUTED LOAD || SUPPORT NODES || NODES WITH LOADS
% Fd=35*10^3*width/(nelx+1); % 30 KN/m
% supd=[(nely+1):(nely+1):(nely+1)*(nelx+1)];
% loadd = [1 : (nely+1) : ((nely+1)*(nelx)+1)];
% fixeddofs=[2*supd-1  2*supd];
% fixeddofs = sort(fixeddofs);
F(2*loadd,1)= -Fd;


alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);

% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;

end
```

A2. Python 3 translation code of BESO MATLAB algorithm.

This code is developed mainly to be used in Rhino/ Grasshopper, is an implicit translation of the MATLAB code. Hence there is space to more efficient in computational time terms.

```
##### Code for calling  BESO algorithm #####
##### Developed by Dimitris Gonidakis #####
#######################################
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

[c, x, i] = beso(30, 30, 0.01, 3., 3., 0.45)
print (‘Objective function’, c , ‘number of iterations’, i)

plt.figure()
   plt.imshow(-x, cmap="gray", extent=[0, 80, 0, 80])
   plt.show()
```

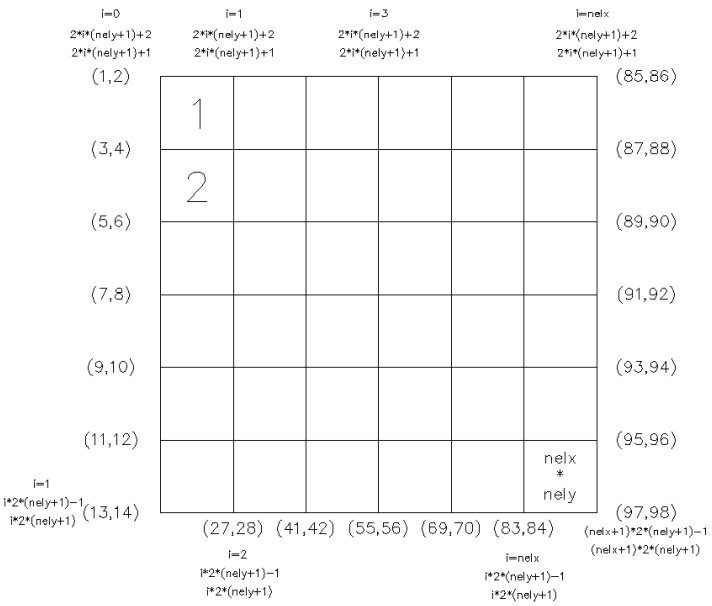
```
#####      BESO algorithm      #####
#############################
from IPython.display import display
from element_stif import lk
import numpy
from FE import FE
from check import check
from OC import add_del

def beso (nelx, nely, er, rmin, penal, volfrac):
    F = numpy.zeros(shape = (2 * (nelx + 1) * (nely + 1), 1), dtype = numpy.float64)
    F [2*(nely+1)*(nelx+1)-nely-1,0] = -1
    x = numpy.ones(shape = (nely, nelx) )
    dc = numpy.zeros(shape = (nely, nelx) )
    olddc = numpy.zeros(shape = (nely, nelx) )
    c = numpy.zeros(200)
    A = numpy.zeros(8, dtype = "int8")
    Ue = numpy.zeros(8)
    U = numpy.zeros(shape = (2 * (nelx + 1) * (nely + 1), 1) )
    vol = 1.0
    i = -1
    change = 1.0
    KE = lk()
    while change > 0.001:
        i = i + 1
        vol = max(vol*(1-er), volfrac)
        if i > 0 :
            olddc = dc
        # FE-Analysis
        U = FE (nelx, nely, x, penal, KE, F)
        # Objective function and sensitivity Analysis
        for ely in range(1+nely):
            for elx in range(1+nelx):
                n1 = (nely + 1) * (elx - 1) + ely
                n2 = (nely + 1) * elx + ely
                A = [2*n1-2, 2*n1-1, 2*n2-2, 2*n2-1, 2*n2, 2*n2+1, 2*n1, 2*n1+1]
                for j in range(8):
                    Ue [j] = U[A[j]]
                Uet = numpy.transpose(Ue)
                if x[ely-1, elx-1] == 1:
                    dc [ely-1, elx-1] = 0.5  * numpy.dot(Ue, numpy.dot(KE, Uet))
                    c [i] += 0.5 * numpy.dot(Ue, numpy.dot(KE, Uet))
                else:
                    dc [ely-1, elx-1] = 0
        # Filtering Sensitivities
        dc = check (nelx, nely, rmin, dc)
        # Stabilization of the evolutionary process
        if i > 0:
            dc = (dc + olddc)/2

        # Update Design
        x = add_del (nelx, nely, vol, dc, x)
```

```
        if i > 10:
            change = abs((sum(c[i-9 : i-5])-sum(c[i-4 : i])))/sum(c[i-4 : i])

        display(i, vol, c[i])

    return [c[i] x, i]


#### FE- Analysis ####
import numpy
import math

def FE (nelx, nely, x, penal, KE, F):
    K = numpy.zeros(shape = (2 * (nelx + 1) * (nely + 1), 2 * (nelx + 1) * (nely + 1)), dtype =
numpy.float64)
    U = numpy.zeros(shape = (2 * (nelx + 1) * (nely + 1), 1), dtype='float64')
    edof = numpy.zeros(8, dtype = "int8")

    for elx in range(1, nelx+1):
        for ely in range(1, nely+1):
            n1 = (nely + 1) * (elx - 1) + ely
            n2 = (nely + 1) * elx + ely
            edof = [2*n1-2, 2*n1-1, 2*n2-2, 2*n2-1, 2*n2, 2*n2+1, 2*n1, 2*n1+1]
            A = (x[ely-1, elx-1]**penal)*KE
            for i in range (8):

                for j in range (8):
                    K[edof[j], edof[i]] += A[j,i]

    # define loads and supports
    fixeddofs = numpy.arange(2*(nely+1),dtype='int')
    alldofs = numpy.arange(2*(nely+1)*(nelx+1), dtype='int')
    freedofs = numpy.setdiff1d(alldofs, fixeddofs)
    kf = numpy.zeros(shape = (numpy.size(freedofs),numpy.size(freedofs)), dtype = numpy.float64)
    for i in range(numpy.size(freedofs)):
        for j in range(numpy.size(freedofs)):
            kf[i, j] += K[freedofs[i],freedofs[j]]

    Ff = numpy.zeros(shape = (numpy.size(freedofs),1), dtype = numpy.float64)
    for i in range(numpy.size(freedofs)):
        Ff[i,0] += F[freedofs[i],0]
    # solving
    af = numpy.linalg.solve(kf, Ff)
    #af = kf/Ff
    for i in range(numpy.size(freedofs)):
        U[freedofs[i]] += af[i]

    return U

#####  Mesh- Independency Filter  #####
##################################
import numpy
import math
import time
```

```
def check (nelx, nely, rmin, dc):

   dcf=numpy.zeros(shape = (nely, nelx), dtype='float32')

   for i in range(1, nelx+1):
      for j in range(1,  nely+1):
         sum = 0.
         for k in range(max(i - math.floor(rmin), 1), min(i + 1 + math.floor(rmin),nelx)+1):
            for l in range(max(j - math.floor(rmin), 1), min(j + 1 + math.floor(rmin),nely)+1):
               fac = rmin - math.sqrt((i-k)**2 + (j-l)**2)
               sum += max(0, fac)
               dcf [j-1, i-1] += max(0, fac) * dc [l-1, k-1]

         dcf [j-1, i-1] = dcf [j-1, i-1] / sum
      return dcf

#####  Optimality Criteria Update  #####
##################################
import numpy

def add_del (nelx, nely, volfrac, dc, x):

   l1 = numpy.min(dc)
   l2 = numpy.max(dc)

   while (l2-l1)/l2 > 10**(-4):
      th = (l1 + l2) / 2
      # for i in range (nelx):
      #     for j in range (nely):
      #        x[j,i] = max (0.001, numpy.sign(dc[j,i] - th))
      x = numpy.maximum(0.001 * numpy.ones(numpy.shape(x)), numpy.sign(dc - th))

      if (numpy.sum(x) - volfrac * (nelx * nely)) > 0:

         l1 = th

      else:

         l2 = th
   return x
#####   Element's stiffness matrix   #####
##################################
import numpy

def lk():
   E = 1.    # eleasticity number
   nu = 0.3  # poisson nummber
   k = [1/2 - nu/6, 1/8 + nu/8, -1/4 - nu/12, -1/8 + 3*nu/8, -1/4 + nu/12,
       -1/8 - nu/8, nu/6, 1/8-3*nu/8]
   k = numpy.array([[k[0], k[1], k[2], k[3], k[4], k[5], k[6], k[7]],\
                    [k[1], k[0], k[7], k[6], k[5], k[4], k[3], k[2]],\
                    [k[2], k[7], k[0], k[5], k[6], k[3], k[4], k[1]],\
                    [k[3], k[6], k[5], k[0], k[7], k[2], k[1], k[4]],\
```

```
            [k[4], k[5], k[6], k[7], k[0], k[1], k[2], k[3]],\
            [k[5], k[4], k[3], k[2], k[1], k[0], k[7], k[6]],\
            [k[6], k[3], k[4], k[1], k[2], k[7], k[0], k[5]],\
            [k[7], k[2], k[1], k[4], k[3], k[6], k[5], k[0]]], dtype='float64')

    KE = (E/(1-nu**2)) * k
    return KE
```

A3. Elemental and Nodal enumeration Key for both MATLAB and Python code.



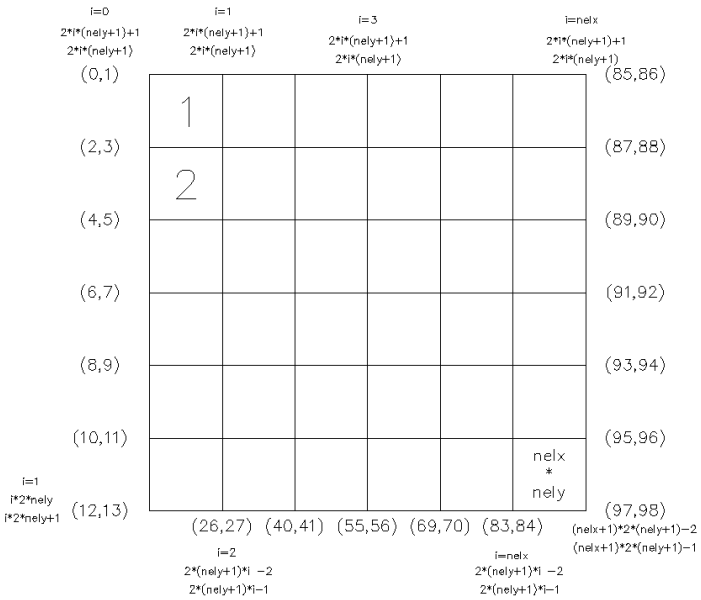


Figure 34
key of mesh reading in BESO algorithm

B1. Daylight Coefficients code.

```
####### Code for calling performing #######
#### daylight analysis with the method #####
######### of daylight coefficients #########
##### Developed by Dimitris Gonidakis #####
#######################################

# this function receives the design variable matrix of BESO and reads it as solid/window pattern
from coordinates import coordinates
from dome import patches
from Li import Li
from S import S
from R1 import R1
from R2 import R2
from R3 import R3r, R3
from collinear import collinearity
from reflectance import reflectance
import numpy
import math
import time
from sun import sun

def illuminance(x):
   # location and time
   # data for Zografou
   lona = '23.782604'# longitude string
   lata = '37.977806' # latitude string
   eleva = 184 # elevation number
   date = '2020/03/05 11:00:00' # string with the form 'year/month/day  hour:min:sec'

   # rooms attributes
   width = 3 # in meters
   depth = 3 # in meters
   height = 3 # in meters
   elx = 0.1 # in meters
   ely = 0.1 # in meters
   elz = 0.1 # in meters
   refw = 0.14 # window's reflectance
   refs = 0.8  # surface's reflectance
   Tw =  0.74

   #sky component
   Rd = 1000 # sky dome radius

   [g_s, a_s] = sun (lona, lata, eleva, date) # altittude and azimuth
   [xf, yf, zf, k1, x, y, z, s_v, x_v, y_v, z_v] = coordinates(x, elx, ely, elz, width, depth, height)
   [xd, yd, zd, bc, th ,ph] = patches (Rd) # band altittude, patch alltittude and azimuth
   Ls = Li(g_s, a_s, th, ph)
   Si = S(Rd)
   R_1 = R1()
   el = elx
   a = len(x)
   col_bol = collinearity (xd, yd, zd, xf, yf, zf, k1, x_v, y_v, z_v, el, height, a, x)
   R_2 = R2 (x, y, z, k1, width, depth, height, th, ph, Tw, col_bol, a)
```

```
    r_f = reflectance(s_v, refw, refs, k1, a)
    R_3 = R3(k1, r_f, x, y, z, elx, ely, elz, depth, width, height)
    C1 = numpy.dot(Si , Ls)
    C2 = numpy.dot(R_1 , C1)
    C3 = numpy.dot(R_2 , C2)
    E = numpy.dot(R_3, C3)

    return E

# R1 matrix in unobstructed environment
import numpy

def R1 ():
    R = numpy.identity(145)
    return R


# R2 matrix
import numpy
import math

def R2 (x, y, z, k, width, depth, height, th, ph, Tw, col_bol, a):
    #print('R2')
    # th alltitude of the sky element
    # ph azimuth

    R2 = numpy.zeros (shape = (k, 145))
    pi = math.pi

    for i in range (k):
        for j in range (145):

            if z[i] == 0 :
                R2 [i, j] = math.sin(th[j]) * Tw
            else:
                if y[i] == width/2:
                    R2 [i, j] = (math.cos(th[j]) * math.cos(ph[j]-pi) * Tw)
                elif y[i] == -width/2:
                    R2 [i, j] = (math.cos(th[j]) * math.cos(ph[j]-2*pi) * Tw)
                if x[i] == -depth/2:
                    R2 [i, j] = 0#(math.cos(th[j]) * math.cos(ph[j]-pi) * Tw)

                elif x[i] == depth/2:
                    R2 [i, j] = (math.cos(th[j]) * math.cos(ph[j]-pi/2) * Tw)
                if y[i] == -width/2:
                    R2 [i, j] = (math.cos(th[j]) * math.cos(ph[j]-pi) * Tw)

    R2 = R2 * col_bol
    RR = numpy.savetxt('R2_250.csv', R2)

    return R2
```

```
# R3 matrix functions, the first one calculates and stoles the form-factor matrix while the second one
# reads a stored one in order to save time

import math
import numpy
from formfactorcheck import formfactorcheck_a

def R3w (m, ro, x, y, z, elx, ely, elz, depth, width, height):
   #print ('R3')

   # x, y, z are the non-corrected coordinates
   # ro is a list which has the elements reflectance factor
   F = numpy.ones(shape = (m, m))
   ll = numpy.zeros(shape = (m, m), dtype = 'float64')

   a = numpy.zeros(2)
   b = numpy.zeros(2)
   c = numpy.zeros(2)
   # ch = numpy.zeros(shape = m)
   x = x + 10**(-4)
   y = y + 10**(-4)

   n = 0
   for i in range(n, m):
      #print(i)
      for j in range(n, m):
         if i == j:
            F[i,j] = 1
         else:
            a [0] = x [i]
            a [1] = x [j]
            b [0] = y [i]
            b [1] = y [j]
            c [0] = z [i]
            c [1] = z [j]
            f_f = (elx*ely)/(2*width*depth + 4*width*height) #formfactorcheck_a(a, b, c, elx, ely, elz,
depth, width, height)
            ll [i, j] = f_f
            ll [j, i] = ll [i, j]
            F [i, j] = -ro[i] * f_f
            F [j, i] = F [i, j]

      n += 1
   ll = numpy.savetxt('ll_0_075m.csv', ll)

   R3_3 = numpy.linalg.inv(F)

   return R3_3


def R3r (m, ro, x, y, z, elx, ely, elz, depth, width, height):

   F = numpy.ones(shape = (m, m))
   ll = numpy.genfromtxt('ll_0_1m333.csv',delimiter=' ')
```

```
F = (-ll.T * ro).T
for i in range(m):
   F [i, i] = 1

R3_3 = numpy.linalg.inv(F)

return R3_3
```

B2. Sky Condition

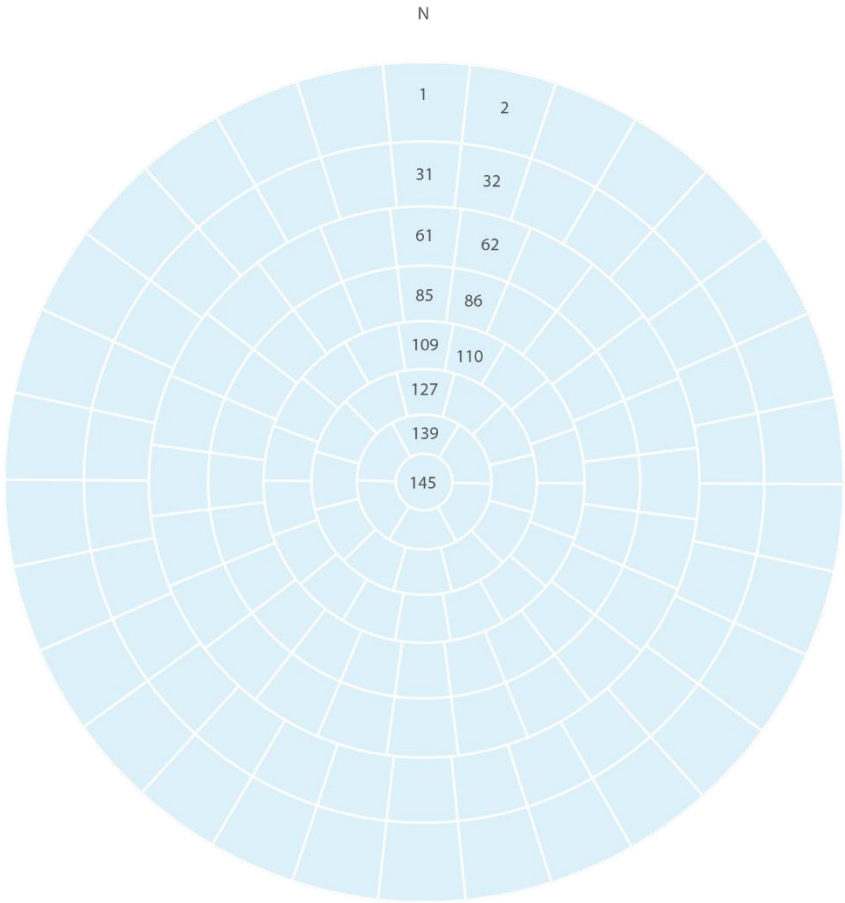


Figure 35
Sky Vault division

In this appendix the schematic sky vault division, CIE standard skies table and codes for computing standard skies, sky's geometry vault and sun's position: (Tergenza 2004, S. Darula, R. Kittler 2002)

```
# sky division scheme Tregenza 145 patches
# using the geometry attributes of sphere
# and having the inputs by Tergenza 2004
# the import data is the radius of sky dome
# which is at least 60 times greater than our project's radius

import math
import numpy

def patches (R):
```

```
    pi = math.pi

    x = numpy.zeros (shape = 145)
    y = numpy.zeros (shape = 145)
    z = numpy.zeros (shape = 145)
    ph = numpy.zeros (shape = 145) #azimuth
    th = numpy.zeros (shape = 145)

    bp = int(8) # number of bands
    np1 = [30, 30, 24, 24, 18, 12, 6, 1]
    bc = numpy.zeros (shape = bp)

    bc [0] = (6 * pi) / 180

    for i in range (1, bp):
        bc[i] = bc[i-1] + (12 * pi) / 180

    np = numpy.array(np1)

    k = 0 # counter

    for i in range(bp):
        a = int(np[i])
        z [k] = math.sin(bc[i]) * R
        st = (2*pi) / a
        y [k] = math.cos(bc[i]) * R
        rb = y [k] # radius band's centers
        x [k] = 0
        ang = 0
        for j in range(a):
            ang += st
            k += 1
            if (k <= 144):
                z [k] = z [k-1]
                x [k] = math.sin(ang) * rb
                y [k] = math.cos(ang) * rb
                th [k] = bc [i]
                ph [k] = ang

    x [144] = 0
    y [144] = 0


    return [x, y, z, bc, th ,ph]


# the procedure is described on Tregenza 2004 and Darula, Kitler 2002

import math
import numpy
from type import type

def Li (g_s, a_s, gp, ap):
    # gp = th ap = ph from dome function
```

```
    # g_s and g_s from the sun
    # a b c d e are constants that define the sky conditions
    h = 4
    [a, b, c, d, e] = type(h)
    Ls = numpy.zeros(shape = (145, 1))

    cos = math.cos
    sin = math.sin
    pi =  math.pi
    exp = math.exp

    zs = pi/2 -  g_s

    fi0 = 1 + a * exp(b)#/ cos(0))
    fzs = 1 + (c*exp(d * g_s) - exp(d * (pi/2))) + e * (cos(g_s))**2

    for i in range(145):
        z = pi/2 -  gp[i]
        x = numpy.arccos(cos(zs) * cos(z) + sin(zs) * sin(z) * cos(abs(ap[i] - a_s)))

        if (z >= 0 and z < (pi / 2)):
            fiz = 1 + a * exp(b / cos(z))
        else:
            fiz = 1

        fx = 1 + ( c*exp(d * x) - exp(d * (pi/2))) + e * (cos(x))**2

        La = fx * fiz
        Lz = fzs * fi0

        Ls [i, 0] = La / Lz

    return Ls

# a definition which provides the code with different set of a, b, c, d, e variables according to the CIE
# standard sky
def type(h):
    if h == 1:
        a, b, c, d, e = 4.0, -0.7, 0, -1.0, 0
    elif h == 2:
        a, b, c, d, e = 4.0, -0.7, 2, -1.0, 0.15
    elif h == 3:
        a, b, c, d, e = 1.1, -0.8, 0, -1.0, 0
    elif h == 4:
        a, b, c, d, e = 1.1, -0.8, 2, -1.0, 0.15
    elif h == 5:
        a, b, c, d, e = 0, -1.0, 0, -1.0, 0
    elif h == 6:
        a, b, c, d, e = 0, -1.0, 2, -1.0, 0.15
    elif h == 7:
        a, b, c, d, e = 0, -1.0, 5, -1.0, 0.3
    elif h == 8:
        a, b, c, d, e = 0, -1.0, 10, -1.0, 0.45
    elif h == 9:
```

```
    a, b, c, d, e = -1.0, -0.55, 2, -1.5, 0.15
  elif h == 10:
    a, b, c, d, e = -1.0, -0.55, 5, -2.5, 0.30
  elif h == 11:
    a, b, c, d, e = -1.0, -0.55, 10, -3.0, 0.45
  elif h == 12:
    a, b, c, d, e = -1.0, -0.32, 10, -3.0, 0.45
  elif h == 13:
    a, b, c, d, e = -1.0, -0.32, 16, -3.0, 0.30

  return [a, b, c, d, e]
```

Type	a	b	c	d	e	Description
1	4.0	-0.70	0	-1.0	0.00	CIE Standard Overcast Sky, alternative form Steep luminance gradation towards zenith, azimuthal uniformity
2	4.0	-0.70	2	-1.5	0.15	Overcast, with steep luminance gradation and slight brightening towards the sun
3	1.1	-0.80	0	-1.0	0.00	Overcast, moderately graded with azimuthal uniformity
4	1.1	-0.80	2	-1.5	0.15	Overcast, moderately graded and slight brightening towards the sun
5	0	-1.00	0	-1.0	0.00	Sky of uniform luminance
6	0	-1.00	2	-1.5	0.15	Partly cloudy sky, no gradation towards zenith, slight brightening towards the sun
7	0	-1.00	5	-2.5	0.30	Partly cloudy sky, no gradation towards zenith, brighter circumsolar region
8	0	-1.00	10	-3.0	0.45	Partly cloudy sky, no gradation towards zenith, distinct solar corona
9	-1	-0.55	2	-1.5	0.15	Partly cloudy, with the obscured sun
10	-1	-0.55	5	-2.5	0.30	Partly cloudy, with brighter circumsolar region
11	-1	-0.55	10	-3.0	0.45	White-blue sky with distinct solar corona
12	-1	-0.32	10	-3.0	0.45	CIE Standard Clear Sky, low illuminance turbidity
13	-1	-0.32	16	-3.0	0.30	CIE Standard Clear Sky, polluted atmosphere
14	-1	-0.15	16	-3.0	0.30	Cloudless turbid sky with broad solar corona
15	-1	-015	24	-2.8	0.15	White-blue turbid sky with broad solar corona

Figure 36
CIE Standard Skies indices

```
# pathces angular area

import math
import numpy

def S (R):

  # pi = math.pi
  #
  # A = 2 * pi * R**2
  # Aa = A / R**2
  # Si = Aa / 145
  S = numpy.zeros(shape = (145,145))
  for i in range (145):
    if i < 30:
      S[i,i] = 0.0435
    elif i >= 30 and i < 60:
```

```
            S[i,i] = 0.0416
        elif i >= 60 and i < 84:
            S[i,i] = 0.0474
        elif i >= 84 and i < 108:
            S[i,i] = 0.0407
        elif i >= 108 and i < 126:
            S[i,i] = 0.0429
        elif i >= 126 and i < 138:
            S[i,i] = 0.0445
        elif i >= 138 and i < 144:
            S[i,i] = 0.0455
        else:
            S[i,i] = 0.0344
    # tergenza says that divides the sky dome on equal pathces that means
    # that every patch has the same area

    return S

# By using the ephem library calculates the sun's potition angles in respect
# an observers potition
import ephem
import math

def sun (lo, la, ele, d):

    A = ephem.Observer()
    A.lon = lo
    A.lat = la
    A.elevation = ele
    A.date = d
    pi = math.pi


    s_un = ephem.Sun(A)

    g_s = float('%.10f' % (s_un.alt))
    a_s = float('%.10f' % (s_un.az))

    print (g_s*180/pi, a_s*180/pi)

    return [g_s, a_s]
```

B3. Form Factors

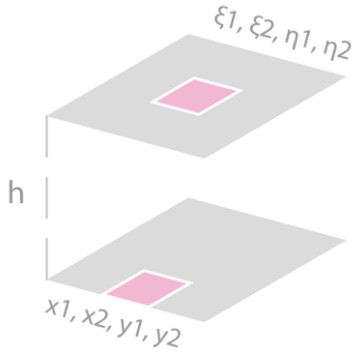
Case 1 Elements on Parallel planes

$$F_{12} = \frac{1}{2\pi A}\sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2}\sum_{l=1}^{2}\left[(-1)^{i+j+k+l}\, G(x_i, y_j, \eta_k, \xi_l,)\right]$$

$$A = (x_2 - x_1)(y_2 - y_1)$$

$$G = vp * \arctan\frac{v}{p} + uq * \arctan\frac{u}{q} - \frac{z^2}{2}\ln\left(u^2 + v^2 + z^2\right)$$

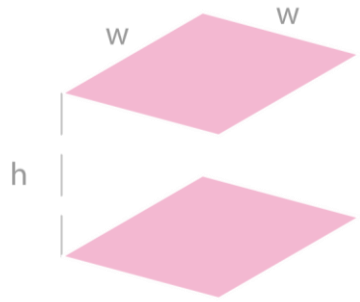
$$u = x - \xi, \qquad v = y - \eta, \qquad p = \sqrt{u^2 + z^2}, \qquad q = \sqrt{v^2 + z^2}$$



Case 2 Parallel Elements sharing shame axes

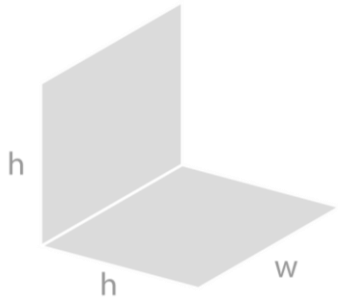
$$F_{12} = \frac{2}{\pi XY}\left[\ln\left[\frac{(1+X^2)(1+Y^2)}{1+X^2+Y^2}\right]^{-\frac{1}{2}} + X\sqrt{1+Y^2}\tan^{-1}\frac{X}{\sqrt{1+Y^2}} + Y\sqrt{1+X^2}\tan^{-1}\frac{Y}{\sqrt{1+X^2}} - Y\tan^{-1}Y\right]$$

$$X = Y = \frac{w}{h}$$



Case 3 Perpendicular equal Elements Sharing one edge

$$F_{12} = \frac{1}{4} + \frac{1}{\pi}\left[\arctan 1 - \sqrt{2}\arctan\frac{1}{\sqrt{2}} - \frac{1}{4}\ln\frac{4}{3}\right]$$



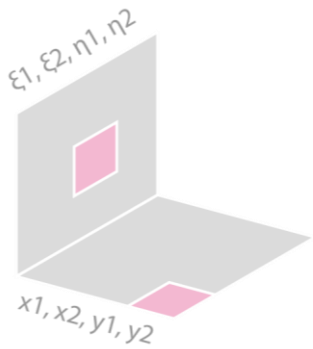
Case 4 Perpendicular Elements

$$F_{12} = \frac{1}{2\pi A}\sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2}\sum_{l=1}^{2}\left[(-1)^{i+j+k+l}\, G(x_i, y_j, \eta_k, \xi_l,)\right]$$

$$A = (x_2 - x_1)(y_2 - y_1)$$

$$G = (y - \eta)\mathrm{Carctan}D - \frac{C^2}{4}(1 - D^2)\ln\left(C^2(1 + D^2)\right)$$

$$D = \frac{y - \eta}{C^2}, \qquad C = \sqrt{x^2 + \xi^2}$$



In radiative heat transfer, a form factor is the proportion of the radiation which leaves a surface A that strikes surface a B. In a complex 'scene' there can be any number of different objects, which can be divided in turn into even more surfaces and surface segments. form factors are also sometimes known as configuration factors, view factors, angle factors or shape factors. The analytical types for form factor (view factor) calculation has been retrieved from the online library (Howell 2010). There are four cases counted in the form factor function. The following function is computing form factors over these four cases:

```
### Analytical calculation of form factors of identical rectangle shape elements

import math
from G import Gpar
from G import Gper

def par_opp_formf(a, b, c): # Case 1

  x = a / c
  y = b / c

  pi =  math.pi

  f1 = math.log(math.sqrt(((1 + x**2) * (1 + y**2))/(1 + x**2 + y**2)))
  f2 = x * math.sqrt(1 + y**2) * math.atan(x / math.sqrt(1 + y**2))
  f3 = y * math.sqrt(1 + x**2) * math.atan(y / math.sqrt(1 + x**2))
  f4 = x * math.atan(x)
  f5 = y * math.atan(y)

  f_0 = (2 / (pi * x * y)) * (f1 + f2 + f3 - f4 - f5)

  return f_0


def per_ax_formf(h , w, l): # Case 3
  # this is the case of two perpendicular elements of the different size

  pi =  math.pi

  H = h / l
  W = w / l

  k1 = ((1 + W**2) * (1 + H**2)) / (1 + W**2 + H**2)
  k2 = (W**2 * (1 + W**2 + H**2)) / ((1 + W**2) * (W**2 + H**2))
  k3 = (H**2 * (1 + W**2 + H**2)) / ((1 + H**2) * (W**2 + H**2))

  f1 = W * math.atan(1 / W)
  f2 = H * math.atan(1 / H)
  f3 = math.sqrt(H**2 + W**2) * math.atan(math.sqrt(1 / (H**2 + W**2)))
  f4 = (1 / 4) * math.log(k1 * (k2**(W**2)) * (k3**(H**2)))

  f_0 = (1 / (W * pi)) * (f1 + f2 - f3 + f4)
```

```
    return f_0


def par_formf(x, y, z, ks, et, elx, ely): # Case 2

    for l in range(1,3):
        for k in range(1,3):
            for j in range(1,3):
                for i in range(1,3):

                    Go = Gpar(x[i-1], y[j-1], et[k-1], ks[l-1], z)
                    Go = Go * ((-1)**(i+j+k+l))
                    Gsum = Gsum + Go

    f_0 = Gsum / (elx * ely)

    return f_0


def per_formf(x, y, ks, et, elx, ely): # Case 4
    # from element with x, y coordinates to element with ksi eta

    Gsum = 0

    for l in range(1,3):
        for k in range(1,3):
            for j in range(1,3):
                for i in range(1,3):

                    Go = Gper(x[i-1], y[j-1], et[k-1], ks[l-1])

                    Go = Go * ((-1)**(i+j+k+l))
                    Gsum = Gsum + Go
                    #print (Gsum)
    f_0 = Gsum / (elx * ely)

    return f_0

# G function Calculation for parallel and perpendicular elements
import math

def Gpar (x, y, ks, et, z):

    pi = math.pi

    A = (y - et) * math.sqrt((x - ks)**2 + z**2)
    B = (y - et) / math.sqrt((x - ks)**2 + z**2)
    C = (x - ks) * math.sqrt((y - et)**2 + z**2)
    D = (x - ks) / math.sqrt((y - et)**2 + z**2)
    E = (z**2 / 1) * math.log((x - ks)**2 + (y - et)**2 + z**2)

    G = (1 / (2 * pi)) * (A * math.atan(B) + C * math.atan(D) - E)

    return G
```

```
def Gper (x, y, ksi, et):

    pi = math.pi

    K = (y - et) / math.sqrt(x**2 + ksi**2)

    A = (y - et) * math.sqrt(x**2 + ksi**2) * math.atan(K)
    B = (1/4) * (x**2 + ksi**2) * (1 - K**2)
    C = math.log((x**2 + ksi**2) * (1 + K**2))

    G = (A - B * C) / (2 * pi)

    return G

#### this set of functions resets the coordinates in order to ####
####  parallel or perpendicular elements' form factors will  ####
####                    be calculated                        ####
####                mapping.py    script                     ####
import numpy
def mapping_par(x, y, z, d, w, h, depth, width, height, elx, ely, elz):

    a = numpy.zeros(2)
    b = numpy.zeros(2)
    m = numpy.zeros(2)
    n = numpy.zeros(2)
    global dista

    if d == depth :
        a [0] = z [0] - elz / 2
        a [1] = z [0] + elz / 2
        b [0] = y [0] - ely / 2
        b [1] = y [0] + ely / 2
        m [0] = z [1] - elz / 2
        m [1] = z [1] + elz / 2
        n [0] = y [1] - ely / 2
        n [1] = y [1] + ely / 2
        dista = d

    elif w == width:
        a [0] = x [0] - elx / 2 # x1
        a [1] = x [0] + elx / 2 # x2
        b [0] = z [0] - elz / 2
        b [1] = z [0] + elz / 2
        m [0] = x [1] - elx / 2
        m [1] = x [1] + elx / 2
        n [0] = z [1] - elz / 2
        n [1] = z [1] + elz / 2

        dista = w

    elif h == height:
        a [0] = x [0] - elx / 2
        a [1] = x [0] + elx / 2
```

```
        b [0] = y [0] - ely / 2
        b [1] = y [0] + ely / 2
        m [0] = x [1] - elx / 2
        m [1] = x [1] + elx / 2
        n [0] = y [1] - ely / 2
        n [1] = y [1] + ely / 2
        dista = h

    return [a, b, dista, m, n]

def mapping_per(x, y, z, d, w, h, depth, width, height, elx, ely, elz):

    a = numpy.zeros(2)
    b = numpy.zeros(2)
    m = numpy.zeros(2)
    n = numpy.zeros(2)

    if (z [0] == 0 or z [0] == height) :
        if (x [1] == 0 or x [1] == depth):
            a [0] = x [0] - elx / 2
            a [1] = x [0] + elx / 2
            b [0] = y [0] - ely / 2
            b [1] = y [0] + ely / 2
            m [0] = z [1] - elz / 2
            m [1] = z [1] + elz / 2
            n [0] = y [1] - ely / 2
            n [1] = y [1] + ely / 2

        else:
            a [0] = y [0] - ely / 2
            a [1] = y [0] + ely / 2
            b [0] = x [0] - elx / 2
            b [1] = x [0] + elx / 2
            m [0] = z [1] - elz / 2
            m [1] = z [1] + elz / 2
            n [0] = x [1] - elx / 2
            n [1] = x [1] + elx / 2

    elif (x [0] == 0 or x [0] == depth):
        if (z [1] == 0 or z [1] == height):
            a [0] = z [0] - elz / 2
            a [1] = z [0] + elz / 2
            b [0] = x [0] - elx / 2
            b [1] = x [0] + elx / 2
            m [0] = y [1] - ely / 2
            m [1] = y [1] + ely / 2
            n [0] = x [1] - elx / 2
            n [1] = x [1] + elx / 2

        else:
            a [0] = x [0] - elx / 2
            a [1] = x [0] + elx / 2
            b [0] = z [0] - elz / 2
            b [1] = z [0] + elz / 2
```

```
            m [0] = y [1] - ely / 2
            m [1] = y [1] + ely / 2
            n [0] = z [1] - elz / 2
            n [1] = z [1] + elz / 2

        else: #(y [0] == 0 or y [0] == width):
          if (z [1] == 0 or z [1] == height):
            a [0] = z [0] - elz / 2
            a [1] = z [0] + elz / 2
            b [0] = y [0] - ely / 2
            b [1] = y [0] + ely / 2
            m [0] = x [1] - elx / 2
            m [1] = x [1] + elx / 2
            n [0] = y [1] - ely / 2
            n [1] = y [1] + ely / 2

          else:
            a [0] = y [0] - ely / 2
            a [1] = y [0] + ely / 2
            b [0] = z [0] - elz / 2
            b [1] = z [0] + elz / 2
            m [0] = x [1] - elx / 2
            m [1] = x [1] + elx / 2
            n [0] = z [1] - elz / 2
            n [1] = z [1] + elz / 2

        return [a, b, m, n]

      #### This function performs analytical computation ####
      ####      of form factors over a simple geometry      ####
      ####          developed by Dimitrios Gonidakis          ####
      #### improvements may needed for more accuracy ####

      import math
      from formfactors import par_opp_formf
      from formfactors import per_ax_formf
      from formfactors import par_formf
      from formfactors import per_formf
      from mapping import mapping_par
      from mapping import mapping_per

      def formfactorcheck (x, y, z, elx, ely, elz, depth, width, height):
        # we take as data the xf, yf, zf which are our room tranfered on the positive
        # grids and so then the absolute valiues are takemn

        d = abs(x[0] - x[1])
        w = abs(y[0] - y[1])
        h = abs(z[0] - z[1])
        h1 = abs(z[0] - z[1])
        h2 = abs(z[0] - z[1])

        global ffa

        if  (x[0] == 10**(-4) and x[1] ==10**(-4)) or (x[0] == depth+10**(-4) and x[1] == depth+10**(-4)) or \
```

```
    (y[0] == 10**(-4) and y[1] == 10**(-4)) or (y[0] == width+10**(-4) and y[1] == width+10**(-4)) or \
    (z[0] == 0 and z[1] == 0) or (z[0] == height and z[1] == height) :
    ffa = 0

else:

    if (d == depth or w == width or h == height) :
        if ((x[0] == x[1]) and (y[0] == y[1])) :

            a = elx
            b = ely
            c = h
            ffa = par_opp_formf(a, b, c)

        elif ((z[0] == z[1]) and (y[0] == y[1])) :
            a = elz
            b = ely
            c = d
            ffa = par_opp_formf(a, b, c)

        elif ((z[0] == z[1]) and (x[0] == x[1])) :
            a = elx
            b = elz
            c = w
            ffa = par_opp_formf(a, b, c)

        else:
            [a, b, dist1, m, n] =  mapping_par(x, y, z, d, w, h, depth, width, height, elx, ely, elz)
            ffa =  par_formf(a, b, dist1, m, n, elx, ely)

    elif x[0] == x[1] and ((w == h) and w == elx/2) :
        h = elz
        w = elx
        l = ely
        ffa = per_ax_formf(h , w, l)

    elif y[0] == y[1] and ((d == h) and d == elx/2) :
        h = elz
        w = ely
        l = elx
        ffa = per_ax_formf(h , w, l)

    elif z[0] == z[1] and ((w == d) and d == elx/2):
        h = ely
        w = elx
        l = elz
        ffa = per_ax_formf(h , w, l)

    else:
        [a, b, m, n] = mapping_per(x, y, z, d, w, h, depth, width, height, elx, ely, elz)
        ffa =  per_formf(a, b, m, n, elx, ely)

formf = ffa
return formf
```

B4 Complimentary functions of Daylight Coefficients

```
# this function examines if three points are collinear and returns a boolean
# list which can activate or deactivate the elements of the R2 matrix with
# direct componetns

import math
import numpy
import time
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

def collinearity (xd, yd, zd, xe, ye, ze, o, xv, yv, zv, el, height, a, xx):
    #print ('collinear')
    #start_time = time.time()

    # xd yd zd are the coordinates of sky patches
    # xe ye ze are the coordinates of room elements
    # xv yv zv are the coordinates of voids of the facade
    ll = 0
    col_bol = numpy.zeros (shape =  (o, 145))

    for i in range (145):
        #print (i)
        for k in range (o):

            for j in range (a):
                if xx [j] == 0 and col_bol[k, i] != 1:

                    if xe[k] == xv[j] and ye[k] == yv[j] and ze[k] == zv[j] :
                        col_bol [k, i] = 0
                    else:
                        x1 = xv [j] - xe [k]
                        y1 = yv [j] - ye [k]
                        z1 = zv [j] - ze [k]

                        x2 = xd [i] - xe [k]
                        y2 = yd [i] - ye [k]
                        z2 = zd [i] - ze [k]

                        if ((x2 != 0) and (y2 != 0) and (z2 != 0)) :
                            a1 = x1 / x2
                            a2 = y1 / y2
                            a3 = z1 / z2
                            if (a1 == a2 and a2 == a3):
                                col_bol [k, i] = 1
                                ll += 1
                            elif (abs(a1-a2)<el/10 and abs(a2-a3)<el/10 and abs(a1-a3) < el/10):
                                col_bol [k, i] = 1
                                ll += 1
                            else:
                                col_bol [k, i] = 0
                        elif ((x1 != 0) and (y1 != 0) and (z1 != 0)):
                            a1 = x2 / x1
                            a2 = y2 / y1
```

```
                    a3 = z2 / z1
                    if (a1 == a2 and a2 == a3):
                        col_bol [k, i] = 1
                        ll += 1
                    elif (abs(a1-a2)<el/10 and abs(a2-a3)<el/10 and abs(a1-a3) < el/10):
                        col_bol [k, i] = 1
                        ll += 1
                    else:
                        col_bol [k, i] = 0
                else:
                    col_bol [k, i] = 0

    print('collinear', ll, 'from', 145*o)

    # plt.figure(1)
    # ax = plt.axes()
    # plt.imshow(-col_bol, cmap="gray", extent=[0, 145, 0, o])
    # plt.show()
    #col_bol = numpy.where()
    #kkk= numpy.savetxt('col_bol.csv', col_bol)
    #print("--- %s seconds ---" % (time.time() - start_time))
    return col_bol

# this function takes void_check output and corresponds to each element
# plaster or glass reflectance

import numpy

def reflectance(s_v, refw, refs, k1, a):
    #print ('reflectance')

    r_f =  numpy.full (k1, refs)

    for i in range(a):
        if s_v[i] == 0:
            r_f [i] = refw

    return r_f


# a function which divides a room which consists of rectangular wall
# into small equal elements and returns their coordinations

import numpy
import matplotlib.pyplot as plt

def coordinates(xt, elx, ely, elz, width, depth, height):

    #print ('coordinates')

    w = int(width / ely)
    d = int(depth / elx)
    h = int(height / elz)
    k1 = 2 * (w * h) + 2 * (d * h) + 2 * (w * d)
```

```
k1 = int(k1)

# xf, yf, zf ara the positive values of coordinates to make it easier for
#form factor relations
xf = numpy.zeros (shape = k1)
yf = numpy.zeros (shape = k1)
zf = numpy.zeros (shape = k1)

# x, y, z are the correction of coordination in order the center of the floor
# to coincide with 0,0

x = numpy.zeros (shape = k1)
y = numpy.zeros (shape = k1)
z = numpy.zeros (shape = k1)

s_v = numpy.ones(shape = len(xt)) # is the updated solid/void for the construction
x_v = numpy.zeros(shape = len(xt))
y_v = numpy.zeros(shape = len(xt))
z_v = numpy.zeros(shape = len(xt))

k = 0
ii = 0

# defining the coordinates of the element's center for the window wall

for i in range (1, h + 1):
   for j in range (1, w + 1):
      xf[k] = depth
      yf[k] = ely*j - ely/2
      zf[k] = elz*i - elz/2
      if xt [k] < 0.5:
         s_v[k] = 0
         x_v[k] = xf[k]
         y_v[k] = yf[k]
         z_v[k] = zf[k]
         #kk +=1
      else:
         z_v[i] = int(-200)
         x_v[i] = int(-200)
         y_v[i] = int(-200)
      k += 1

#2 defining the coordinates of the element's center for the first side wall
for i in range (1, h + 1):
   for j in range (1, d + 1):
      yf[k] = 0
      xf[k] = elx*j - elx/2
      zf[k] = elz*i - elz/2
      k += 1

#3 defining the coordinates of the element's center for the back wall
for i in range (1, h + 1):
   for j in range (1, w + 1):
      xf[k] = 0
```

```
        yf[k] = ely*j - ely/2
        zf[k] = elz*i - elz/2
        k += 1

#4 defining the coordinates of the element's center for the second side wall
for i in range (1, h + 1):
    for j in range (1, d + 1):
        yf[k] = 0
        xf[k] = elx*j - elx/2
        zf[k] = elz*i - elz/2
        k += 1

#5 defining the coordinates of the element's center for the ceiling
for i in range (1, d + 1):
    for j in range (1, w + 1):
        zf[k] = height
        xf[k] = elx*i - elx/2
        yf[k] = ely*j - ely/2
        k += 1

#6 defining the coordinates of the element's center for the floor
for i in range (1, d + 1):
    for j in range (1, w + 1):
        zf[k] = 0
        xf[k] = elx*i - elx/2
        yf[k] = ely*j - ely/2
        k += 1
# k2 = k
# for i in range (1, d + 1):
#     for j in range (1, w + 1):
#         zf[k] = 0.5
#         xf[k] = elx*i - elx/2
#         yf[k] = ely*j - ely/2
#         k += 1

for i in range(k): # room correction to centered
    x[i] = xf[i] - depth/2
    y[i] = yf[i] - width/2

z = zf

plt.figure(1)
ax = plt.axes(projection='3d')
ax.scatter3D(x_v, y_v, z_v)


return [xf, yf, zf, k1, x, y, z, s_v, x_v, y_v, z_v]
```

||References||

Alshaibani, K.A. *PREDICTION OF INTERIOR DAYLIGHT UNDER CLEAR SKY CONDITIONS.* PhD Dissertation, 1996.

Aremu, A., I. Ashcroft, R. Hague, R. Wildman, and C. Tuck. "Suitability of SIMP and BESO Topology Optimization Algorithms for Additive Manufacture." 2010.

Bendsøe, M.P. "Optimal shape design as a material distribution problem." *Struct. Optim. 1*, 1989: 193-202.

Bendsoe, M.P., and M. Kikuchi. "Generating Optimal topologies in structural design using a homogenization method." *Computer Methods in Applied Mechanics and Engineering 71*, 1988: 197-224.

Bendsoe, M.P., and O. Sigmund. *Topology Optimization: Theory Methods and Applications.* Berlin: Springer, 2003.

Chwng, K., and N. Olhoff. "An investigation concerning optimal design of solid elastic plates." *International Journal of Solids and Structures 17(3)*, 1981: 305-323.

CIE. " Natural daylight, Official recommendation." *Compte Rendu, CIE 13th Session, Committee E-3.2, vol. II, parts 3–2, II–IV&35–37*, 1955.

—. "Standardization of luminance distribution on clear skies." *Pub. CIE No.22, TC-4.2*, 1973.

Darula, S., and R. Kittler. "CIE GENERAL SKY STANDARD DEFINING LUMINANCE DISTRIBUTIONS." 2002.

Fuller, M. *Fuller, M. 1985. Concepts and Practice of Architectural Daylighting.* New York: Van Nostrand Reinhold Company, 1985.

Futrell, B.J., E.C. Ozelkanb, and D. Brentrupc. "Optimizing complex building design for annual daylighting performance and evaluation of optimization algorithms." *Energy and Buildings 92*, 2015: 234-245.

Gherri, B. *Assessment of Daylight Performance in Buildings.* WITPRESS, 2015.

Heschong, L., M. Saxena, S. Wayland, and T. Perry. *DAYLIGHT METRICS: PIER Daylighting Plus Research Program.* 2012.

Hopkinson, R.G., P. Petherbridge, and J. Longmore. *Daylighting.* London: Heinemann, 1966.

Howell, J.R. *A Catalog of Radiation Tranfer Configuration Factors.* 2010. https://web.engr.uky.edu/rtl/Catalog/.

Huang, X., and Y.M. Xie. "Convergent and mesh-independent solutions for bi-directional evolutionary structural optimization method." *. Finite Elements in Analysis and Design 43(14)*, 2007: 10391049.

—. "Evolutionary topology optimization of continuum structures including design-dependent self-weight loads." *Finite Elements in Analysis and Design, Volume 47, Issue 8*, August 2011: 942-948.

Iversen, A., et al. "Daylight calculations in practice: An investigation of the ability of nine daylight simulation programs to calculate the daylight factor in five typical rooms." 2013.

Kimball, H.H, and I.F. Hand. "Sky brightness and daylight illumination measurements." *Monthly Weather Rev., 49, 9*, 1921: 481-488.

Kimball, H.H. "Daylight illumination on horizontal, vertical and sloping surfaces." *Transactions of IES (N.Y) 18, 5*, 1923: 434-474.

—. "Daylight illumination on horizontal, vertical and sloping surfaces." *Transactions of IES (N.Y) 18, 5*, 1923: 434-474.

Kimball, H.H., and I.F. Hand. "Daylight illumination on horizontal, vertical and sloping surfaces." *Monthly Weather Rev., 50, 12*, 1922: 615-628.

Kittler, R. "Standardization of outdoor conditions for the calculation of daylight factor with clear skies. In: Proceedings of the CIE International Conference on Sunlight in Buildings." *Bouwcentrum International, Rotterdam*, 1967: 273–285.

Kohn, R., and G. Strang. " Optimal design and relaxation of variational problems (part i)." *Commun Pure Applied Math 39(1)*, 1986: 113-137, 139-182, 353-377.

Kota, S., and J.S. Haberl. "Historical Survey of Daylighting calculation method and their use in energy." *Energy Systems Laboratory*, 2009.

Mardaljevic, J. "Daylight simulation: Validation, Sky models, and daylight coefficients." *Ph.D. thesis, De Montfort University, Leicester, UK*, 2000.

Marler, R.T., and J.S. Arora. "Survey of multi-objective optimization methods for engineering." *Structural and Multidisciplinary Optimization 26(6)*, 2004: 369-395.

McDermott, L.H., and G.W. Gordon-Smith. "Daylight illumination recorded at Teddington." *Proc. Build. Res. Congr., Division 3, Part III*, 1951: 156.

Michell, A.G.M. "The limits of economy material in frame structures." *Lvii: The London, Edinburg, and Dublin philosophical Magazine and Journal of Science 8 (6)*, 1904: 589-597.

Moon, P., and D.E. Spencer. "Illumination from a nonuniform sky." *Illum. Eng. (N.Y.), 37*, 1942: 707-726.

Nabil, A., and J. Mardaljevic. "Useful daylight illuminance: a new paradigm for assessing daylight in buildings." *Lighting Research and Technology*, 2005: 37(1): p. 41–57.

Neufert, E. *Οικοδομική & Αρχιτεκτονικη Σύνθεση.* Μ. Γκιούρδας, 2007.

Osher, S., and J. Sethian. "Fronts Propagating with Curvature- Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations." *JOURNAL OF COMPUTATIONAL PHYSICS 79*, 1988: 12-49.

Perez, R., R. Seals, and j. Michalsky. "All-weather model for sky luminance distribution—preliminary configuration and validation." *Solar Energy 50 (3)*, 1993: 235–245.

Pokrovsky, G.I. "On optical studies of humus in soils." *Soviet Soil Science, 1-2*, 1929: 124-130.

Prager, W., and G.I.N Rozvany. "Optimal Layout of Grillages†." *Journal of Structural Mechanics*, 1977.

Querin, O.M, G.P. Steven, and Y.M. Xie. "Evolutionary structural optimisation (ESO) using a bidirectional algorithm." *Engineering Computations 15(8)*, 1998: 1031–1048.

Radman, A. *Bi-directional evolutionary structural optimization (BESO) for topology optimization of material's microstructure.* Diss RMIT Univesity, 2013.

Reinhart, C., and S. Herkel. "The simulation of annual daylight illuminance distributions- a stateof-the-art comparison of six RADIANCE-based methods." *Energy and Buildings 32*, 2000: 167-187.

Rietz, A. "Sufficiency of a finite exponent in SIMP (power law) methods." *Struct. Multidisc. Optim. 21:*, 2001: 159–63.

Rogers, Z. "Daylighting Metric Development Using Daylight Autonomy Calculations In the Sensor Placement Optimization Tool. Boulder, Colorado, USA." *Architectural Energy Corporation*, 2006.

Rozvany, G.I.N. "Optimal load transmission by flexure." *Computer Methods in Applied Mechanics and Engineering 1(3)*, 1972: 253-263.

—. "Grillages of maximum strength and maximum stiffness." *International Journal of Mechanical Sciences 14(10)*, 1972: 651-666.

—. "A critical review of established methods of structural topology optimization." *Struct Multidisc Optim 37*, 2009: 217-237.

Tregenza, P. "Subdivision of the sky hemisphere for luminance measurements." *Lighting Research and Technology 19(1)*, 1987: 13-14.

—. "Analysing sky luminance scans to obtain frequency distributions of CIE Standard General Skies." *Lighting Research and Technology 36, 4*, 2004: 271-281.

Tregenza, P., and M. Wilson. "Daylighting: Architecture and lighting design." *Routledge Taylor & Francis Group*, 2011: 182.

Tregenza, P.R., and I.M. Waters. "Daylight Coefficients." *Lighting Res. Technology*, 1983: 15(2) 65-71.

Tzempelikos, A. *A METHODOLOGY FOR DETAILED CALCULATION OF ILLUMINAICE LEVELS AND LIGHT DIMMING FACTORS IN A ROOLM WITH MOTORIZED BLINDS LNTEGRATED LN*

AN ADVAXCED WINDOW. Master of Applied Science, Concordia University, Montreal, Quebec, Canada , 2001.

Walsh, J.W.T. "The early years of Illuminating engineering in Great Britain ii ." *Transactions of the Illuminating Engineering Society 15(3)*, 1951: 49-60.

Ward, G., and F. Rubinstein. "A new technique for computer simulation of illuminated spaces." *Lawrence Berkeley Laboratory Report 23042*, 1988.

Xia, L., Q.·Xia, X. Huang, and Y.M. Xie. "Bi-directional Evolutionary Structural Optimization on Advanced Structures and Materials: A Comprehensive Review." *Arch Comput Methods Eng* , 2016.

Xie, Y.M., and G.P. Steven. " A simple evolutionary procedure for structural optimization." *Computers & Structures 49*, 1993: 885-896.

Zhou, M., and G.I.N Rozvany. "The COC algorithm. Part 2: topology, geometry and generalized shape optimization." *Comp. Meth. Appl. Mech. Eng. 89*, 1991: 197-224.

||*Glossary||

Split Flux: The split flux formula is a simple algorithm derived from a manual calculation method established by BRE (The English Building Research Establishment). This method is based on the principle that the global illumination at a certain point in a room is the result of three distinctive components of daylight:

1. the direct sky component (SC)
2. the reflections from exterior surfaces (ERC)
3. the reflections from internal surfaces (IRC).

Each component is calculated separately and then added up to obtain the global illumination in each sensor point. The internally reflected component is determined by an equation using the average reflectance of interior surfaces, the total glazing area and a correction factor for the external obstruction. Given these approximations, this method is likely to overestimate or underestimate the amount of daylight. It is only recommended to use this method for spaces in which the window openings are parallel to the walls. (Iversen, et al. 2013)

Daylight: Daylight is the combination of all direct and indirect sunlight during the daytime.

Data tree: A data Tree is a hierarchical structure for storing data in nested lists. Data trees are created when a grasshopper component is structured to take in a data set and output multiple sets of data. Grasshopper handles this new data by nesting it in the form of sub-lists. These nested sub-lists work in the same way as folder structures on your computer in that accessing indexed items require moving through paths that are informed by their generation of parent lists and their own sub-index.

Brep: In solid modeling and computer-aided design, boundary representation—often abbreviated as B-rep or BREP—is a method for representing shapes using the limits.