

# **A Novel Generative Design System for Ceiling Lighting Layouts**

ابتكار نظام تصميم توليدي جديد لمخططات إضاءة الأسقف الداخلية

by

**SARA J M AIHAMARNA**

**Dissertation submitted in fulfilment  
of the requirements for the degree of  
MSc SUSTAINABLE DESIGN OF THE BUILT  
ENVIRONMENT**

at

**The British University in Dubai**

**August 2020**

## **DECLARATION**

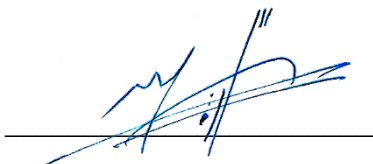
I warrant that the content of this research is the direct result of my own work and that any use made in it of published or unpublished copyright material falls within the limits permitted by international copyright conventions.

I understand that a copy of my research will be deposited in the University Library for permanent retention.

I hereby agree that the material mentioned above for which I am author and copyright holder may be copied and distributed by The British University in Dubai for the purposes of research, private study or education and that The British University in Dubai may recover from purchasers the costs incurred in such copying and distribution, where appropriate.

I understand that The British University in Dubai may make a digital copy available in the institutional repository.

I understand that I may apply to the University to retain the right to withhold or to restrict access to my thesis for a period which shall not normally exceed four calendar years from the congregation at which the degree is conferred, the length of the period to be specified in the application, together with the precise reasons for making that application.



Signature of the student



## **COPYRIGHT AND INFORMATION TO USERS**

The author whose copyright is declared on the title page of the work has granted to the British University in Dubai the right to lend his/her research work to users of its library and to make partial or single copies for educational and research use.

The author has also granted permission to the University to keep or make a digital copy for similar use and for the purpose of preservation of the work digitally.

Multiple copying of this work for scholarly purposes may be granted by either the author, the Registrar or the Dean only.

Copying for financial gain shall only be allowed with the author's express permission.

Any use of this work in whole or in part shall respect the moral rights of the author to be acknowledged and to reflect in good faith and without detriment the meaning of the content, and the original authorship.

## **Abstract**

This dissertation proposes a novel approach to a generative system that produces reflected ceiling layouts in commercial spaces. Indoor lighting is a key element in interior design, it plays an important role on spaces and occupants. The current lighting design process mostly depends on human's perception and individual experience, performed manually based on 2D CAD drawings, which can be time consuming, especially in proposing alternatives in the early phase of design.

Hence, this paper suggests a novel design approach that generates reflected ceiling layouts. Presenting an algorithmic system scripted Dynamo in the form of a BIM plug-in. The system employs various design approaches through transferring selected design concepts into mathematical scripts in Dynamo software, resulting in a production of two-dimensional lighting layouts in Dynamo.

The structure had followed a proposed sustainable framework for generative systems. The system has three key approaches: (a) splitting the surface into smaller surfaces, (b) forming boundaries around the surface, (c) creating two types of grids. And various sub-approaches developed based on those three key approaches: (a) creating lighting patterns, (b) boundary lighting, (c) drop down ceilings. Every sub-approach is divided into different categories as well. The default status of the generative system combines all the several approaches and sub-approaches into a single system that produces more than 500 different reflected ceiling layouts in a matter of minutes.

In this paper, the system was assessed and validated on three-phases, as Phase 01 tested it on Conceptual and Actual case-studies. Results showcased 147 different ceiling designs for conceptual case-studies, and 218 generated designs for the actual case-studies. Exhibiting visible results that validated the system's flexibility, diversity, innovative designs, and providing various novel design tools in the generative system. Phase 02 validated the system's comprehensiveness through connecting two geometrical results - ceiling (a) and ceiling (b) - in Dynamo to lighting families in Autodesk Revit. Lighting calculations achieved proper outcomes that were suitable to each selected design. Sustainability was targeted through benefit-cost analyses established by conducting a cross-sectional study on a sample of 72 interior designers and architects and comparing the outcomes to the performed case-studies. Results had reduced 99% of the design cost. As time was decreased around 98%, with an increase of 96% in the generated design quantity.

## ملخص

تستعرض هذه الرسالة تقنية جديدة تبتكر تصميمات هندسية مختلفة لإضاءة الأسقف في المكاتب، حيث تعتبر الإضاءة الداخلية عنصراً أساسياً في التصميم الداخلي، وتلعب دوراً مهماً بالنسبة للمساحات والأفراد. إن التصميم الحالي للإضاءة يعتمد على نظرة المصمم الشخصية و خبرته العملية، حيث يقوم برسم التصميم يدوياً مستخدماً البرامج الحاسوبية المختلفة ، و هذا بدوره يستغرق الكثير من الوقت خاصة في مراحل التصميم الأولى.

تقدم هذه الرسالة تقنية مبتكرة عبارة عن نظام توليدي لتصميم إضاءة الأسقف، من خلال ابتكار نظام خوارزمي تمت كتابته باستخدام برنامج داينامو (Dynamo) و يقوم النظام التوليدي بتحويل أساليب التصميم المختلفة إلى معادلات رياضية و نصوص منطقية تؤدي إلى إنتاج تصاميم إضاءة متعددة و مختلفة.

وقد اعتمد الهيكل الأساسي لهذا النظام على طريقة منهجية تطبق إطار العمل المستدام المقترح لهذه النوعية من النظم التوليدية. يحتوي النظام على ثلاث طرق رئيسية لتحديد شكل السقف: (أ) تقسيم السطح إلى أسطح أصغر، (ب) تشكيل حدود حول السطح، (ج) إنشاء نوعين من شبكات الإضاءة، وتم تطوير مناهج فرعية مختلفة بناءً على هذه الأساليب الثلاثة الرئيسية: (أ) تصميم أشكال إضاءة مختلفة ، (ب) تحديد مواضع الإضاءة الحدودية للسقف (ج) تصميم أشكال هندسية للسقف. و تقسيم كل نهج فرعي إلى أفرع مختلفة. ثم يجمع النظام التوليدي بين جميع الطرق والأساليب الفرعية المتعددة لإنتاج أكثر من خمسمئة تصميم مختلف لإضاءة الأسقف في غضون دقائق.

في هذه الرسالة تم تقييم النظام والتحقق من صحته على ثلاث مراحل، حيث اختبرت المرحلة الأولى النظام من خلال تطبيقه على نماذج نظرية و أخرى واقعية. أظهرت النتائج 147 تصميمًا مختلفًا للنماذج النظرية، و 218 تصميمًا تم ابتكاره لنماذج الحالات الواقعية، كما أثبتت مرونة النظام وتنوعه وتصميماته المبتكرة وتوفيره لأدوات تصميم جديدة متنوعة.

أثبتت المرحلة الثانية من التقييم صحة شمولية النظام من خلال اختيار تصميمين منتجين في داينامو (Dynamo) - السقف (أ) والسقف (ب) - و ربطهما مع مجموعات الإضاءة في أوتوديسك ريفت (Autodesk Revit) و تنفيذ حسابات الإضاءة لهما ، و كانت النتائج الحسابية للإضاءة ملائمة لعدد و نوعية المصابيح الموجودة في كل تصميم.

تم استهداف الاستدامة من خلال إجراء تحليل التكلفة والمنفعة ، و ذلك عبر تطبيق استبيان على عينة من 72 مصممًا داخليًا ومعماريًا ومقارنة نتائج الاستبيان بنتائج المرحلة الثانية ( الدراسات النظرية والواقعية ) حيث حققت النتائج تخفيضًا بنسبة 99% من تكلفة التصميم، مع توفير الوقت بنحو 98%، و زيادة كمية التصميم المنتجة بنسبه 96%.

*To My Parents, Asmahan & Jamil*

## **Acknowledgment**

First and foremost, I would like to convey my sincere gratitude to my professor and mentor, Dr. Riad Saraiji for his guidance throughout this dissertation, his unlimited support, patience and advise, along with his unlimited encouragement and faith in me as a student. I am truly fortunate to have been given the opportunity to deliver this topic with him as my supervisor.

I would like to express my deepest appreciated for the faculty of Engineering & IT at the British University in Dubai, Prof. Bassam Abu-Hijleh and Dr. Hanan Taleb for their amazing education skills and continuous support throughout the period of my studies.

It is with the most profound love I express my gratitude to my parents. They are the reason for all my accomplishments. Their support and patience are key pillars that made this work possible and I am forever indebted to them.

Lastly, I would like to thank my sisters and brothers for enduring me during this dissertation and for their endless emotional and practical support.

## List of Contents

DECLARATION.....	
COPYRIGHT AND INFORMATION TO USERS.....	
ABSTRACT.....	
ملخص.....	
DEDICATION.....	
ACKNOWLEDGMENT.....	
LIST OF CONTENT.....	i
LIST OF FIGURES.....	vii
LIST OF TABLES.....	xvii
<b>1 Chapter 1: Introduction.....</b>	<b>2</b>
1.1 Background .....	2
1.2 Problem statement.....	4
1.3 Goals and Objectives .....	4
<b>2 Chapter 2: Literature review.....</b>	<b>8</b>
2.1 Introduction.....	8
2.2 Literature review .....	9
2.2.1 Generative design overview.....	9
2.2.2 The evolution of algorithmic design in architecture.....	13
2.2.3 Algorithmic design sustainability approaches for commercial artificial lighting .....	18
2.2.3.1 Algorithmic design in optimizing light uniformity.....	19
2.2.3.2 Algorithmic design in optimizing indoor lighting control systems.....	21
2.2.4 Algorithmic design for generating designs in commercial spaces .....	23
2.2.5 Algorithmic design utilized in commercial ceiling designs.....	25
2.3 Identifying the literature gap .....	28
<b>3 Chapter 3: Methodology - The generative system outline.....</b>	<b>31</b>
3.1 Introduction.....	31

3.1.1	User-System interactions .....	34
<b>3.2</b>	<b>The generative system components.....</b>	<b>36</b>
3.2.1	Key Step: Importing a ceiling from Autodesk Revit to Dynamo.....	36
3.2.2	Subsystem 01: Three approaches to identify a surface.....	37
3.2.2.1	Approach 01: Splitting the surface .....	38
3.2.2.2	Approach 02: Forming boundaries .....	39
3.2.2.2.1	Boundaries sub-approach: Creating boundary lighting .....	40
3.2.2.2.2	Boundaries sub-approach: Creating drop down ceilings .....	42
3.2.2.3	Approach 03: Creating a grid .....	43
3.2.2.3.1	Stage 01: Creating a plug-in grid .....	43
3.2.2.3.2	Stage 02: Scripting a uniform/quad grid.....	44
3.2.2.3.3	Stage 03: Scripting a non-uniform/diamond grid.....	44
3.2.2.3.4	Stage 04: Updating the scripts .....	45
3.2.2.3.5	Grid sub-approach: Creating light patterns.....	47
3.2.3	Subsystem 02: Counting the generated light quantities.....	50
3.2.4	Subsystem 03: Exporting the results back to Autodesk Revit .....	51
<b>3.3</b>	<b>Arranging the generative system .....</b>	<b>51</b>
3.3.1	Arranging the results horizontally and vertically .....	51
3.3.1.1	Creating proper spacing between the generated ceilings.....	51
3.3.1.2	Arranging and placing the generated light patterns on the original ceiling.....	52
3.3.2	Coloring the surfaces and the produced lights .....	53
<b>4</b>	<b>Chapter 4: Development of the generative system.....</b>	<b>55</b>
<b>4.1</b>	<b>Introduction.....</b>	<b>55</b>
4.1.1	Software background - Dynamo's anatomy and previous implementations .....	56
4.1.2	Justification for selecting Dynamo-Revit as a programming software.....	59
<b>4.2</b>	<b>The Algorithms - description, development, and shortfalls .....</b>	<b>61</b>
4.2.1	Key Step: Importing the ceiling from Autodesk Revit to Dynamo.....	63
4.2.2	Subsystem 01: Three approaches to identify a ceiling .....	64

4.2.2.1	Approach 01: Splitting the surface .....	65
4.2.2.1.1	Calculating the angles' degrees of the surface .....	66
4.2.2.1.2	Investigating the proper splitting sequence.....	68
4.2.2.1.2.1	Creating a proper splitting sequence using permutation.....	68
4.2.2.1.2.2	Permutations shortfall and using combinations as a solution.....	70
4.2.2.1.2.3	Merging between permutations and combinations as an optimized approach .....	73
4.2.2.1.3	Applying a division cut to the surface.....	76
4.2.2.2	Approach 02: Forming boundaries .....	78
4.2.2.2.1	Creating a polygon line as boundaries.....	78
4.2.2.2.2	Choosing between splitting the surface or forming boundaries .....	81
4.2.2.2.3	Boundaries sub-approach: Creating Boundary lighting.....	83
4.2.2.2.3.1	Creating a script for full surfaces.....	83
4.2.2.2.3.2	Shortfalls of boundary lighting algorithm on split ceilings .....	85
4.2.2.2.3.3	Creating central line lighting for narrow ceilings.....	88
4.2.2.2.3.4	Boundary lighting controllers.....	89
4.2.2.2.4	Boundaries sub-approach: Creating drop down ceilings .....	90
4.2.2.2.4.1	Drop ceilings script.....	91
4.2.2.2.4.2	Drop ceiling controllers.....	92
4.2.2.3	Approach 03: Creating a grid .....	94
4.2.2.3.1	Creating a grid using a plug-in.....	94
4.2.2.3.2	Shortfalls and solutions in the plug-in grid.....	96
4.2.2.3.3	Scripting a new uniform/quad grid design.....	101
4.2.2.3.4	Scripting a new diamond/non-uniform grid design.....	106
4.2.2.3.5	Switching between the two types of the scripted grids.....	109
4.2.2.3.6	Updating the scripted grids upon testing it .....	110
4.2.2.3.7	The grid controllers.....	113
4.2.2.3.8	Grid sub-approach: Creating Light patterns .....	115
4.2.2.3.8.1	Uniform rectangular patterns.....	115
4.2.2.3.8.1.1	Key issues in creating uniform rectangular lighting patterns .....	117
4.2.2.3.8.1.2	Uniform rectangular patterns script explanation.....	119



4.2.2.3.8.1.3	The produced uniform rectangular patterns.....	123
4.2.2.3.8.2	Non-uniform rectangular light patterns.....	125
4.2.2.3.8.2.1	The scripted error that produced the non-uniform patterns.....	125
4.2.2.3.8.2.2	Non-uniform patterns script explanation.....	126
4.2.2.3.8.2.3	The produced non-uniform rectangular patterns.....	128
4.2.2.3.8.3	Uniform circular and polygon patterns.....	129
4.2.2.3.8.3.1	Uniform circular and polygon script explanation.....	130
4.2.2.3.8.3.2	The produced uniform circular and polygon patterns.....	132
4.2.2.3.8.4	Non-uniform circular and polygon patterns.....	134
4.2.2.3.8.4.1	Non-uniform circular and polygon script explanation.....	134
4.2.2.3.8.4.2	The produced non-uniform circular and polygon patterns.....	136
4.2.2.3.8.5	The light patterns controllers.....	138
4.2.3	Subsystem 02: Counting the generated light quantities.....	141
4.2.4	Subsystem 03: Exporting the results back to Autodesk Revit.....	143
4.2.5	Arranging the generative system.....	145
4.2.5.1	Arranging the results horizontally and vertically.....	145
4.2.5.1.1	Creating proper spacing between the generated ceilings.....	145
4.2.5.1.2	Arranging and placing the generated light patterns on the original ceiling.....	146
4.2.5.2	Coloring the surfaces and the produced lights.....	147
<b>5</b>	<b>Chapter 5: Results and discussion - Testing the generative system.....</b>	<b>151</b>
<b>5.1</b>	<b>Introduction.....</b>	<b>151</b>
<b>5.2</b>	<b>Phase 01: Case studies.....</b>	<b>156</b>
5.2.1	Conceptual case studies.....	156
5.2.1.1	Office 01: The green room - Utrecht, Netherland.....	156
5.2.1.1.1	Office 01 limitations and shortfalls.....	158
5.2.1.1.1.1	Unable to identify ceilings with openings.....	158
5.2.1.1.1.2	Limiting the generated results due to laptop capacity.....	159
5.2.1.1.1.3	Curved corners shortfall.....	161
5.2.1.1.2	Office 01 results.....	162

5.2.1.2	Office 02: Commerzbank headquarters - Frankfurt, Germany .....	170
5.2.1.2.1	Office 02 limitations and shortfalls .....	173
5.2.1.2.1.1	Producing unparalleled shapes with the surface outline .....	173
5.2.1.2.2	Office 2 results.....	175
5.2.1.3	Summary of the results from both conceptual case studies.....	183
5.2.2	Field case studies.....	184
5.2.2.1	Office A101 - Building 5 - Dubai Design District.....	184
5.2.2.1.1	Office A101 results .....	187
5.2.2.2	Office A202 - Building 7 - Dubai Design District.....	196
5.2.2.2.1	Office A101 results .....	199
5.2.3	Summary of both conceptual and field case studies.....	207
<b>5.3</b>	<b>Phase 02: Connecting the results to Autodesk Revit .....</b>	<b>209</b>
5.3.1	Exporting the lights from Dynamo and connecting it to Revit light families .....	209
5.3.2	Lighting calculations in Autodesk Revit .....	211
5.3.2.1	Lighting calculations for ceiling (a).....	213
5.3.2.2	Lighting calculations for ceiling (b) .....	218
5.3.3	Phase 02 out-turns .....	221
5.3.4	Lighting power density calculations.....	223
<b>5.4</b>	<b>Phase 03: Sustainability elements of the proposed generative design sysetm .....</b>	<b>224</b>
5.4.1	Proving the system's sustainability with-in itself.....	225
5.4.2	Proving the system's sustainability by utilizing results from a cross-sectional study...227	
5.4.2.1	Survey participants .....	227
5.4.2.2	Survey questions outline .....	229
5.4.2.3	Survey results .....	232
5.4.2.4	Time-Cost analysis .....	237
<b>6</b>	<b>Chapter 6: Conclusions.....</b>	<b>240</b>
<b>6.1</b>	<b>Introduction.....</b>	<b>240</b>
<b>6.2</b>	<b>Summary.....</b>	<b>240</b>
<b>6.3</b>	<b>Conclusions.....</b>	<b>244</b>

6.4	Research limitations .....	249
6.5	Further work .....	251
7	References .....	257

## List of Figures

Figure 1: Number of times generative design showed in literature from 1978 up to 2018(Caetano, Santos & Leitão 2020).....	13
Figure 2: a sequence of generated 3D models by the first generative project of Medieval Towns in Italy (Soddu 1989) .....	14
Figure 3: Different steps of geometric variations (Magna et al. 2013) .....	17
Figure 4: Optimization techniques found in the literature (Baloch et al. 2018) .....	19
Figure 5:Design metrics in the algorithmic system: (from left to right: adjacency preference, work style preference, buzz, productivity, daylight and views to outside). (Nagy et al. 2017) .....	23
Figure 6:Simulation results in an office using the created algorithm (Abdulbaki, Abdulbaqi & Mohialden 2018).....	24
Figure 7: From left to right: University of Iowa Concert Hall, Conga Room dance club, Resonant Chamber (Badino, Shtrepi & Astolfi 2020).....	25
Figure 8: BIM model of college of human ecology building and the lecture room along with the results (Kim et al. 2016) .....	27
Figure 9: The final algorithm with the scripts and variables sliders created in Dynamo (author) .....	31
Figure 10: Outline for the generative system discussed in this chapter (author) .....	33
Figure 11: All the interactive sliders in the generative system (author).....	35
Figure 12: The transition between Autodesk Revit to Dynamo (author) .....	36

Figure 13: Splitting script key steps and sub-steps (author) .....	38
Figure 14: The steps in forming boundaries approach (author) .....	39
Figure 15: Different ceiling layouts showing boundary lighting method generated in Dynamo (author) .....	40
Figure 16: The scripts outline for Creating Boundary Lighting approach (author) .....	41
Figure 17: The outline for creating drop down ceilings (author) .....	42
Figure 18: Light patterns outline (author) .....	47
Figure 19: The Code Blocks that show the counting in the generative system (author) .....	50
Figure 20: The arrangement of results in Dynamo for Splitting the ceiling design approach (author) .....	52
Figure 21: The different coloring Nodes for the different approaches (author) .....	53
Figure 22: Nodes and Scripts that are generated with sequences and ranges (Dynamo 2.0.3) .....	58
Figure 23: The Interaction level between different design Sub-approaches and their types (author) .....	61
Figure 24: Outline for the scripted design approaches and their functionalities (author) .....	62
Figure 25: Dynamo Nodes that collect one ceiling element from Revit and isolating the bottom horizontal face (author) .....	63
Figure 26: Selecting one ceiling in Revit-model to transfer to Dynamo (author) .....	64
Figure 27: The steps to calculate the angles of the surface (author) .....	66

Figure 28: The scripted Code Block for calculating the angles between consecutive edges in Dynamo (author).....	68
Figure 29: The scripted Code Block for combinations for arranging the splitting angles in Dynamo (author).....	70
Figure 30: Flow progress of creating the final list using combinations (author) .....	72
Figure 31: Merging combinations and permutations Nodes in Dynamo (author) .....	74
Figure 32: The definition script for splitting the surfaces in Dynamo (author).....	75
Figure 33: Splitting results on a ceiling that has four angles above 90 and the final script for splitting in Dynamo (author) .....	77
Figure 34: Testing a simple shape offset command in Dynamo (author).....	79
Figure 35: The surface after the polygon offset with the Nodes for it in Dynamo (author)....	80
Figure 36: Split controls sliders and script for turning on and off the split by the end user in Dynamo (author).....	81
Figure 37: Splitting the surface or not script in Dynamo (author) .....	82
Figure 38: Offset inwards and outwards explanation in boundary lighting (author) .....	83
Figure 39: The created script for running the algorithm on full surfaces in Dynamo (author)....	84
Figure 40: The issue that appeared when applying boundary lighting on split ceilings in Dynamo (author).....	86
Figure 41: Explanation for the algorithm approach to fix the discontinuity issue (author).....	87

Figure 42: The algorithmic script approach to creating central line lighting for narrow ceilings in Dynamo (author).....	88
Figure 43: Boundary lighting controllers in Dynamo (author).....	89
Figure 44: Drop down ceiling option with its controllers in Dynamo (author).....	90
Figure 45: The script for drop down ceilings and minor definitions for boundary lighting in Dynamo (author).....	91
Figure 46: Drop Ceiling Controllers created in Dynamo (author).....	92
Figure 47: Figure 23: Produced drop down ceiling in Dynamo (author) .....	93
Figure 48: First shortfall of the inserted grid in Dynamo (author).....	96
Figure 49: Second shortfall of the inserted grid in Dynamo (author) .....	97
Figure 50: The Quad Node connected to the intersecting Code Block in Dynamo (author)...	97
Figure 51: The final Plug-in Grid algorithm in Dynamo 2.0.3 (author).....	98
Figure 52: A Surface with the final grid algorithm with the patched offset turned on in Dynamo 2.0.3 (author) .....	99
Figure 53: A Surface with the final grid algorithm with the patched offset turned off in Dynamo 2.0.3 (author) .....	99
Figure 54: Different grid designs with the panels turned on in Dynamo 2.0.3 (author) .....	100
Figure 55: The first script for the uniformed grid in Dynamo 2.0.3 (author).....	101
Figure 56: The first sequence produced by uniformed grid on a ceiling in Dynamo 2.0.3 (author).....	103

Figure 57: Applying lights on the first sequence for uniform grid in Dynamo 2.0.3 (author)	104
Figure 58: The uniform grid after applying the reverse lists in Dynamo 2.0.3 (author)	105
Figure 59: Grid 2 – the diamond/non-uniform grid in Dynamo 2.0.3 (author)	106
Figure 60: The first script and grid controls for the non-uniformed grid in Dynamo 2.0.3 (author)	107
Figure 61: Swapping between Grid1 and Grid2 script and slider in Dynamo.2.0.3 (author)	109
Figure 62: The quad grid results after applying it on a split surface in Dynamo 2.0.3 (author)	110
Figure 63: The quad grid results after applying it on a split surface using the new approach in Dynamo 2.0.3 (author)	111
Figure 64: The final quad and diamond grids scripts in Dynamo 2.0.3 (author)	112
Figure 65: Grid Controllers Group in Dynamo 2.0.3 (author)	113
Figure 66: Ceiling with a grid points with Limit Grid-size of 500 in Dynamo 2.0.3 (author)	114
Figure 67: Ceiling with a grid points with Limit Grid-size of 500 in Dynamo 2.0.3 (author)	114
Figure 68: Simplified concept of creating rectangular uniform patterns (author)	115
Figure 69: The lights sizes input in the algorithm for creating patterns in Dynamo 2.0.3 (author)	116



Figure 70: The algorithm error due to the dimensions of the lights and proposed solution (author).....	117
Figure 71: Lights orientation issue in uniform rectangular lighting algorithm (author) .....	118
Figure 72: The steps to change the orientations of the lights (author) .....	118
Figure 73: Uniform rectangular patterns script in Dynamo 2.0.3 (author).....	120
Figure 74: Uniform rectangular patterns output generated in Dynamo 2.0.3 (author).....	123
Figure 75: Uniform rectangular patterns output generated in Dynamo 2.0.3 (author).....	124
Figure 76: The script error that resulted in creating non-uniform patterns in Dynamo 2.0.3 (author).....	125
Figure 77: Non-uniform pattern script in Dynamo 2.0.3 (author).....	127
Figure 78: Non-uniform rectangular pattern outputs generated in Dynamo 2.0.3 (author)...	128
Figure 79: The different shapes the polygon algorithm develops into .....	129
Figure 80: Uniform circular and polygon patterns script in Dynamo 2.0.3 (author) .....	131
Figure 81: Uniform circular patterns outputs generated in Dynamo 2.0.3 (author).....	132
Figure 82: Triangle and quad uniform patterns outputs generated in Dynamo 2.0.3 (author) .....	133
Figure 83: Pentagon, hexagon, and decagon uniform pattern outputs generated in Dynamo 2.0.3 (author).....	133
Figure 84: Non-uniform circular and polygon patterns script in Dynamo 2.0.3 (author) .....	135
Figure 85: Non-uniform circular patterns outputs generated in Dynamo 2.0.3 (author).....	136

Figure 86: Non-uniform hexagon patterns outputs generated in Dynamo 2.0.3 (author) .....	136
Figure 87: Non-uniform triangular and quad patterns outputs generated in Dynamo 2.0.3 (author).....	137
Figure 88: All the sliders for creating light patterns in Dynamo 2.0.3 (author) .....	138
Figure 89: Explanation of the three on/off sliders for the light pattern controllers in Dynamo 2.0.3 (author) .....	140
Figure 90: Boundary lights list count connected to a watch node to display the numbers in Dynamo 2.0.3 (author) .....	141
Figure 91: The light counts results and the scripted algorithm for the Pattern Rectangle Lights in Dynamo (author).....	142
Figure 92: The Script to export the results in Dynamo back to Revit (Dynamo 2.0.3).....	143
Figure 93: The grouping Nodes created in Dynamo 2.0.3 (author) .....	144
Figure 94: Close up shot showing the outlined selected ceiling compared to the other ceiling options in Dynamo (author) .....	145
Figure 95: Dynamo workspace after the arrangement of the splits results and the patterns (author).....	146
Figure 96: Changing the opacity and borders for the surfaces in Dynamo 2.0.3 (author) ....	147
Figure 97: Lights patterns color display on a surface and the connected Nodes in Dynamo 2.0.3 (author).....	148
Figure 98: Lights patterns color display on a surface in Dynamo 2.0.3 (author).....	148

Figure 99: Boundary lights color display on a surface and connected Nodes in Dynamo 2.0.3 (author).....	149
Figure 100:Grid lights color display on a surface and connected Nodes in Dynamo 2.0.3 (author).....	149
Figure 101:Inserted lighting sizes in the algorithm in Dynamo Nodes (author).....	151
Figure 102: The framework outline for Results and discussion Chapter - Testing the generative system (author) .....	155
Figure 103: The green room layout (Levy 2019).....	156
Figure 104: The green room layout after modeling in Autodesk Revit (author) .....	157
Figure 105: The modeled layout in Autodesk Revit and the selected ceiling to export to Dynamo (author).....	157
Figure 106: The error list that appeared from the ceiling shape in Dynamo 2.0.3 Nodes (author) .....	158
Figure 107: The inserted ceiling before and after patching from Dynamo 2.0.3 workspace (author).....	159
Figure 108: The algorithmic generated results on splitting the ceiling exhibited in Dynamo 2.0.3 (author) .....	160
Figure 109: Limiting results script created in Dynamo 2.0.3 (author).....	160
Figure 110: Curved corners shortfalls for Office 01 generated from the algorithmic system in Dynamo 2.0.3 (author).....	161

Figure 111: The diversity example of the produced layouts for Office 01 generated in Dynamo 2.0.3 (author) .....	165
Figure 112: Comparison between boundary lighting design output for Office 01 by changing variables in Dynamo (author).....	167
Figure 113: Different ceiling patterns for Office 01, generated solely by changing the pattern type in Dynamo 2.0.3 (author).....	168
Figure 114: Commerzbank Headquarters - Frankfurt, Germany (foster and partners 2018)	170
Figure 115: Commerzbank Headquarters layout and elevation (foster and partners 2018) ..	171
Figure 116: Commerzbank Headquarters modeled layout in Revit 2018 (author) .....	172
Figure 117: The imported ceiling in Dynamo for Office 02 (author) .....	172
Figure 118: Bad drop-down ceiling results for Office 2 generated in Dynamo 2.0.3 (author) .....	173
Figure 119: The modeled ceiling in Revit and the bounding box around the ceiling in Dynamo workspace (author).....	174
Figure 120: The applied grid using the boundary box on Revit axes in Dynamo 2.0.3 against the Revit model (author) .....	174
Figure 121: The development of the same pattern in Room 01 in Office 02 generated in Dynamo 2.0.3 (author).....	179
Figure 122: Generated results for Room 01 in Office 02 from Dynamo 2.0.3 (author).....	180
Figure 123: The development of different lighting layouts for Office 02 generated in Dynamo 2.0.3 (author) .....	181

Figure 124: Building 5 floor plan - The office A101 floor plan (Tecom Group) .....	184
Figure 125: Site images of Office A101 (author).....	185
Figure 126: Office A101 modeled in Autodesk Revit 2018 (author).....	186
Figure 127: Development of the produced patterns in Dynamo 2.0.3 for Office A101 (author) .....	191
Figure 128: Different generated ceilings using spotlights in Dynamo 2.0.3 for Office A101 (author).....	193
Figure 129: Generated mixed lighting design methods in Dynamo 2.0.3 for Office A101 (author).....	194
Figure 130: 24: Building 7 floor plan and the office A202 floor plan (Tecom Group) .....	196
Figure 131: Site images of Office A202 against its location in the layout (author).....	197
Figure 132: Office A202 modeled in Revit 2018 (author) .....	198
Figure 133: Same pattern merged with different pattern generated in Dynamo 2.0.3 for Office A202 (author) .....	204
Figure 134: Different types of designs using the same light generated in Dynamo 2.0.3 for Office A202 (author).....	205
Figure 135: Division / combination example for both directions generated in Dynamo 2.0.3 for Office A202 (author).....	206
Figure 136: Nodes that creates light lists to connect to Revit in Dynamo 2.0.3 (author).....	209
Figure 137: The chosen ceilings for the lighting calculations generated in Dynamo 2.0.3 (author).....	211

Figure 138: The counting for ceiling (a) and for ceiling (b) from Dynamo 2.0.3 (author) ...	212
Figure 139: ElumTools Plug-in in Revit Workspace (Autodesk Revit 2018) .....	213
Figure 140: Dynamo produced results for ceiling (a) and application of them in Revit workspace.....	213
Figure 141: The inserted lights and the Revit model with the layout model (Autodesk Revit 2018) .....	214
Figure 142: Adding Calculations Points (Autodesk Revit 2018).....	215
Figure 143: Dynamo produced results for ceiling (b) and application of them in Revit workspace.....	218
Figure 144: The light specifications for the produced ceiling (Autodesk Revit 2018) .....	219
Figure 145: Ceiling (a) and ceiling (b) comparison (author).....	222
Figure 146: The survey participants demographic (author).....	228
Figure 147: Question 6 results/ How many reflected ceiling layouts do you usually propose in a project? (author).....	232
Figure 148: Question 8 results /How many hours would you spend on creating a ceiling layout for a 7x7m meeting room? (author).....	233
Figure 149: Question 7 results / How many hours do you usually take to alter the ceiling's proposed designs after the client's feedback? (author).....	233
Figure 150: Question 9 results / Based on your experience; how many cumulative hours do you take in getting the counts of lights while doing a B.O.Q for large ceilings such, e.g.: theaters, dining halls, places of workshops, office towers, museums ... etc.? (author)	234

Figure 151: Human cost and the generated system cost comparison .....	238
---	-----

## List of Tables

Table 1: The development of the stages of the grid (author).....	46
Table 2: Sample of the generated patterns for uniform and non-uniform rectangular pattern (author).....	48
Table 3: Sample of the generated patterns for uniform and non-uniform polygon pattern (author).....	49
Table 4: The results of p1 and p2 combinations command (author).....	71
Table 5: The resulted grids using Lunch Box in Dynamo (author).....	95
Table 6: Pattern Type Slider Details (author) .....	139
Table 7: Office 01 generated ceilings designs results from the algorithmic system in Dynamo 2.0.3 (author) .....	162
Table 8: Office 01 limitations and advantages (author) .....	169
Table 9: Office 02 generated ceilings designs results from the algorithmic system in Dynamo 2.0.3 (author) .....	175
Table 10: Office 02 limitations and advantages (author) .....	182
Table 11: The summary of the results from the conceptual case studies (author).....	183
Table 12: Office A101 produced options from the generated ceilings designs in Dynamo 2.0.3 (author).....	187
Table 13: Office A101 limitations and advantages (author).....	195



Table 14: Office A202 produced options from the generated ceilings designs from Dynamo 2.0.3 (author) .....	199
Table 15: Office A101 and Office A202 limitations and advantages (author) .....	207
Table 16: The selected lights from Acuity Brands Lighting (author) .....	210
Table 17: Ceiling (a) Lighting Calculations Results (Autodesk Revit 2018) .....	216
Table 18: Ceiling (a) luminance Generated Results from Autodesk Revit 2018 (author) ....	217
Table 19: Ceiling (b) Lighting Calculations Results (Autodesk Revit 2018) .....	220
Table 20: Ceiling (b) luminance Generated Results from Autodesk Revit 2018 (author) ....	221
Table 21: Lighting fixtures details for Ceiling (a) and (b) and LPD calculations (author) ...	223
Table 22: ideality strategies proposed as a framework that fosters sustainability (Cohen & Reich 2017) .....	224
Table 23: The proposed sustainable framework with the algorithm's sustainable ability for each point (Cohen & Reich 2017) .....	225
Table 24: Comparison between the designer's ability and the generative system ability - Questions (6-9) .....	235
Table 25: Results for questions (10-12) .....	236

# Chapter 1

## Introduction

# **1 Chapter 1: Introduction**

## **1.1 Background**

This dissertation dwells between the fields of interior design, lighting design, sustainability, and computation. A method is proposed to incorporate generative systems in the primary phases for lighting layout design process. Designed from the relationships between primary interior design approaches and the mathematical tactics to transform design concepts into algorithmic generative approaches. Throughout this dissertation, various design criteria were investigated to construct the most suitable approaches that generates a variety of options, while considering flexibility and user-machine interaction as key design pillar. The most important contribution for this dissertation is the method it proposes, with various specifications and different experiments that were developed and tested to validate it.

Genetic Algorithms acted as a search engine for this Generative System. Genetic algorithms were chosen as the engine because of their ability to perform continuous random and organized mathematical for-loops and while-loops to deliver high quantity of diverse designs in a relatively short time. It also enabled the generative system to become an adaptive and flexible system, since most of the mathematical scripts relied on inserted variables from the end-user.

Although generative systems had distinguished itself from the other available heuristic methods as one of the great utilized search engines due to their great ability to produce many results instead of one. The human mind of the end-user is ultimately the decision maker. As in some cases, the optimal result is selected based on a set of variables that might not be considered in the design process of the algorithm. Such as cost, client's feedback and so on. Hence, human's interaction with the computational system is critical to create an optimal generative system.

The illumination of interior spaces ranges far beyond merely technical executions and mathematical calculations. As interior design spaces are largely impacted by the selected type of lighting, aesthetical design, and functionality of it.

Lighting is one of the key elements that impacts space, can highlight certain design elements. For example, artificial lighting in commercial spaces can diverse based on the functionality. As offices would largely require bright settings and glare-free lighting. Whereas commercial stores would target lighting temperature and colors, highlighting the presented product. Whilst cinemas or small cozy cafés would use pleasingly soft, warm lighting, creating a full experience for occupants. Hence, lighting planning is dictated by the various requirements of functionality, and expectations of the end user (Skowranek 2017).

Most of functional-lighting systems can be in forms of uniform lighting, general lighting with local lighting, or fully localized lighting systems (Karlen, Benya & Spangler 2017). These systems are mostly designed for offices, schools and industrial environments, where the key aim in those spaces is to provide good visual conditions.

That is why this dissertation had tested the created generative design system in commercial spaces. Due to the fact that a large amount of current constructed office towers employs functional-lighting systems and neglects the aesthetic side of those systems. Aiming for functionality only, since it is more efficient and less time consuming to apply direct lighting using simple approaches, than to place the time and effort on providing new designs for uniform functional-lighting systems. In addition, highlights generative design as one form of the recent utilized technologies in lighting design and showcasing a variety of previous researches that employed it mostly in lighting control systems. while indicating the gap in the current literature that merely addressed new approaches in generative design in creating functional systems with the aesthetic aspect in consideration.

## **1.2 Problem statement**

One of the key problems in most applications of mathematical lighting models and their implementations to lighting design layouts is the limitation to space settings. Neglecting the design matter and functionality to individual inputs. Lighting generative applications roughly suggest uniform illumination for a predefined area, that adopts a typical categorical approach to light planning, overlooking the design and the aesthetic aspect of it and limiting a holistic solution that would address the design requirements and functionality at the same time.

The generative design system created in this dissertation embraces an approach that links functionality to a number of flexible design approaches that can be practically implemented to achieve a targeted lighting design aim, contrarily to applications that only consider lighting uniformity, or optimizing lighting intensity or lighting systems and don't consider the vital relationship between interior design and the aesthetical aspect of interior lighting.

## **1.3 Goals and Objectives**

The main objectives for this dissertation can be summarized as the following:

- 1- To propose a computational generative design in producing various solutions for reflected ceiling layouts designs.
- 2- To exhibit a methodological framework that presents an outline for a comprehensive, interactive generative system that produces relatively high quantities of lighting layouts for interior ceilings. performing a full design cycle. Starting from generating various and diverse reflected ceiling designs, all the way to producing numerical data for lighting counts and artificial lighting calculations, all within the same software.
- 3- Demonstrate different aspects in sustainability in generative systems, such as saving time, cutting cost, increasing diversity, providing flexibility, delivering innovative designs, creating various design tools and much more.

- 4- Targeting sustainability through cost reduction and benefits increasement, through statistically proving the usage of the software to produce results is more beneficial compared to a designer.
- 5- To integrates generative lighting design with a BIM module.

The aim of the discussed chapters is to create a generative system that can produce solutions that develops a certain aspect of interior design, namely in indoor artificial lighting design. It obtains six chapters, contents of each are discussed briefly in the following paragraphs.

Chapter 1: Introduction; provides an introductory note that highlights the selection of generative design as the research engine for artificial indoor lighting and commercial spaces as the implementation space for the system. The chapter provides the problem statement along with the key goals and objectives of this paper.

Chapter 2: literature review; discusses the published literature papers regarding algorithmic design. Highlighting its definition, its limitations, and advantages. It provides a review of the most influential work addressing the evolution of algorithmic systems throughout the years. And exhibits various approaches, previous applications, and optimization systems that were attained to achieve different means of sustainability for commercial artificial lighting through the usage of algorithmic design. Most importantly, this chapter identifies the literature gap that this dissertation aims to fulfill by creating the generative design system.

Chapter 3: Methodology - The generative design outline; this chapter outlines the process of developing the generative system, it briefly discusses the structure of the system, identifying the design concept behind each design approach that created the generative system, along with highlighting the taken steps to arrange the algorithmic scripts, along with identifying the approaches taken to exhibit the generated results to be easier to use, alter and view.

Chapter 4: Development of the generative system; talks about the components of the algorithmic system in details, it explains the mathematical approaches and showcases the algorithmic scripts that create different design approaches and sub-approaches, it highlights the structure of the entire system and discusses in details the development of every approach, along with the shortfalls, the limitations, and the created solutions in every script. It also exhibits the created methods that allowed the overall results for all the design approaches to be produced at once and showcases the approach that created the proper flexibility to turn on and off algorithmic scripts.

Chapter 5: Results and discussion - Testing the generative system; tests the scripted generative design system using different approaches that are identified in three phases: Phase 01, Phase 02, and Phase 03. Ensuring that the generative algorithm presents feasible outputs in terms of the diversity in the mechanisms for selection, limitation and mutation of the outputs and sustainability implementation.

Chapter 6: Conclusions and further work; summarizes the work presented in this dissertation, a comprehensive briefing for the generative system its structure, providing an overview for the different phases selected to test the system along with the presented results. Along with the contribution of the research in this field. Highlighting the faced limitations while scripting and testing the system and most importantly, it proposes further work that is founded based on this research.

## Chapter 2

### Literature Review



## **2 Chapter 2: Literature review**

### **2.1 Introduction**

Various chapters in this dissertation includes brief literature review sections. Chapter 4, which discusses in detail the development of the generative design mentions previous studies that implemented the same proposed software in various fields. Chapter 5, that exhibits and evaluates the generated results through testing the generative design, employs sustainability within the system itself through literature.

In this chapter, a review of the published literature papers regarding algorithmic design is conducted, providing a comprehensive definition of generative design, along with an explanation of its limitations and advantages.

Followed by a review of the most influential work that highlights the evolution of algorithmic systems throughout the years. The chapter also addresses various approaches that were attained to achieve different means of sustainability for commercial artificial lighting through the usage of algorithmic design. Earlier applications found in literature that aimed to increase energy efficiency in artificial indoor lighting were discussed, whether through optimizing light uniformity or lighting control systems.

In addition, the optimization of algorithmic systems found in generative designs that's located in commercial spaces was discussed as most cases were conducted through optimizing space planning. Commercial ceiling designs that were formed through algorithmic systems were addressed, along with ceiling designs case-studies that were published in literature.

The final section in this chapter highlights the different studies that were reviewed and identifies the literature gap that this dissertation aims to fulfill by creating the generative design.

## **2.2 Literature review**

### **2.2.1 Generative design overview**

Over the past years, the digital revolution and computational tools had allowed designers from various fields to be able to use groundbreaking form-finding techniques under various terminologies such as: Generative Design, Algorithmic Design, and Parametric Design.

Various descriptions are published throughout the literature defining those terminologies based on different architectural theorist. It seemed that a single definition for each terminology cannot be pinned, due to the shared principles that are found in all the proposed definitions.

(Caetano, Santos & Leitão 2020) were able to analyze the existing literature related to computational design to clarify the most used terms. They have identified Generative Design terminology as the utilization of an algorithm to generate a design. Which falls under the terminology of Algorithmic Design, which enables a correlated relationship between an algorithm and a generated design. While outlining Parametric Design as a design that relies on a set of parameters to define different sets of designs.

A comprehensive terminology can be identified for Generative Design as a computational design tool that generates several designs and forms built from logical rules or algorithms, originating from scripting programs, such as: Processing, Rhinoceros and Grasshopper (Agkathidis 2015). It is a tool that backs up a generated design by an entire system, exposing designers to certain parameters that creates multiple variations in the produced design (Nagy et al. 2017).

Algorithms can be presented differently according to the employed algorithmic medium. They can be created using different digital tools such as: codes, graphics nodes or verbal scripts. Algorithms are key components in computers, since any program can be identified as an algorithmic system that uses a set of instructions written in a certain language such as:

numerical inputs, alphabets data or geometric elements to command the computer to perform a certain task in order to solve a problem (Fernandes 2013).

(El-khalidi 2007) states that algorithmic approaches are designed through a rationalization process, leading the designers to mathematically convey their design concepts into tasks and sequences. They require practical, clear instructions that would usually be translated into mathematical scripts. These mathematical scripts can be defined as the engine that builds up the systems and generates outputs. Different forms of outputs are produced based on the inserted variables, which can be shapes, locations, numbers, sizes, etc. (Fernandes 2013).

While (DINO 2012) Defines it as systems that targets formation over form, using logic into designing and modeling, instead of the traditional approach which develops geometrical forms. It is an algorithmic system that offers a large area of adaptability in changing the design criteria, resulting in an instant adaptation in the final solution, making it ideal for complex designs (DINO 2012; Henriques et al. 2019; Oxman 2017a). Algorithmic designs disregard the predictable human's experience and creativity, in favor of delivering hundreds, if not thousands of complex, practical, and implementable solutions of designs. Furthermore, they are efficient in fabricating overly complex projects, as it would be difficult and time-consuming if they were produced manually without the help of computational designs (Caetano & Leitão 2018).

Exploring computational abilities do not only provide unthinkable solutions in a short period of time but expands creativity as well. Since the common concept of the human's creativity aims to improve the product quality more than developing various possibilities during the creation process of solutions (Henriques et al. 2019). As designers would usually utilize their skills and experiences to create a single solution, leaving the optimization process to the end (Henriques et al. 2019; Nagy et al. 2017). Relying on the human's intuition as a source of

creativity, while algorithmic systems relies on the logic to generate the computer's creativity (Bukhari 2011).

Furthermore, these computational design approaches had led to developing the normal means of thinking and modeled a new body of thoughts that is now called Design Parametric Thinking, producing its own line of terminologies under it (Oxman 2017a). And impacting the creation of forms and the creativity approach of parametric design thinking for designers (Oxman 2017b).

However, (DINO 2012; Gunagama 2018) argue that algorithmic designs are systems that are built solely on variables and parameters, which have a correlated relationship to limiting the generated designs based on the data input and the selected boundaries. In addition, algorithmic systems would produce limiting results if verbal ideas were not translated into mathematical ones. (Terzidis 2006) affirms the authors inputs, stating that the relationship between numbers and concepts are too deterministic. Adding that many designers are simply not interested in the mathematic of the design composition as much as the composition itself.

While (Kolarevic 2003; Terzidis 2006) addresses the shortfall of algorithmic design systems from a different perspective, stating that the creative process of the generated design is developed in the human mind. Hence, computers are just methods to produce the design. And because of that, algorithmic results will go through evaluation metrics that refers to the human mind. As designers select the results not the algorithm.

The use of Generative Design was adopted by architects widely in literature (Agkathidis 2015). As they have used generative algorithmic systems to create various parametric designs. It has been noticed in the preliminary design stages (Touloupaki & Theodosiou 2017). Which is considered one of the strengths of algorithmic design systems. As they're mostly utilized in the primary stages that focus on the "process" over the "end-product" (DINO 2012).

(Bukhari 2011; Terzidis 2006) state that Generative Design examples such as; genetic algorithms, evolutionary designs, stochastic search, cellular automata fall under the category of algorithmic designs. With (Barreto 2016; Caetano & Leitão 2018; Feist et al. 2016) affirming some algorithmic-tools such as Dynamo, Generative-Components, Rosetta-BIM, Rhino-BIM, Hummingbird, and Lyrebird, that transformed the static norms of the orthodox process in creating complex geometry into dynamic digital algorithmic systems (Feist et al. 2016; Kolarevic 2003). For example, (Gunagama 2018) had utilized Grasshopper -a digital algorithmic programming software- to model one story building on sloped land. While (Lee & Kim 2016) explored the spatial diversity of the traditional Korean house Han-ok, using an algorithmic system to modernize the house through using Cellular Automata (CA) as a generative design tool.

Whereas (Caetano & Leitão 2018) had combined the algorithmic design with Building Information Modeling (BIM) through integrating algorithmic design in the design process of a simple design studio. While (Khan & Awan 2018) proposed a generative design technique under the name of Space-Filling-Generative Design Technique that generates all the possible shape variations for an inserted computer-aided design model.

Furthermore, (Tsiamis, Oliva & Calvano 2017) had replicated the same configurations from the geometric rules that generate folded origami tessellations, to create a free-form folded tessellation structural system that supports surfaces on an architectural scale. While (Barreto 2016; Caetano & Leitão 2016; Feist et al. 2016) had employed and evaluated RosettaBIM tool in supporting the development of generative design in Building Information Modeling.

### 2.2.2 The evolution of algorithmic design in architecture

(Caetano, Santos & Leitão 2020) had proposed in Figure 1 a graph which proposes an approximate time-line for algorithmic design that appeared in scientific sources between 1978 until 2018. The authors had only sampled 65 publications regarding algorithmic design, since generative design obtains a wider perspective that leads to an overlap in literature with algorithmic design and parametric design, leading to some inconsistencies in the definitions and the categories.

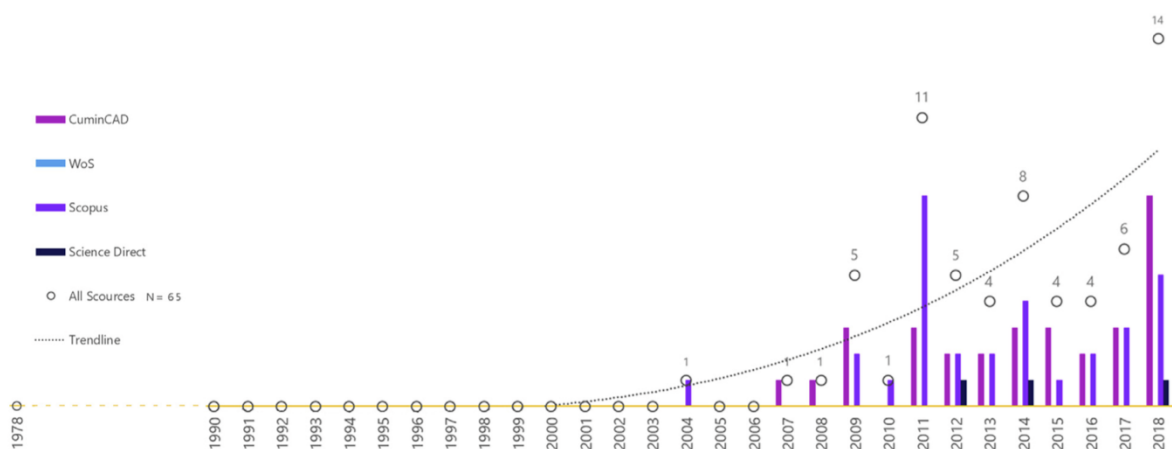


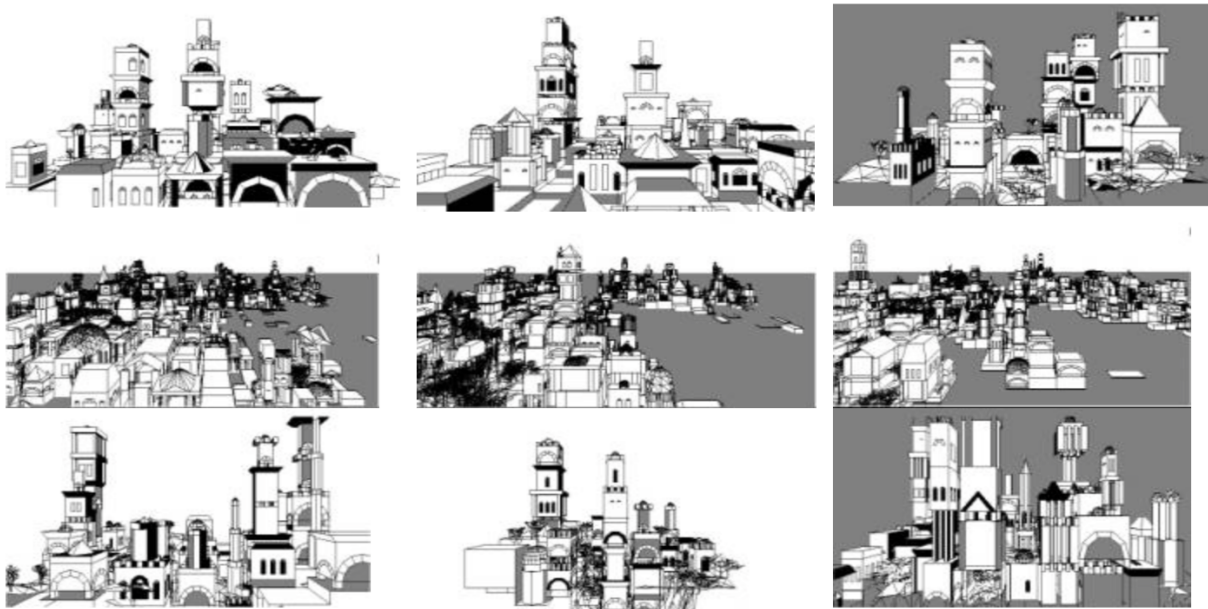
Figure 1: Number of times generative design showed in literature from 1978 up to 2018(Caetano, Santos & Leitão 2020)

(Terzidis 2006) highlights in his book the necessity of rationality in the design process. Which started to appear in the 1960s, due to the development of computers as logical devices.

Different researches in literature (Caetano, Santos & Leitão 2020; Fernandes 2013; Terzidis 2006) had pointed out that (Stiny & Mitchell 1978) paper in bilateral symmetries can be considered the entail major step in algorithmic architecture design. Which described a parametric shape grammar that generates the ground plans of Palladio's villas. Authors highlighted the papers rational description of the process, along with its clear instructions had led to the production of algorithms.

More than a decade later, (Soddu 1989) published in his book a morphological simulation with a research software that uses algorithms to write specific rules that define the dynamic and the chaotic systems to generate 3d town models, results had showed differences in the generated designs but similarities in the sense of the environmental form of the generated town.

Figure 2 is showcasing the first case study that was tested in 1988, which was a medieval town in Italy. As the logical structure of this virtual city controlled its formalization, and its identity was inspired by Simone Martine and Giotto paintings (Soddu 1988).



*Figure 2: a sequence of generated 3D models by the first generative project of Medieval Towns in Italy (Soddu 1989)*

Two years later, (Soddu 1991) published two software that work as simulation tools for learning approaches to a dynamic evolution of town shapes, architecture, and industrial designs. In 1993, the same author created a new search software employing the idea of design evolution, stating it as a sequence of logical procedures that increases performance and complexity. The proposed software generates the simulation of the increased complexity of an idea. The idea can target three parameters: architecture, environment and industrial designs. As every logical approach to design builds a possible path to simulate a dynamic evolution that address of one of those three parameters (Soddu 1993).

At the beginning of the 21<sup>st</sup> century, literature had started to focus more on generative design, including a literature shift into algorithmic design (Caetano, Santos & Leitão 2020).

Whereas, in 2001 (Caldas & Rocha 2001) had published a new generative system that uses genetic algorithm to investigate the integration of architectural design targets and its impact on the generative design system. Allowing the generated solutions to be within certain design intentions. It was tested on Alvaro Siza's School of Architecture at Oporto, Portugal, as the system worked on three-dimensional narrative of the building which included the geometry, orientation, spatial characteristics, materials and so on. Proving its flexibility as a system that includes different constraints which allows the designer to manipulate certain design targets.

(Chien & Flemming 2002) exhibited in the next year a comprehensive approach that helped navigating and filtering the generated solutions in generative design systems that dealt with human spatial design. While (Frazer et al. 2002) had focused on proposing a new initial design implementation for a building envelope system utilizing generative approaches and the use of algorithms.

One of the key researches that were highly cited in different papers throughout literature was (Kolarevic 2003). The author investigated the architectural applications that utilized computational design which included algorithmic design and fabrication technologies. Providing a comprehensive understanding of the relationship between architecture and its suitable methods of production. Similarly, (McCormack, Dorin & Innocent 2004) was a highly influential research, reviewing a variety of processes that can be investigated by designers including generative grammars. Providing different examples of applications to creativity and design. A year later, (Shea, Aish & Gourtovaia 2005) had described the initial combination of a generative design tool through the use of XML models. Exhibiting a discussion of the interoperated relationship between associative modeling and generative systems.



Fasting forward to 2010, algorithmic design had become numerous addressed in literature. (Kotnik 2010) offered a theoretical framework that provides a comprehensive approach that explains the logical correlation and concepts that is translated to algorithms and mathematical scripts in digital design tools, to teach digital design in architecture.

As exhibited earlier in Figure 1; based on (Caetano, Santos & Leitão 2020) evaluation, the year of 2011 had witnessed a noticeable incensement in algorithmic design researches in the literature. (Krish 2011) proposed a Computer Aided Design (CAD) design generative exploration system. Addressing complex multi-criteria design issues. It is constructed based on a genotype structure of the design with parametric CAD system which verifies its parameters randomly through utilizing identified limits in order to generate various designs. Krish's design philosophy behind this generative approach was later utilized by various papers in the literature.

While 2012 had various important publications about algorithmic design in architecture, as (Dunn 2012) book: Digital Fabrication in Design had addressed the importance of computational designs that utilized parametric design and algorithmic design to transform concepts into forms, with the focus on the formation process over the final product. Highlighting the enormous capacity provided by the current computational modeling methods, that impacted the traditional creative way of thinking for designers, enabling them to create unthinkable designs that would've been hard to achieve through the traditional methods. Additionally, (Singh & Gu 2012) provided one of the key steps towards supporting multiple generative design techniques through creating a framework for an integrated system. The paper reviewed five different generative design approaches: Cellular automata (CA), Genetic algorithms (GA), Shape grammars (SG), L-systems (LS) and Swarm intelligence (SI) and compared between their characteristics and design systems. Painting a framework that aims to develop an integrated system that provides high levels of flexibility.

In the year of 2013; (Jabi 2013) published a book called: Parametric Design for Architecture, where Part I in the book had discussed algorithmic thinking in detail. Identifying the overall structure, logic behind it, the functions that utilizes it and much more. The book explained how algorithmic thinking key structure relies on logic, which allows the computer to generate any desired solution. It provided simple, easy to understand framework on how to transfer logic into a programming language digitally. While (Magna et al. 2013) work was highly referenced in literature. The paper exhibited a project as a case-study to identify the methodological steps that led into generating a design utilizing biomimetic principles in developing structural and architectural matters, Figure 3 shows the different design steps that were developed in an FE model, simulating cell elements.

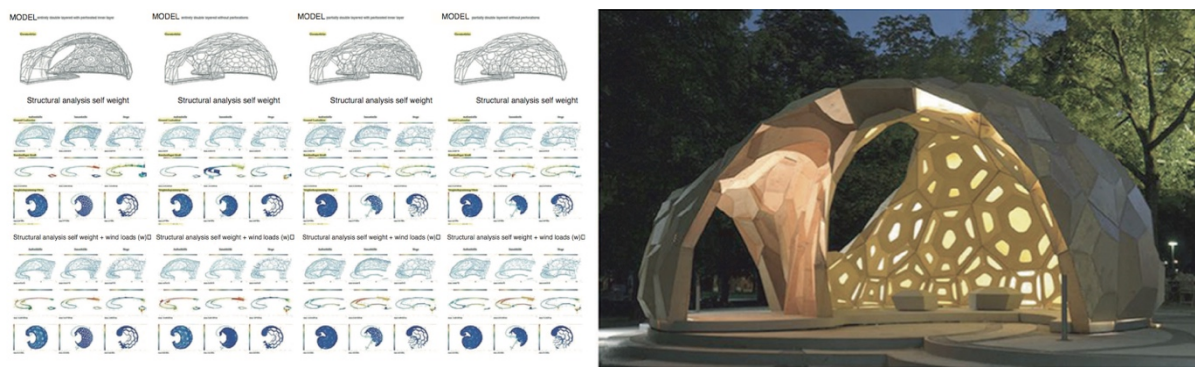


Figure 3: Different steps of geometric variations (Magna et al. 2013)

In the following year; (Svoboda et al. 2014) revealed a new tool for complex architectural objects under the name of algorithmic forms. It utilized two open sources written in Grasshopper algorithm editor and C++, which are linked to scripting-based architectural modeling software Rhinoceros, IntelliCAD. Generating various structural characteristics that matches the targeted design objectives. The paper demonstrated three case studies that showcased the capacity of the algorithmic system. Algorithmic design is developing throughout the years, generating novel approaches for various design solutions. Published papers in literature from 2015 up to 2020 are addressed further throughout this chapter.

### **2.2.3 Algorithmic design sustainability approaches for commercial artificial lighting**

Algorithmic Design had utilized artificial lighting as a mean of approaching sustainability in the literature, optimizing various solutions that targets energy efficiency in different fields. 25% to 35% of the total energy consumption in architecture is consumed by artificial lighting (Madias, Kontaxis & Topalis 2016). Consequently, several research papers in literature targeted artificial lighting calculations in commercial sectors to investigate the generated energy efficiency. Where a 23% reduction of energy savings was achieved through utilizing different lighting systems and control strategies tested on some open-plan office layouts (Xu et al. 2017), showcasing the importance of simulation analyses studies that investigate artificial lighting, leading to 77% of energy savings in some cases such as (Larsen et al. 2017).

Different means of energy savings regarding office building lighting were proposed in the literature that took advantage of the promising potential of generative design; starting from designing the optimal space layout that allows merging between day light and artificial lighting, impacting the office energy performance (Du et al. 2019), finding the most suitable configurations for lighting control systems (Wagiman & Abdullah 2018), or identifying the reflecting materials to redirect interior lighting in office spaces (Wu 2018). Whilst harvesting daylighting to illuminate indoor areas through applying light guiding systems (Chen, Oh & Burhan 2020), all the way to examining a number of lighting technologies merged with daylight harvesting systems (Doulos et al. 2019).

However, occupants play an important role that has an active impact of energy efficiency (Gandhi & Brager 2016). That's why (Lowry 2016) argues that the published studies for energy efficiency concerning lighting optimization systems that are provided in energy software might fall in the trap of overstating saving potentials in automatic control. Since manual control is not highlighted enough. It is more realistic and depends on the occupants' behaviors, that usually has a critical impact on the lighting use.

### 2.2.3.1 Algorithmic design in optimizing light uniformity

(Mendes et al. 2017) had identified that decreasing the energy usage whilst keeping an adequate lighting levels can be an optimization problem in offices. Hence, genetic algorithms have been exploited in artificial lighting (Madas, Kontaxis & Topalis 2016). Aiming to achieve uniformity in indoor lighting spaces while still achieving sustainability through energy efficiency. (Wagiman et al. 2020) had produced results that concurred with (Mendes et al. 2017) statement, identifying the Optimization-Based Control Technique as the most used approach in commercial lightings upon reviewing the state of the art interior lighting system control techniques in commercial buildings.

(Baloch et al. 2018) work presented state of art systematic review of indoor lighting simulations related to the building research in the literature. Where the levels of various methods that targeted indoor lighting simulations are showing in Figure 4 , with genetic algorithms (GA) attaining the highest levels.

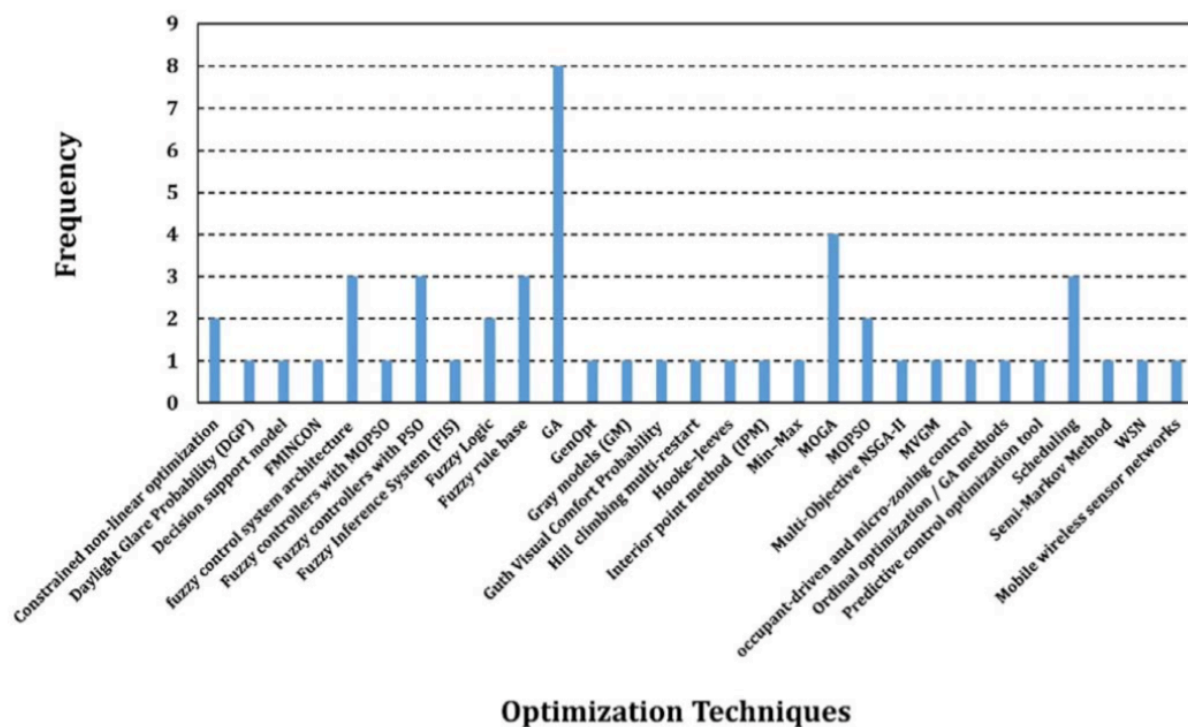


Figure 4: Optimization techniques found in the literature (Baloch et al. 2018)

Algorithmic optimization lighting approaches were widely found throughout literature, where (Wang & Tan 2013) had reduced energy consumption through optimizing the illumination levels, creating a lighting energy optimization algorithm through proposing a neural network that defines the complicated relationship between the dimming levels and the measured illuminance of the task area. While (Liu et al. 2017) proposed a gene density genetic algorithm (GDGA) to obtain an optimal light layout to attain uniform power, that serves the needs of both Light Emitting Diode (LED) lamp coordinates with the optical signal power's uniformity and population's diversity on the communication plane. Furthermore, the same authors had improved the genetic algorithmic system in order to improve the uniformity coverage through minimizing the fluctuation power and enhance the fairness without changing LED layout or position (Liu et al. 2019).

(Madias, Kontaxis & Topalis 2016) had created multi-objective genetic algorithmic system for indoor lighting; through formulating two mathematical functions that decrease energy consumption while increasing uniformity and sustaining the illuminance levels. (Mendes et al. 2017) on the other hand had proposed a bio-inspired metaheuristic optimization algorithmic system that addresses the problems of reducing required energies to provide lighting levels that matches a set of targets. The proposed algorithmic system was put to the test using the standard benchmark functions, followed by real time lighting issues.

While (Dupláková et al. 2019) created 15 sequences of rationalization algorithms for redesigning lighting in manufacturing enterprises. Where direct measurements were carried out in a selected machine engineering company to input the acquired data to the rationalization algorithms. Verifying the beneficial levels of the algorithmic system through Dialux Evo 6.1.

### **2.2.3.2 Algorithmic design in optimizing indoor lighting control systems**

Plenty of optimization methods have been applied in indoor lighting control systems using algorithmic techniques in commercial spaces. As (Soori & Vishwas 2013) had proposed a lighting control algorithmic system that allows a control strategy for office lighting systems with the target of achieving energy efficiency. The authors had examined a typical office building in Dubai, United Arab Emirates. And designed the algorithm taken into consideration the following logical parameters; day lighting, integrated lighting control systems and space cooling systems, along with investigating the level of comfort for occupants in order to address thermal conditions and lighting, with other elements that contribute to the occupants' productivity levels.

Furthermore, (Gunay et al. 2017) was able to formulate an algorithmic system that learns the illuminance preferences of the occupants, and employs that data to control five different light control systems and the closing-opening behaviors of the blinds in the office. Resulting in a substantial reduction in lighting loads while sustaining the occupant's visual comfort. Another novel adaptive lighting control system is proposed by (Seyedolhosseini et al. 2020); as the system proposed a smart lighting control approach for both controllable (dimmable) and uncontrollable light sources. The system was able able to adapt the different lighting conditions on different areas when tested on an indoor environment, responding to irregular variations in daylight and reducing energy consumption.

Additionally, (Plebe & Pavone 2017) had created a new system based on multi-objective optimization criteria for interior lighting. The selected criteria targeted illuminance levels, uniformity and electrical saving energy. The paper had utilized 3D graphic software Blender as a direct engine to provide a three-dimensional interior space, and a simulation platform for the lighting calculations. The system developed an algorithm that generate tailored solutions

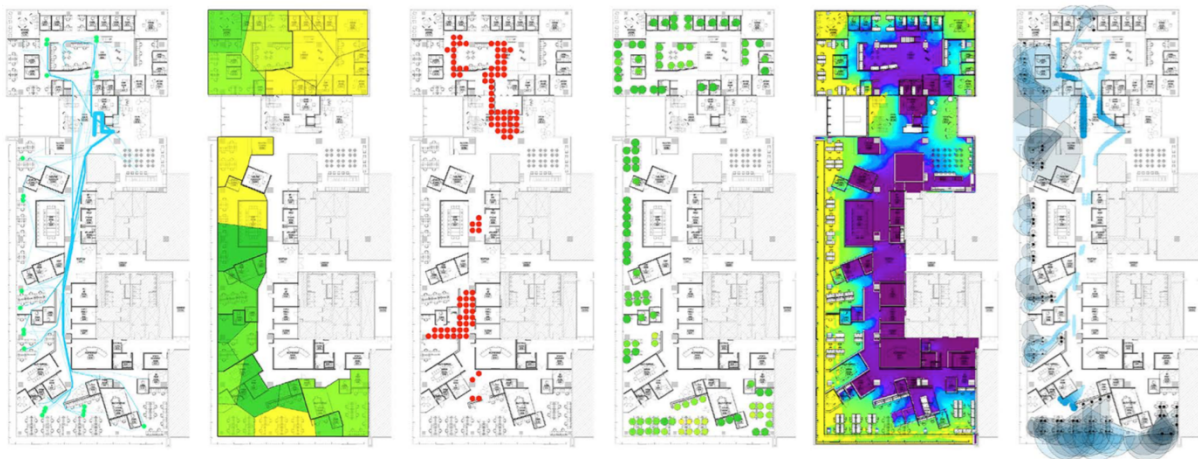
for lighting design optimization, utilizing potential illumination configurations, lamp specifications, lamp position and orientation, intensity, color temperature and much more.

A lot of papers had focused on optimizing lighting systems through linking independent versions to current software for modeling and lighting calculations in offices (Duplaková et al. 2019). (Şahin, Oğuz & Büyüktümtürk 2016) had created an artificial neural network (ANN) model that estimates the illuminance distribution in interior spaces. The model efficiency had been proven through a results comparison with long-duration experimental measurements, contributing to energy-saving interior lighting development. Similarly, (Ilbeigi, Ghomeishi & Dehghanbanadaki 2020) had utilized ANN-based energy consumption prediction model in an office building. Optimizing energy through a genetic algorithm that takes under consideration essential variables, revealing that artificial lighting is one of the key factors that directly impacts the total energy demand, due to its operation time.

Furthermore, (Tagueu & Ndzana 2019) compared using Dialux Evo simulation between the different approaches in lighting optimizing systems; employing dual Fluo/LED luminaires, a neural network model that exemplified the relationship between dimming levels and illumination, along with a similar model that was built through a mathematical/algorithmic approach. The evaluated results revealed the advantage of the mathematical/algorithmic model over the neural network model, however both had optimized lighting in a selected building. (Natephra et al. 2017) on the other hand had developed an approach that utilizes a virtual reality software to simulate daylighting and artificial lights through connecting it to a Dynamo script -an algorithmic visual programming software- allowing users to compare design scenarios while obtaining energy consumption feedback. Moreover, (Bangali 2018) had calculated discomfort glare and the unified glare ratings in an office conference room through utilizing a Python developed program that calculates the Unified Glare Rating through a Discomfort Glare Index equation, and comparing the results to acquired data from Dialux Evo software.

## 2.2.4 Algorithmic design for generating designs in commercial spaces

Research proposals that targeted office interiors from a generative design aspect proposed different approaches to optimize various solutions. Most papers in literature targeted spatial layouts; as (Nagy et al. 2017) developed a computational design model that generates a variety of office layouts, built on identifying all the desired interactions through some metric inputs. Figure 5 shows the selected design metrics that produced the generated values which are used in the generative process. The generative process is identified as a search algorithm that evaluates the designs. The design metrics can be summarized into 6 metrics: the adjacency preference, work style preference, buzz, productivity, daylight, and views to outside.



*Figure 5: Design metrics in the algorithmic system: (from left to right: adjacency preference, work style preference, buzz, productivity, daylight and views to outside). (Nagy et al. 2017)*

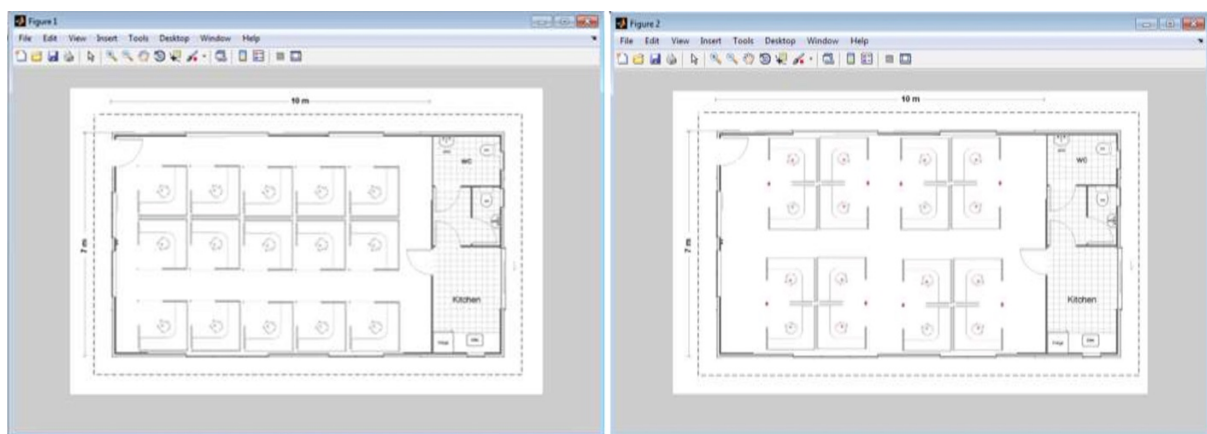
Relatedly, Autodesk had included generative design feature in Revit 2021. Where one of the features is workspace layout; creating various desk configurations in a room (Smith et al. 2020). Allowing end users to input the required parameters in order to produce the most suitable desired solution.

(Anderson et al. 2018) had presented various practical algorithms for space planning in commercial offices. As the algorithmic system was tested on 13,000 offices designed by architects. Performing equally as good as an architect in 77% of the time and producing



promising results that utilized space standards. In addition, (Phelan, Davis & Anderson 2017) proposed a new approach to utilize meeting rooms spaces through an artificial neural network that outperformed human designers. Affirming that recognizing patterns in built architecture using machine learning algorithms can help designers to establish better design decisions.

Furthermore, (Abdulgaki, Abdulbaqi & Mohialden 2018) had also proposed a new method for office space planning based on Monte Carlo technique that is correlated to a Density-Based firefly algorithm. Generating the optimal space planning and assigning furniture in the office through utilizing various parameters. Figure 6 shows simulation results in an office working area that is 7x10 m<sup>2</sup> using the created novel approach for office space planning. Similarly, (Tachikawa & Osana 2012) had proposed an algorithmic system that generates two key outputs for offices, room arrangement plans, and layout plans for workspace. The system structure is based on the combination of genetic algorithms, where an adaptive algorithm, and a genetic algorithm with a search area adaptation are utilized to produce a variety of layout plans in a small amount of time.



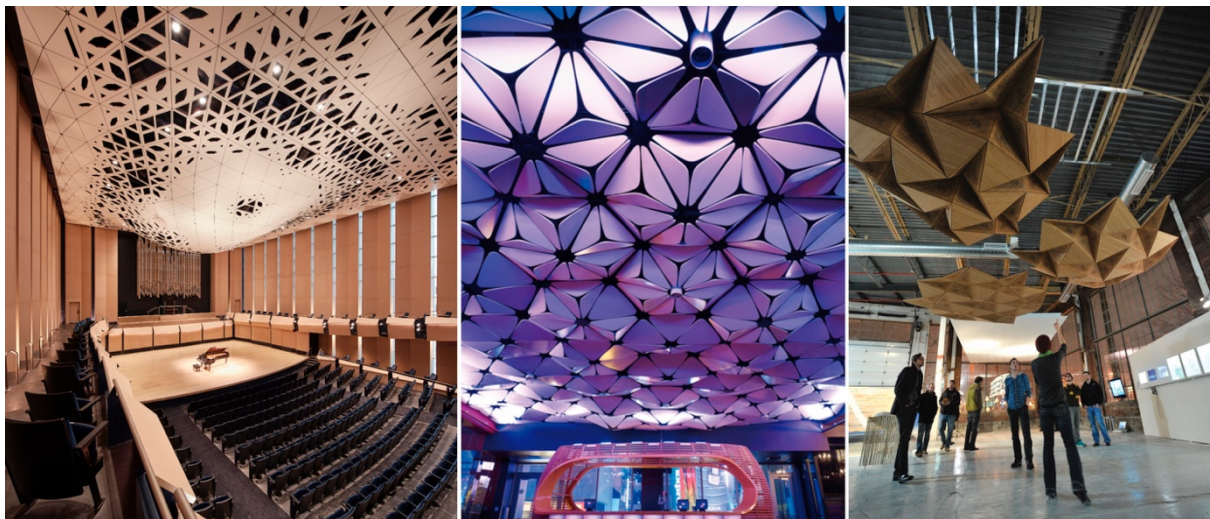
*Figure 6: Simulation results in an office using the created algorithm (Abdulgaki, Abdulbaqi & Mohialden 2018)*

While, (Coelho, Amaral & Guimarães 2018) had optimized the quantity of light sources in interiors using a genetic algorithm, achieving an appropriate light intensity while minimizing the quantity of light sources.

### 2.2.5 Algorithmic design utilized in commercial ceiling designs

As for reflected ceilings designs, generative design was mostly utilized through exploring shape variations, mostly for the purpose of optimizing the space's acoustics. (Rumpf et al. 2018) had customized an acoustical structure that combined various digital design methods in an office space. Employing various modeling techniques that formulated the inserted data to geometry, with a work scope that included structure, light, sound and other parameters of energy and matter.

Achieving desired acoustical performance through using generative designs was mostly found employed in large spaces in design. (Badino, Shtrepi & Astolfi 2020) explored the existing architectural applications of Performance-Based Designs, where generative design covers an essential part in creating the acoustical three-dimensional geometry. The paper had given various case-studies of ceiling designs that were created by international architectural companies, utilizing algorithmic design in order to create unique shapes that covered various performance parameters. Figure 7 is showing three selected cases that highlighted the usage of algorithmic design in creating various shape variations.



*Figure 7: From left to right: University of Iowa Concert Hall, Conga Room dance club, Resonant Chamber (Badino, Shtrepi & Astolfi 2020)*

The first example is the University of Iowa Concert Hall, where the suspended structure is created from folded aluminum composite modules. The design was generated through a collaborative parametric model developed in Grasshopper. Which later was transferred into Sketch-up and Autodesk Revit models as well. The structure had proposed solutions for five different categories: acoustics, stage lighting, house lighting, audio-visual and fire protection while still fabricating a unique, one of a kind ceiling design. The second ceiling example shown in Figure 7 is the ceiling for Conga Room dance club in Los Angeles. Made of CNC-milled plywood panels that target almost the same categories as the first example. The design was generated utilizing CATIA software, Rhinoceros which utilizes grasshopper and Ecotect. While the third example seen in Figure 7 is Resonant Chamber; which is a responsive acoustic envelope system that's designed based on origami principles. Created from a combination of different plywood panels, and generated through Grasshopper, Kangaroo software to create the deformation of the ceiling, and ray-tracing tool for acoustics (Badino, Shtrepi & Astolfi 2020). The examples given highlights the importance of algorithmic design in creating three-dimensional ceiling designs that solves various problems while still creating unique forms.

However, little literature was found about creating two-dimensional ceiling layouts that utilized algorithmic design to address generating lighting layout designs.

(Kim et al. 2016) had proposed an automated lighting layout system for indoor space using properties included in Building Information Modeling (BIM) model to generate lightings layouts. The created system employee information taken from the BIM model to generate solutions. It arranges lighting fixtures automatically in space based on the calculation of illuminance and the quantity of lights that are recommended based on the Korea Industrial Standards. The system was tested on a literature room as seen in Figure 8, it produced two alternatives; scenario A was chosen as a suitable alternative, since scenario B produced a luminous flux that was too low to meet the Korea Industrial standards.

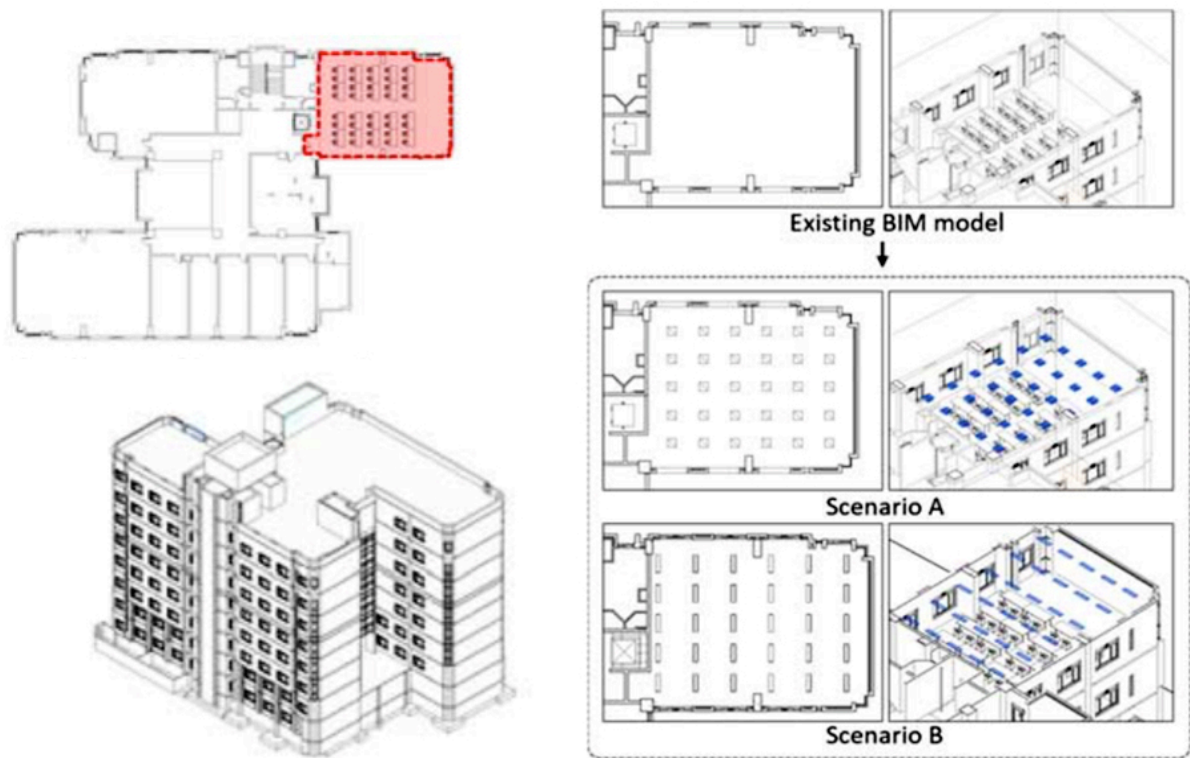


Figure 8: BIM model of college of human ecology building and the lecture room along with the results (Kim et al. 2016)

A lot of lighting design software provide certain tools to help designers distribute lighting throughout the space, as DIALux software for example presents an interactive approach for distributing light luminaries, providing different methods that arrange uniform lighting on the constructed model in the software. As the end user is able to draw a rectangular, polygon or a line arrangement for the lights. Selecting the appropriate space within each light. However, based on (Shikder, Price & Mourshed 2009) evaluation for different four artificial lighting simulation tools, DIALux displayed shortage in building complex geometric description.

Apart from (Kim et al. 2016) paper, there wasn't any papers that addressed generating two-dimensional ceiling designs using any form of generative design, nor did the authors discussed any formal papers found in that field. Most papers had focused on optimizing lighting calculations, utilized algorithmic design in lighting uniformity and in control systems. In regard to office designs, most novel algorithmic systems were built for space utilization, while algorithmic systems addressing ceiling designs targeted creating three-dimensional shapes.

### **2.3 Identifying the literature gap**

Reviewing the current literature, it is noticeable that a lot of different approaches employed generative design in artificial lighting in commercial spaces. Utilizing different lighting strategies in offices (Chen, Oh & Burhan 2020; Doulos et al. 2019; Du et al. 2019; Larsen et al. 2017; Xu et al. 2017). And optimizing indoor lighting control systems using algorithmic systems (Gunay et al. 2017; Plebe & Pavone 2017; Seyedolhosseini et al. 2020; Soori & Vishwas 2013; Wagiman & Abdullah 2018), with the focus on linking independent versions to current software for modeling and lighting calculations (Bangali 2018; Ilbeigi, Ghomeishi & Dehghanbanadaki 2020; Natephra et al. 2017; Şahin, Oğuz & Büyüktümtürk 2016; Tagueu & Ndzana 2019). While attaining sustainability through energy efficiency by utilizing the generative design to achieve light uniformity in indoor spaces (Baloch et al. 2018, Dupláková et al. 2019; Liu et al. 2017; 2019; Madias, Kontaxis & Topalis 2016; Mendes et al. 2017; Wagiman et al. 2020; Wang & Tan 2013).

While research proposals that targeted office interiors from a generative design aspect proposed different approaches to optimize various solutions. Most papers in literature targeted spatial layouts (Abdulbaki, Abdulbaqi & Mohialden 2018; Anderson et al. 2018; Coelho, Amaral & Guimarães 2018; Nagy et al. 2017; Phelan, Davis & Anderson 2017; Smith et al. 2020; Tachikawa & Osana 2012), As for reflected ceilings designs; generative design was mostly utilized through exploring shape variations, mostly for the purpose of optimizing the space's acoustics (Badino, Shtrepi & Astolfi 2020; Rumpf et al. 2018)

It was clear that literature had exploited generative design in targeting artificial lighting through various approaches that aim to produce and optimize lighting uniformity, lighting calculations, and different several artificial lighting systems that address decreasing energy levels to achieve sustainability.

While commercial interior design was mostly highlighted in the generative design field through two aspects: office space layout distribution, which included the optimal furniture distribution that relied on different variables. And creating three-dimensional acoustical ceilings that utilized algorithmic systems.

Hence, a gap was identified that this dissertation aims to fill. Where generative design is employed to merge between two-dimensional ceiling designs and artificial lighting in commercial spaces. Creating an algorithmic system that embodies a practical approach to develops and generate various reflected two-dimensional ceiling layouts, using various design approaches. Furthermore, allows for lighting calculations to ensure the selection of the optimal design.

It was found that (Kim et al. 2016) paper proposed a similar concept as this dissertation, however, it is much more like the available tools found in lighting calculations software. Thus, is it not filling the gap found in literature, as it is producing only uniform lighting layouts, and proposing a two to three layouts as solutions that matches the Korea Industrial standards.

The aim of this dissertation is to generate various reflected ceiling designs through a multi-algorithmic system. It works through selecting certain inputs by the end user to generate a large number of ceiling designs in a short time, while providing diversity in the generated designs in three design aspects: light patterns, used design approaches and lighting fixtures quantities. It has the capacity to produce various ceiling layouts using one design parameter only, in addition to producing layouts that combine different designs parameters at the same time. Not to mention that it is an interactive system that gives the designer's the ability to alter and fix the visual results immediately and turn on-off the appropriate design methods he/she desire.

## Chapter 3

### Methodology: Generative System Outline

### 3 Chapter 3: Methodology - The generative system outline

#### 3.1 Introduction

The generative system is created using Dynamo-Revit, which can be defined as a comprehensive programming application hosted by Autodesk Revit. It is an inserted plug-in that became embedded within Autodesk Revit in 2020 version.

The system was formed using two primary means, visual programming, and Dynamo textual programming. Due to Dynamo-Revit correlation; the end users will be able to insert the ceiling selection from Revit into Dynamo and run the created algorithms to generate various lighting layouts for the selected ceiling. The developed algorithm shown in Figure 9 can be defined as an interactive system that generates reflected ceiling results immediately based on the inserted variables. This system does not rely on one algorithm only, as it is a combination of various Dynamo-scripts that can be used individually as tools to create specific design outputs, or as a whole to generate various reflected ceiling designs outputs.

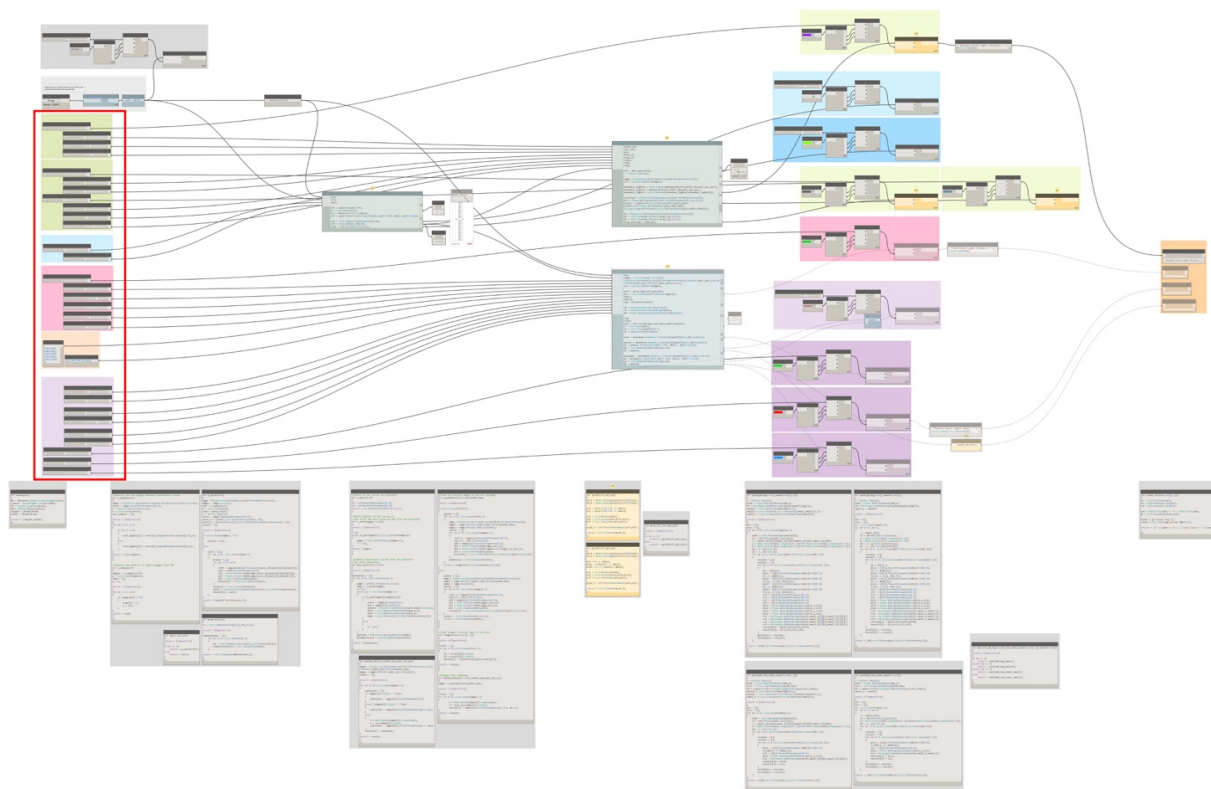


Figure 9: The final algorithm with the scripts and variables sliders created in Dynamo (author)



The red rectangular outlined in Figure 9 contains all the variables the designer can manipulate to produce different results. Every variable changed by the designer allows a different output to be generated.

This algorithm is a primary outline of what can be developed into a proper software that is able to produce all different design possibilities for different types of ceilings. Presenting feasible outputs in terms of the diversity in the mechanisms for selection, limitation and mutation of the outputs and sustainability implementation.

This chapter outlines the process of developing the generative system, it briefly discusses each subsystem and the different approaches that every subsystem has. It also gives an overview of the shortfalls and the limitations that every sub-system encountered. The detailing for every script is found in the next chapter. As this is merely an outline of what is the next chapter specifies in detail.

Figure 10 shows a comprehensive outline for the generative system, which aims to simplify the structure of the generative system. However, the algorithmic system has a high interaction level between its components that might not be clearly identified in the outline.

Relying heavily on visual outputs and instant results, this generative system displays an interaction section between human's creativity and the algorithmic production. Where great amount of flexibility in design is given to the end user, which vary based on the selected design approach. The system also provides the flexibility in altering results and updating different values using different system-interactions means, without the need for the designer to alter the algorithm scripts to update or change anything.

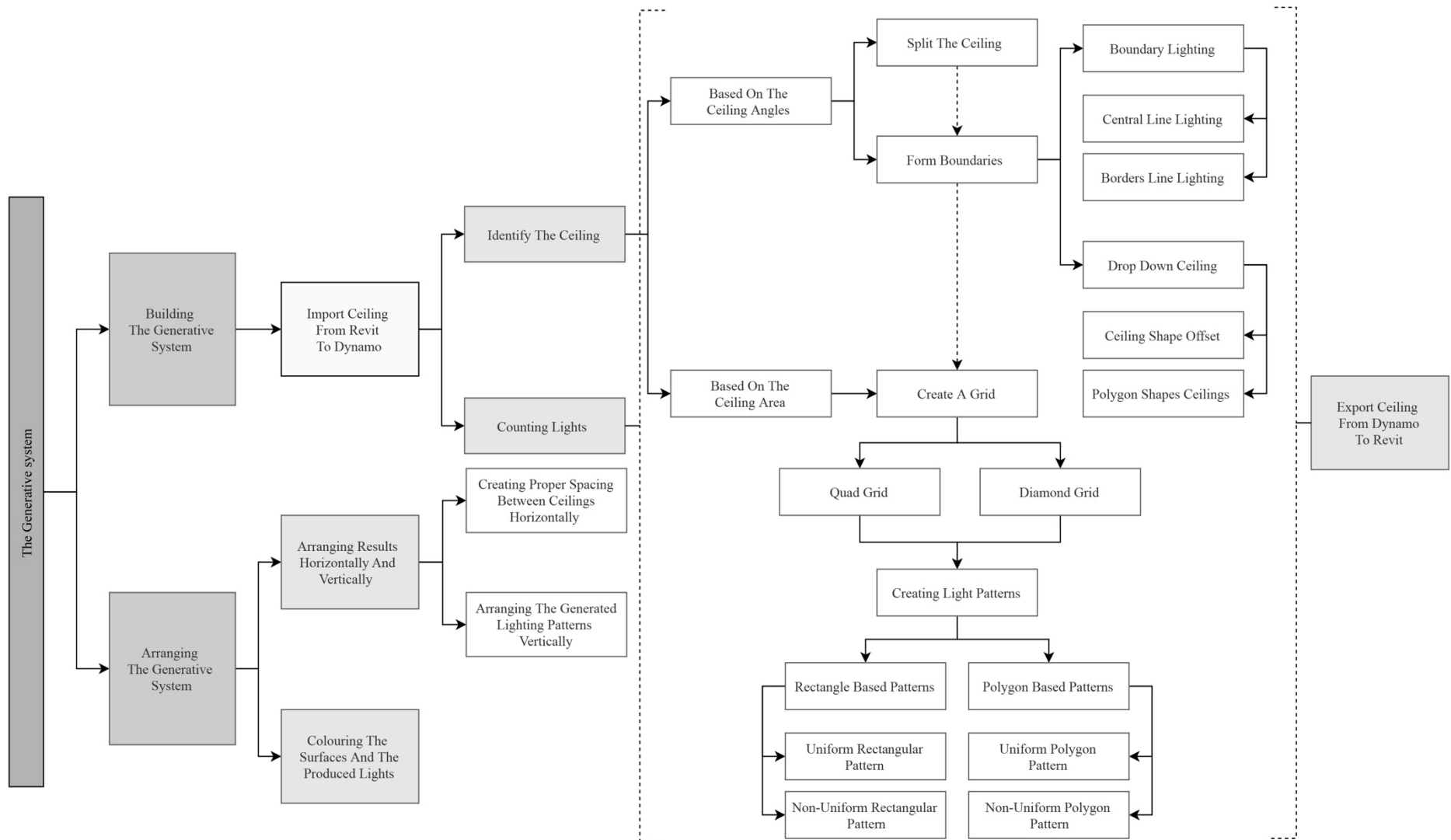


Figure 10: Outline for the generative system discussed in this chapter (author)

### **3.1.1 User-System interactions**

Arranging the algorithmic system was created through utilizing different interactive means. The default status of the generative system combines all the several functions of the subsystems into a single system that produces more than five hundred different reflected ceiling layouts. However, the end user can exclude auxiliary functions that support the main function, by turning them off, without affecting the performance of the main function. This approach was created to give the end user the flexibility to target certain designs approaches he/she desires.

Figure 11 presents all the different sliders that were created in generative system in order to achieve that flexibility. As they allow the end-user to select certain parameters, turning on and off other parameters, adjusting or altering different values and inputting other values to achieve certain dimensions; leading every alteration to connect to a selected script in the generative system and run simultaneously.

Every slider is connected to a script to create the interactive system between the designers and the algorithmic scripts. Allowing them to decide the option that fulfill their design needs. They ensure that the generative algorithm presents feasible outputs in terms of the diversity in the mechanisms for selection, limitation and mutation of the outputs and sustainability implementation.

The sliders influence different algorithmic tasks, such as the splitting slider that will divide the surface area into smaller areas, while switching it off will sustain the inserted ceiling as is and treat it as one entity. Figure 11 displays all the approaches such as boundary lighting, drop down ceilings, which starts from a simple offset of the surface to creating a decagon drop down ceiling. The algorithm is a complex system that aims to generate all the possible designs for the user to use.



## 3.2 The generative system components

### 3.2.1 Key Step: Importing a ceiling from Autodesk Revit to Dynamo

The first step in designing the generative system was to create an easy, user friendly transition between Autodesk Revit and Dynamo. To do so, 3 Nodes were created to make the Autodesk-Dynamo transition.

The model of the interior scene is designed in Autodesk Revit workspace, when activating Dynamo and importing the scripted algorithm, the first Node in the system will require to select a Model element. Where the end user will simply select the required ceiling. Figure 12 shows Autodesk Revit with Dynamo's workspace showing the red outlined Nodes that shows the ceiling selection. The Nodes will automatically isolate the lower surface of the selected ceiling, and transfer it to a surface, in order for the algorithm to process it to the other subsystems.

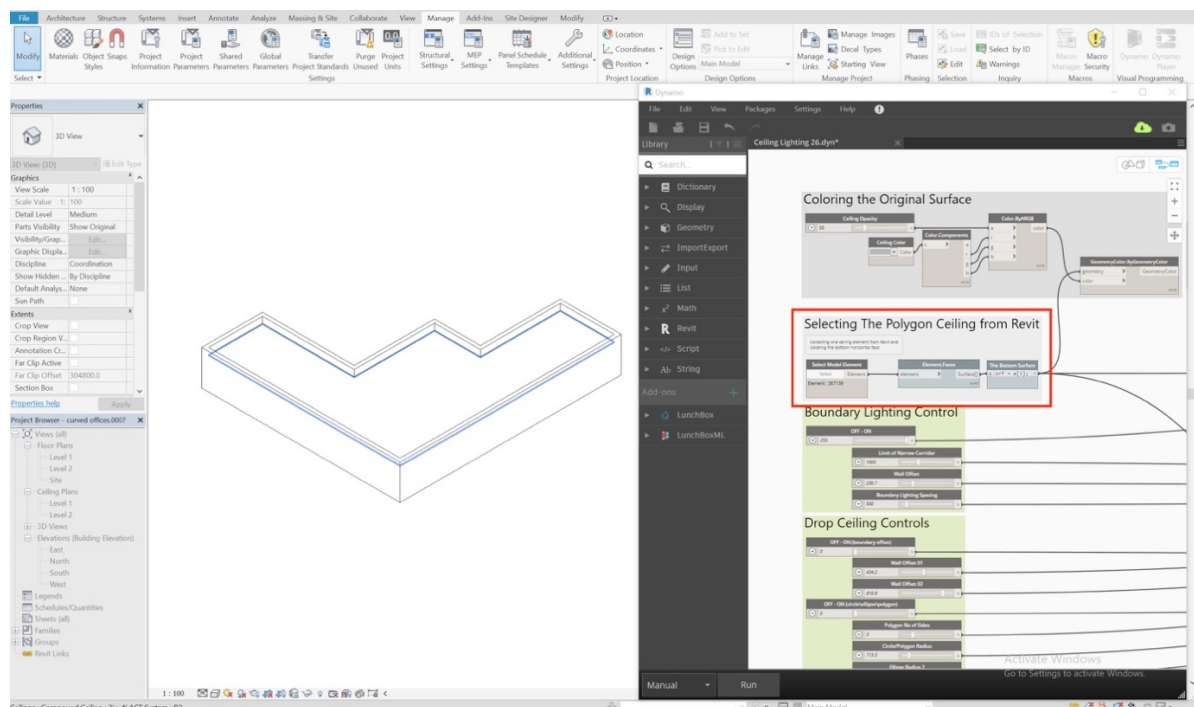


Figure 12: The transition between Autodesk Revit to Dynamo (author)

The only limitation in this approach, its inability to select more than one ceiling at a time. If the end user wishes to do so, different ceilings needs to be joined in Autodesk Revit then transferred into Dynamo.

### 3.2.2 Subsystem 01: Three approaches to identify a surface

One of the key issues that needed to be researched was the variety of shapes that ceiling designs can form. It is almost impossible to predict all various layouts and shapes that a designer might input to the algorithm. Writing a separate algorithm for every possible shape is not a practical process and might result in various errors. Therefore, it was important to figure out a logical way to address all the possible shapes that might be inserted, the best thinking strategy was to study all the common characters in shapes.

The surface's overall boundaries, the degree with the quantity of the angles, and the area that every shape has are considered the common elements throughout numerous surfaces that were selected in this dissertation. Different approaches were build using these three elements as a foundation, allowing the generative system to understand any given ceiling surface.

Utilizing the degree and the quantity of the angles any surface has resulted in creating two approaches: *Approach 01: Splitting the surface*, and *Approach 02: Forming boundaries*. While addressing the area of the surface created *Approach 03: Creating a grid*.

Since the first two approaches rely on the same parameter, a condition was created to prevent overlapping the generative results. The condition prevents splitting the surfaces if the surface has 4 angles or less, as it will just identify the surface using the boundaries it has. For example; in the case of a rectangular, square or a round layout, the algorithm will not run split the ceiling command, as it is considered as a simple shape that is suitable for applying lighting designs methods on it directly. Nonetheless, If the surface has more than 4 angles; the end user is given the flexibility to choose between the two different approaches.

The third approach relays on a different parameter, which is the area of the surface instead of the angles. Thus, the generative system gives the option to apply a grid regardless of the inserted surface's shape.

### 3.2.2.1 Approach 01: Splitting the surface

This approach splits the surface into smaller surfaces using the corners of the shape in both x axis and y axis, which leads to dividing the total surface into smaller surfaces and generates different outputs for the same layout.

The degrees of the angles in the surface is a key element in this approach, merely because it is easy to split lines from different angles to create shapes, resulting in potential spaces that simplify the shape of the surface, while indicating different zoning in an open plan layout.

The scripted algorithm was built on the following logical statement: If the measured angle in any corner of the surface is bigger than  $90^\circ$ , the shape will split from that corner creating a simpler shape.

This approach has three key steps to generate various splitting options: *Calculating the angles' degrees of the surface*, *Investigating the proper splitting sequence* and *Applying a division cut to the surface*. Figure 13 outlines these steps along with its sub-steps that are discussed in detail in every section.

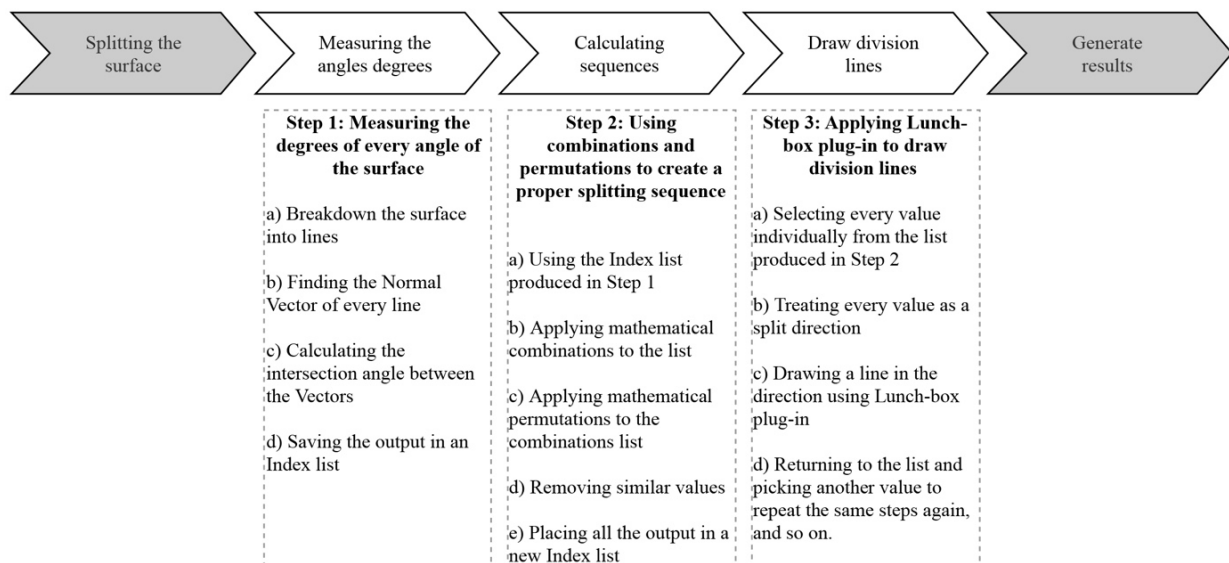


Figure 13: Splitting script key steps and sub-steps (author)

### 3.2.2.2 Approach 02: Forming boundaries

Dynamo's default approach to surfaces does not immediately form constant line or a boundary around the surface's shape by itself. It breaks the shape down into lines that are connected, with every line having its own individual characteristics.

Forming a proper constant boundary is a fundamental step and considered an important approach to identify the surface. It is essential, as it will allow the generative system to run any design inside the surface only. Hence, this approach can investigate the inserted surface and the attributes for each line forming it.

Since Dynamo does not give the user a direct information about every line's attribute, a testing method was created to investigate the characteristics of the connected lines. The testing method basically offset the lines that forms the surface, visually producing results that indicate the attributes of each line. If the lines are showing a random offset in multiple directions, it means that every line has a different direction. That problem cannot be ignored because it causes a lot of random issues further in the generative system, since this is one of the three approaches to of identify the inserted surface.

To fix the issue; lines were joined into a polygon. Resulting in unifying all the characteristics of them. After that, the polygon was exploded to revert the lines to their original status while sustaining the new attributes created by the polygon. Figure 14 outlines the basic steps for forming boundaries.

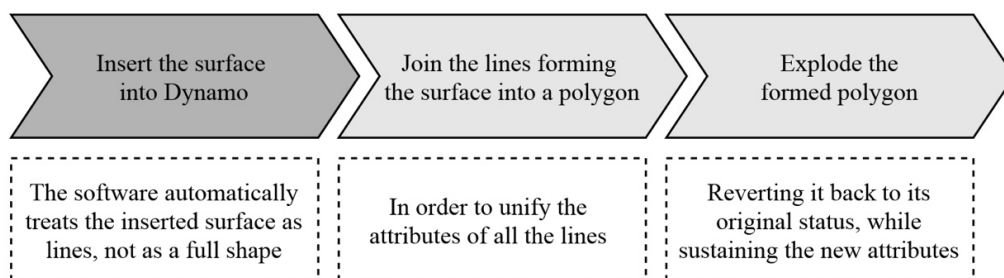


Figure 14: The steps in forming boundaries approach (author)



### 3.2.2.2.1 Boundaries sub-approach: Creating boundary lighting

After the offset test method was successful in *Approach 02: Forming boundaries*, it was developed into its own lighting design method. This method identifies the surface's boundaries and explode it. Then uses a value given by the end user as a wall offset, offsetting every border line towards the inside. Where the new offsetted line will be used to place lights coordinates around the ceiling, or as an indication to a drop-down ceiling.

In order to place light coordinates on the new offsetted line, the algorithm will check the length of each exploded offsetted line that forms the surface, and divide the length of it by the spacing between the lights which is determined by the end user. Figure 15 shows the result of boundary lighting method on different ceilings, the algorithm works on any shape or curved surfaces.

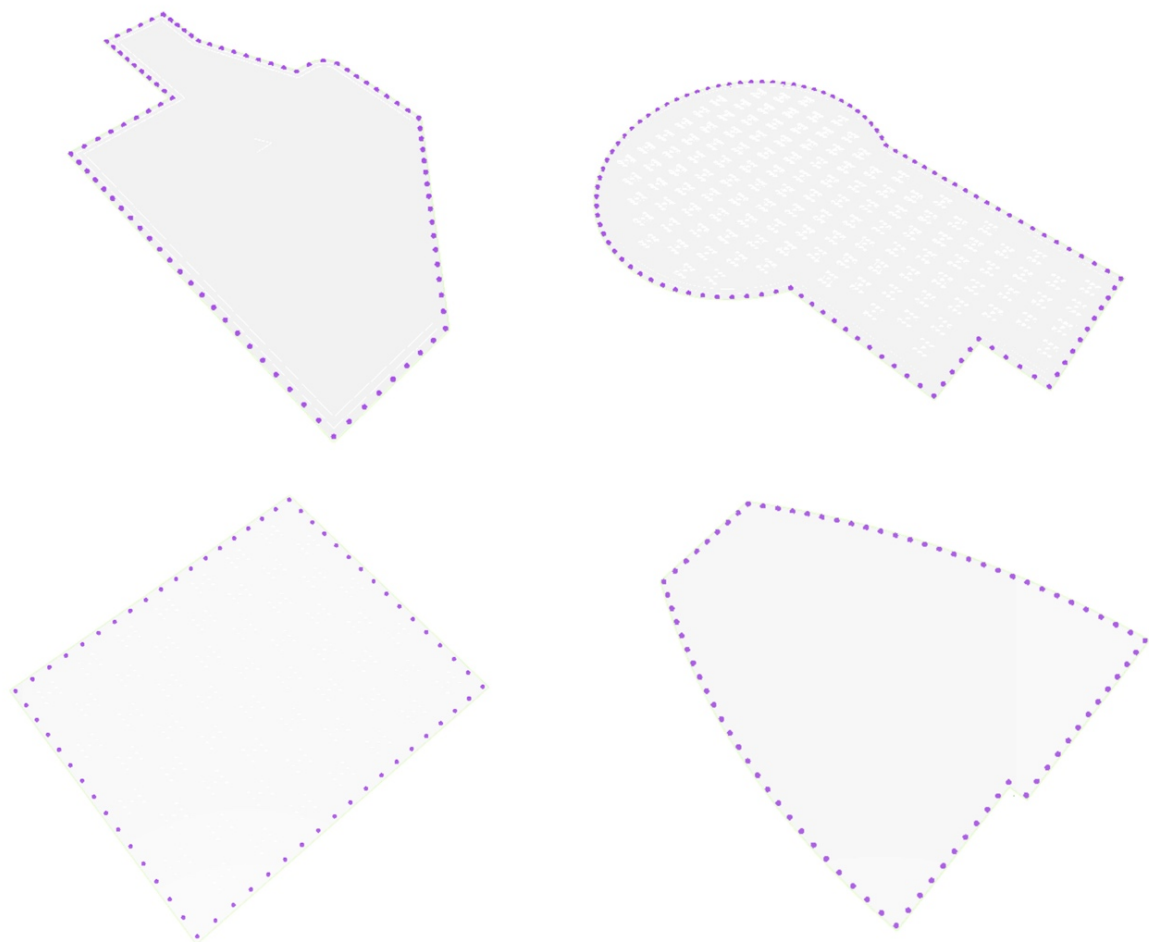


Figure 15: Different ceiling layouts showing boundary lighting method generated in Dynamo (author)

The generative system for boundary lighting is divided into three main Dynamo scripts. The first script runs the algorithm when the ceiling is not split and treated as one surface as seen in Figure 15.

While the second script works in the condition of splitting the surfaces, as an individual script was required to be written to prevent a reoccurring error that appears when the splitting approach is turned on. The error showed inaccuracy in dividing the spaces between the lighting coordinates, creating discontinuity in the light quantities between the original line that forms the surface and the added line that forms the splitting command from the surface.

Narrow ceilings and corridors were required to be considered individually. As it is not visually appealing nor practical to have two rows of lights on a narrow ceiling. Thus, in the case of a narrow corridors, a script was created to center a line in the middle of any corridor, which is discussed in detail in *Creating central line lighting for narrow ceilings*.

Figure 16 shows the scripts and the structure of creating boundary lighting in the generative system.

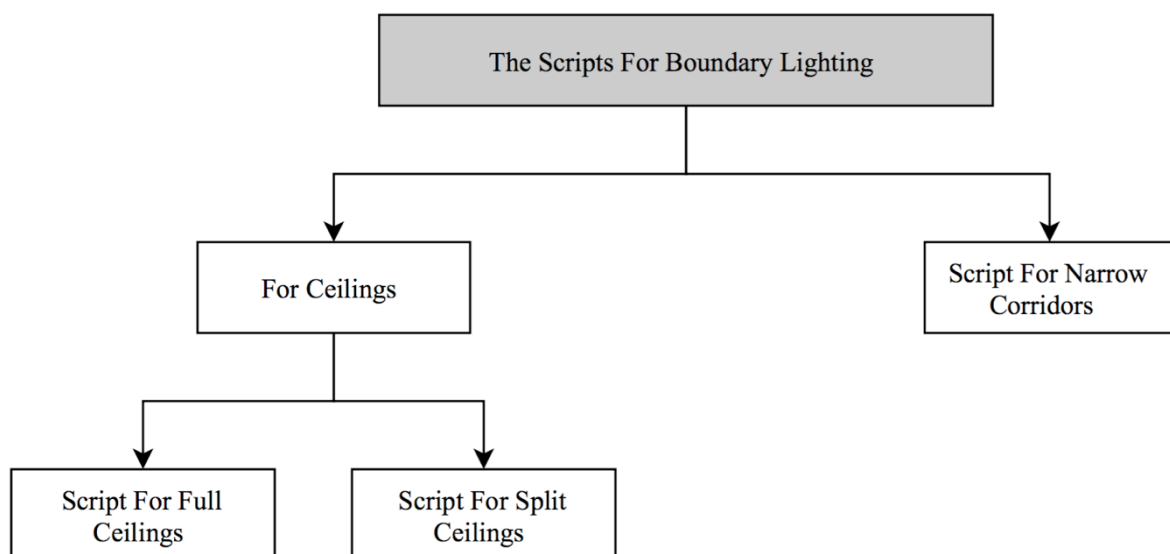


Figure 16: The scripts outline for Creating Boundary Lighting approach (author)

### 3.2.2.2.2 Boundaries sub-approach: Creating drop down ceilings

Drop down ceiling method was extracted from boundary lighting method. As it adds the option in creating various ceiling designs using two-dimensional geometrical lines instead of generating designs through placing light coordinates as the other methods. This approach can help the end user to indicate cove lighting and other geometrical designs for the ceilings.

This sub-approach generated two different key outputs seen in Figure 17. The first approach will offset the same surface shape inwards, twice if desired. Where the end user can input the desired space between the original ceiling wall and the new offset using different sliders.

While the second approach will create different polygon shapes in the center of the surface. The written script will identify the center of the inserted ceiling and draw a circle in using the center point. A circle can develop into an ellipse, then a triangle, quadrilateral, pentagon, hexagon, heptagon, all the way to decagon. It is the same approach that was done while creating polygon light patterns.

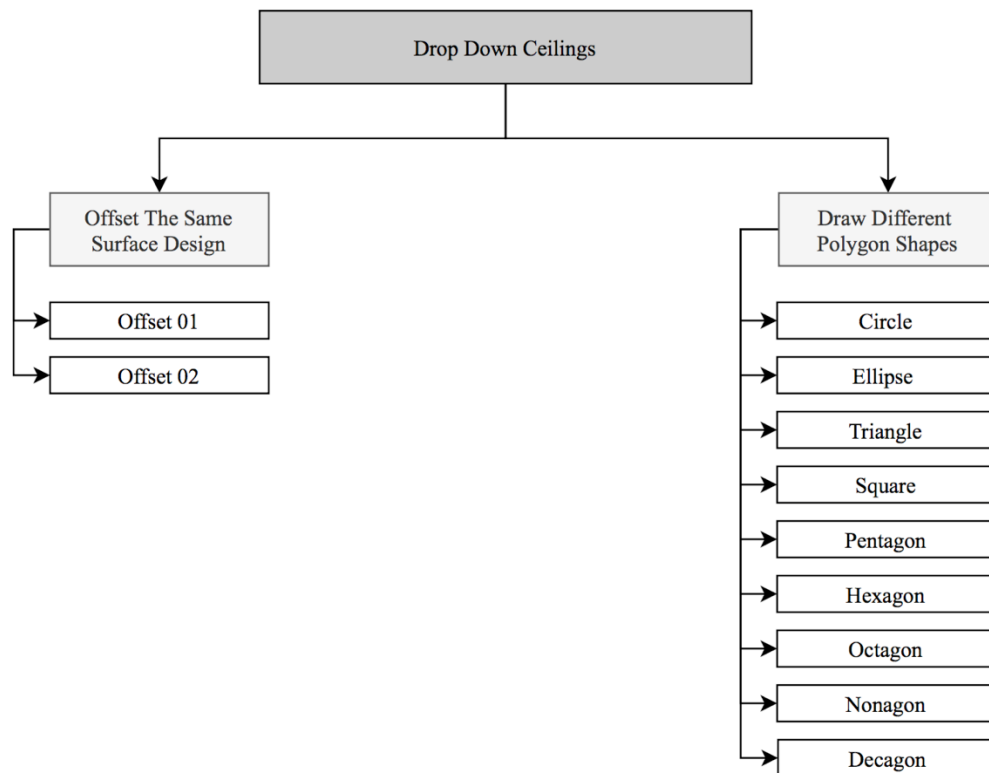


Figure 17: The outline for creating drop down ceilings (author)

### 3.2.2.3 Approach 03: Creating a grid

A grid can be identified as a pattern of horizontal and vertical lines, that usually forms shapes on a selected surface. Creating a grid was important as it serves a design approach that both Splitting the Surface and Forming Boundaries didn't address, it's created to serve lighting design in large spaces where the end user aims to have a uniformed distributed lighting coordinates throughout the ceiling. Creating a grid went through different stages that are mentioned in detail in Chapter 4: *Approach 03: Creating a grid*

The main issue in Stage 01 that resulted in scripting a new grid, is the approach the plug-in creates points through. As it was created to form geometric designs, not points, thus overlapping points occur as every 4-sided shape will have 4 points on its corners. Since the grid is a repetition of the same shape; overlapping points occur. The points do show as one point visually in Dynamo's workspace, but when connected to a light in Revit or to another Node; errors occur.

#### 3.2.2.3.1 Stage 01: Creating a plug-in grid

Stage 01 was through a downloaded plug-in called Lunch box; which resulted in creating 6 different types of grids: (1) Quad grid, (2) diamond grid, (3) hexagon grid, (4) Triangle grid shape A, (5) Triangle grid shape B, (6) Triangle grid shape C.

Although results showed an interesting variety, some types didn't produce the inserted numbers in U and V directions when testing the generated grids. As the visual output was half of the inserted input value, while other types produced the points as the input value. In addition, it only worked efficiently on four corners' surfaces, as errors showed when testing it on asymmetrical surfaces. Furthermore, the grid points were applied of the surface's borders regardless of the number of angles the surface had. *Shortfalls and solutions in the plug-in grid* discuss each shortfall in detail.

#### **3.2.2.3.2 Stage 02: Scripting a uniform/quad grid**

Stage 02 scripted a uniform/quad grid using mathematical calculations. In order to create a grid, the algorithm will assume an imaginary four angled surface that works like a boundary box to the inserted surface from Dynamo, then it will provide divisions for both x and y axis that creates the imaginary surface. After that, the intersected lines between the divisions are considered the coordinates that the points are placed on.

The grid in stage 02 relied on three inputs: the inserted surface, and two values taken from two numerical sliders called: Division 01 and Division 02 sliders. These two sliders provide the two directions of divisions for the grid that distribute itself out equally on both axes. The inserted value in them is controlled by the end user.

#### **3.2.2.3.3 Stage 03: Scripting a non-uniform/diamond grid**

Stage 03 showed the other design that was chosen to be scripted, the diamond grid. Where one point in the second line of the grid lies is in the center area of two points in the first line of the grid, creating a grid that is like a zigzag pattern.

The mathematical process that allowed this script to produce a proper diamond grid is more complicated than the quad grid. It was essential to find the proper formula that allows the second line to start at an offsetted value to create an accurate pattern. This is discussed in detail in *Scripting a new diamond/non-uniform grid design*.

A script was created to allow the swapping between both designs of the grid, it was created through an if statement, and connected to a slider that gives the end user 2 options. The selection of value 1 in the slider will produce the quad grid, while value 2 will result in the second produced type which is the diamond grid.

#### **3.2.2.3.4 Stage 04: Updating the scripts**

The grid is a fundamental step to approaching ceiling designs, as its intersection points are used as light coordinates to apply various uniform lighting patterns, and a key element in lighting distribution. Thus, various tests were conducted to ensure that it is working properly.

The previous finalized grid's algorithm ran through taking the two divisions input from the end user through two sliders called Divisions 01 and Divisions. Diving it based on the length and width of the ceiling. Which worked without any issues when applied on a one surface. However, since a section of the algorithm offers splitting the ceiling into smaller ceilings to create different zones in open plans layouts. Errors in the grid occurred upon splitting.

Testing showed different densities of lights in different split surfaces. Although the divisions inputs are the same. This issue occurred since the system distributes the inserted values on the length and the width of the surface. Hence, after applying the splitting option to the surface, the algorithmic system will take every length and width for every split surface and distribute the inputted values on those two variables. Resulting in creating ceilings with various densities while the inserted values taken from the sliders are the same for all the surfaces.

The new update the fed the generative system replaced the new values. Instead of Division 1 and Division 2 sliders, new sliders called Spacing in length and Spacing in width is fed to the system. This value is not the final amount of lights on a surface as the previous approach. It is now the amount of space between every light. Examples are discussed in detail in 4.2.2.3.6.

Table 1 shows the different stages of developing the grid with the advantages, disadvantages, and highlights the issues that were fixed in creating every stage. Due to its errors Stage 01 was removed and no longer used. And Stage 02 with Stage 03 were introduced, followed by Stage 04 update.

Table 1: The development of the stages of the grid (author)

Development stages of the grid	Advantages	Disadvantages	Highlighted issues in the stage
Stage 01: Creating a grid using a plug- in	<p>1- Offered 6 grid designs for distributing the lights</p> <p>2- Distributed the points without the need to script the calculations to the algorithm, thus saving time and effort</p>	<p>1- The coordinates by the points are uncontrolled as they don't always match the inserted U and V values. Not producing clear numbers for the lights coordinates in a surface</p> <p>2- Over lapping points that caused a lot of errors</p>	<p>1- Removing the points coordinates that appeared on the boundaries of the inserted surface</p> <p>2- Intersecting the surface with the grid to remove the points on the outside of the surface</p>
Stage 02: Scripting a new uniform/quad grid design	1- Produced a uniform grid	1- When merged with splitting the surface approach, the grid produced different light densities in each split ceiling	1- Finding a proper division approach for uniform lighting
Stage 03: Scripting a new non-uniform grid	2- Produced a non-uniform grid		2- Reversing the sequence in order to create different grid patterns that are able to alternate the two types of lights.
Stage 04: Updating the scripts	<p>1- Produced a uniform grid</p> <p>2- Produced a non-uniform grid</p> <p>3- The grid displays suitable light densities and distribution when merged with splitting the surface approach</p>	None	1- Replacing the grid calculations from relying on a slider that gives the number of lights on each line, to a slider that gives the spacing between each line in both axes

### 3.2.2.3.5 Grid sub-approach: Creating light patterns

After the grid was created, it was important to place lights on the points the grid had provided. The initial idea was to only place points as spotlight indicators in the axis intersections between x and y. However, this option was quite simple, so in order to allow the algorithm to generate various designs for uniform lighting, four scripts were developed to create all the possible shapes and movements using certain variables as inputs for lighting sizes.

The patterns are designed through a primary concept. The pattern is created based on a point; this point is taken from the axe's intersections in the grid. There are two primary types of patterns: the rectangle-based patterns, and the polygon-based patterns. Each type contains two approaches that generates patterns; uniformed and non-uniformed, Figure 18 shows the structure of the final four generated types.

True to their names, the rectangle based pattern uses an invisible rectangle to create a distribution of lights around the parameter of the point, while the polygon based patterns starts from a circle which gradual to a triangle, as the end user starts adding more sides using a given slider, the shape develops to a quadrilateral, pentagon, hexagon, heptagon, all the way to decagon.

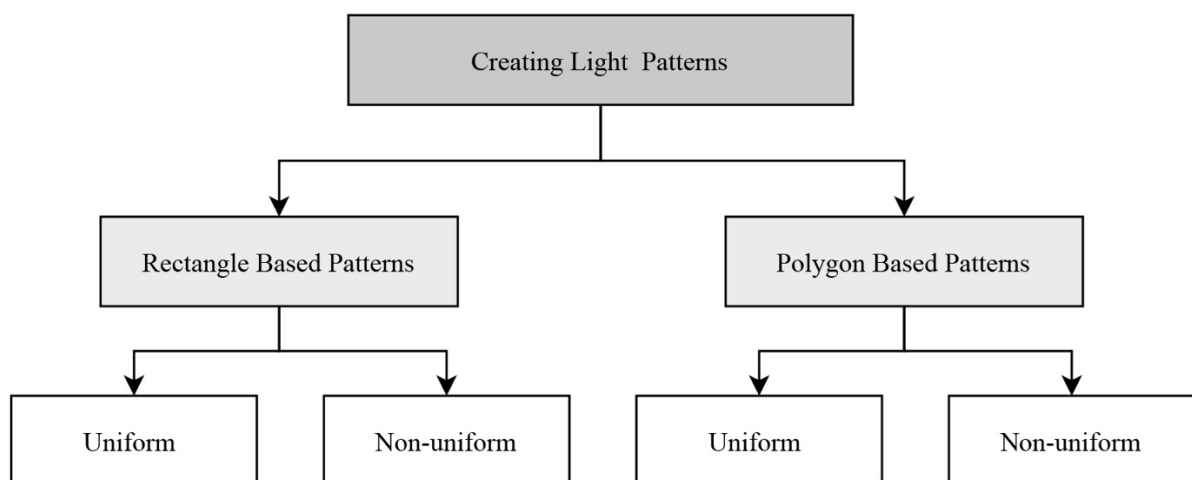


Figure 18: Light patterns outline (author)













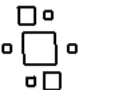


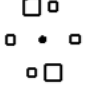




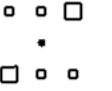

The main difference between the two categories is the uniformity in creating the patterns; as the rectangle holds the properties of having a different length to the width, while all the polygon shapes can be treated as having equal length.

This difference is the reason for creating two algorithms instead of developing a rectangle from the polygon algorithm. Where the rectangle script treats two equal sides, and the other two sides as one due to its properties. As for the polygon algorithm, the script addresses only one side of any shape and automatically repeats it on the other sides. However, since the circle has one consistent edge, the script treats it as one side only.

Table 2 shows a sample of the created patterns for the uniform rectangular patterns. The generative system had produced around 470 light patterns.









































































In addition, it produced 409 different non-uniformed rectangular patterns using the following light dimensions: 500x500, 300x300, 600x600mm in addition to adding points as well.

Table 2: Sample of the generated patterns for uniform and non-uniform rectangular pattern (author)

The Pattern	Image				
Uniform Rectangular Pattern					
					
Non-uniform Rectangular Pattern					
					

The produced patterns were around five hundred and thirty patterns for all the uniform polygon shapes. Table 3 shows a sample of them. And around two-hundred non-uniform polygon patterns that can be used as light patterns applied on a grid or as center lights.

Table 3: Sample of the generated patterns for uniform and non-uniform polygon pattern (author)

	Circle	Triangle	Square	Pentagon	Hexagon	Heptagon	Octagon	Nonagon	Decagon
Uniformed polygon patterns									
									
									
									
Non-uniformed polygon patterns									
									
									
									

### 3.2.3 Subsystem 02: Counting the generated light quantities

After producing all the lights using different generating methods, the quantities of each light type were provided. This procedure provides the counting for the end-user in order to create proper bill of quantities for the design. In addition, it saves time and optimizes the usage of the algorithm.

The counting of the lights was created through two approaches; the first approach was through connecting the produced results directly to a Node that counts the numbers of points, producing an accurate count. This was applied in Boundary Lighting and Forming a Grid approaches.

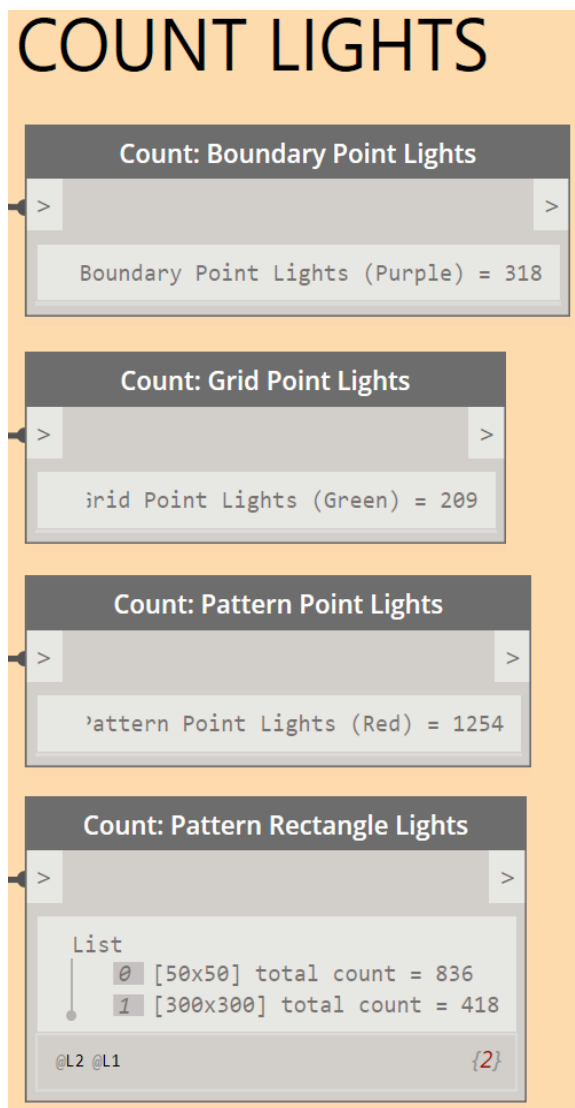


Figure 19: The Code Blocks that show the counting in the generative system (author)

The second approach had grouped the types of lights upon measuring the dimensions of each used light. This approach is used when the end-user switches-on Creating Light Patterns approach.

The light types were grouped as per their dimensions (300x300, 600x600, 100x300, 100x600, 100x1200, 50x50 millimeters).

These groups are the same groups that were inserted into the generative system to create lighting patterns. After grouping every category, a counting Node was connected to each group which produced the different counts as seen in Figure 19

### **3.2.4 Subsystem 03: Exporting the results back to Autodesk Revit**

After producing all the designs in Dynamo, results were carried to Autodesk Revit and connected to different light families that represented all the inserted light dimensions in Dynamo.

In order to do so, another grouping approach was scripted, that automatically resulted in creating a connection between Dynamo and Revit. As all the geometrical shapes that represents different lights in Dynamo are connected to actual lighting families in Autodesk Revit. And any alteration for the design that occurs in Dynamo will result into an automatic alteration in Revit.

### **3.3 Arranging the generative system**

Nodes were created to arrange Dynamo workspace to allow the end user to view the generated ceilings effortlessly, and control or alter the results efficiently.

Dynamo scripts and - if statements - were created to arrange the results in the workspace of the software for better visibility. Splitting the ceiling results were arranged horizontally while generated patterns of lights were arranged vertically.

Furthermore, different Nodes and scripts were created to color the surfaces and to the lights, creating an easier identification for every light approach.

#### **3.3.1 Arranging the results horizontally and vertically**

##### **3.3.1.1 Creating proper spacing between the generated ceilings**

In order to avoid the various generated results from intersecting on top of each other, a logical script with an - if statement - is created; allowing the ceilings to distribute horizontally throughout the workspace without overlapping as seen in Figure 20.

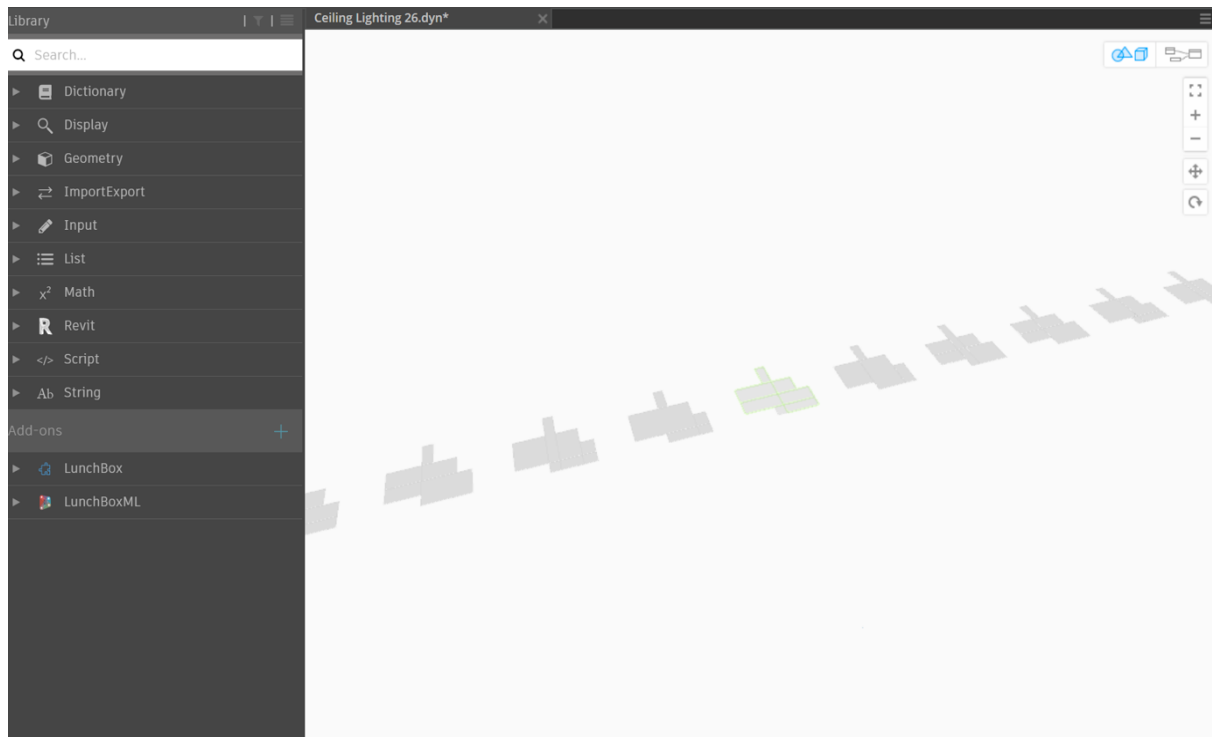


Figure 20: The arrangement of results in Dynamo for Splitting the ceiling design approach (author)

Figure 20 shows Dynamo's workspace after the arrangement script is applied; where only the selected ceiling option is colored with green borders to be distinguished.

It is important to note that the selected ceiling surface with the green borders will always be in the original ceiling location that was imported from Revit. So, when the end user applies the lighting results Dynamo and returns to Revit, they will be applied on the ceiling's location exactly and not on a random area.

### 3.3.1.2 Arranging and placing the generated light patterns on the original ceiling

In addition to arranging split surfaces' results, a cross section arrangement for the light patterns was created, giving a visual representation in Dynamo's work space, where both ceiling outputs and light pattern outputs are seen in the opposite axes, allowing the user to have an overall view of the results. Details of the arrangement are discussed in *Arranging and placing the generated light patterns on the original ceiling*.

### 3.3.2 Coloring the surfaces and the produced lights

Different Nodes were created to allow visual indication for every light type. Figure 21 shows the Nodes that were connected to the different scripts. Every color is shown in Figure 21 with the title of the connected generative method.

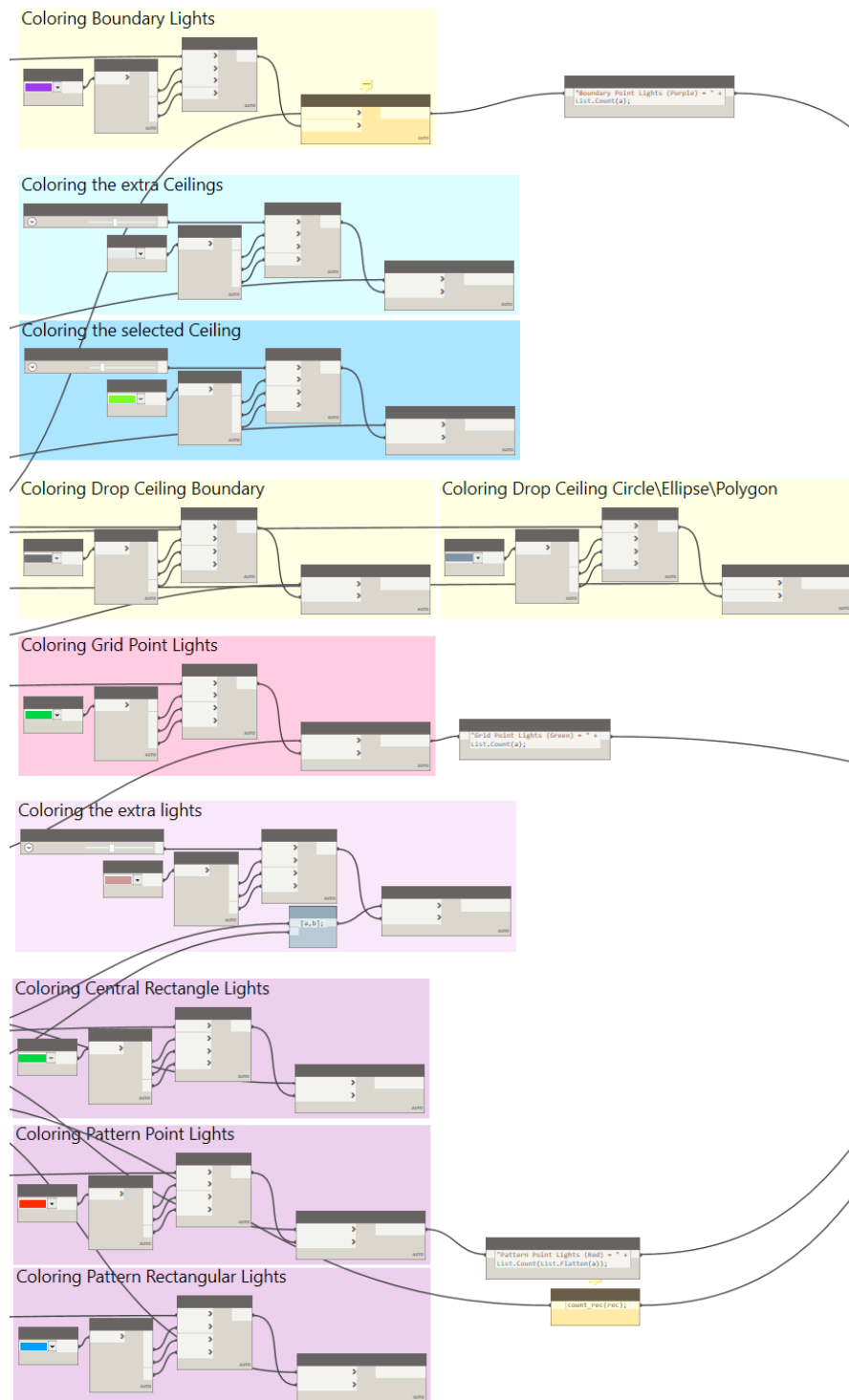


Figure 21: The different coloring Nodes for the different approaches (author)

## Chapter 4

### Development of the generative system

## **4 Chapter 4: Development of the generative system**

### **4.1 Introduction**

This chapter addresses the components of the generative design system and the development processes of it, which aims to generate various reflected ceiling designs through an independent approach that targets ceilings as a surface only. The system works through selecting certain inputs by the end user and running it produce various outcomes in a short amount of time.

Different approaches to investigate a ceiling layout are described and discussed, undertaking three primary design elements: the surface's overall boundaries, the degree and the number of angles, and the area every shape. Resulting into creating three foundations the algorithm runs on; splitting the surface into smaller surfaces; which can be used in the case of open ceiling layouts. Forming boundaries around the surface and applying lighting design methods directly on it. In addition to creating two types of grids enabling the placement of lights coordinates on the surface.

Various sub-approaches were built on these three foundations. Dynamo-scripts were written to generate four key types for lighting patterns using specified lighting dimensions. In addition, boundary lighting method is created for the end user to be able to place lights around any given ceiling design, or after splitting it. And manipulate various inputs that insures a required design. Furthermore, a script for drop down ceilings was created which develops from the original shape of the inputted surface to different polygon shapes all the way to a decagon.

The different algorithms were incorporated in an approach that reduced calculations time for each script and allowed the overall results for all the design methods to be produced at once. Creating a key outline for a generative system that can produce different ceiling design layouts and can be developed further into a proper comprehensive plug-in.



#### **4.1.1 Software background - Dynamo's anatomy and previous implementations**

The initial version of Dynamo was created by Ian Keough while he was working in Buro Happold Consulting Engineers in New York. The current software is owned by Auto Desk and developed to its existing version through its engineers. The first stable Dynamo version - Dynamo 0.6.3- came out in 2013 for public, and ever since; Dynamo versions had been rapidly developing to the latest downloaded version - Dynamo 2.0.3 - before including Dynamo in Auto Desk Revit 2020 (Dynamobim 2014).

The software can be defined as a comprehensive programming application hosted by Autodesk Revit; allowing developers and designers to create their customized tools using several programming means. Dynamo is mostly used as a visual programming application, where the instructions and the relationships between different elements in the software are approached through a graphical algorithm user interface. Nevertheless, textual programming can also be used in the application and will produce the same visual outputs (The Dynamo Primer 2019).

It is found in literature that Dynamo-Revit was widely used in addressing structural analyses as (Basta, Serror & Marzouk 2020) had developed a framework using the software to enhance the assessment of the de-constructability of steel structures, while (Sheikhhoshkar et al. 2019) had reduced the complexity of manual planning of concrete pouring in large construction project through developing a 4D Building Information Approach (BIM) that created automated concrete joint positioning solutions, and much more.

Dynamo had also shown that visual programming is able to support sustainable design studies in the primary stages of design by computing certain metrics relating to energy such as, solar gain and air movement (Nastasi et al. 2018), to simulate and investigate the relationship between the future building and the environment (Peters 2018). Studies such as (Lim et al. 2019) had used a multi objective optimization algorithm using Dynamo-Revit to achieve

44.78% reduction in overall thermal transfer value (OTTV) and 19.64% incline in the construction cost by creating an algorithmic Building Information Management (BIM) based optimization model. Furthermore (Gao et al. 2019) had developed using Revit Auto Desk and Dynamo an energy simulation and optimization BIM-based building system, helping architects to optimize the building form, orientation and window to wall ratio in the early stages of the design, through using the building energy performance.

Furthermore, Daniel Knott an associate in Buro Happold had developed the [Dynamo Master Planning Tool] that investigates key building form parameters such as building height and typology, alongside different mechanical, electrical and public health systems. Resulting in a direct effect to various sustainable metrics such as passive ventilation, external shading, and others (Peters 2018).

The anatomy of Dynamo is shown in Figure 22 . It is formed using two primary components (Nodes and Wires) that connect to create and run a visual program.

Nodes can be outlined as the commands the developer and designer use to perform an operation. Due to the embedded library in Dynamo and the available packages created by Dynamo community; the operations that are performed are numerous. Starting from a simple mathematical equation to a complex command that generates geometry in the workspace.

While Wires are the connections between nodes that results in creating the visual relationship and establishing the logical flow that the developer or the designer require. They are similar to electrical wires that transports the data and builds on the commands to create a certain output.

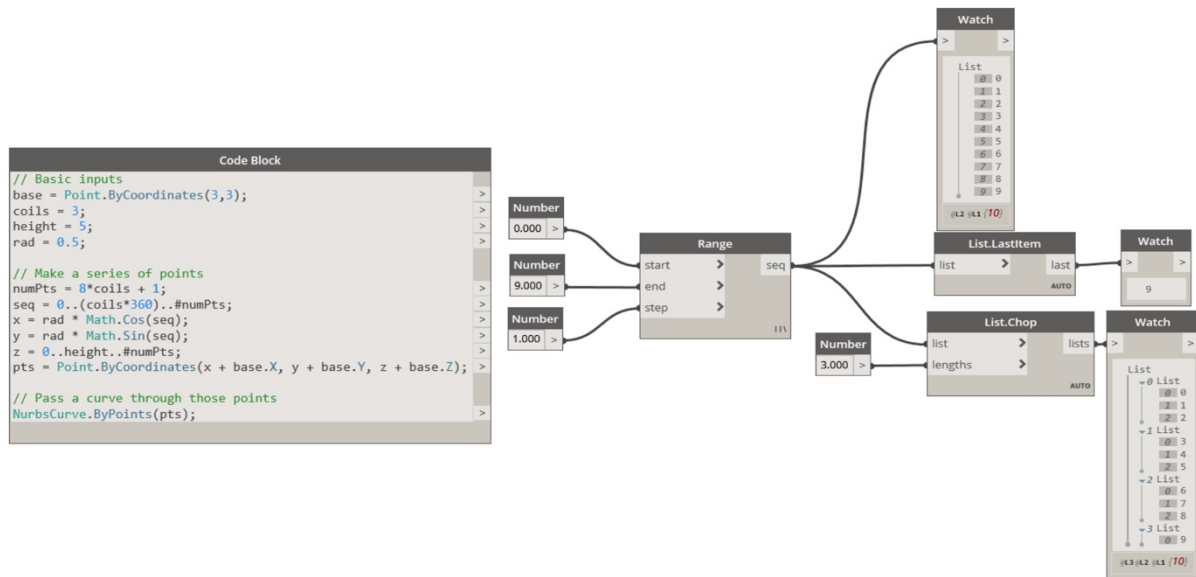


Figure 22: Nodes and Scripts that are generated with sequences and ranges (Dynamo 2.0.3)

Dynamo allows writing algorithm scripts directly using certain Nodes called Code Block displayed in Figure 22, where the scripted algorithm will run and produce a helical curve at a base point. The software also supports Iron Python - which is a programming language - by using one or two types of Python Nodes.

A direct relationship between visual programming and textual programming can be created, to avoid the increased number of Nodes and Wires that might lead to complicating the logical visual flow in the workspace.

The written Code Blocks can be referred to using the visual Nodes, if the long as Code Block containing the algorithm script is in the same workspace. This relationship is showcased in this dissertation while writing the proper algorithm script in a lot of design methods, allowing the workspace not to clutter with wires and Nodes.

#### **4.1.2 Justification for selecting Dynamo-Revit as a programming software**

Dynamo-Revit is often the choice for building simulations and building performance analysis by various studies in the literature review. Its high degree of flexibility in terms of the features that can be modeled and ability to manipulate the parameters of Revit Auto Desk to create a new level of associativity and generate various opportunities for cross-platforms (Lim et al. 2019) and customized form-generating algorithms had made it a popular choice along with Grasshopper, and Marionette.

It is the only used visual programming plug-in for Autodesk Revit. Compared to Grasshopper - another visual programming software that's widely used – that is identified as the visual programming plug-in to Rhino. Both software provide a similar variety of programming advantages, as both of them uses various scripting languages, both of them includes Nodes and Wires as a mean for visual programming, and in most cases, both of them are able to create, manipulate and generate various results based on the scripted commands.

However, the characteristics of Dynamo and the characteristics of Autodesk Revit Dynamo were the key contributors for selecting Dynamo as the visual programming software for this dissertation. As Dynamo offers a range of scripting methods between visual programming and textual programming, which was utilized in this dissertation. The merge between both scripting approaches created high flexibility and ease compared to other software. Although Dynamo has the ability to script different commands using various programming textual languages as the other present cross-platforms, such as Java and Python. It offer the ability of scripting using Dynamo-scripts, which was used to create textual scripts for the design approaches. This language is much simpler to script compared to Python and Java.

In addition, one of the main reasons for the selection of Dynamo-Revit is the presence of one character that Autodesk Revit provides compared to the other software. Autodesk Revit is able

to identify its model components, such as ceilings, walls, floor, and different furniture, lighting families. It holds definitions with-in itself that enables the end user for easy selection of components in any model. While other software on the other hand doesn't offer this feature. This is a critical feature since this dissertation key concept is to provide designers with a flexible generative approach that creates ceiling lighting layouts. The first step of running the algorithm will be selecting the ceiling. If the designer is not able to separate the ceiling component, the whole approach fails. Hence, Dynamo was the selected visual software for this concept. This important feature allows the end user the ease of selection.

Furthermore, Revit offers a key feature in providing lighting families to the end-user. It offers existing light families with its own characteristics in its library, it also allows the users to insert specific lighting families as an IES format. This format is usually available with any lighting supplier. This is a vital feature in this dissertation as it gives a great flexibility for the designer to select whatever lighting family with its specific characteristics he/she desires.

Not to mention that creating an algorithmic Dynamo-Script in a plug-in format makes it very appropriate for designers and architects to include it in their original models. As users will only require to open the created algorithmic script and run it on their original model created in Autodesk Revit in order to provide a variety of results that can be adjusted and manipulated upon creation based on specific aims and goals the designer aims to create.

The discussed characteristics that Dynamo offers along with Autodesk Revit had allowed this dissertation to achieve its aims and objectives, where the usage of other visual programming software might have resulted in creating unfixable difficulties that would work as a barrier that prevents reaching of notion of this research.

## 4.2 The Algorithms - description, development, and shortfalls

The selected approach to generate various lighting layout designs neglects the relationship between furniture and luminaires, and investigates the different shapes of the ceiling surface, and the connecting walls to it.

Figure 23 shows the level of interactions across the components of the algorithm. The dynamic interaction between these design approaches are the key element in generating a high number of different designs. Resulting in an outcome of multiple generative layouts and designs. Figure 23 shows four approaches the algorithm utilizes for generating ceiling designs, splitting ceilings approach was not added due to its complexity in interacting with all the different approaches. While Figure 24 outlines the functionality for the design approaches created in this system.

Furthermore, every subsystem has multiple approaches and different options in it, that results in further interactions and further possibilities. They are briefly discussed in this chapter, along with the main issues that were faced and solved in each one.

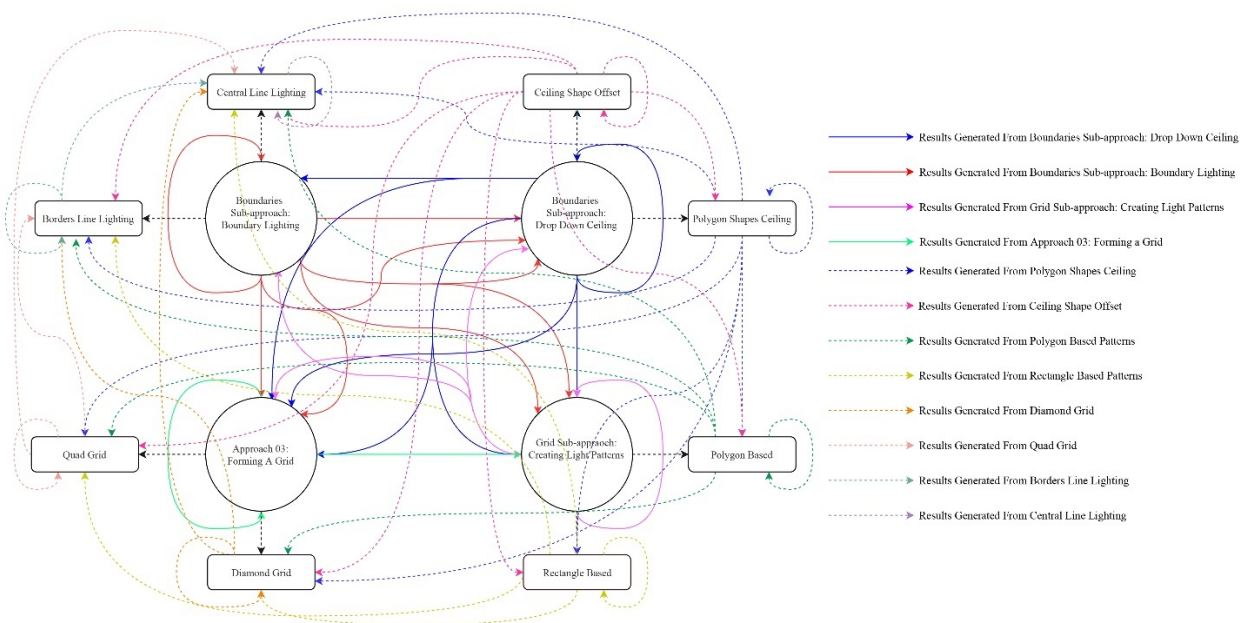


Figure 23: The Interaction level between different design Sub-approaches and their types (author)

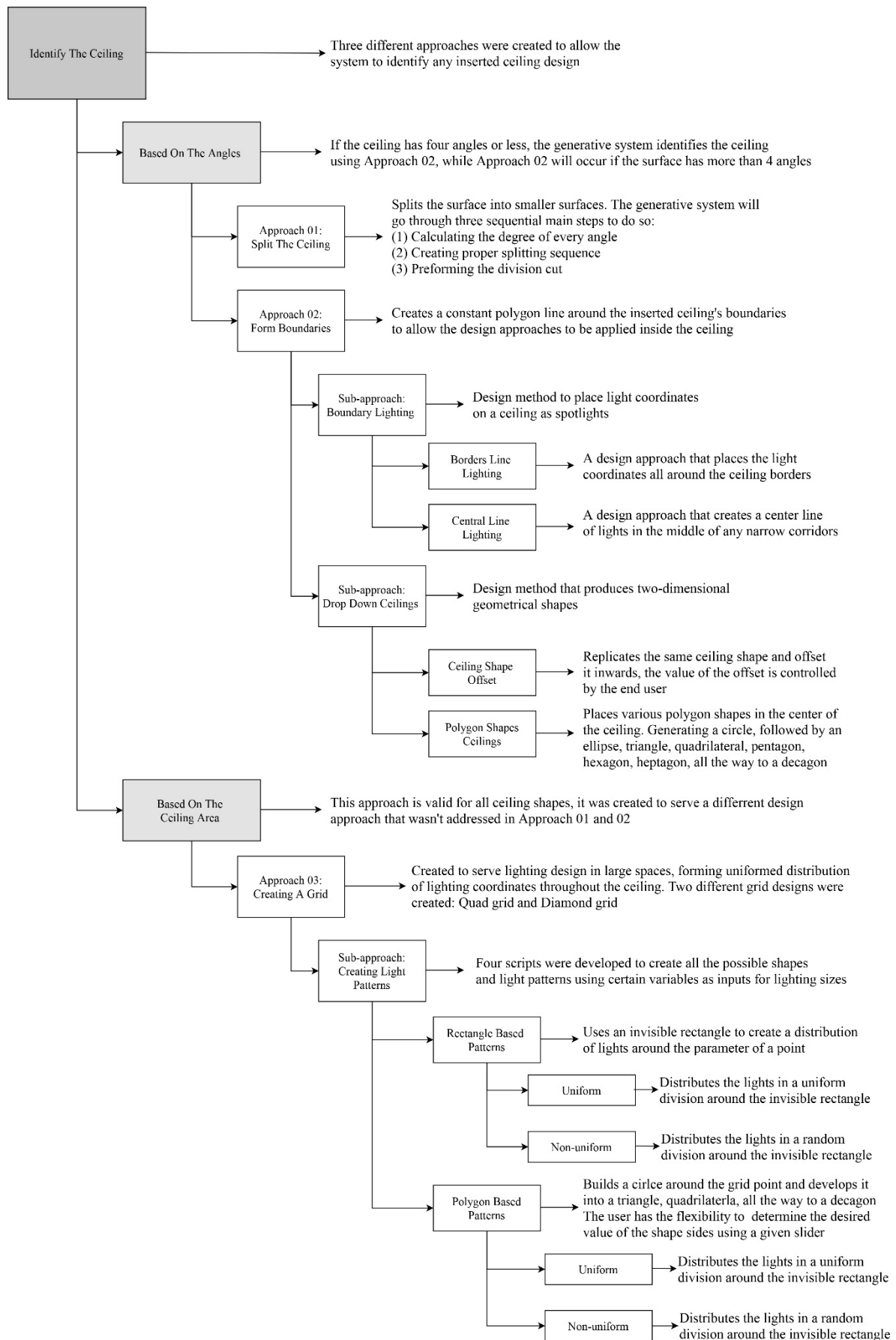


Figure 24: Outline for the scripted design approaches and their functionalities (author)

#### 4.2.1 Key Step: Importing the ceiling from Autodesk Revit to Dynamo

Autodesk Revit can identify and work with different structural elements, whereas Dynamo works with surfaces regardless of the type of the surface. An easy transition is created to enable selecting any ceiling from a model created in Revit to Dynamo. The first Node of the algorithm allows the selection as seen in Figure 25. However, since the ceiling element is three-dimensional, another step was required to isolate the bottom surface only, where the location of the lights can be identified.

After various tries on various models, it was found that the second surface placed in the list which transformed the inserted ceiling element into multiple surfaces - shown in Figure 25 as (1)Surface - was to be the bottom surface in the majority of cases. Thus, it was isolated as an individual two-dimensional surface using another Node, allowing the various applications of lighting design methods on it.

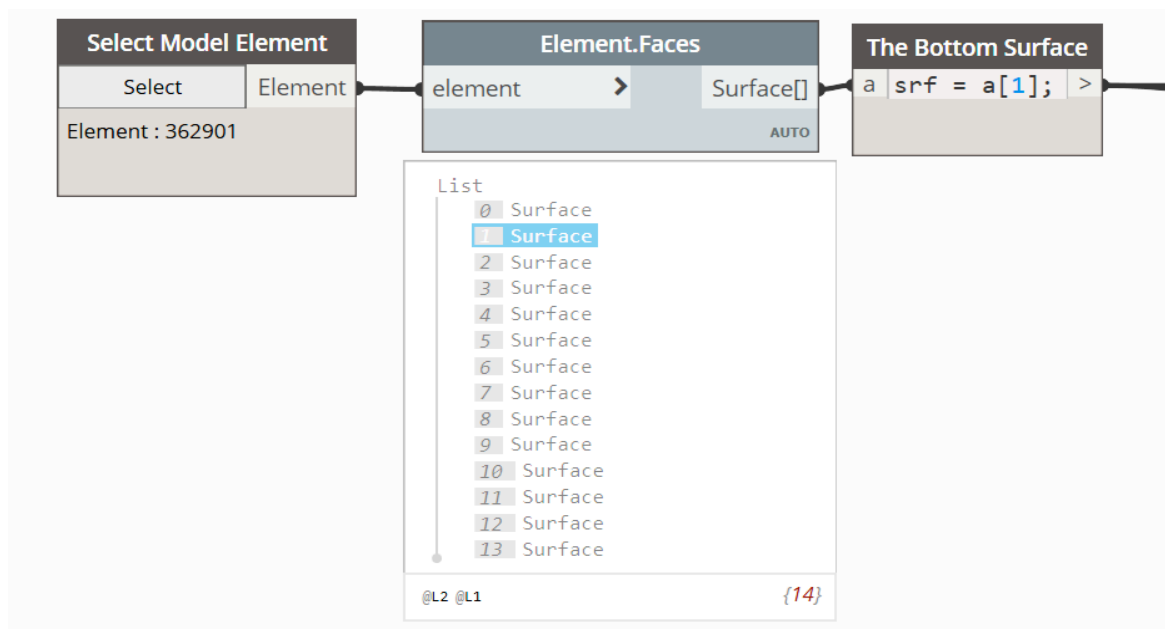


Figure 25: Dynamo Nodes that collect one ceiling element from Revit and isolating the bottom horizontal face (author)

This method will only insert one selected ceiling surface at a time and will run the entire algorithm on the inserted surface only. For example, if the layout has an open area ceiling and closed rooms such as Figure 26. The designer will be able to only select one of the available



ceilings, either the open space area or one closed room ceiling and so on. Allowing the algorithm to run on the selected ceiling to produce design options.

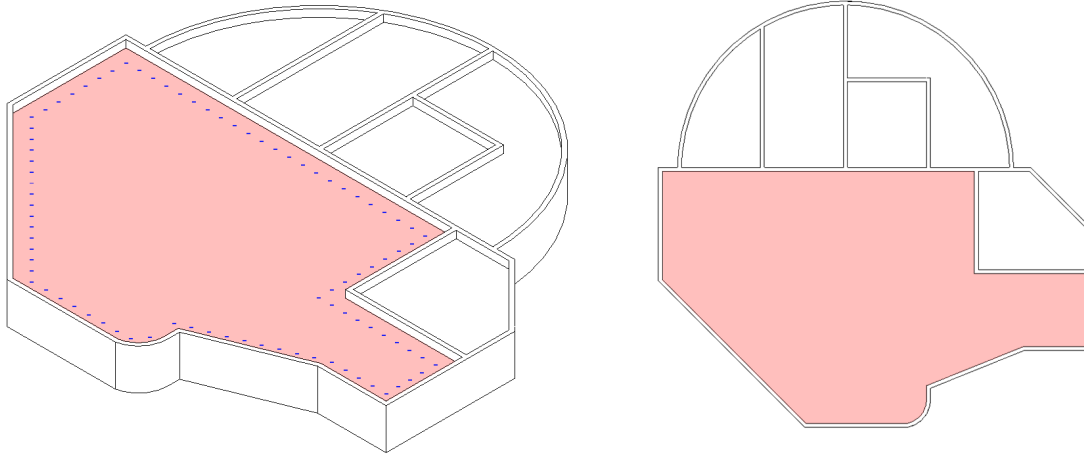


Figure 26: Selecting one ceiling in Revit-model to transfer to Dynamo (author)

#### 4.2.2 Subsystem 01: Three approaches to identify a ceiling

Three elements are considered in this dissertation as the key elements to address any ceiling design: (1) The surface boundaries (2) The degree and the quantity of the angles the surface has (3) The area of the surface, which created three approaches for the algorithm to investigate any inserted ceiling design; the approaches are discussed in details in *Approach 01: Splitting the surface*, *Approach 02: Forming boundaries*, and *Approach 03: Creating a grid*.

If the surface has four angles or less; it will just identify the surface using the boundaries it has; Which is discussed in detail in later in section *Approach 02: Forming boundaries*. While splitting the surface command will occur if the surface has more than four angles. The steps that were made to reach splitting a surface, along with the issues it had are mentioned in following section.

If the surface has more than four angles; the end user is able to choose between the two approaches that investigate the surface from its angles, he/she is given the flexibility to choose

to what suits the design's vision. For the end user to be able to make the choice, a scripted algorithm for the selection is discussed under *Arranging the generative system*.

#### **4.2.2.1 Approach 01: Splitting the surface**

Different attempts analyzing the surface were conducted and resulted in a technique that worked on almost every ceiling design. It became one of the key initial steps that the algorithm performs prior to applying any scripted algorithm that results in placing light coordinates on the surface.

In order to split the surface into smaller areas; the generative system will go through three sequential main steps: (1) Calculating the degree of every angle of the surface, (2) Creating a proper splitting sequence, and (3) Performing the division cut.

Each one of these steps are has its own sub-steps that are explained in details in its own sections, *Calculating the angles' degrees of the surface* has four sub-steps the scripted algorithm performs in order to get the degree of all the angles of the inserted surface, which is explained in Figure 27.

While *Investigating the proper splitting sequence* has three sub-steps: The first step *Creating a proper splitting sequence using permutation*, explains the first approach to create sequences, followed by *Permutations shortfall and using combinations as a solution*, which is a good approach to create sequences. And it was optimized in *Merging between permutations and combinations as an optimized approach* section.

The final step was to draw a line on the inserted surface to indicate the different splitting options, which occurred through downloading a plug-in called Lunchbox to Dynamo (Nathan 2013). It is further discussed in detail in section *Applying a division cut to the surface*.

#### 4.2.2.1.1 Calculating the angles' degrees of the surface

Figure 27 displays a simple surface created to explain the steps the algorithm takes to calculate the degree of the angles in any shape.

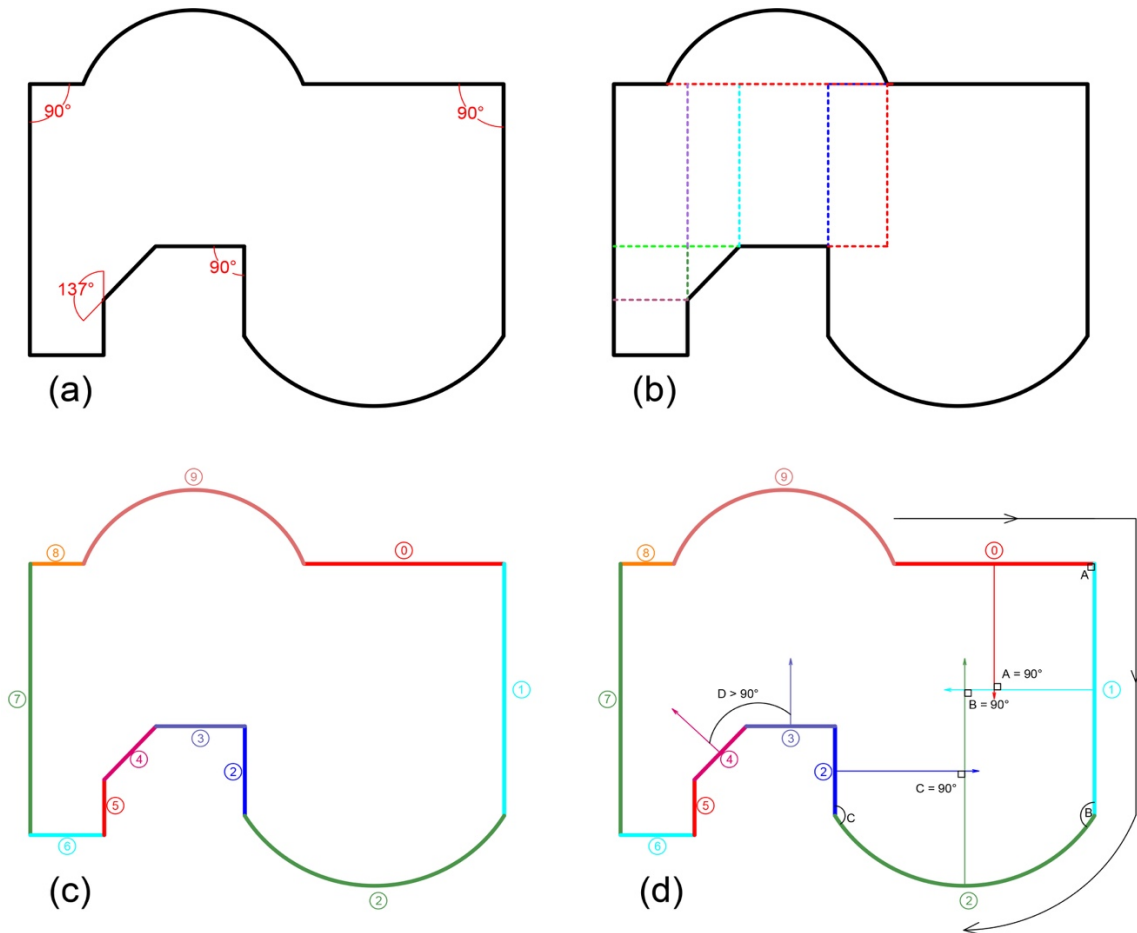


Figure 27: The steps to calculate the angles of the surface (author)

(a) Shows the surface which contains different lines with different angles such as a semi-circle polygon in some areas, and a diagonal line in a corner, along with different angle measurements shown.

(b) Displays the different possibilities of splitting the surface, showcasing the logical part of splitting the surface from a corner or an edge rather than from the center of a line. Those various possibilities shown in different colors have a correlated relationship with the number

of the corners the surface has. The more  $> 90^\circ$  corners the surface holds, the more generative options the algorithmic system will produce.

Point (c) in Figure 27 illustrates the advantage of Dynamo used in this technique, as the software will break down the surface into lines referred to in numbers and different colors in Figure 27 and combine them together in a list called Index, zero being the first number in Index. It's important to mention that every line has a direction that is linked to its coordinates, for example; if the line's coordinate is (0,1)(0,2), it means that line's direction is from left to right.

(d) shown in Figure 27, explains how the algorithm calculated the angles between each line. The software will start at Index 0 and move down the list. Checking the Normal Vector of every line. The Normal Vector is defined as  $90^\circ$  projection vector that comes from the center of the line, it is perpendicular from the direction of the line (The Dynamo Primer 2019).

When two consequent Normal Vectors from two lines intersect, the degree of the angle of the intersection is calculated, the same angle is usually reflected on the corner between these two lines. For example; Figure 27 shows that A angle =  $90^\circ$  calculated upon the intersection of (0) line Normal Vector and (1) line Normal Vector from the Index. Whereas line (3) and line (4) has an angle that is bigger than  $90^\circ$ . However, B angle will be calculated as  $90^\circ$ , since the Normal Vector will come from the center of the semi-circle line, not from the corner.

The textural writing for calculating the angles is displayed in Figure 28. It was essentially created from visual Nodes and Wires, then transferred into textural script to reduce the clutter of the workspace in Dynamo.

```

Calculate Angles Between Consecutive Edges
//Returns all the angles between consecutive curves
def n_angles(srf)
{
edges = PolyCurve.ByJoinedCurves(srf.PerimeterCurves()).Curves();
vects = edges.NormalAtParameter(0.5);
z = Vector.ZAxis().Reverse();
n = List.Count(vects);
vect_angles = [];

return = [Imperative]
{
for (i in 0..n-1)
{
if (i <= n-2)
{
vect_angles[i] = vects[i].AngleAboutAxis(vects[i+1],z);
}
else
{
vect_angles[i] = vects[i].AngleAboutAxis(vects[0],z);
}
}
return = vect_angles;
}
}

//Returns the indices of angles bigger than 90
def n_index(srf)
{
angles = n_angles(srf);
n = List.Count(angles);
indx = [];
j = 0;

return = [Imperative]
{
for (i in 0..n-1)
{
if (angles[i] > 91)
{
indx[j] = i;
j = j+1;
}
}
return = indx;
}
};

```

Figure 28: The scripted Code Block for calculating the angles between consecutive edges in Dynamo (author)

After the algorithm calculates all the angles of the surface, it saves the outputs in a list that will go to the second part of the process. The list is organized as the sequence of Indexes mentioned earlier.

#### 4.2.2.1.2 Investigating the proper splitting sequence

##### 4.2.2.1.2.1 Creating a proper splitting sequence using permutation

The second step is sorting out the values of the gathered angles. The algorithm will sequentially check the given list from the first step, starting with Index (0). If the value of the angle is not more than 90°, the algorithm will just ignore it. If it is more than 90°, it will attempt to split the

surface from that corner. The splitting will be from two directions only: x and y axes. However, the positive or negative direction of each axis will be determined based on the suitable direction that goes inside the boundaries of the surface. As there is no point to split in a direction that will only redraw the existing surface's boundaries.

Each angle that is more than  $90^\circ$  has two directions, hence two possibilities of splitting. The scripted algorithm will combine the possibilities of splitting in each direction using mathematical permutation for the angles. For example, if the shape has five angles that are more than  $90^\circ$ , each angle will have (Direction1, Direction2). So Angle-1 will have (d1,d2), Angle-2 will have (d1,d2), and so on. The algorithm will run through the angles and create different possibilities of splitting. The first possibility will split to Direction1 only in all the angles of the shape. So, Angle-1 and will be split from d1, Angle-2 will also be split from d1, and so on. Creating a splitting sequence like this: (d1,d1,d1,d1,d1).

Then, the algorithm will run again and focus this time on Direction2, repeating the same thing where it will split all the angles to Direction2. Creating a splitting sequence like this (d2,d2,d2,d2,d2). After that, the algorithm will run again splitting only Angle-1 to Direction1, and splitting all the remaining four angles to Direction2, resulting in a sequence like this (d1,d2,d2,d2,d2). Next, it will run again, and split Angle-1 and Angle-2 in Direction1, while keeping the other angles to Direction2, creating a sequence like (d1,d1,d2,d2,d2). The algorithm will keep looping until it reaches all the possibilities of the sequences.

In other terms, the algorithm is calculating the permutations of the angles. For the given example, the permutation of 5 is  $5 \times 4 \times 3 \times 2 \times 1 = 120$ , so the algorithm will produce 120 possibilities on how to combine the splitting angles, to get all the options to divide the ceiling.

#### 4.2.2.1.2.2 Permutations shortfall and using combinations as a solution

One of the issues faced in using permutations, its inability to replicate any values. Although it is an effective method since it picks each value and combine it differently with other values. The method had failed in splitting surfaces that have two or one angles above 90°, because it does not allow any values to be repeated in the sequence. So if a surface has two angles that are more than 90°, using permutations will result in two splitting options only;  $2 \times 1 = 2$  these two options will split in two directions that are (d1,d2)(d2,d1). Whilst in reality the possibilities of splitting two angles will result in four options (d1,d1)(d1,d2)(d2,d2)(d2,d1). That's why combinations were added to the scripted algorithm, because combinations are able to pick the same value twice. Dynamo has a default Node in its library for combinations. But when it was tested, the default Node was not providing all the possible combinations, because it wasn't shuffling the values as it should.

```
Combinations for arranging the Splitting Edges
def perm_lst(itt,lim)
{
p1 = List.Combinations([0,1],itt,true);
p2 = List.Combinations([1,0],itt,true);
p0 = List.Join([p1,p2]);
permutations = List.UniqueItems(p0);

return = [Imperative]
{
combinations = [];
i = 0;
while (i < lim)
{
perm = [];
for (j in 0..List.Count(permutations)-1)
{
perm[j] = List.Shuffle(permutations[j]);
}
r1 = List.Join([permutations,perm]);
permutations = List.UniqueItems(r1);
i = i+1;
}
return = permutations;
}
};
```

Figure 29: The scripted Code Block for combinations for arranging the splitting angles in Dynamo (author)

The issue was fixed through creating two combinations lists, **P1** and **P2** that are highlighted in yellow and red in Figure 29 that's showing the script for the combinations algorithm.

The first combination list (P1) is highlighted in red and the statement in it is written as follows: **P1=List.Combinations([0,1],itt,true);** where **List.Combinations** is a command to create a combinations list, and **0** refers to the first direction, while **1** refers to the second direction, and **itt** refers to the number of angles that are above 90° in the surface.

To explore all the possibilities of how combinations can work; **P2** list is created and highlighted in red in Figure 29. P2 list is the opposite of P1 list. After creating the two combinations lists, the algorithm will join the the two lists together in a new list called **P0**.

Shown in Figure 29; the command **Perminations=List.UniqueItems(p0);** which is highlighted in green, is created to eliminate the repeated values in the final list. It will go through P0 list and remove any repetition that will occur from joining the two lists (P1 and P2).

For example, the first combination list (P1) will produce the sequences shown in Table 4. Presenting the values for a surface with five angles above 90°, starting with (0,0,0,0,0) and ending with (1,1,1,1,1), While the opposite combination list (P2) shown in Table 4 will produce the opposite values. Those values represent the possibilities of directions coming out from the five angles, which will split the surface. However, as clearly shown in Table 4, the results are in a specific range of options. It is missing the sequence where the numbers are shuffled in the center, such as (1,1,0,0,1), (0,1,1,0,1,0), (0,1,0,1,1) and so on.

Table 4: The results of p1 and p2 combinations command (author)

P1 Combination list Values [0,1]	P2 Combination list Values [1,0]
(0,0,0,0,0)	(1,1,1,1,1)
(0,0,0,0,1)	(1,1,1,1,0)
(0,0,0,1,1)	(1,1,1,0,0)
(0,0,1,1,1)	(1,1,0,0,0)
(0,1,1,1,1)	(1,0,0,0,0)
(1,1,1,1,1)	(0,0,0,0,0)



To find the missing sequences that are not presented in the Table 4, the algorithm will create a new list that will shuffle all the values in the list of P0; which is the list that joins P1 and P2. After shuffling all the values, the outcome of that list will be joined to P0 to create a bigger list. And then, the algorithm will remove any repeated values that were caused by the joining of the two lists.

However, since creating a list of values from shuffling is considered a random process, running the shuffling process one time will not produce all the possible values for the directions of splitting. That is why the algorithm will run the process of [creating a list using shuffling, and joining the outputs to P0, then excluding the repetitive values] multiple times. After experimenting different numbers of repeating the process. the outputs showed that the best repetition was from seven to ten times, where the values produced covered almost all the possibilities. It was decided then, that the algorithm will repeat the shuffling process for ten times. The dynamic flow of these operations is displayed in Figure 30.

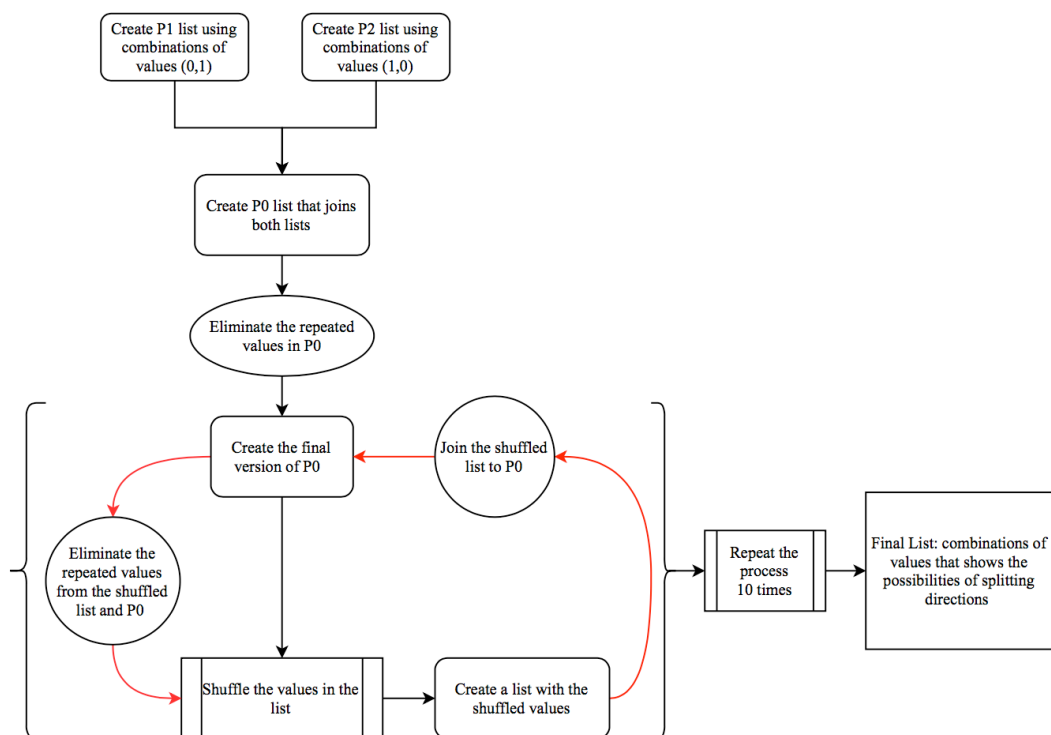


Figure 30: Flow progress of creating the final list using combinations (author)

#### **4.2.2.1.2.3 Merging between permutations and combinations as an optimized approach**

The algorithm can produce proper sequences using combinations method only, however a trial was conducted to see what might occur if both methods were combined. As discussed previously, permutations method had failed due to the fact that it doesn't produce all the maximum sequences from a surface that only had two or one angles above 90°, because it doesn't allow any values to be repeated in the sequence. And the complexity in combination is defined in the shuffling process that occurs around ten times in order to produce unpredictable possibilities of sequences using the number of angles that are  $> 90^\circ$  from the surface as the index reference.

The merging of both methods was created due to a minor error that was detected a while after creating the combinations method. The error showed that whenever the software is closed and re-opened, the combinations algorithm will run again. Due to the random procedure of shuffling in the script, the produced results are not constant. For example, if the end user had selected a ceiling that is spilt in a curtain design and saved the software and closed it. Once the file is re-opened, the algorithm will run again automatically, resulting in creating different unpredictable designs, indicating that the initial approved design might be lost. This procedure will always happen as the shuffling process is random. Thus, unexpected, and uncontrollable results occur.

The merging of both methods had produced proper results, because although using combinations can lead to random results, adding permutations with it allows the results to be constant and controlled. Figure 31 shows how the two lists are combined in a simplified approach.

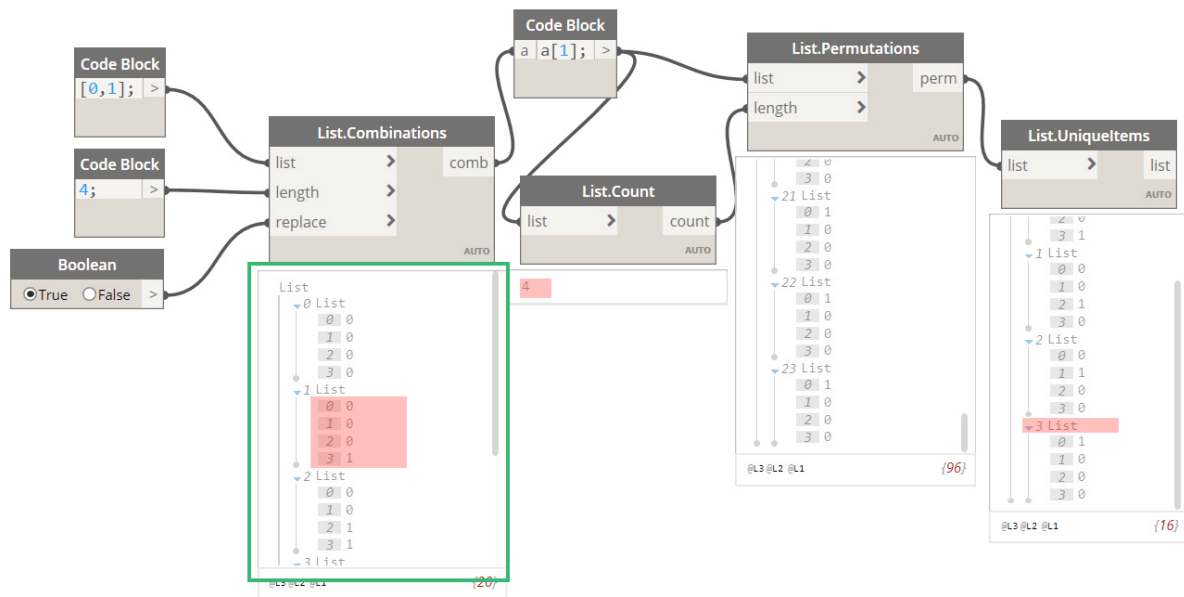


Figure 31: Merging combinations and permutations Nodes in Dynamo (author)

Assuming that **[0,1];** shown in Figure 31 is the list that represents the Index, allowing the algorithm to roam throughout the inserted surface to investigate the number of angles that are more than 90°. While **4;** is the actual number of angles that are found to be above 90°. These inputs are connected to the combinations Node found in Dynamo's library. The produced list is outlined in green in Figure 31, as the generated sequence will have 5 lists in the normal state of combinations without the added shuffling script. The combinations Node essentially counts the amount of times the algorithm can use a certain value from the Index list. As mentioned earlier in details in *Permutations shortfall and using combinations as a solution*, it cannot mix the values in between the produced lists. That is why the shuffling method applied previously resulted in unpredictable results every time the algorithm ran. To merge permutations with combinations, the written script will apply permutations on every list resulted from combinations. For example; only one list from the combination output containing 4 values, which are (0,0,0,1) -shaded in red in Figure 31- is connected to the permutations Node, resulting in four lists of permutations. This example is to clear the connection between permutations and combinations for one list only that is taken from the combinations output. The actual algorithmic script takes all the lists from the combinations' outputs.

In summary; instead of applying permutations on the original index list which is **[0,1]**; in Figure 31, the algorithm will now apply permutation on the output lists of combinations. This allows a controlled output that will have consistency every time the algorithm runs. However, permutations treat every value as its own, creating its individual lists for it. So in Figure 31, although zero is a recurring value from the combinations list, permutations Node treats every zero as a different value, resulting in repeating the same lists in some cases again and again. In order to fix the issue, **list.uniqueItems** Node is added; where it will remove the repeated values found in permutations as seen in Figure 32.

A written script in Figure 32 is added to the workspace as a replacement for the Nodes shown in Figure 31, since the permutations will be looping on every list created by the combinations command. In the previous example shown in Figure 31, the output of combinations was 5 lists; each had 4 values. The permutation will loop on those five produced lists, creating outputs for each, and then the lists will be filtered using **List.UniqueItems** command. It's easier to add the looping definition in a script than Nodes as it allows easier definitions, in addition to sustaining a clear workspace in Dynamo.

```
Combinations/Permutations for splitting the edges
def perm_lst(itt)
{
c1 = List.Combinations([0,1],itt,true);

return = [Imperative]
{
combinations = [];
    for (i in 0..List.Count(c1)-1)
    {
        p1 = List.Permutations(c1[i],List.Count(c1[i]));
        combinations[i] = List.UniqueItems(p1);
    }

return = List.Flatten(combinations,1);
}
};
```

Figure 32: The definition script for splitting the surfaces in Dynamo (author)

The importance of these lists falls in its ability to generate various splitting options for any inserted ceilings' designs. The algorithm will calculate the angles that are above 90° and pick one value from the final list that is created by merging combinations and permutations. Splitting the surface is based on the values given in the final list, as it treats those values as directions.

#### **4.2.2.1.3 Applying a division cut to the surface**

After the proper lists are produced, the results will be shown through drawing the different options of splitting lines on the inserted ceiling surface. However, the graphics tool in Dynamo is not as developed as the other Adobe software. It does not have a good tool to draw lines that indicates splitting the surface in different directions at the same time. it can however split an angle towards one direction only, but not all the angles at once consecutively. To fix this issue; new Nodes from a plug-in called Lunch Box were added to the script. This plug-in is created by a third party called The Proving Ground; and used for both Grasshopper and Dynamo. It is used as a tool for managing data, and it allows geometry behaviors such as generative form making (Nathan 2013).

The algorithm will select every value individually from the final list and will treat it as a direction to produce ceilings' designs that are split differently.

For the same example given earlier in Figure 31, if the inserted surface has four angles above 90°, the outputs will be 16 options of splitting, equal to the values in the final list. Each option is split in a unique different way from the other, as every sequence created will produce a ceiling that is divided differently from the others. Generating all the possible splitting options for one surface. The 16 results from splitting the surface is shown in Figure 33 along with the final scripts; the quantity of splitting outputs are coherent with the amount of the angles that are above 90° in the surface.



#### **4.2.2.2 Approach 02: Forming boundaries**

The number of corners the shape has will lead the algorithm to which method to use, if the surface has more than 4 angles; the algorithm will run *Approach 01: Splitting the surface*, regardless of the degree of the angles. If it has less than 4 angles; it will identify the surface through the boundaries it has and run this method.

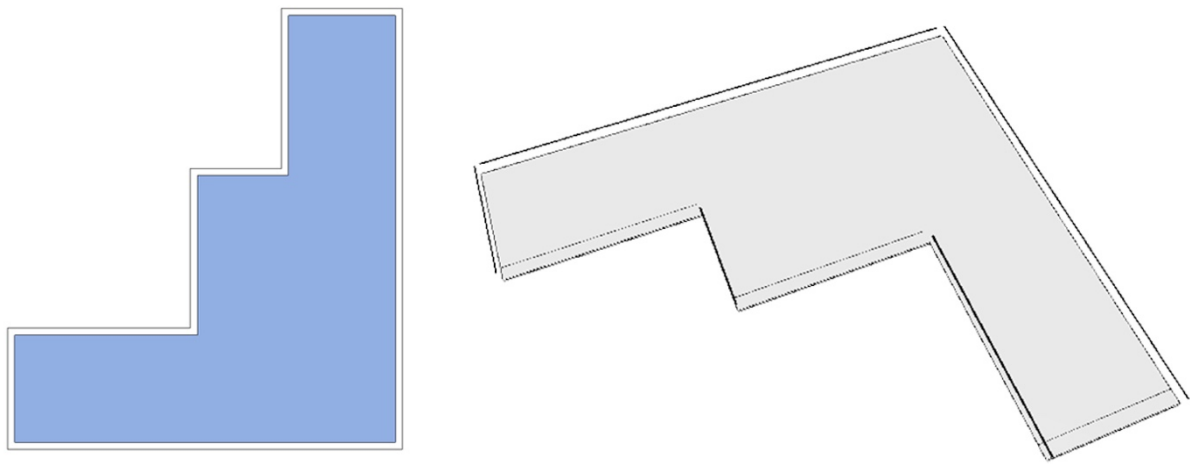
##### **4.2.2.2.1 Creating a polygon line as boundaries**

An offsetting testing method was created, to allow the investigation of the characteristics of the connected lines that create surfaces in Dynamo. It intends to stop any lighting design methods further applied in the generative system, from acting outside the borders of the inserted surface. Clearly identifying the borders of the surface.

When the algorithm ran the offset command on the surface immediately, the attributes of each line were visually clear.

As seen in Figure 34; a simple ceiling shape was tested through applying a direct Offset Node in the workspace of Dynamo after selecting the ceiling, and the results showed that some lines were offsetting outside or inside the surface, on positive or negative Z axis directions.

Concluding that regardless of the surface, the characteristics of the lines forming the surface are not similar, and each line had offset based on its own direction. Which is based on the line's Normal Vector. The Normal Vector is defined as 90° projection vector that comes from the center of the line, it is perpendicular from the direction of the line (The Dynamo Primer 2019).



*Figure 34: Testing a simple shape offset command in Dynamo (author)*

For the offset to function as expected, the characteristics of the lines that created the surface needed to be unified. To do that, the Normal Vector of every line needs to be aligned towards the center.

After various trials, the solution was to take all the lines of the surface and join them into a polygon. Due to the fact that a polygon is a continuous line, it forces every line to have consistent direction, as it starts from one corner and goes around the shape - clockwise direction or anti-clockwise depending on how it is joined - until it reaches the starting corner again.

After that, since a normal consistent direction for the lines is created, running the offsetting testing method again on the surface resulted in a proper offset shown in Figure 35.

It successfully offsetted towards the inside with negative 600mm, forcing all the lines of the polygon to inherit the properties of a closed shape, where they have a consistent similar direction. The value of the offset in Figure 35 is negative, because closed shapes in Dynamo will offset positively towards the outside, and towards the inside if it's negative.



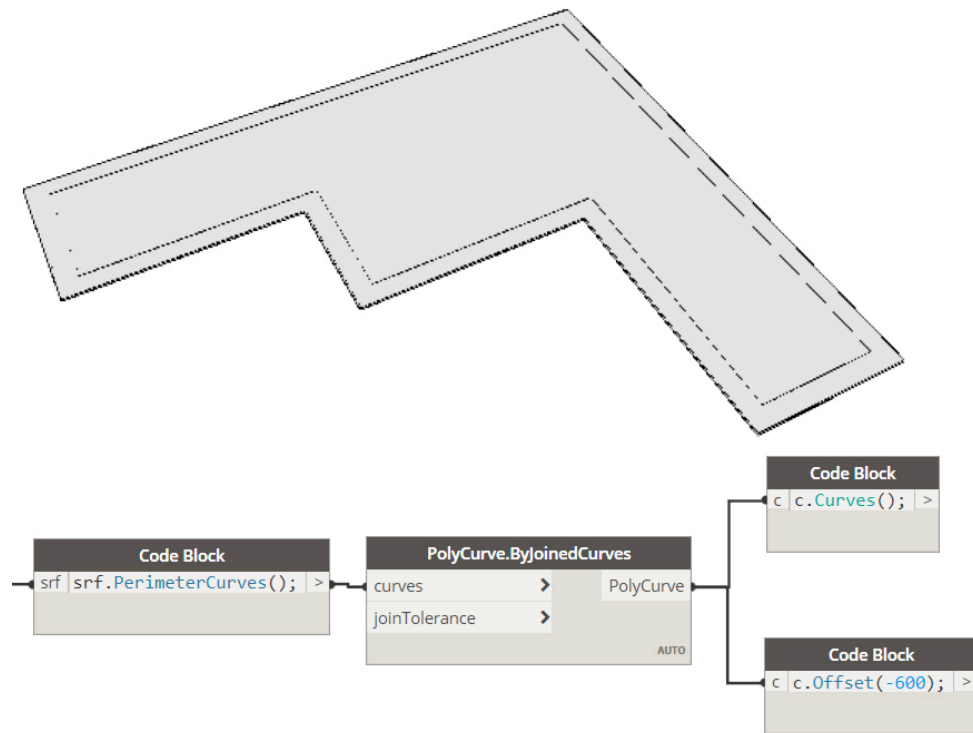


Figure 35: The surface after the polygon offset with the Nodes for it in Dynamo (author)

The offset testing method discussed was merely a test to investigate the characters of each line forming the surface. Different ways were tested to show and identify the attributes of the lines; however, the offsetting testing method showed its ability to showcase the attributes of every line visually. It is not applied in the final scripted algorithm nor required to create boundaries.

The testing method had proved its ability to identify the Normal Vector of every line, which controls the line's direction. While creating a polygon had resulted in unifying the Normal Vectors of all the lines to be aligned towards the center of the shape.

Since Dynamo needs to treat the lines that creates the surface individually for it to run different design methods, an explode command is applied to the polygon line, reverting it back to individual lines. However, these lines are retaining the same properties as the polygon, thus creating a uniformed proper direction for all the lines and establishing proper boundaries to work with in different lighting methods later on in this chapter.

#### 4.2.2.2.2 Choosing between splitting the surface or forming boundaries

Figure 37 shows the script that allows the end user to decide whether to give the command to the algorithm to split the ceiling; which was discussed in details in *Approach 01: Splitting the surface* or to treat the ceiling as a full surface through creating a polygon line around the ceiling and applying direct design methods on it, found in details in *Approach 02: Forming boundaries*.

The script will take the inputs from an added slider under a group shown in Figure 36 called: split controls. If the slider is at 1, it will allow the algorithm to split. If it is at 0, the algorithm will run choosing the boundary option which will treat the surface as one entity and not split it. This method saves calculations time as it does not showcase all the options at once. It only shows the option picked by the slider.

For the slider to be able to determine whether the splitting algorithm is selected or not. A simple if statement was scripted shown in Figure 36, where the if statement is defined as the following: If the split is equal to 1, the algorithm script that defines the splitting will run. Or else; return it to the boundary surface option.

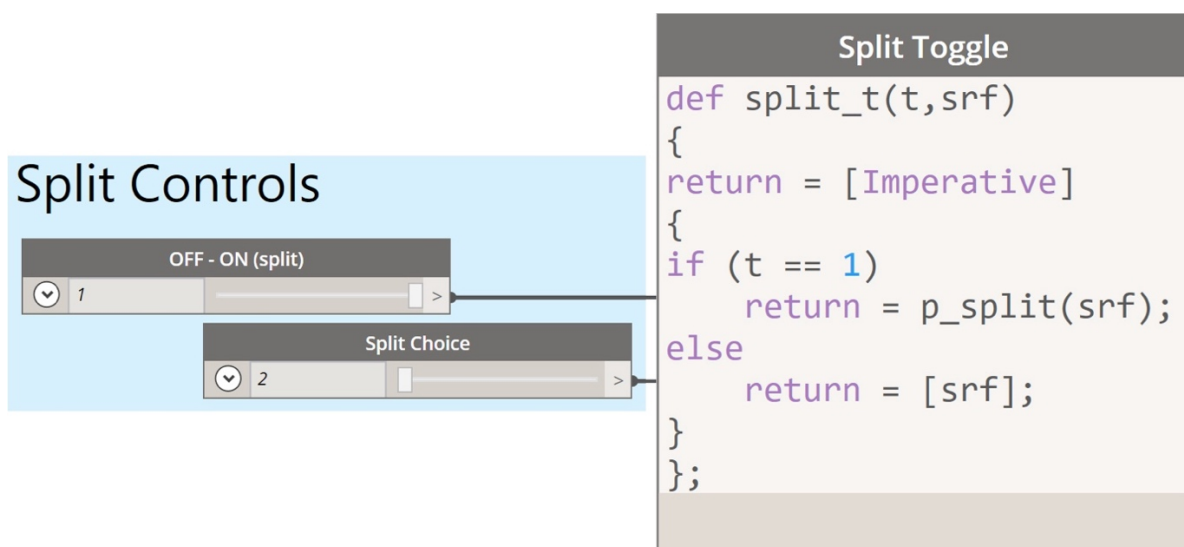


Figure 36: Split controls sliders and script for turning on and off the split by the end user in Dynamo (author)

The split choice slider in Figure 36 allows the end user to see the different designs for the splitting output, as the user changes the value, the splitting design will change. The total number of the produced splitting designs are based on the number of angles the surface has.

The issue faced while writing the script in Figure 37, was its inability to determine how many options the ceiling might produce. So, it is impossible to create a proper end for the slider, as it starts from zero and can lead up to perhaps 60 or 90 options of splitting. So, the slider can't be limited as design options might be lost. However, if the split options produced are little and the slider has more values than the actual produced options, an error will result in the algorithm when the end user moves the slider. In order to fix that, an if statement is added to the script, highlighted in red in Figure 37; where if the choice that the end user is inputting is more than the number of options produced, it will return to the number of options that exists. So, whenever the end user goes beyond the last option, it will always return into the last option of the splitting output, therefor removing the error in the algorithm.

**Sft** definition is highlighted in yellow in Figure 37, it defines the split option; where if the split grid input equals zero, it will return to the original surface state, otherwise if the surface equals one, it means that there are various outputs based on the splitting.

Splitting the ceiling or not		
len	len;	>
srf	srf;	>
splt	splt;	>
choice		
	sfs = List.Clean(split_t(splt,srf),false);	>
	n = List.Count(sfs);	>
	ch = choice>=n-1?n-1:choice;	>
	sft = splt==0?srf:(sfs.Translate(0,-len*1.5*ch..#n+1..len*1.5,0));	>
	sf1 = List.RemoveItemAtIndex(sft,ch);	>
	sf2 = n==1?sft[0]:sft[ch];	>
	sf2e = sf2.PerimeterCurves();	>

Figure 37: Splitting the surface or not script in Dynamo (author)

#### 4.2.2.2.3 Boundaries sub-approach: Creating Boundary lighting

##### 4.2.2.2.3.1 Creating a script for full surfaces

The algorithm will take the boundaries of the inserted surface from Revit, convert it into a poly-curve, and offset the polygon inwards and outwards using the same offset value inserted by the end user using a slider.

The reason of this offset method is demonstrated in Figure 38, where this approach allows the small and narrow corridors - similar to Figure 38 (a) - to be excluded from applying lights around it.

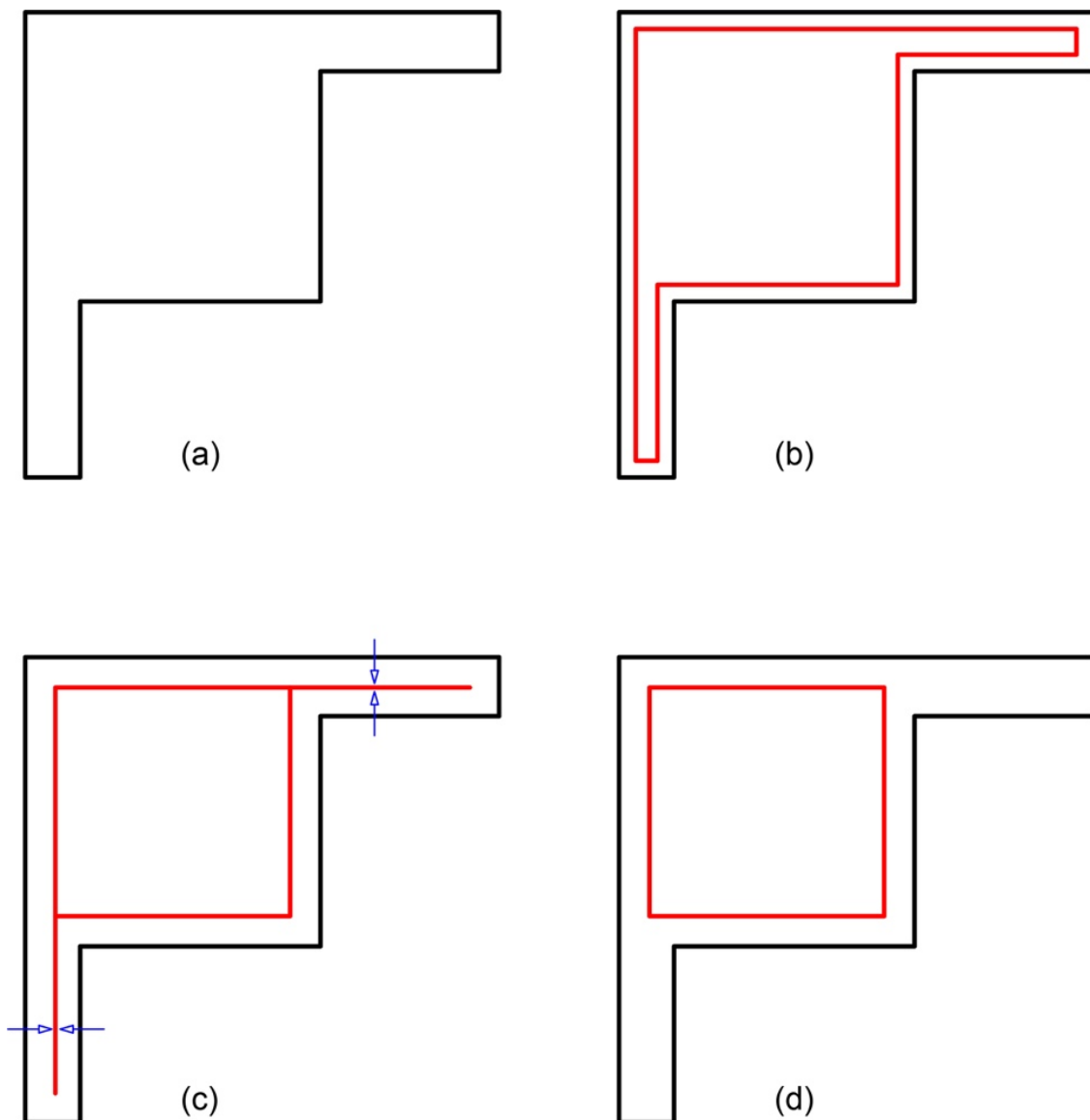


Figure 38: Offset inwards and outwards explanation in boundary lighting (author)

If the inserted value of the offset is relatively low, Figure 38 (b) will result. However, if the offset value is equal to half the width of the narrow corridor in the inserted ceiling, the inwards and outwards offset will eliminate this corridor as seen in Figure 38 (c). Resulting in a ceiling that is shown in Figure 38 (d). This step is important as placing a double line of lights in a small corridor does not serve the design. In addition, another approach is created to address the excluded areas such as small corridors explained in *Creating central line lighting for narrow ceilings*.

After that, the outwards offset is exploded into lines instead of a polygon with the same attributes, the reason for this approach is previously discussed in detail *Approach 02: Forming boundaries*. Outlined in red in Figure 39, a for-loop will run for each exploded line that created the surface, treating them as independent values.

**Script 01: Lights around the ceiling**

```
def method_01(srf,width_lim,wall_spc,spc)
{
    edge1 = PolyCurve.ByJoinedCurves(srf.PerimeterCurves())
    .Offset(-width_lim).Offset(width_lim);
    edges = edge1.Offset(-wall_spc).Explode();
    result = [];

    return = [Imperative]
    {
        for (i in 0..List.Count(edges)-1)
        {
            subresult = [];
            if (edges[i].Length < 2*spc)
            {
                subresult = edges[i].PointAtParameter(1);
            }
            elseif (edges[i].Length < 3*spc)
            {
                subresult = edges[i].PointAtParameter([0.5,1]);
            }
            else
            {
                n = Math.Round(edges[i].Length/spc);
                a = spc/edges[i].Length;
                subresult = edges[i].PointAtParameter(a..1..#n);
            }
            result[i] = subresult;
        }
    }
    return = result;
}
};
```

Figure 39: The created script for running the algorithm on full surfaces in Dynamo (author)

Highlighted in yellow in Figure 39; the algorithm will check the length of the line, and evaluate if the length is less than two times the inserted value for the spacing between the lights. It will only place a point at the end of the line. Preventing overlapping points that might occur if points are placed at the start and at the end of every line that creates the surface.

Otherwise, if the measured length is less than three times the inserted value for the spacing between the lights, the algorithm will place a point at the center and at the end of the line. The else statement is highlighted in blue in Figure 39.

Furthermore, if the measured length is more than the two discussed conditions, it will divide the length by the inserted value for the spacing between the lights. After that, the results are rounded to avoid fractions that leads to unnecessary points. This statement is highlighted in Figure 39. Although Dynamo places points not lights, these points will be used by Revit as coordinates for lights.

#### **4.2.2.2.3.2 Shortfalls of boundary lighting algorithm on split ceilings**

Figure 40 shows the issue that started to appear if the boundary lighting algorithm is tested on split ceilings. Although the scripted algorithm divides the lines correctly, results showed errors in the spaces between the lights, creating inaccurate results. The black arrows in Figure 40 highlights the locations where the division had failed.

After various testing and investigations, the inaccuracy in division occurred because of the splitting algorithm. Since the splitting algorithm draws an extension line from the original surface to do the split, Dynamo treats this case as two individual lines: dividing both lines separately. So, although they might appear as one extended line visually, Dynamo is treating it as two separate lines, creating this discontinuity in the division as the calculations depends on the length of the line.

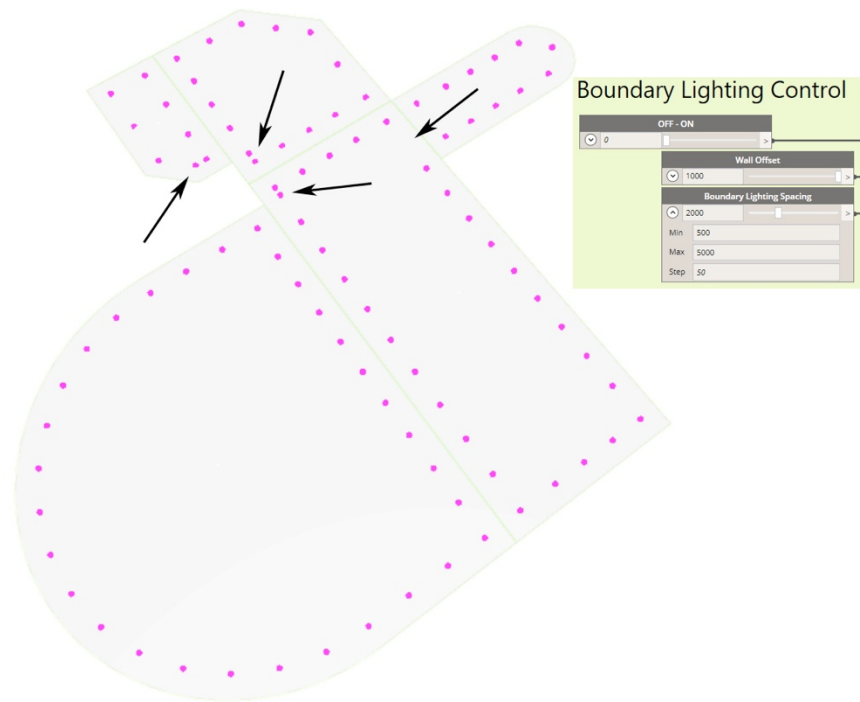


Figure 40: The issue that appeared when applying boundary lighting on split ceilings in Dynamo (author)

Figure 41 shows the steps the algorithm takes to fix the discontinuity issue when placing boundary lights. Where (a) shows a random ceiling before applying the splitting algorithm. While (b) shows what happens when the split algorithm is successful, where it creates division lines that are colored in light green. When applying the boundary algorithm, it will address each split surface on its own as (c) shows. However, due to the extended split lines; the surface will appear to have multiple lines creating one side of the surface, hence the discontinuity issue.

(d) in Figure 41 is showing the first step to fix the issue, where if two lines showed a parallel condition, this condition can tell the algorithm which lines are on the same axis. For the algorithmic system to know which lines have this condition, Normal Vectors were extruded from all the lines and checked if they were parallel, since there are not any Nodes in Dynamo to check for parallel lines directly. The output will be fed into (e).

(e) displays the next step of the algorithm in Figure 41, where a script is written to arrange the indexes of the lines, arranging them through checking each index and shifting it to a position that results in creating sequential indexes that are coherent with the surface lines.

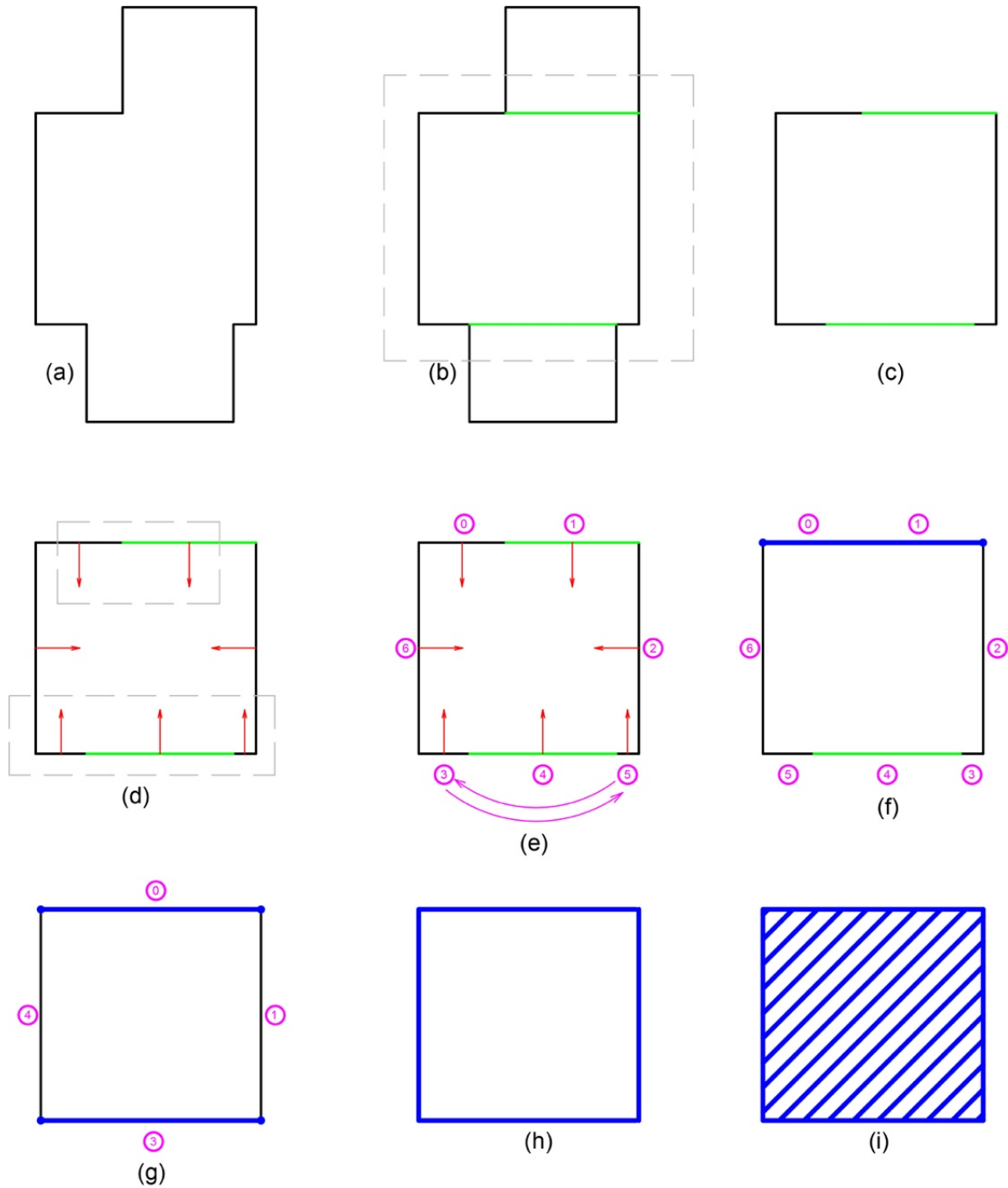


Figure 41: Explanation for the algorithm approach to fix the discontinuity issue (author)

(f) is showing the next step, where the algorithm will extract the points of the start and the end of the parallel index and joins these two points using a poly-curve line.

Due to the fact that a new line is drawn, this adds up to the original indexes, so the algorithm will remove the old indexes as seen in (g) in Figure 41. A while-loop is created in this condition rather than the standard for-loop as not all the lines that forms the surface requires this step.



(h) in Figure 41 joins all the new clean lines and creates a poly-curve, While (i) is the final step that patches the surface. Resulting in creating the same split surfaces but without the discontinuity issue. After that, the patched ceiling is fed to the first algorithm and the boundary lighting method is created.

#### 4.2.2.2.3.3 Creating central line lighting for narrow ceilings

Figure 42 shows how the algorithm approaches the narrow ceilings and places a center line of lights in the middle. The same steps that were explained in Figure 38 is conducted again in this approach, resulting in having a ceiling as (d), where the narrow corridors are excluded from the layout.

(e) in Figure 42 is showing the next step, as the algorithm will apply Normal Vectors throughout the entire surface, in addition to the reversed Normal Vectors as well.

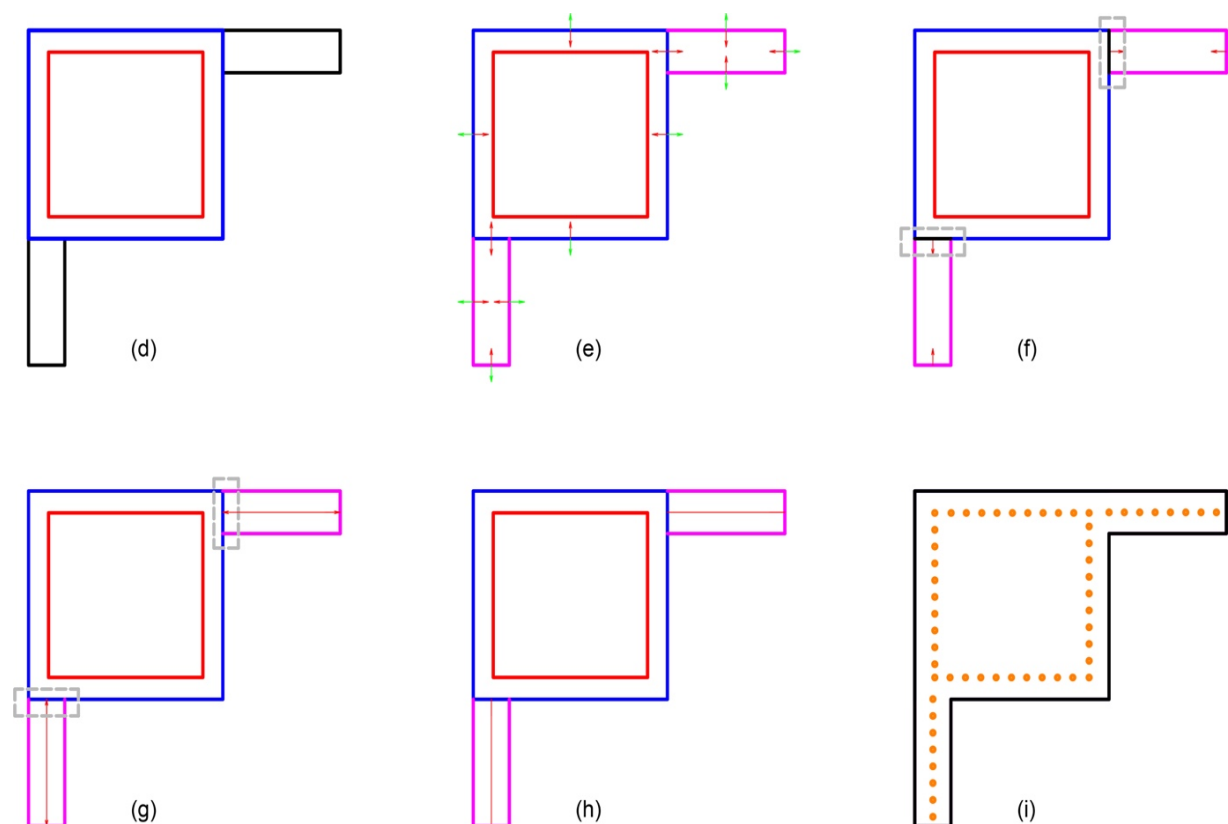


Figure 42: The algorithmic script approach to creating central line lighting for narrow ceilings in Dynamo (author)

Since the corridors were excluded in the beginning due to the inwards-outwards offset explained in Figure 38 previously. The algorithm is treating the blue outlined surface as a surface by its own and the purple surfaces as surfaces by their own individually.

(f) in Figure 42 is highlighting where both surfaces had intersected. The script applied a condition on the Normal Vectors to identify if the applied Normal Vectors are projecting from one ceiling to another. If so; they will remain as seen in (g) in Figure 42. Otherwise they will be discarded.

After that; (h) in Figure 42, is showing the created poly-curve using the points on the intersections, and (i) is the final step that takes the poly-curve and divide it based on the inserted value by the Boundary Lighting Spacing Slider.

#### 4.2.2.2.3.4 Boundary lighting controllers

Figure 43 shows the controllers that are used by the end user to allow easy and instant design alterations based on the visual outputs. The first slider can turn on and off boundary lighting method to reduce calculations on the total algorithm.

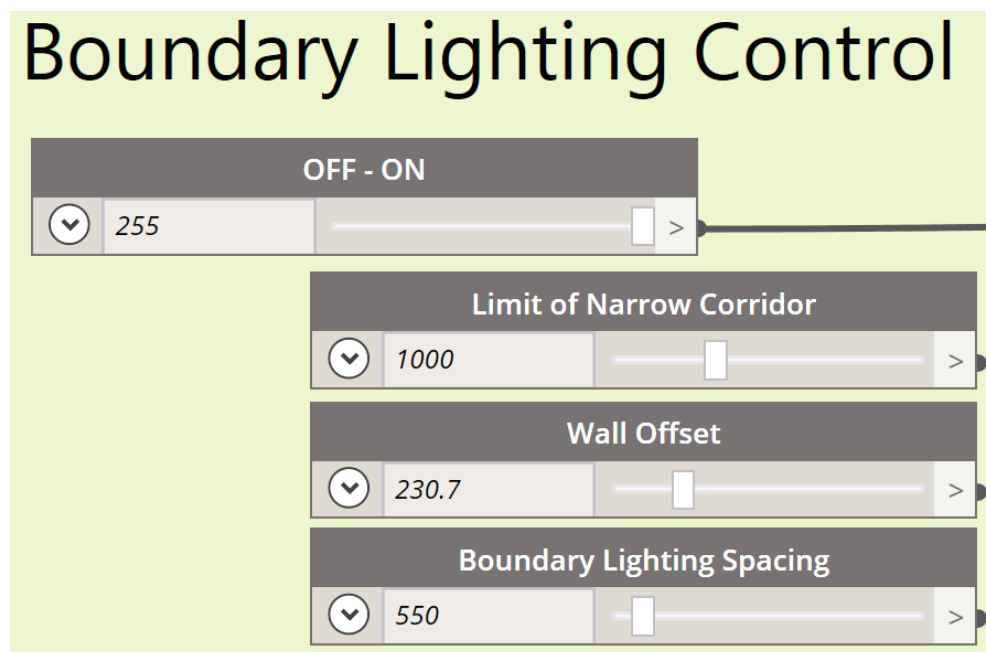


Figure 43: Boundary lighting controllers in Dynamo (author)

Limit of Narrow Corridor slider shown in Figure 43 refers to the amount of narrow corridors the end user would like to exclude from the boundary ceiling approach. If the slider value increased, the number of corridors or narrow ceilings that applies the boundary lighting method will decrease.

Wall Offset Slider controls the distance between the lighting and the borders of the ceiling. The increased spacing works in a direct correlation with the value in the slider. While Boundary Lighting Spacing Slider shown in Figure 43 is basically the value that states the spacing between every light coordinates. This value is also used for the division part in the algorithm.

#### 4.2.2.2.4 Boundaries sub-approach: Creating drop down ceilings

Figure 44 shows one option produced by the algorithm, along with the sliders controlled by the end user.

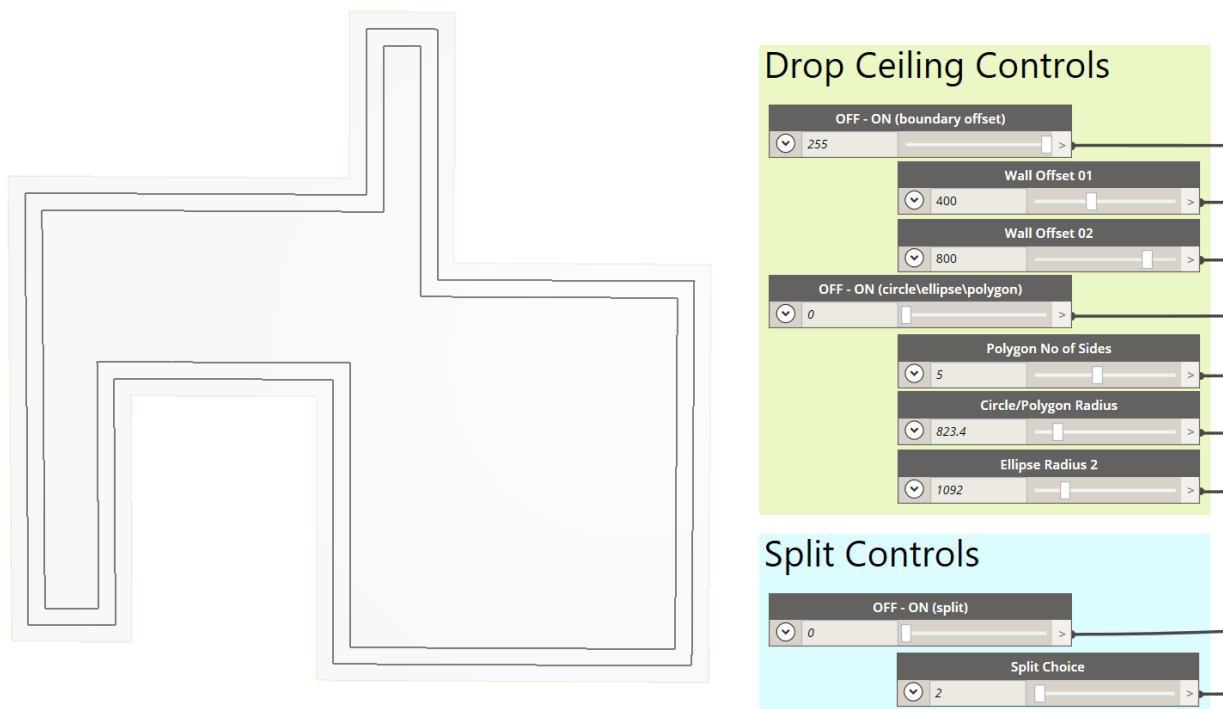


Figure 44: Drop down ceiling option with its controllers in Dynamo (author)

The Wall Offset 01 is at a value of 400 and Wall Offset 02 is at a value of 800 producing the double lines shown in Figure 44.

#### 4.2.2.2.4.1 Drop ceilings script

Figure 45 shows the script for the drop ceilings method that is outlined in red. Where the algorithm is using the surface that is inserted from Revit to form boundaries around the ceiling using a polygon line. Then, the created polygon line is offset inwards, the value of the offset is controlled by the end user using two sliders. As two offsets are created to increase the variety of options.

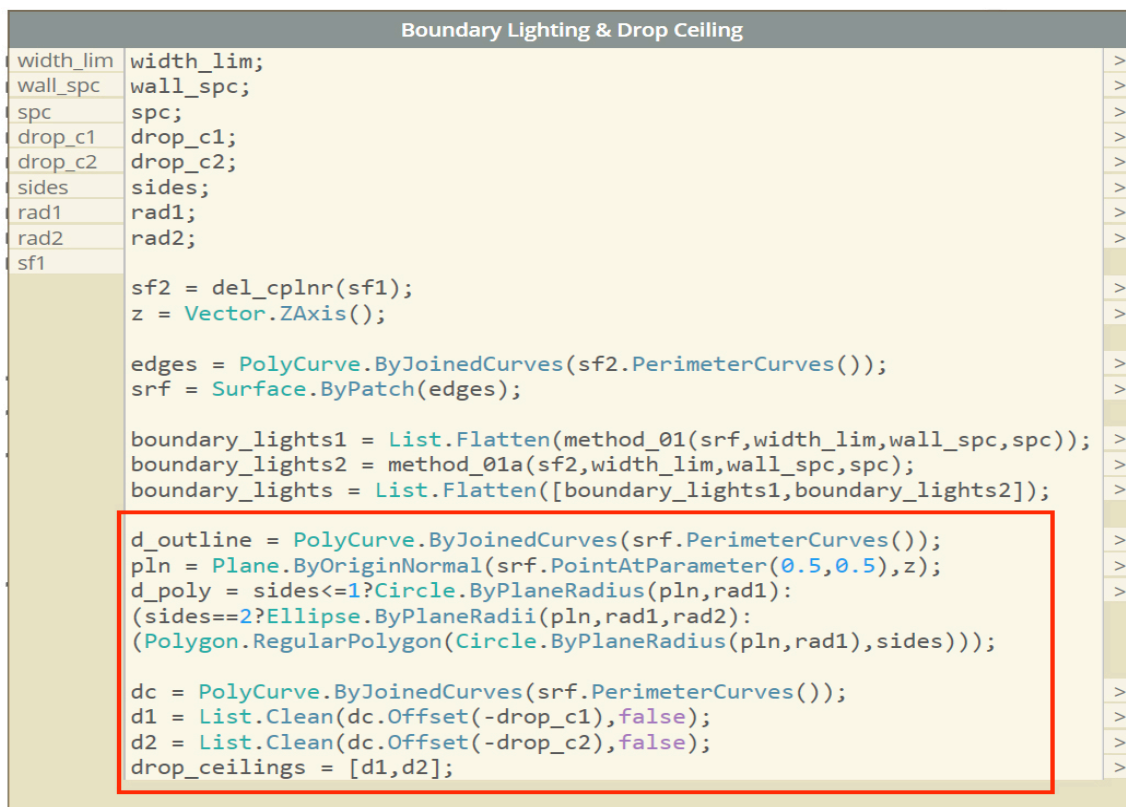


Figure 45: The script for drop down ceilings and minor definitions for boundary lighting in Dynamo (author)

To place various polygon shapes in the center of the ceiling; the center point of the shape is created, and a plane is placed on that center. By using a plane, the algorithm can insert a **Circle.ByPlaneRadius** command, that created a circle in using the center point.

Using the circle created, different shapes can be generated from the circle. Generating a circle followed by an ellipse, then a triangle, quadrilateral, pentagon, hexagon, heptagon, all the way to decagon.

However, the created ellipse is scripted in an individual line in the algorithm since it is using two axes: a minor axis and a major axis. The minor axis is considered as the radius of the circle and uses the same slider of the Circle/Polygon Radius slider. While the major axis is controlled individually using the Ellipse Radius 2 Slider.

#### 4.2.2.2.4.2 Drop ceiling controllers

As all the other algorithms, drop ceiling controllers allows a flexible, instant, visual design approach for the end user. The first slider in Figure 46 turns on and off the method. While the Wall Offset 01 slider determines the first offset that creates the drop ceiling design. The Wall Offset 02 slider controls the second offset design. The offset space from the boundaries that creates the ceiling to the new created line increases as the value in the slider rises.

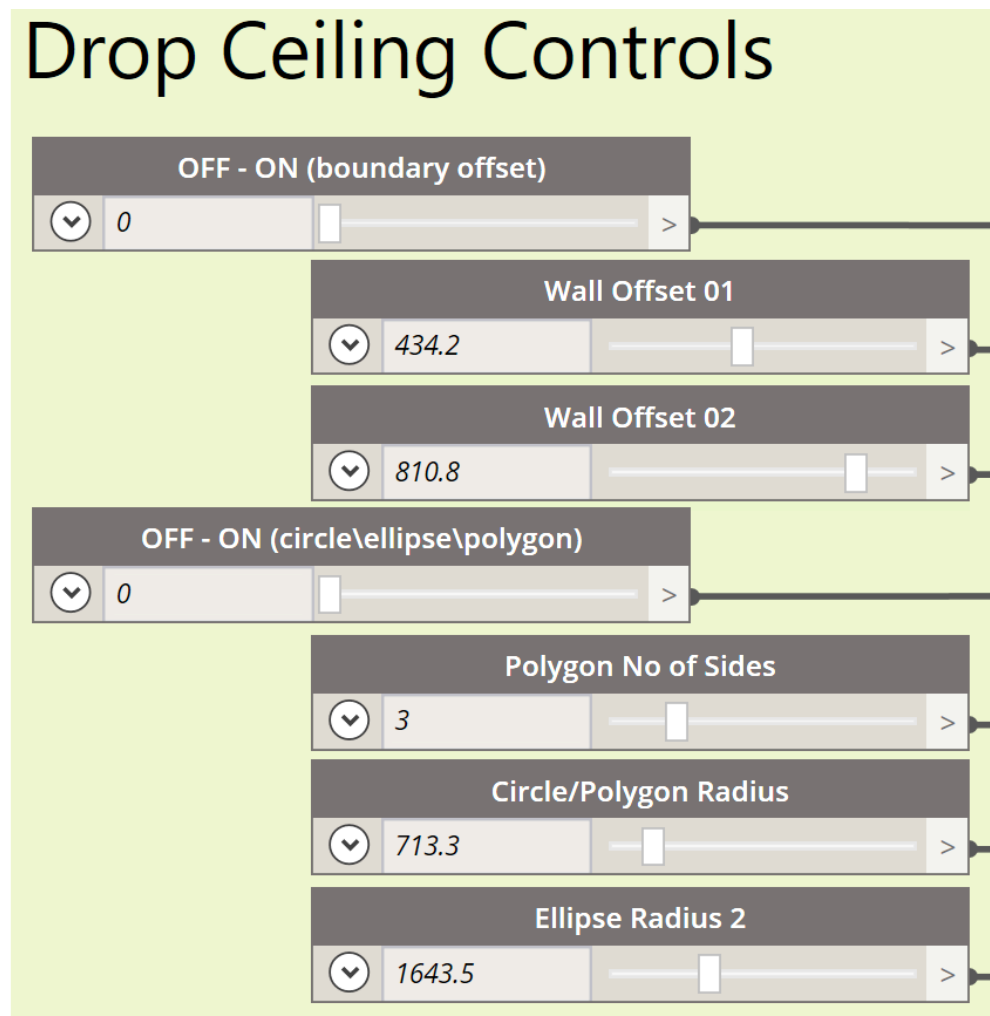


Figure 46: Drop Ceiling Controllers created in Dynamo (author)

The Off-On(Circle\ellipse\polygon) slider in Figure 46 turns on and off the other drop ceiling designs instead of an offset that has the same identical outline as the ceiling.

The other sliders are self-explanatory, as the Polygon No of Sides slider changes the created shapes based on the value in the slider. As 1 creates a circle, 2 creates an ellipse, 3 creates a triangle and so on, the number in the slider matches the number of the sides that creates the shape.

The Circle/Polygon Radius controls the radius of the polygon shape as well as the circle radius. While the Ellipse Radius 2 allows the user to manipulate with the ellipse dimensions as seen in Figure 47.

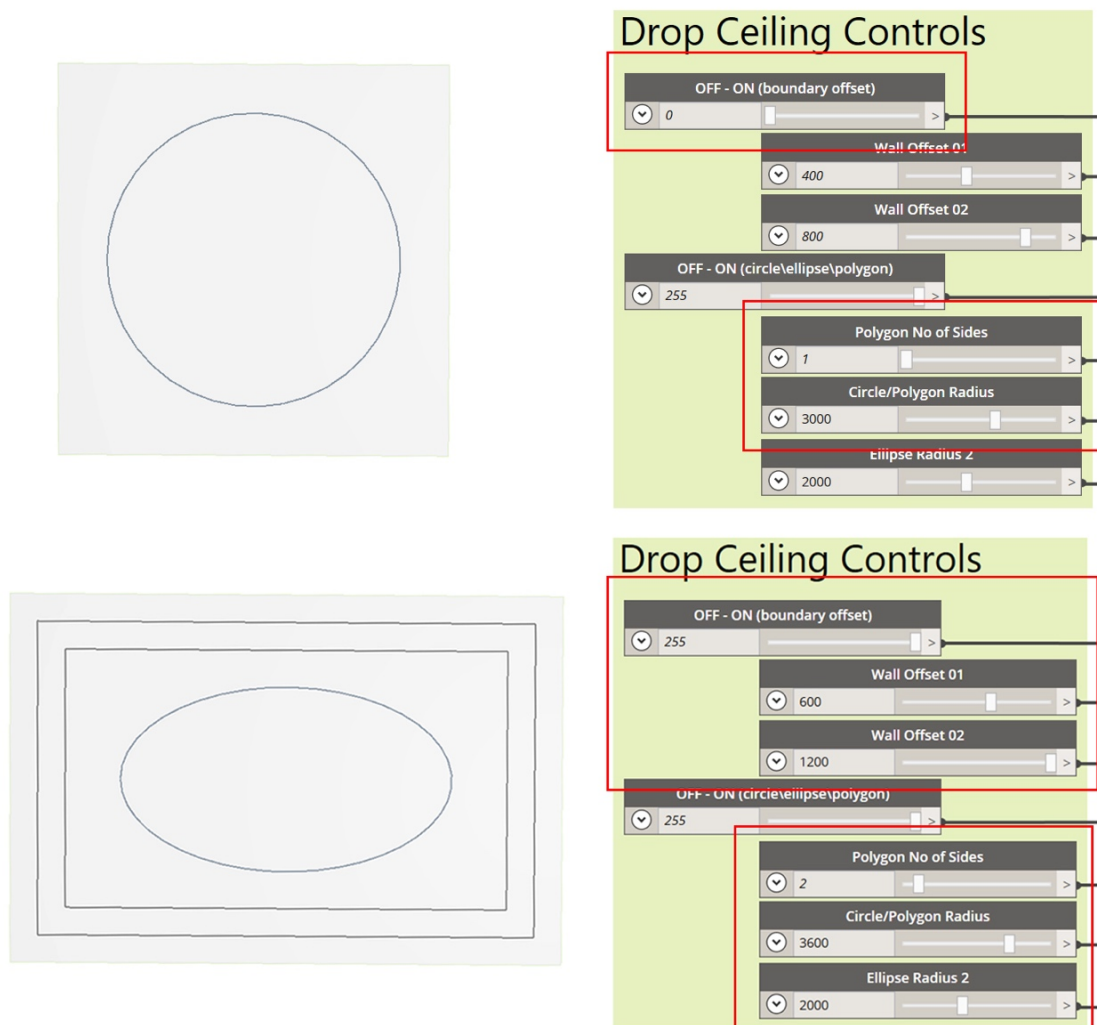


Figure 47: Figure 23: Produced drop down ceiling in Dynamo (author)

### 4.2.2.3 Approach 03: Creating a grid

#### 4.2.2.3.1 Creating a grid using a plug-in

For creating a grid, a plug-in called Lunch Box was used, it is also previously used in adding the splitting lines in *Approach 01: Splitting the surface*. The grid tool is a part of the plug-in and it is usually used for creating three dimensional shapes and for controlling the geometry while creating different masses. Thus, it offers a variety of different shapes for the grid. Different Nodes were added to the algorithm, a quad grid, a diamond grid, a hexagon grid and three different shapes of triangle (a,b and c grids). If the diamond Node is used for example, it will result in placing points on the shape of a diamond distributed uniformly throughout the ceiling.

After adding the Nodes from Lunch Box, each Node required three main inputs: the surface, the U direction, and the V direction. Those directions are numerical inputs that control the amount of points inserted. Once these three elements are fed to the Node, it will automatically create points on the selected surface using the referred Node shape. These points are treated as light coordinates.

When testing the different grids, results showed an interesting variety. As not all the grid Nodes produced the inserted numbers in U and V directions. For example, for the Quad, and Triangle B grids; if the V or U direction input was 10, the number of points shown on the surface will be 9 on each axis.

While the Hexagonal and Triangle B produce 10 points on both axis as the input. While the Diamond grid produced 4 points on a line, followed by 5 points on the next line to create the diamond shape on both U and V axis, however the inserted input was 10.

While Triangle C grid produced the opposite of the diamond grid starting with 5 points on the U axis, followed by 4 points on the V axis.

Table 5 shows the Nodes with the results of each grid. In the shown Nodes, the three inputs are visible. The U and V inputs are inserted through a Slider Node, that shows 10 in all the grids.

Table 5: The resulted grids using Lunch Box in Dynamo (author)

Grid Type	The Node in Dynamo	The Resulted Points Grid
Quad Grid		
Diamond Grid		
Hexagonal Grid		
Triangle Grid A		
Triangle Grid B		
Triangle Grid C		

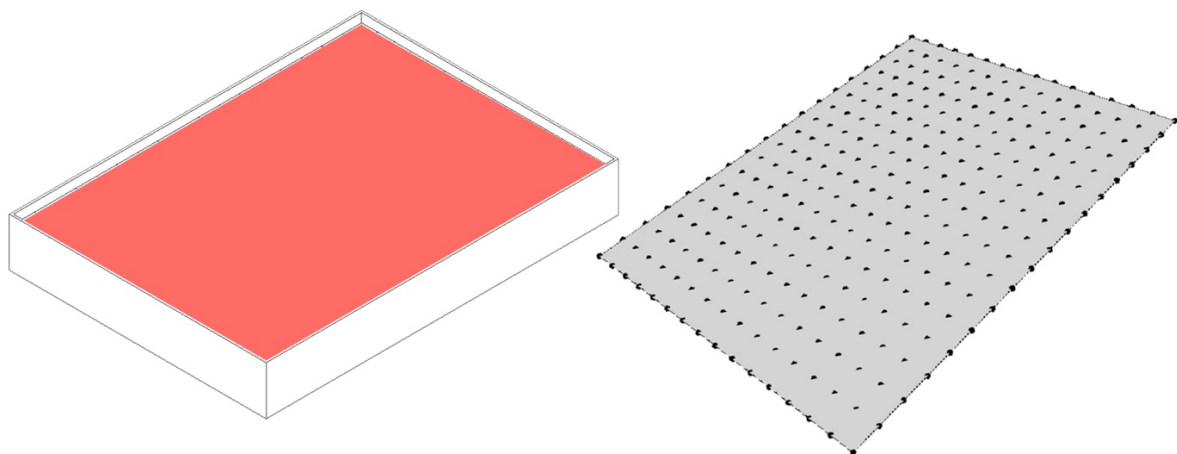


#### 4.2.2.3.2 Shortfalls and solutions in the plug-in grid

The plug-in grid was able to distribute the points without the need to feed the division calculations to the algorithm, thus saving time and effort in figuring out proper calculations. In addition, it offered a variety of designs for distributing the points on the grid.

However, the plug-in grid had failed in distributing points in the surfaces that had more or less than four corners, furthermore it had applied points of the surface's borders in all the tested shapes regardless of the number of angles the surface had.

The first issue occurred when the grid Node was tested on rectangular and square ceilings. The algorithm was able to apply the coordinates of the points using the different designs, however the boundaries of the surface had also points on it as seen in Figure 48. Since the points refers to light coordinates, it is essential for the boundaries not to have any points on it.



*Figure 48: First shortfall of the inserted grid in Dynamo (author)*

The second issue occurred when the algorithm was tested on ceilings that had more or less than four angles. As seen in Figure 49, in order for the algorithm to achieve uniformed distribution for the points with different designs, it created a boundary box that applied an imaginary four edges surface on top of the selected shape, then the algorithm measured the length of the U and the V directions from the boundary box, and divided it mathematically to be able to distribute the points on it equally. Resulting in locating a lot of points outside the surface.

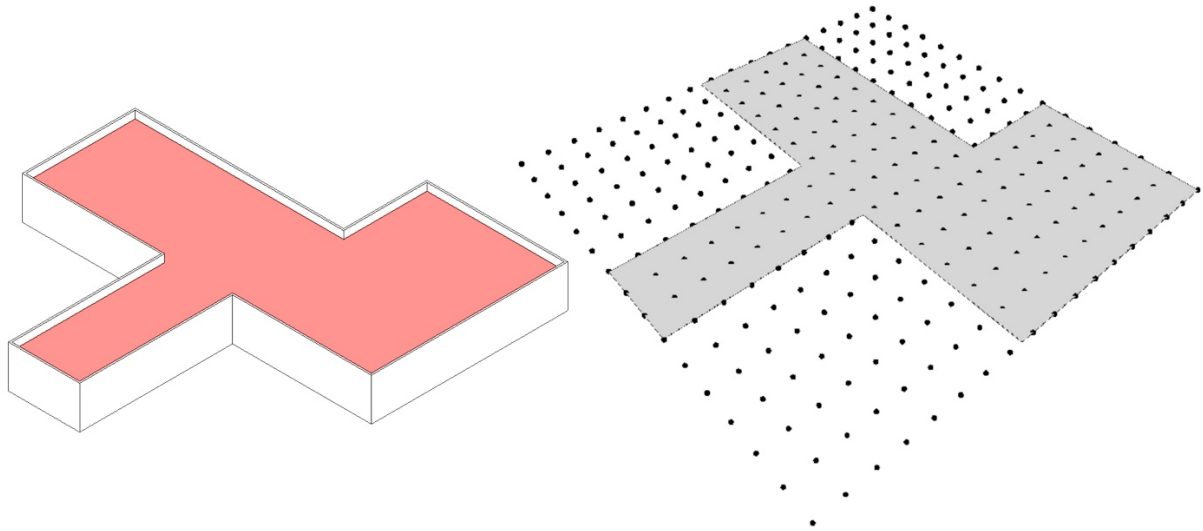


Figure 49: Second shortfall of the inserted grid in Dynamo (author)

The plug-in tool produces the proper distribution of the points using the imaginary boundary box, which is important to keep, as it creates equal spaces between each point. However, to delete the points that are located outside the surface, a Code Block was added to the algorithm.

The script is highlighted in red in Figure 50, where it intersects the produced points with the shape of the surface that was inserted in Dynamo. Upon intersecting the points with the surface; the algorithm will sustain the points that are attached to the surface or touching the boundaries, while ignoring the points that didn't.

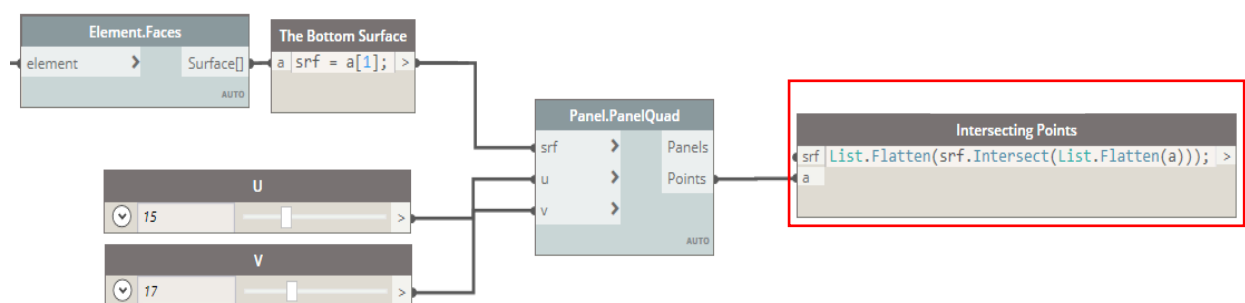


Figure 50: The Quad Node connected to the intersecting Code Block in Dynamo (author)

The script shown in Figure 50 had eliminated the extra points that did not intersect with the surface. Nevertheless, the points found on the boundaries are considered intersecting points, thus this script cannot remove it. To fix that issue another script was created and added to the algorithm.

The added script is shown in Figure 51, where it creates a polygon line that forms the boundaries of the surface. Then, offsets that line towards the inside based on a given value from the end user - shown in Figure 51 as a numerical slider - creating a smaller surface inside the original surface by patching the offsetted polygon.

After that, the new patched ceiling that resulted from the offset towards the inside of the shape connects to the intersection script. Finally, the preview of the offset line is turned off which is outlined in Figure 51. To allow displaying only the points inside without showing the points on the boundaries nor the geometry of the boundaries inside the ceiling that indicates the polygon offset.

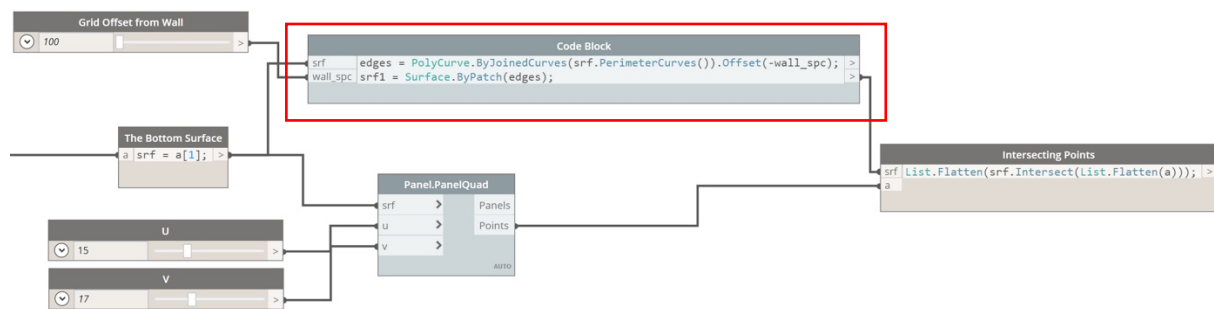


Figure 51: The final Plug-in Grid algorithm in Dynamo 2.0.3 (author)

Figure 52 shows the full final algorithm for the quad grid, and the surface if the patched offset is turned on, where the offset line that created a new surface is clearly visible, and the grid points are only applied to the new surface. Resulting in having a clear frame that doesn't have any points in the original surface.

While Figure 53 shows how the surface looks when the preview of the offset line is turned off, resulting in creating points inside the original surface without showing the new added surface, the offset is 2000mm in both figures and it can be controlled and changed to increase the grid points to the surface.

In Figure 53; turning off the line of the offset doesn't mean that the algorithm is not using the script and creating a new smaller surface, it simply means that the line can't be seen visually, however the result of creating it is clearly visible.

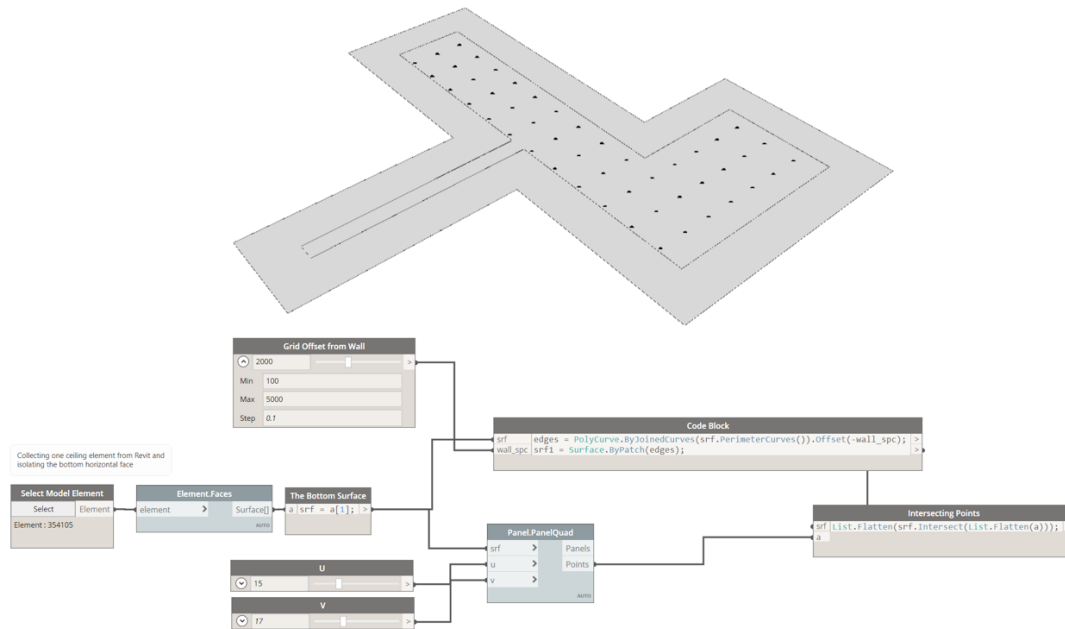


Figure 52: A Surface with the final grid algorithm with the patched offset turned on in Dynamo 2.0.3 (author)

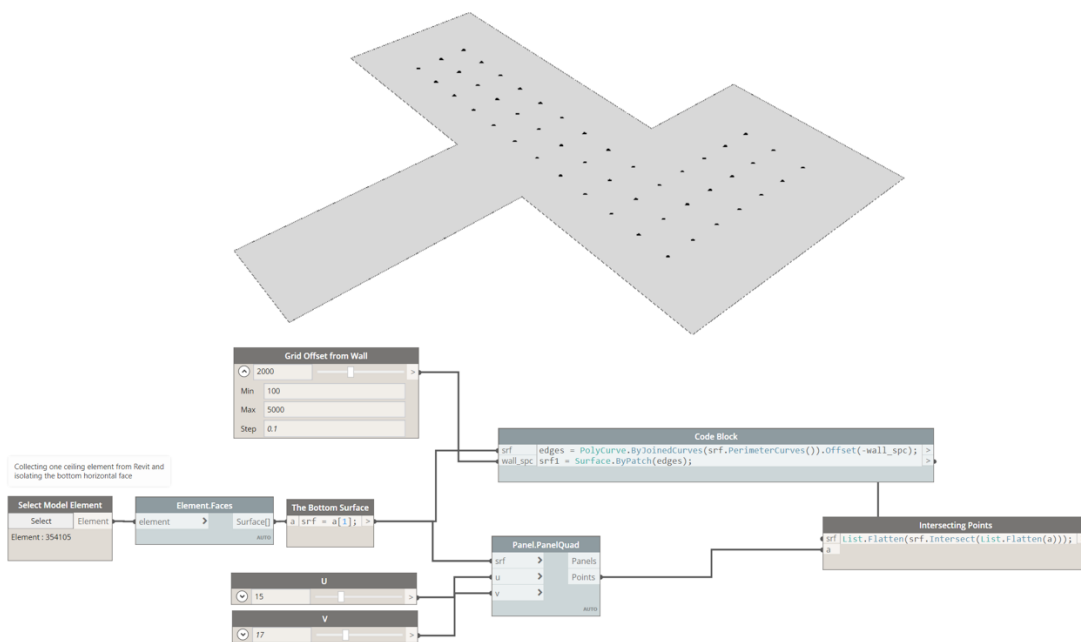


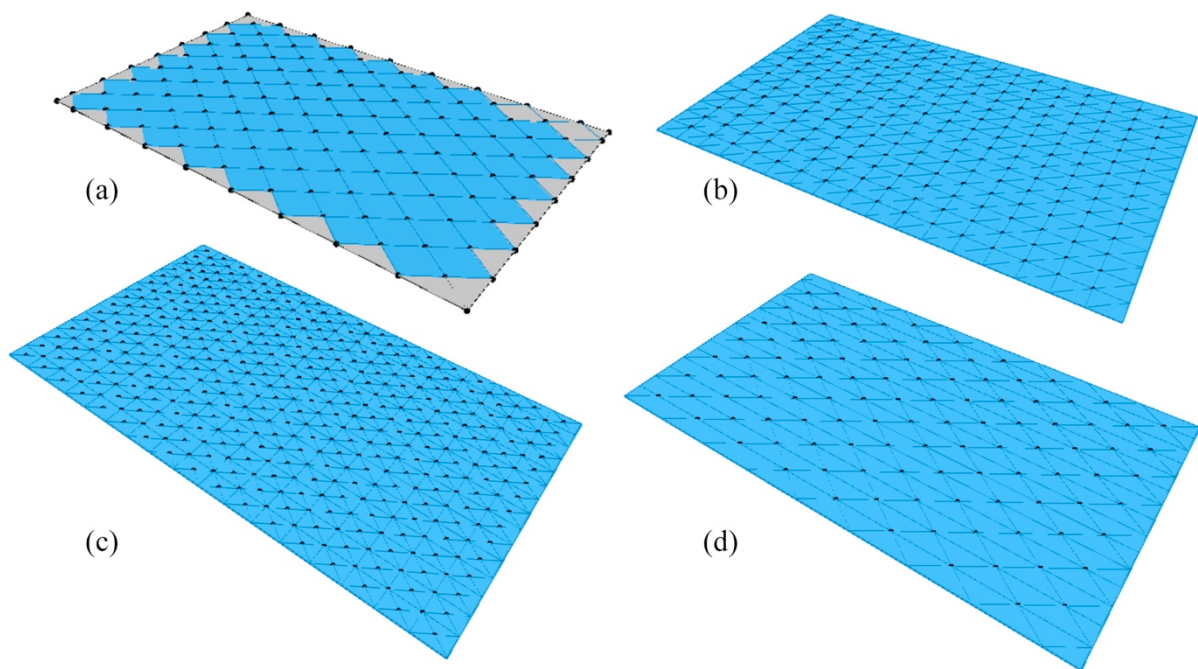
Figure 53: A Surface with the final grid algorithm with the patched offset turned off in Dynamo 2.0.3 (author)

Although the plug-in grid had shown promising results, it had a few issues that resulted in changing the grid approach completely. Nevertheless, this grid can be used widely if the algorithm is addressing masses and geometrical shapes.

Due to fact that the aim of the algorithm is to create light coordinates only, the output results did not serve that purpose as required. The coordinates by the points are uncontrolled as they do not always match the inserted U and V values. Not allowing the end user to determine a clear number for the lights coordinates in a surface.

Although the different design Nodes had shown great differences when the points are connected through panels, exhibiting the potential different results the grids can make - as seen in Figure 54 - these options tend to have a very similar results when the lines or panels are removed.

Figure 54 displays the different four types of grids when the panels are turned on and connected, as (a) presents the diamond grid (b) is the triangle A grid, (c) is showing the triangle B grid and (d) is the triangle C grid.



*Figure 54: Different grid designs with the panels turned on in Dynamo 2.0.3 (author)*

#### 4.2.2.3.3 Scripting a new uniform/quad grid design

Due to the errors that occurred in the previous stage, it was simpler to take the two distinguished grid designs (the quad and diamond shape grids) and script down the grid calculations manually in order to avoid overlapping points.



Figure 55: The first script for the uniformed grid in Dynamo 2.0.3 (author)

The scripting of the two grid designs had been developed through different stages. The first grid is called Grid 1, shown in Figure 55; it's for the uniformed grid, which resembles the quad grid. While Grid 2 is the non-uniformed grid, which imitates the diamond grid discussed earlier.

The end user will have three types of sliders as seen in Figure 55; the grid type, which basically switches between the quad/uniformed grid and the diamond/non-uniformed grid. Due to the fact that this algorithm is working with the number of divisions on each side; slider Division1 and slider Division2 shown in Figure 55 determine the number of points on both axis. Rather than having it produced randomly as the plug-in grid. This gives the end user the flexibility to customize their grid based on the shape of their inserted ceiling.

The scenario of choosing number one only in Division 1 and Division 2 sliders was added to the script under an if statement. For example; the division starts normally if the end user placed in the both division sliders number 2, as it will basically pick the starting point of the surface;

which is always 0 in all surfaces in Dynamo and the end point of the surface; which is 1. Creating coordinates for two points on each axis, where x axis will have (0,0),(1,0) coordinates, and y axis will have (0,0),(0,1) coordinates. Whenever the coordinates are intersecting, a point is created. So in this case; the two intersection points between the two axis will be (0,0) and (1,1), creating two-point division as per the input by the end user. However, if only number 1 is picked in the division sliders, the algorithm will automatically place one point in the center of the surface with coordinates of (0.5,0.5).

These commands can be seen in Figure 55; in the script. In the start of the script, the algorithm has three input elements highlighted in yellow: **srf** which refers to the surface, **div1** and **div2**; that refers to the division sliders in both axes. While **Pr1** is the list of coordinates created based on Division1 slider that divides the surface in the first direction, and **Pr2** is a similar list that divides the the surface in the second direction.

The if statement is highlight in blue in Figure 55, **Pr1** definition is the following:

**pr1=div1==1?0.5** states that if the inserted value of Division1 is 1, the coordinate point will be 0.5, Otherwise - created using the colon mark in the statement - **0..1..#div1**; that means the algorithm will create a sequence starting from zero all the way to 1, depending on the inserted number of divisions. **#div1** inserted using the slider, the divisions will distribute itself out equally on the axis.

In order to place visual points in the divided axis, the used tool in Dynamo is **surface.point.parameter**, which takes a surface, and produces a point at a certain given coordinates, it's highlighted in green as the final step in Figure 55.

Normally the grid will produce points seen in Figure 56 when tested on a ceiling. The final grid output intersects the points with the surface only, so points will be only on the surface.

Figure 56 shows the entire imaginary four angled surface to highlight the sequence of the coordinates in it. As it's clearly seen in Figure 56 the numbers are going in one direction only as the arrows show. Although this is not a bad sequence, but due to the fact that this grid is applied for uniform lighting, creating light coordinates in this sequence will work in applying one light pattern normally, but it can't alternate patterns sequentially.

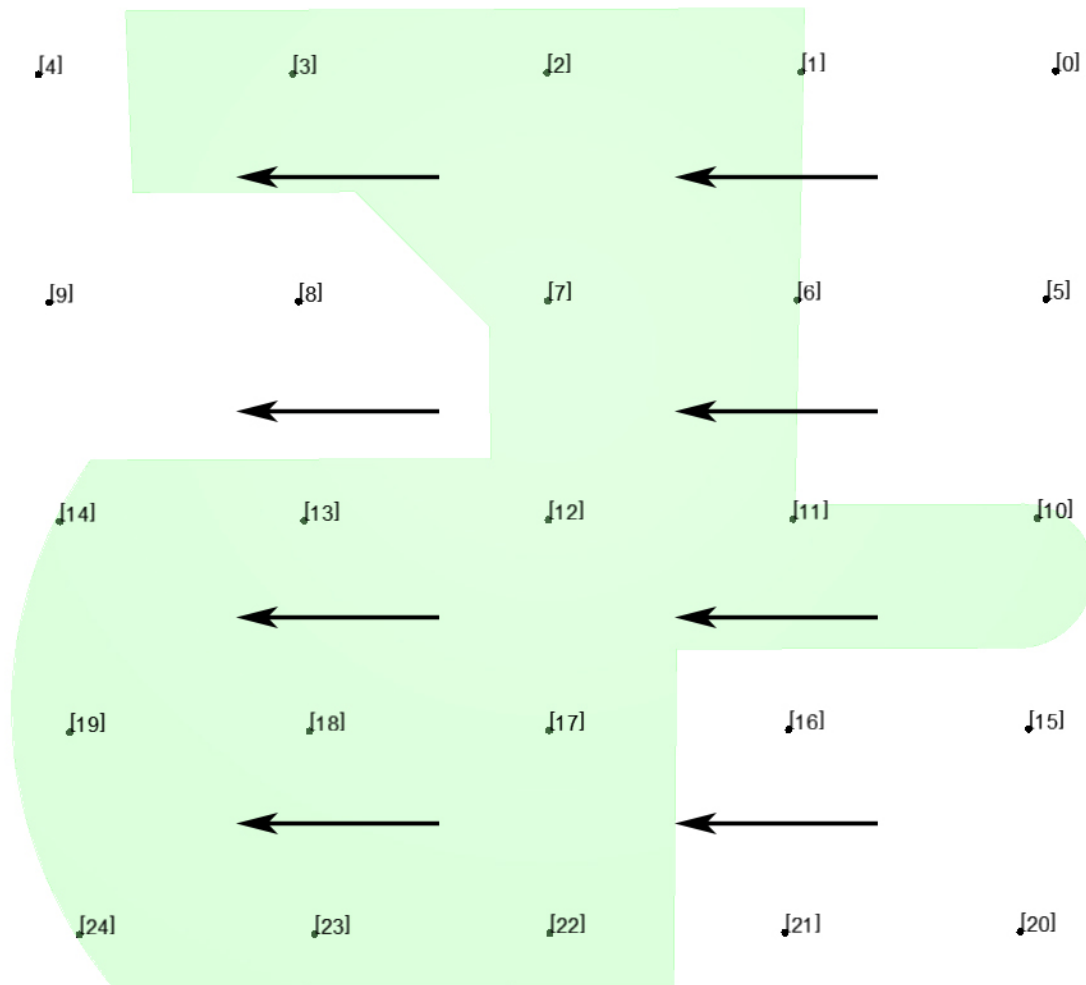


Figure 56: The first sequence produced by uniform grid on a ceiling in Dynamo 2.0.3 (author)

For example; as seen in Figure 57; if an LED spotlight is to be placed on zero-point, two-point, and four-point coordinates and a square LED light is placed on one-point and three-point coordinates. Dynamo will repeat the same sequence again and again as this sequence is located in one list that's outlined in orange in Figure 57; where coordinates (0 to 4) are automatically



placed in one list and numbered as the first list. And that list repeats itself for a second, third, and fourth list in this example. The number of lists depends on the input number of divisions.

This sequence will result in the output seen in Figure 57, where a line of spotlights is seen vertically followed by a line of square LED.

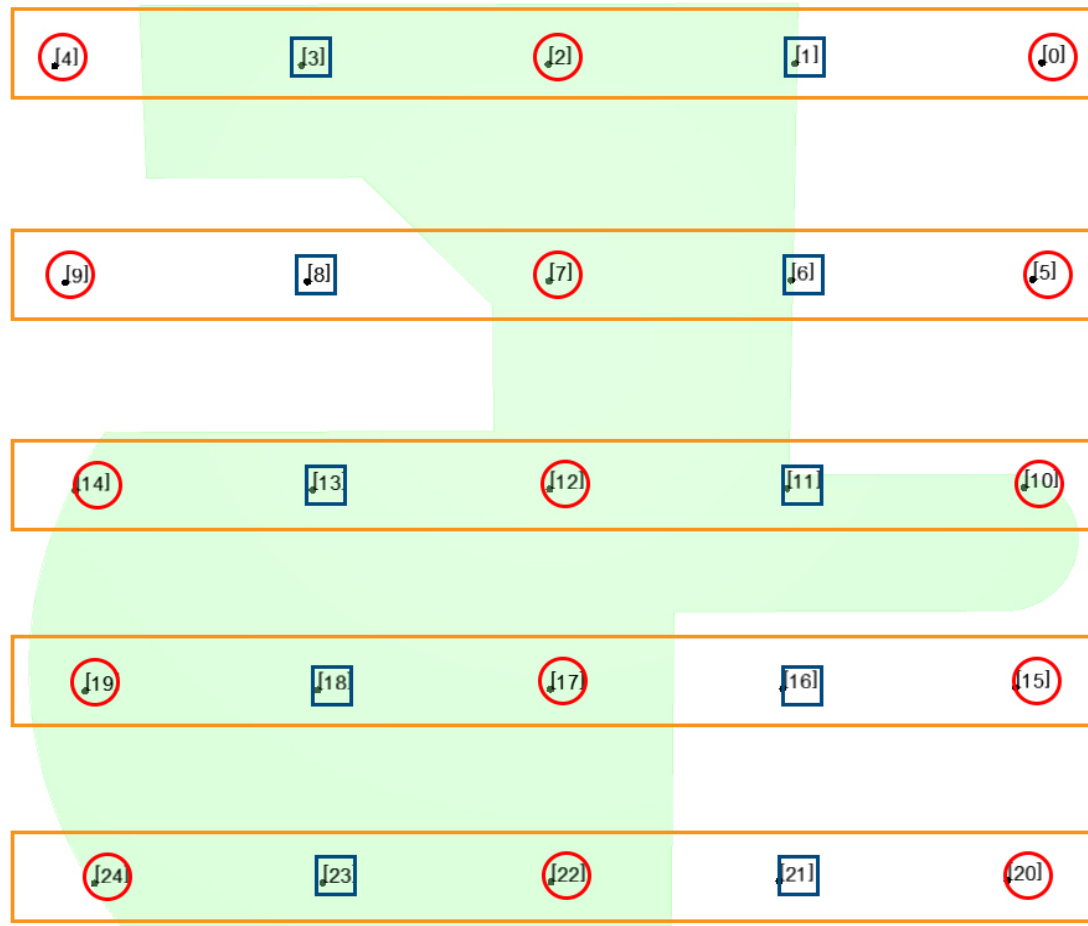


Figure 57: Applying lights on the first sequence for uniform grid in Dynamo 2.0.3 (author)

In order to change that sequence, the following script was added, showing in Figure 55. Where **pr3 = List.Reverse(pr2)**; Which means that the second list **Pr2** is reversed and defined under a variable named **Pr3**. So, if the list is zero to four, it will be four to zero.

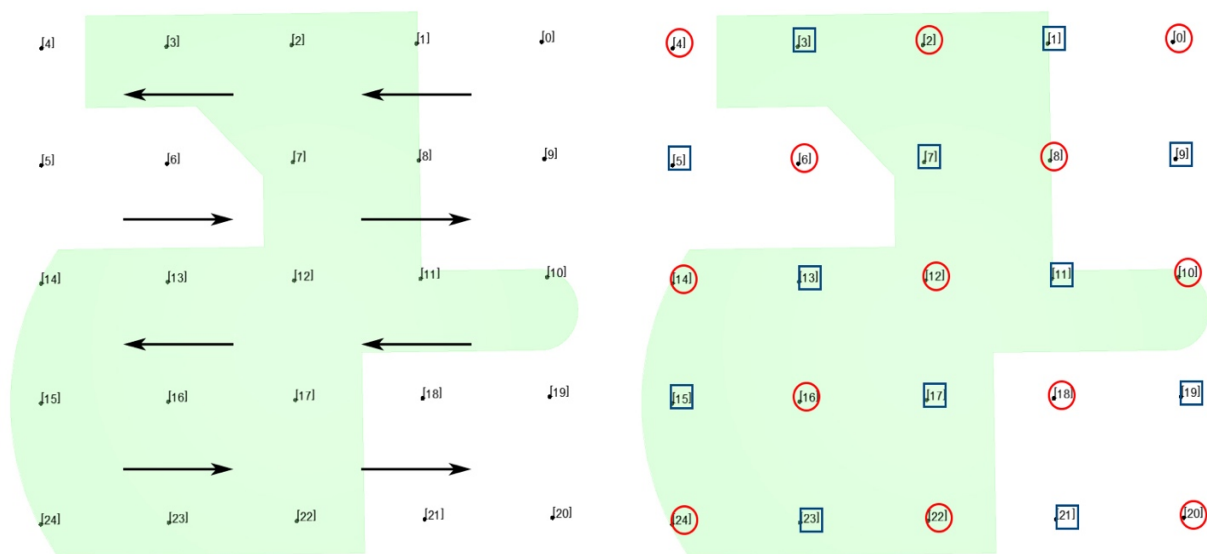
In this step, the grid had an issue where it had reversed the sequence but the coordinates didn't reach to the end of the surface in the division direction produced by Pr3, while reaching the end of the surface in other direction that was produced by Pr1.

After various trials the solution was to add another variable called Pr4 that combines Pr2 with the new reversed list Pr3, it is written in the following form: **pr4 = List.Cycle([pr2,pr3],div1);** After combining Pr2, and Pr3, these two lists are cycled. The two lists will cycle to reach the same division value in Pr1, thus **div1** is added.

In other terms, Pr4 will cycle itself over and over to reach the same number inputted in Division1. Division 1 is discussed earlier as the input value the end user places using the slider creating a large list of numbers.

Since Pr4 is a combination between the original list Pr2, and the reverse list Pr3. The list became double the values compared to Pr1. So to equalize Pr4 list to Division 1 value, Pr5 variable is added, where it will take the values that are not repeated to match the same amount of as Pr1, it is written as the following: **pr5 = List.TakeItems(pr4,div1);**

These added scripts had changed the sequence shown as seen in Figure 58, where the coordinates on the grid are reversed in the second and the fourth lists. Creating a different grid pattern that is able alternate the two types of lights.



#### 4.2.2.3.4 Scripting a new diamond/non-uniform grid design

Figure 59 shows the diamond/non-uniform grid with the imaginary boundary box showing. As mentioned before; the final step in the grid script intersects the boundary box with the surface and only shows the points that are located on the surface.

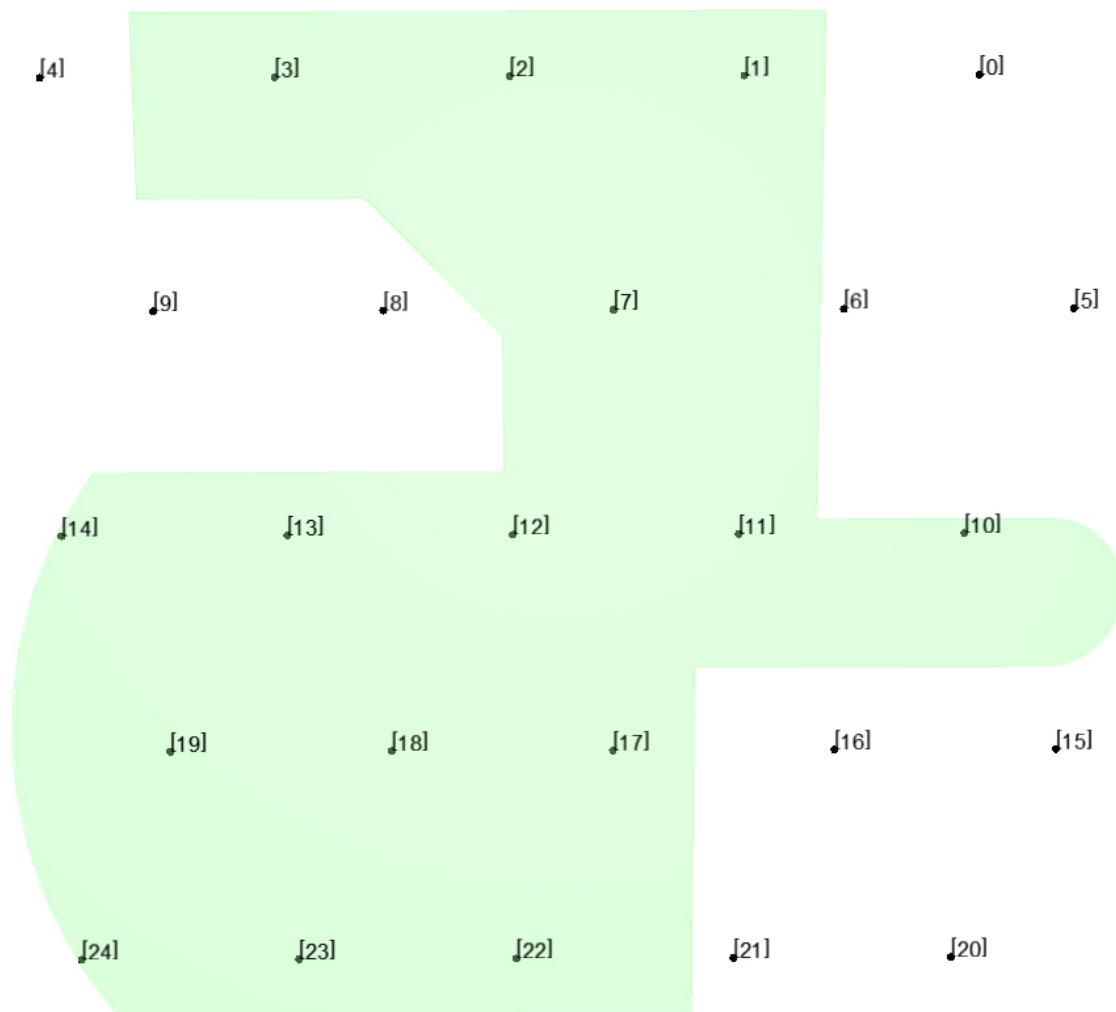


Figure 59: Grid 2 – the diamond/non-uniform grid in Dynamo 2.0.3 (author)

To achieve the diamond grid, the first line starts at an offsetted value, and goes all the way to 1, as 1 refers to the end of the imaginary boundary box. While for the second line; it starts from zero and does not end at 1.

Figure 60 shows the written script for the diamond algorithm where **Pr1=0..1..#div1;**. **Pr1** is defined as a one direction of the division, **Pr1** can be the division script x or y axis. Because

there is no valid way to identify which division is for x axis and which is for y axis. It can only be seen once the algorithm is tested on a certain surface, and that may vary based on the dimensions of the inserted surface.

Assuming that **Pr1** is x axis; in that case the script states that the border of the imaginary boundary box that intersect with x axis is divided starting from zero, which is the starting point of the axis until the end of the of the imaginary boundary box which is 1, diving that axis with the value inputted in **#div1**. This value is obtained from the end-user, where a slider shown in Figure 60 is placed in the workspace under the name of Division 01.

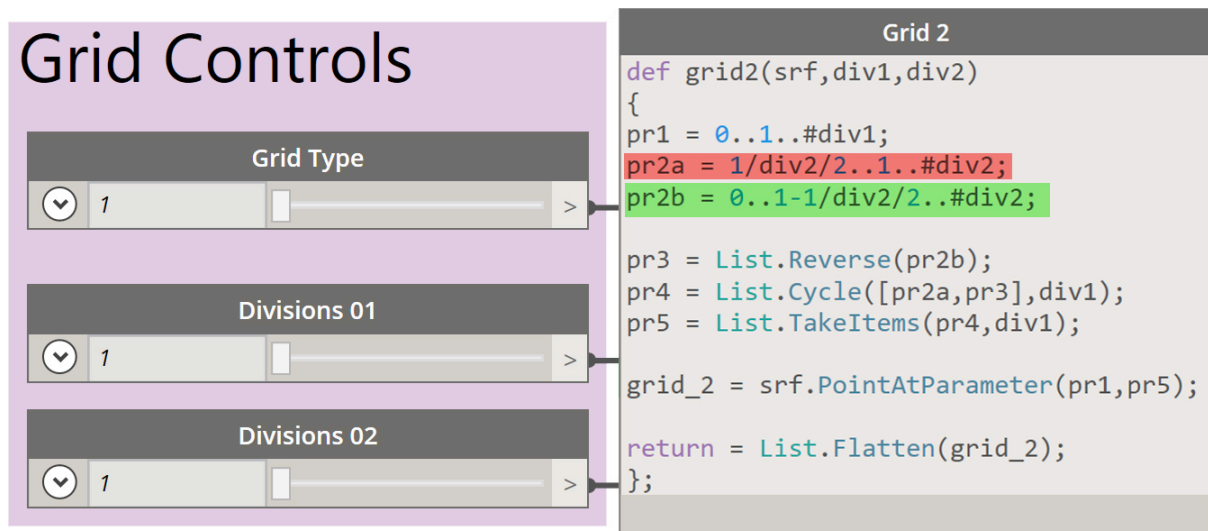


Figure 60: The first script and grid controls for the non-uniformed grid in Dynamo 2.0.3 (author)

**Pr2** is split into two mathematical equations: **Pr2a** and **Pr2b**. **Pr2a** refers to the line in the grid that doesn't start from zero, it starts from an offsetted value. It is highlighted in red in Figure 60 and written as the following in the script: **Pr2a=1/div2/2..1..#div2**; which means: 1 is divided by **div2**, which is the value inserted from Division 02 slider shown in Figure 60, divided by 2 ; to get the point to be in the center of the following line to create the zigzag effect.

For example, if the inserted division by the end user was 4. The list in Pr1 will be (0,0.25,0.5,1) which is the normal grid line that starts from zero. While Pr2a will be  $[(1 \div 4) \div 2]$  presenting

the script of  $1/\text{div}2/2$ . This will create the coordinates to start the second grid line, that is in the center of the space between the two points in the first line. In other terms, it creates the offset value to start the second line with. So, in this example, the second grid line will start at 0.125, which is the center of 0.25. where **#div2;** in the script refers to the number of divisions inserted.

Pr2b is highlighted in green in Figure 60, which starts from zero but the points don't reach the end of the surface in the grid, because Pr2a will get to the end of the surface since it started with an offsetted value and not from zero. In order to keep Pr2b from reaching to the end -1 is added, the script is written as the following:  $0..1-1/\text{div}2/2.. \text{#div}2;$  as it starts from zero coordinates and run until 1-1, which means it doesn't reach the end, which is 1. **div2** refers to the second division slider input that is visible in Figure 60.

The rest of the script is like the uniform grid script, however in this case Pr2b is the list that is reversed under the definition of Pr3.

Then the same procedure is carried on, as Pr3 and Pr2a are combined and cycled to the number of the inserted divisions from Divisions 01 slider, to create a bigger list that is called Pr4.

After that, the final step is the same step as the quad grid script, where P5 is the final list that will only take the same number of unrepeated coordinates from Pr4 as the inserted division value.

As mentioned earlier, the tool in Dynamo to place points is seen in Figure 60 is called **srf.PointAtParameter**. It will take coordinates from the final two lists Pr1 and Pr5 and place points on those coordinates, creating the visual diamond grid.

**List.Flatten** command basically cleans out all the inner loops that the algorithm is doing. It comes after the calculations script is closed, in this the list is flattened because the cycling script loops various times and can impact the efficiency of Dynamo.

#### 4.2.2.3.5 Switching between the two types of the scripted grids

After creating both grids, a slider was added to the workspace to allow the end user to choose an option for the grid design. In order to do that, a simple algorithm is scripted in Figure 61 and connected to the slider, allowing the swapping between the grids.

As seen in Figure 61; it is defined as **grid\_t(t,srf,div1,div2)**, where **t** refers to the slider that chooses the type of grid, **srf** is for the selected surface, and **div1,div2** is for the divisions slider, **srf,div1,div2** are the same inputs written in the definitions of the two grids as will.

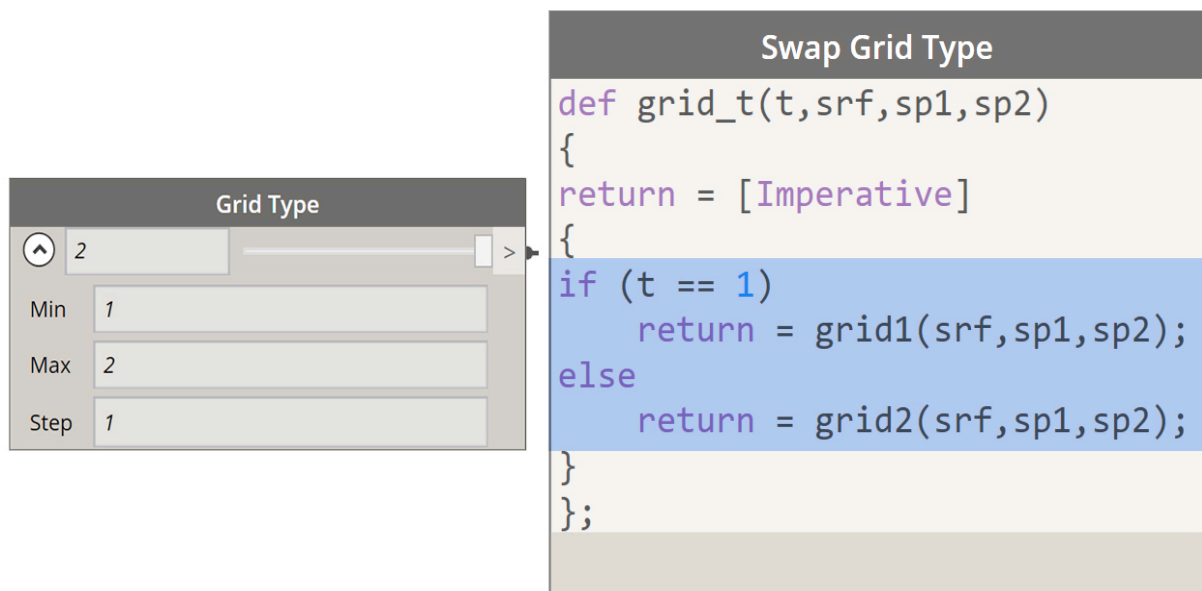


Figure 61: Swapping between Grid1 and Grid2 script and slider in Dynamo.2.0.3 (author)

An if statement is created in this script, allowing the selection between the two types of grids 1 or 2 (uniformed or non-uniformed). The highlighted script in Figure 61 states:

- If the slider selection is 1, if **t=1** then run the algorithm on grid 1, which is the quad option
- Or else run the algorithm on grid 2. Since the inserted slider is set to only move between one or two, **else** in the script refers to the second option.

#### 4.2.2.3.6 Updating the scripted grids upon testing it

Figure 62 shows the error that occurred when the grid is applied after splitting the ceiling, although Division 01 and Division 02 had very close inputs 8 and 7. The grid had divided each split space differently. In addition, the designer doesn't have the control to alter that division.

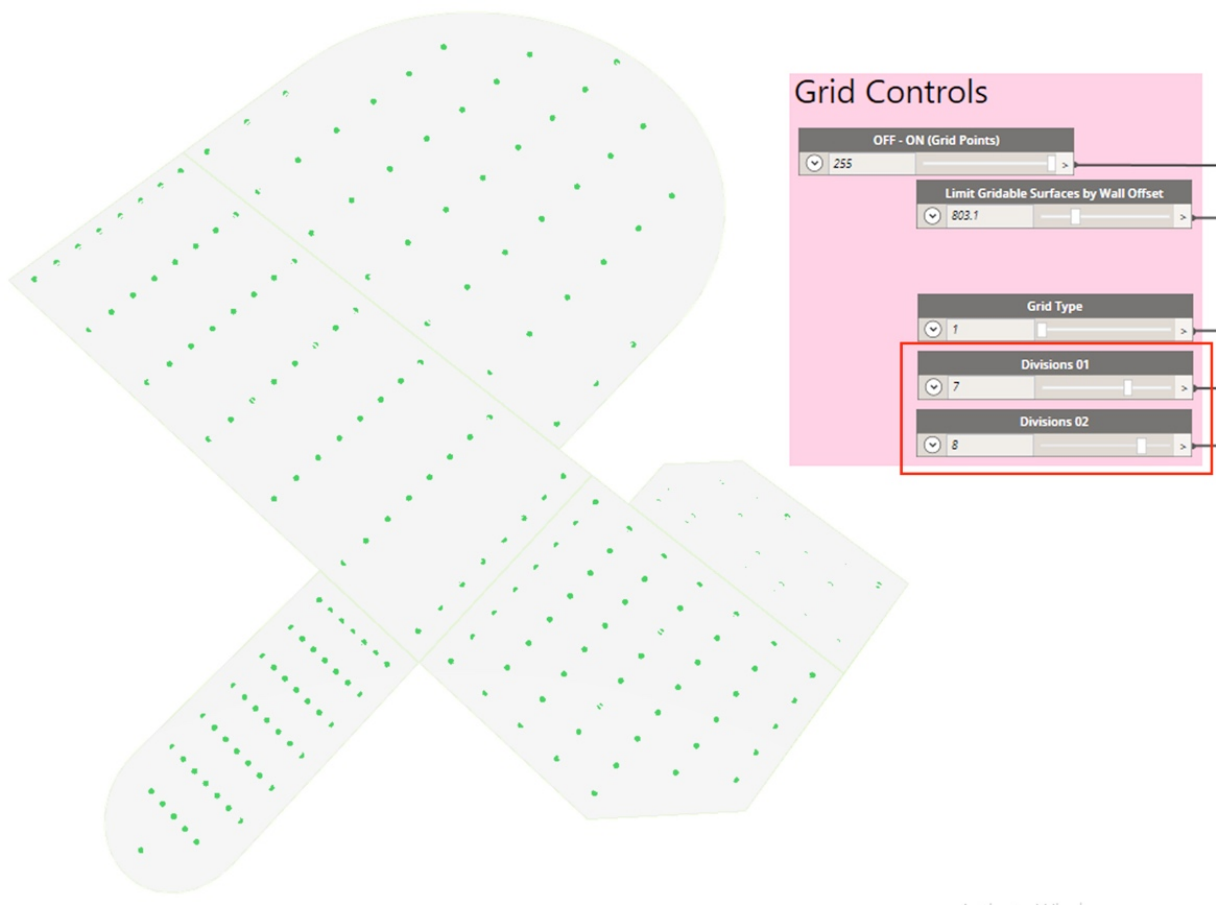


Figure 62: The quad grid results after applying it on a split surface in Dynamo 2.0.3 (author)

In order to fix this issue, the divisions controllers were replaced to become Spacing in Length and Spacing in Width as seen in Figure 63. The old approach inserts the numbers the end user wants. So, for example, if the surface 6000x6000mm, and the end user inserted in Division 01 a value of 300 points. The algorithm will automatically find x variable [ $x = \text{inserted value} / \text{length of the line}$ ], which will be 0.05 to place 300 points on the line. So, the end user is giving the generative system the quantity of lights they desire on the line.

The new approach replaced the value from the quantity of lights, to the spacing between each line. Now the algorithm finds x variable through [  $x = \frac{\text{the length of the line}}{\text{the inserted value called spacing}}$  ], in this case  $6000/300 = 20$  points on the line. And a parallel approach is applied on the width of the surface.

After that, the division will result in lines from both directions that will intersect to create the grid and proper intersection points. The same procedure is repeated on all the split surfaces. Resulting in an even distribution on all the split surfaces.

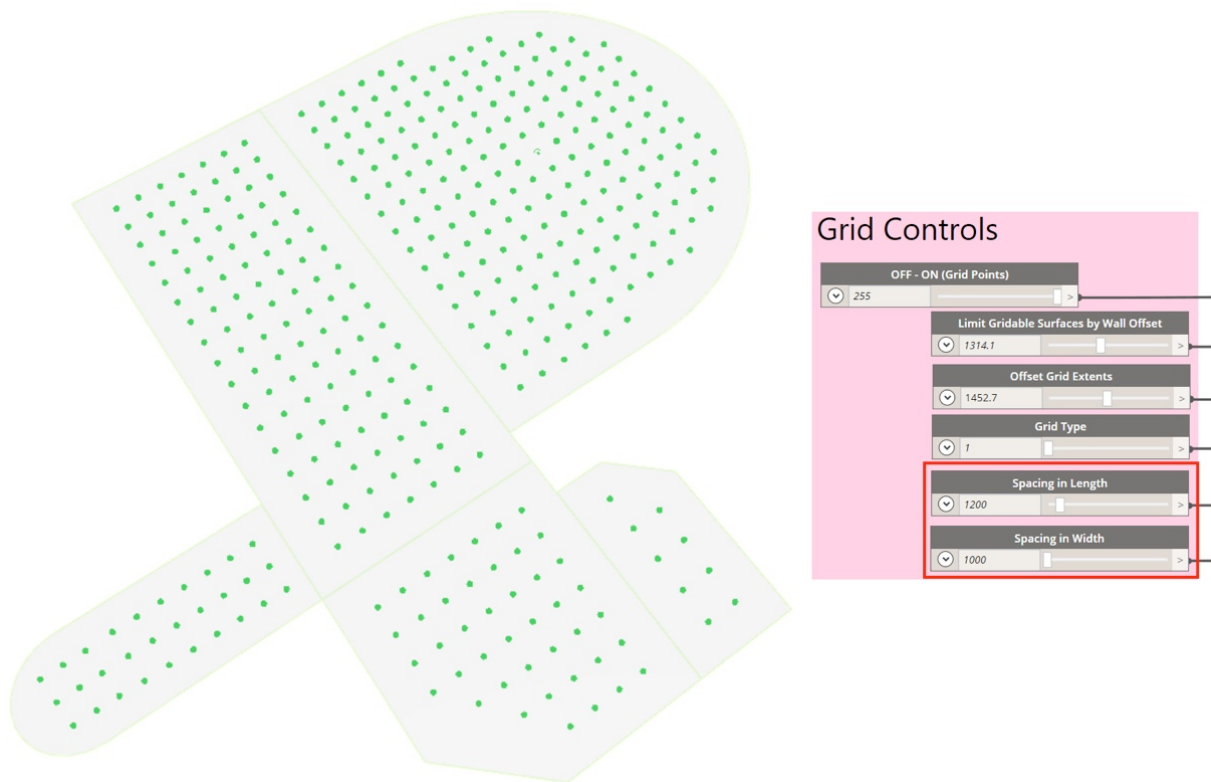


Figure 63: The quad grid results after applying it on a split surface using the new approach in Dynamo 2.0.3 (author)

In addition, different sliders were added that are shown in Figure 63 , allowing a better mobility by the end user. The sliders are discussed in detail in the following section *The grid controllers*.

Figure 64 is showing the final scripts for the both the quad and the diamond grids, with the added script highlighted in the script. The three variables fed to the algorithm now are: the



ceiling surface, the spacing in the length, and the spacing in width that are taken from the sliders.

Quad Grid	Diamond Grid
<pre>def grid1(srf,sp1,sp2) { div1 = Math.Ceiling(exnts(srf)[0]/sp1); div2 = Math.Ceiling(exnts(srf)[1]/sp2);  pr1 = div1==1?0.5:0..1..#div1; pr2 = div2==1?0.5:0..1..#div2;  pr3 = List.Reverse(pr2); pr4 = List.Cycle([pr2,pr3],div1); pr5 = List.TakeItems(pr4,div1);  grid_1 = srf.PointAtParameter(pr1,pr5);  return = List.Flatten(grid_1); };</pre>	<pre>def grid2(srf,sp1,sp2) { div1 = Math.Ceiling(exnts(srf)[0]/sp1); div2 = Math.Ceiling(exnts(srf)[1]/sp2);  pr1 = 0..1..#div1; pr2a = 1/div2/2..1..#div2; pr2b = 0..1-1/div2/2..#div2;  pr3 = List.Reverse(pr2b); pr4 = List.Cycle([pr2a,pr3],div1); pr5 = List.TakeItems(pr4,div1);  grid_2 = srf.PointAtParameter(pr1,pr5);  return = List.Flatten(grid_2); };</pre>

Figure 64: The final quad and diamond grids scripts in Dynamo 2.0.3 (author)

The highlight parts in Figure 64 are added to the original script, where **div1** is for calculating the length divided by the inserted value, in the length direction. And **div2** calculates the width and divides it by the inserted value from the second slider (Spacing in Width).

While **Math.Ceiling** that's seen in Figure 64 rounds up the division output to be a whole number instead of a fraction.

The rest of the script remains the same as before, however in the previous calculations explained prior to this section, the variables fed to the calculations were the values taken from Divisions 01 and Divisions 02 sliders, but in Figure 64; **div1** and **div2** are now the output of dividing the length or width by the inserted variable from the Spacing in length and Spacing in width sliders.

#### 4.2.2.3.7 The grid controllers

The grid controllers are a group of sliders that are connected to the discussed scripts earlier. Allowing the end user to have more flexibility over the shape and design of the grid.

The Off-On Grid Points slider seen in Figure 65 is a tactic to reduce the amount of calculations required on the total algorithm. This slider is applied almost on all the different light generating methods, channeling the algorithm to select specific design methods that the user chooses. The Off-On Grid Points slider has two values: as zero switches the grid approach off and 255 switches it on.

The Limit Grid-able Surface Slider can limit the grid to certain areas instead of the whole surface. And the Offset Grid Extents slider allows the end user to determine the desired offset distance from the wall, as zero value will result in applying the grid starting from the borders of the ceiling, while increasing the value in the slider will automatically result in increasing the offset space between the border of the ceiling and the grid.

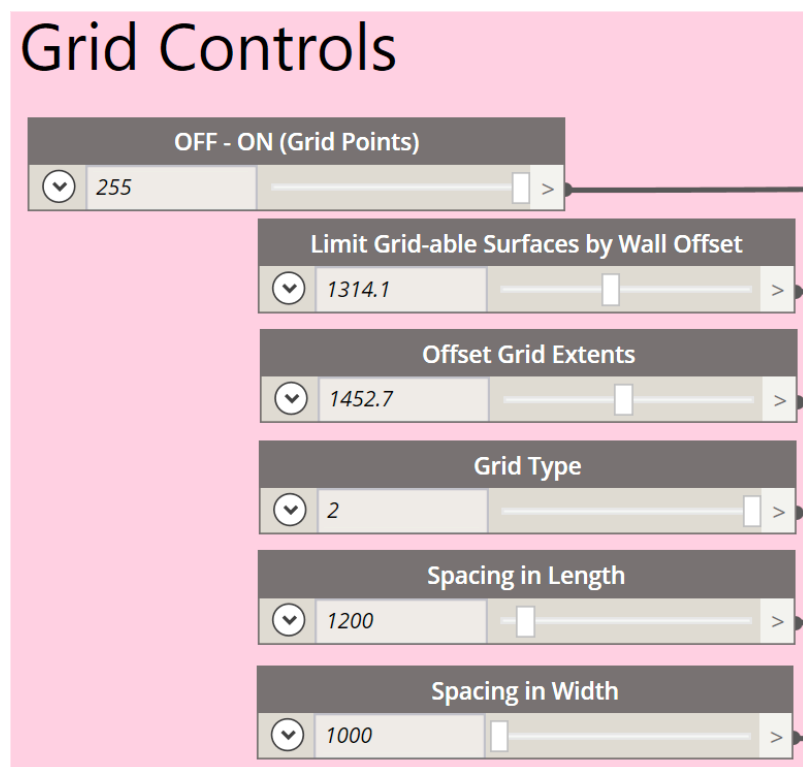


Figure 65: Grid Controllers Group in Dynamo 2.0.3 (author)

Figure 66 shows a ceiling with the Limit Grid-able with a value off 500. As the grid points applied throughout the ceiling.

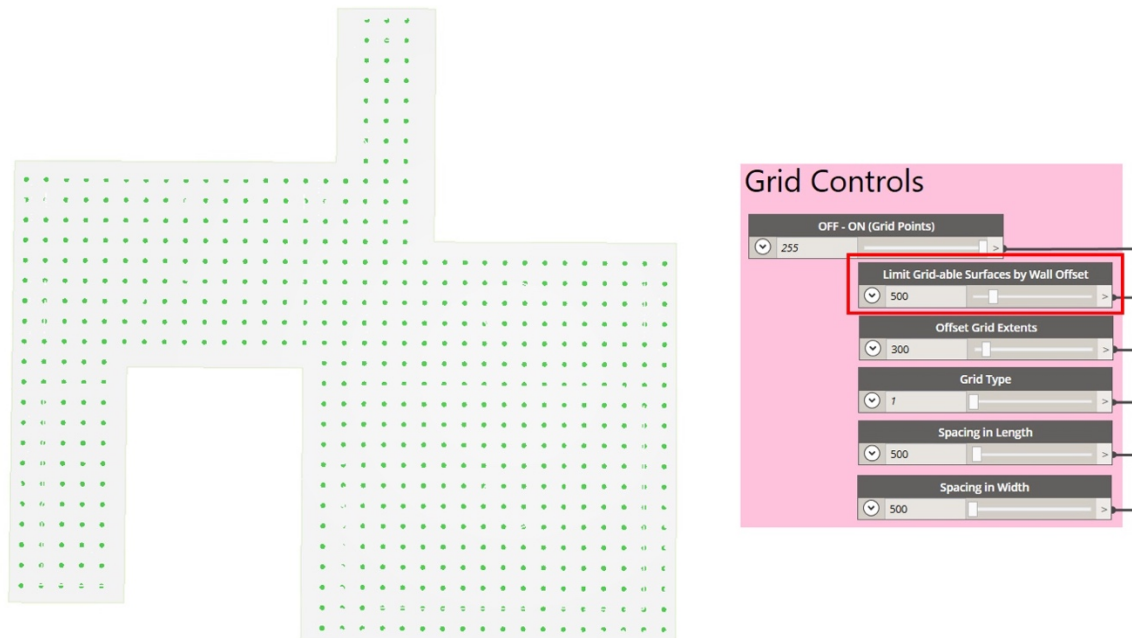


Figure 66: Ceiling with a grid points with Limit Grid-able of 500 in Dynamo 2.0.3 (author)

Figure 67 shows the same ceiling with the Limit Grid-able with a value of 1500, showing the grid points reduced in the narrow areas of the ceiling.

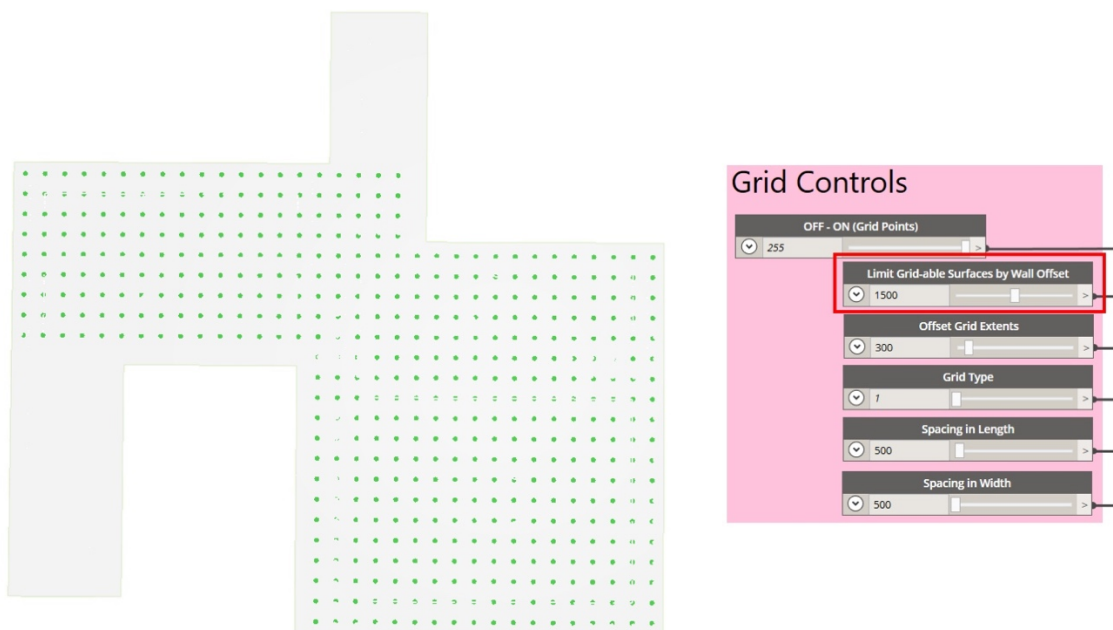


Figure 67: Ceiling with a grid points with Limit Grid-able of 500 in Dynamo 2.0.3 (author)

#### 4.2.2.3.8 Grid sub-approach: Creating Light patterns

##### 4.2.2.3.8.1 Uniform rectangular patterns

In order to place lights forming a uniform rectangular pattern, an outline of an imaginary rectangle was created with the center of it being a point taken from the grid's x and y intersections. The algorithm takes the length of the imaginary rectangle outline which is inserted by the end user using a slider, and dividing it to obtain the number of points that will be distributed on one side of the imaginary rectangular outline. These points refer to the lights coordinates depending on the required spacing between each light, which is also controlled by the end user.

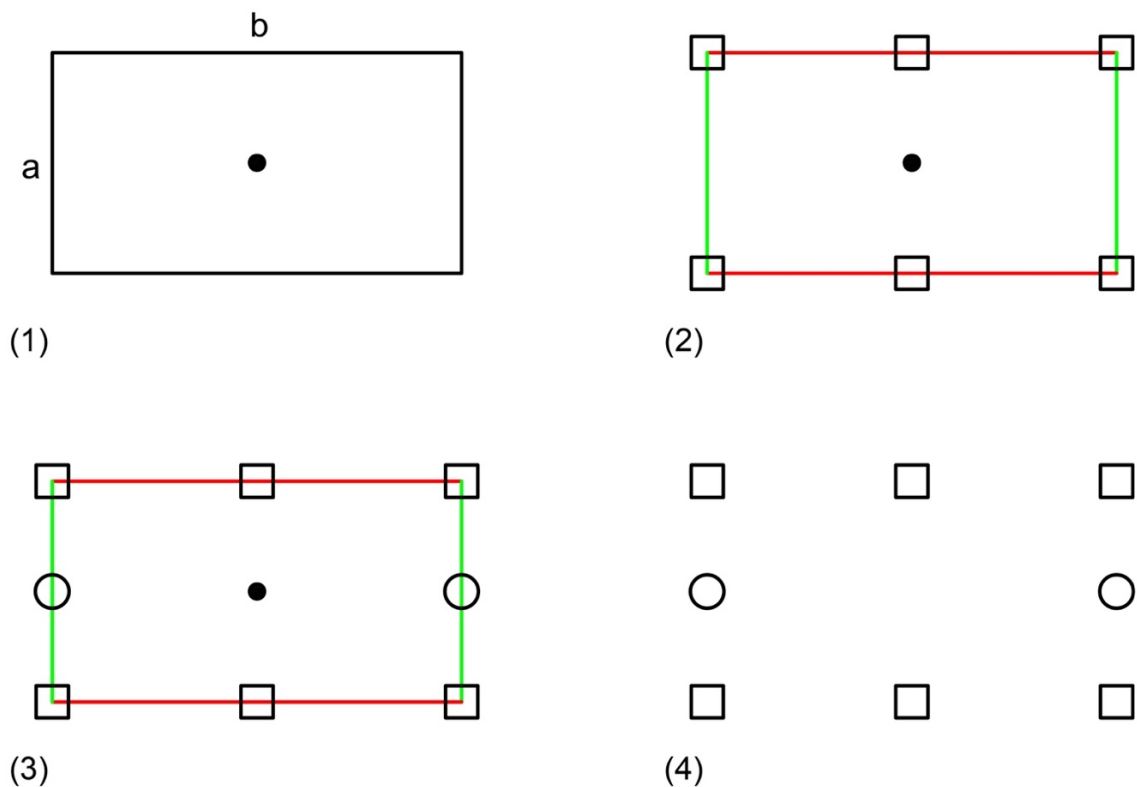


Figure 68: Simplified concept of creating rectangular uniform patterns (author)

Figure 68 shows a simplified concept of creating a uniform rectangular pattern, (1) shows the center point that's taken from the grid intersection, and the two parallel sides creating the imaginary rectangle outline, in this example one side is called a and the other is b.

(2) is showing how the algorithm will study the shape, as it will treat every side as it's own, hence the two different colors for the sides. For b, the algorithm had placed three square lights, including the edges of the line in the division.

While (3) shows in Figure 68 that -side a- had only placed one circular light in the center, neglecting the edges of the side, the numbers of the lights can be increased based on the end user input. The algorithm will always run this method; where if one side had included lights at the starting and at the end of it, it will automatically not place lights at the start and at the end of the other side. Preventing an overlapping in the corners between the two opposite sides. (4) in Figure 68 is showing the result, as the imaginary rectangular will not show and a rectangular light pattern is created.

Furthermore, the user's input to create the imaginary rectangle is controlled by the slider where the end user has access to manipulate both sides of the rectangular. The center point can change to become a center light.

In order to limit the amount of outputs the algorithm is able to produce, all the used lights to create patterns and the center lights had fixed dimensions which are shown in Figure 69, the end user has the ability to change the dimensions of the lights shown in millimeters, however a possibility of an overlapping might occur.

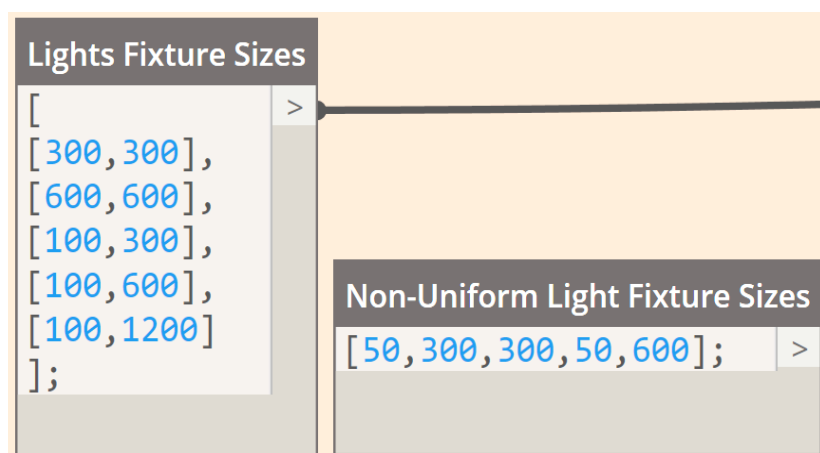


Figure 69: The lights sizes input in the algorithm for creating patterns in Dynamo 2.0.3 (author)

#### 4.2.2.3.8.1.1 Key issues in creating uniform rectangular lighting patterns

Different errors occurred while scripting the algorithm for the creating light patterns. The first issue happened once the lights' dimensions were added to the algorithm, as some of the lights had dimensions of 100x600mm and 100x1200mm. When the lights were applied using the explained method earlier, the results showed overlapping patterns shown in Figure 70 (1), where the red square represents 300x300mm light and the blue rectangles are 100x600mm, while the green hatched line is the imaginary rectangular outline.

In order to eliminate this issue, Figure 70 (2) shows the solution; as a script is added in the algorithm to measure the longest dimension of the placed light, and use that value to do the offset of the imaginary outline. So regardless of the direction of the light, the default offset will remain equal to the longest dimension of the light.

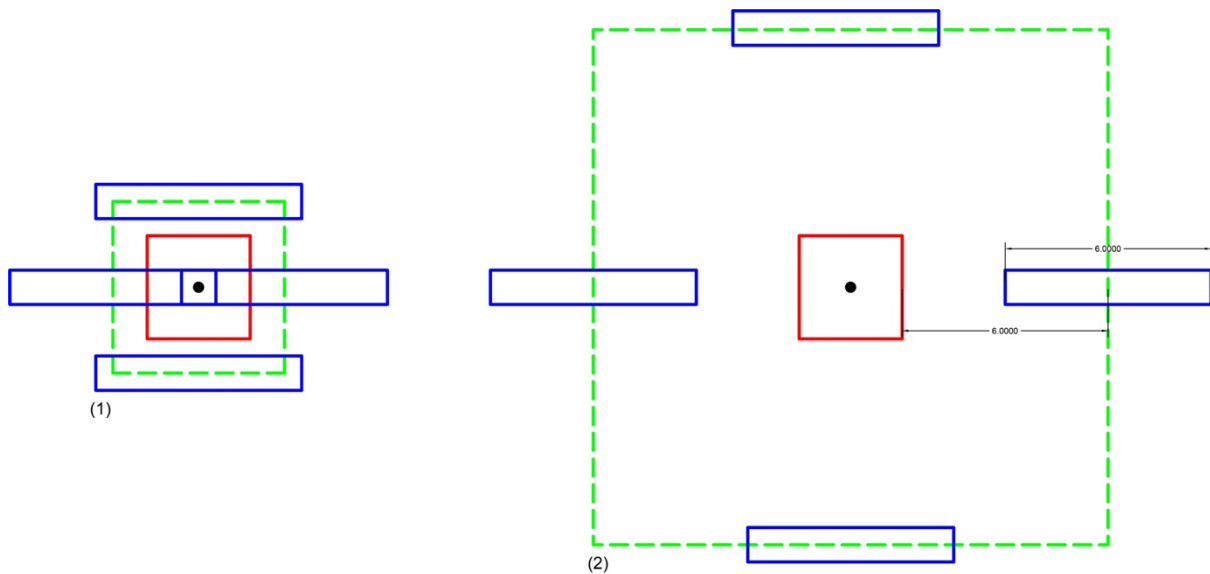


Figure 70: The algorithm error due to the dimensions of the lights and proposed solution (author)

Due to the fact that there isn't a specific rule for the spacing between each light, different sliders were given to allow manipulations in the patterns spacing. Although the default offset is equal to the longest length of the placed light, that offset can be changed using the sliders, as the imaginary rectangle can change its length using the slider (Pattern Scale 1) and its width using slider (Pattern Scale 2) that are shown later in *The light patterns controllers*.

The second issue in creating rectangular patterns, was the orientation of the lights that are placed using the imaginary rectangle outline. Where the algorithm had managed to produce lights that are shown in Figure 71 (1) but didn't produce option (2).

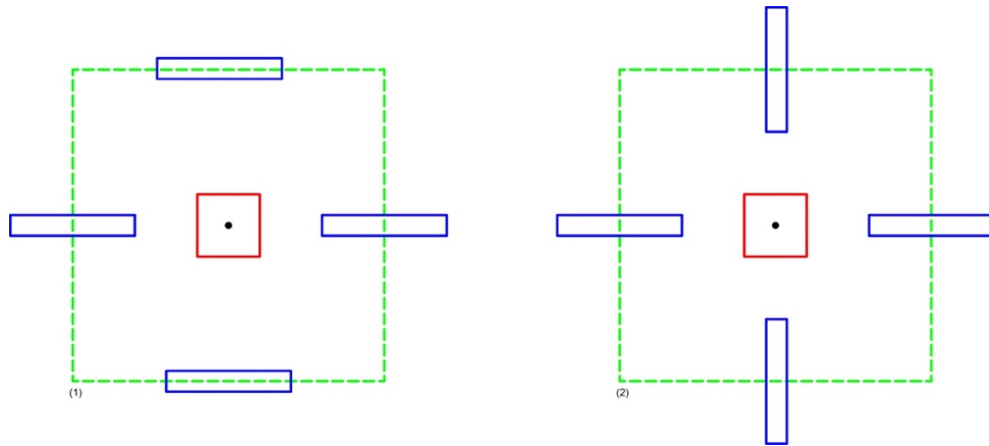


Figure 71: Lights orientation issue in uniform rectangular lighting algorithm (author)

After various trials, this issue was fixed through scripting different steps that are explained in Figure 72.

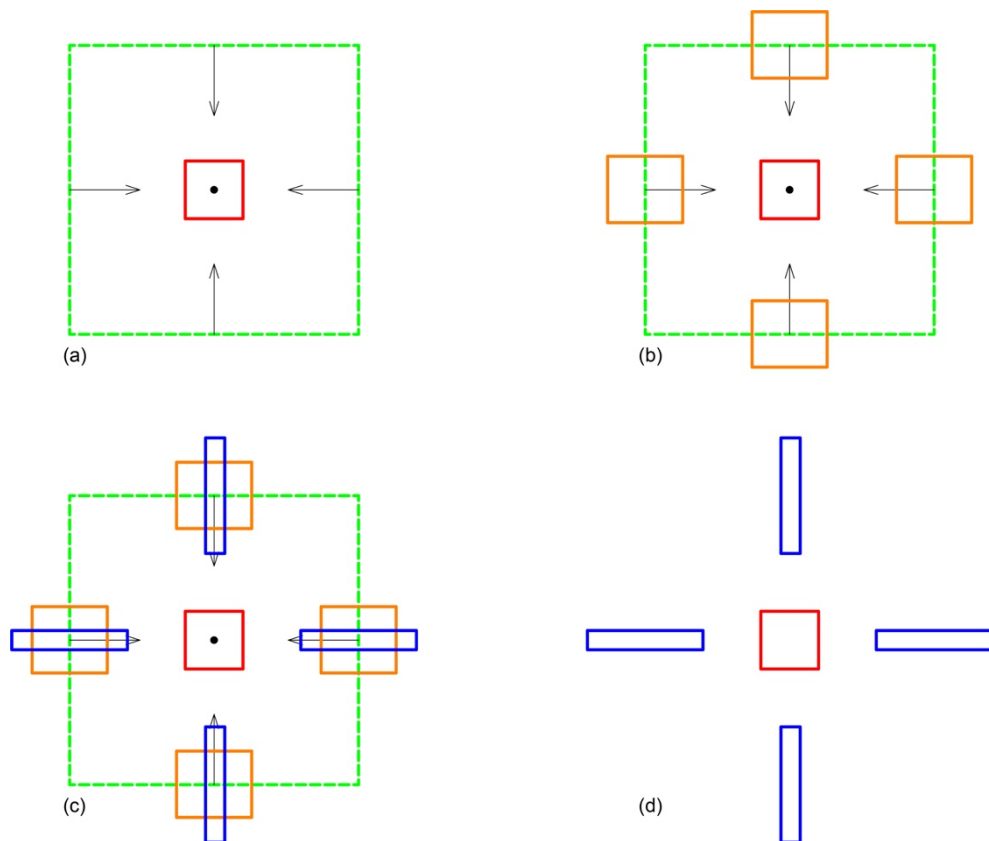


Figure 72: The steps to change the orientations of the lights (author)

The first step was to find the Normal Vector of every line that created the imaginary rectangle outline, colored in black in (a) in Figure 72. The Normal Vector is defined as 90° projection vector that comes from the center of the line towards the inside of the shape (The Dynamo Primer 2019).

The second step was to create a plane for every line that creates the imaginary rectangle outline, they're colored in orange in (b) in Figure 72. The plane is created from a Node found in Dynamo's library that's called **Plane.ByOriginNormalXAxis**; where the Normal Vector is fed into that plane, forcing every plane to be aligned with the axis of each line created the imaginary rectangle outline.

To create a rectangle that represents the light with the required dimensions, a Node called **Rectangle.ByWidthLength** is added; where the plane direction is fed into that Node, along with the required dimensions of the light. Resulting in creating rectangles on top of the planes that holds the same orientation as the plane as seen in Figure 72 colored in blue.

(d) in Figure 72 shows the final results; where the planes nor the Normal Vectors are displayed for the end user and only the created pattern will appear.

#### **4.2.2.3.8.1.2 Uniform rectangular patterns script explanation**

Figure 73 shows the script that explains the division process to produce the uniform rectangular patterns. **Small1** highlighted in green in Figure 73 is taking the input lists of lights dimensions shown in Figure 69 and reversing it, as some lights will be drawn as 100mm being the length, and 1200 being the width, so the reversing option flips the lights dimensions, then **small\_t** uses a command that's called **List.UniqueItems** which removes any repeated lights, as a reversed 300x300mm dimensions will result in an unnecessary repetition of lights. And applies **List.Transpose** command which will take the length of the input lights and place them in a list, and the width of the lights and place them in another list. Those two lists will feed the



rectangle the two required dimensions to create it. These commands are not connected to the for-loop and will be constant in any pattern creation, the for-loop starts after **return=[Imperative]** and ends after **return** that is found at the end of the script.

```

UNIFORM RECTANGULAR PATTERNS
def pat1(p0,big:var[],small:var[].[.])
{
  z = Vector.ZAxis();
  pln0 = Plane.ByOriginNormal(p0,z);
  r0 = Rectangle.ByWidthLength(pln0,big[0],big[1]);
  small1 = List.Reverse((small)@L2<1>);
  small2 = List.Flatten(List.Transpose([small,small1]),1);
  small_t = List.Transpose(List.UniqueItems(small2));

  return = [Imperative]
  {
    pts = [];
    recs = [];
    for (h in 0..List.Count(small2)-1)
    {
      wid1 = List.MaximumItem(small2[h]);
      r1 = r0.Offset(wid1).Explode();
      c = small_t[0][h]>small_t[1][h]?small_t[0][h]:small_t[1][h];
      a = Math.Floor(r1[0].Length/(c*1.5))>3?3:Math.Floor(r1[0].Length/(c*1.5));
      b = Math.Floor(r1[1].Length/(c*1.5))>3?3:Math.Floor(r1[1].Length/(c*1.5));
      n0 = 1..(a<=1?1:a);
      n1 = 1..(b<=1?1:b);
      for (i in 0..(List.Count(n0)==1?0:(List.Count(n0)-1)))
      {
        resultp = [];
        resultr = [];
        for (j in 0..(List.Count(n1)==1?0:(List.Count(n1)-1)))
        {
          jj = n1[j]+1;
          pts1 = r1[0].PointAtParameter(n0[i]==1?[0.5]:
            (0..1..#n0[i]));
          pts2 = r1[1].PointAtParameter(n1[j]==1?[0.5]:
            (1/jj..1-1/jj..#n1[j]));
          pts3 = r1[2].PointAtParameter(n0[i]==1?[0.5]:
            (0..1..#n0[i]));
          pts4 = r1[3].PointAtParameter(n1[j]==1?[0.5]:
            (1/jj..1-1/jj..#n1[j]));
          vc1 = r1[0].NormalAtParameter(0.5);
          vc2 = r1[1].NormalAtParameter(0.5);
          vc3 = r1[2].NormalAtParameter(0.5);
          vc4 = r1[3].NormalAtParameter(0.5);
          pln1 = Plane.ByOriginNormalXAxis(pts1,z,vc1);
          pln2 = Plane.ByOriginNormalXAxis(pts2,z,vc2);
          pln3 = Plane.ByOriginNormalXAxis(pts3,z,vc3);
          pln4 = Plane.ByOriginNormalXAxis(pts4,z,vc4);
          rc1 = Rectangle.ByWidthLength(pln1,small_t[0][h],small_t[1][h]);
          rc2 = Rectangle.ByWidthLength(pln2,small_t[0][h],small_t[1][h]);
          rc3 = Rectangle.ByWidthLength(pln3,small_t[0][h],small_t[1][h]);
          rc4 = Rectangle.ByWidthLength(pln4,small_t[0][h],small_t[1][h]);
          resultp[j] = [pts1,pts2,pts3,pts4];
          resultr[j] = [rc1,rc2,rc3,rc4];
        }
        pts[h][i] = resultp;
        recs[h][i] = resultr;
      }
    }
    return = [r0,List.Flatten(pts,2),List.Flatten(recs,2)];
  }
};

```

Figure 73: Uniform rectangular patterns script in Dynamo 2.0.3 (author)

Figure 73 displays the three for-loops created where the algorithm will pick only one variable from the input light dimensions, ignoring the rest of the dimensions. And run the all the for-loops on that variable only, then place the results it in a list. After that, the algorithm will pick the second light and do the same procedure and add it to the final list. So, in this case, since the input lights have five different dimensions, the algorithm will for-loop five times, every for-loop has three for-loops inside of it. That overall for-loop process is seen in **(h in 0..List.Count (small2)-1)** highlighted in blue; as **h** index takes the lists created from the transpose command, and runs the for-loop using it. That is why **h** is repetitive throughout the script. Without the sub-level lists of **h**. The for-loop will have shortage of looping.

**R0** highlighted in yellow; shows the creation of the center light. while **R1** shows the offset of R0 to create the imaginary outline rectangular. It is exploded to get individual sides. One side of the imaginary rectangular outline is called a and the other is b. To create proper division, the two parallel lines of both a and b were grouped, and the division was applied on the groups immediately.

**C** highlighted in orange; allows the algorithm to pick the first value from the first list in **small\_t** list and checks if that value is bigger than the first item in the second list in **small\_t**. If it is bigger it will return to the first list, using the value from the first list. Otherwise, the algorithm will to the second list. This procedure feeds the created rectangular light shape only one dimension, while the opposite dimension is fed to the rectangular from **wid1**; which refers to original light inputs found in **small\_2** list. **C** definition essentially created to identify which length of the light dimensions from the two lists is longer, to feed it into the division process of the imaginary rectangle outline. For example, if the division calculations did not consider the longest length of the light, and the short length of the lights had been placed in the divisions, once the light rectangles will be placed, they will overlap. This ensures the overlapping within the lights in one side of the imaginary rectangle outline does not occur.

Highlighted in purple; (a) represents one of the groups that creates the imaginary rectangle outline, assuming that it is the length groups. While (b) is the opposite group which will be the width group that forms the imaginary rectangle outline.

The division calculations of (a) side that is applied on the imaginary rectangular outline; where the length of **R1** -which is the offsetted rectangle- is measured, divided by **C** -which is the longer dimension of the light- and multiplied by 1.5 to increase the space a bit more, so the pattern will not end up with lights that are very close to each other. **Math.Floor** command rounds down the results, as the division might result in fractions. Rounding it down allows the algorithm to lose the extra fraction that might result in having more lights. After that, the algorithm will check if the patterns results are more than 3, it will revert to three. Otherwise, the calculations will run again if it is less than 3.

The 3 restriction is added after the algorithm was created and done, to limit the patterns options. The algorithm was able to produce up to 4000 options, which did not feel beneficial and simply time consuming to view by the end-user. So, limiting the patterns to only the first 3 from every sub-list had reduced the overall options. (b) side has also the same calculations as (a) but for the opposite group of the imaginary rectangle outline.

Highlighted in red in Figure 73, **n0** for group (a) and **n1** is for group (b) and **I** for-loop; where these three definitions creates the varieties of designs in the patterns. **n0** creates a list starting from 1 all the way to 3 -which is the added limit- where the first option the division for group (a) will be by 1, after that; lights will be placed and patterns will result from only one division in one side of the rectangular. Then using **I** for-loop it will run again and divide by 2; placing two lights on the points of the divisions and producing patterns, and finally the algorithm will run 3 divisions as well. All these divisions are only for group (a). Then **n1** will repeat the same scenario for group (b). **I** definition basically counts and creates an index of different values of

divisions to divide the rectangle. A statement is created in **I** where if the index is zero, which means 1 because the Index list in Dynamo starts at 0. Then this list can't run more than once, otherwise; if **n0** count is not 1 then count it and subtract 1. The -1 is added because the index starts from zero in Dynamo.

Another for-loop called **j** is created outlined in black in Figure 73. Where **pts1**, **pts2**, **pts3** and **pt3** that are highlighted in pink in Figure 73, will check if the division for both **n0** and **n1**- which refers back to both sides of the imaginary rectangle outline- is 1; then the division point will be placed at the middle of the line, but if more than 1, then run the division algorithm to place the points on equal spaces.

While the gray shaded scripts are the Normal Vector, Plane and the light rectangular creations that was explained previously using Figure 72.

#### 4.2.2.3.8.1.3 The produced uniform rectangular patterns

This script had produced around 470 light patterns. Figure 74 and Figure 75 show some of the produced lighting patterns.

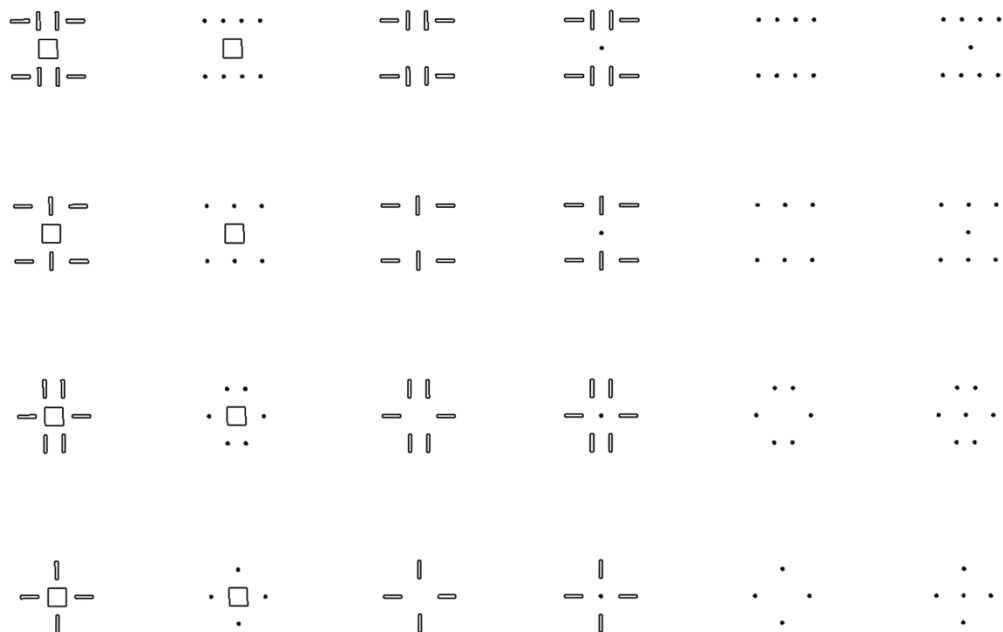


Figure 74: Uniform rectangular patterns output generated in Dynamo 2.0.3 (author)

Every pattern will be placed on the grid and repeated to create uniform lighting that was created using generative designs. The grid will not mix the patterns together in one space, although this option can be scripted, it is not beneficial from a design or execution perspective.



Figure 75: Uniform rectangular patterns output generated in Dynamo 2.0.3 (author)

#### 4.2.2.3.8.2 Non-uniform rectangular light patterns

The non-uniform rectangular patterns were an output of a scripting mistake, which resulted in mixing different sizes of lights, generating irregular patterns that were unique and different from the uniformed patterns.

##### 4.2.2.3.8.2.1 The scripted error that produced the non-uniform patterns



Figure 76: The script error that resulted in creating non-uniform patterns in Dynamo 2.0.3 (author)

While scripting the green highlighted part in the uniform lighting algorithm as seen in Figure 76; instead of refereeing **small\_t** to index **[0][h]** in **rc1**, the script had referred to index **[h]** only; which is the transposed list as seen in Figure 76. So, writing an **[h]** index resulted in picking all the values in every sub-list that are outlined in black, and running the algorithm using only one dimension. So, if the first list in **[h]** has 100 value, the algorithm will place that value to **pts1** and create a rectangle that is 100x100 dimension. Then it will go to the second value found in the first list in **[h]** list, which is 300 in Figure 76 and repeat the same procedure. Until it reaches to a point where the values inside the **[h]** sub-lists ran out, while **pts** count is still running, because **pts** is using **[0]** index not **[h]** index. Which will result in creating only center points instead of shapes. The repetitive results of producing only center points and not patterns had highlighted an error in the algorithm.

#### 4.2.2.3.8.2.2 Non-uniform patterns script explanation

Since [h] list had only fed the algorithm one value at a time in the error. A simpler lighting input were created for the non-uniform lighting as shown in Figure 69, which takes every variable as the width and the length. So, for example 50mm will create a spotlight that is 50x50mm, and so on.

Figure 77 shows the non-uniform rectangular pattern, where all the reverse lists written in the uniform lighting script were removed as the lighting inputs are only one variable for both length and width. After that, a for-loop is running based on the number of lights inputted in the algorithm, since the non-uniform lights input are 5 values, creating 5 types of lights. Then the for-loop will run 5 times using a different value every time.

**C** highlighted in red in Figure 77 is now the value of h only, while in the uniform lighting script **c** had checked the longer length of the light to add it in the division. And **R1**, highlighted in green will create the offset and explode it to get each line of the imaginary rectangle outline exactly as the uniform lighting script.

The calculations of both (a) and (b) groups that are highlighted in blue in Figure 77 are calculated as the uniform lighting, where group (a) refers to one side of the rectangle and (b) refers to the opposite side of the rectangle that creates the imaginary rectangle outline. As the calculations takes the length of **R1** -which is the offsetted rectangle- and divide it by **C** -which is light input - and multiplied by 1.5 to increase the space between the lights and. **Math.Floor** is also applied in this script to rounds down the results.

The gray shaded scripts are as the uniform lighting script that were explained in detail previously using Figure 72.

# NON-UNIFORM RECTANGULAR PATTERNS

```
def pat2(p0,big:var[],small1:var[])
{
  z = Vector.ZAxis();
  pln0 = Plane.ByOriginNormal(p0,z);
  r0 = Rectangle.ByWidthLength(pln0,big[0],big[1]);
  small_t = small1;

  return = [Imperative]
  {
    pts = [];
    recs = [];
    nt = List.Count(small_t);
    for (h in 0..nt-1)
    {
      c = small_t[h];
      r1 = r0.Offset(c).Explode();
      a = Math.Floor(r1[0].Length/(c*1.5));
      b = Math.Floor(r1[1].Length/(c*1.5));
      n0 = 1..(a<=1?1:(a>nt?nt:a));
      n1 = 1..(b<=3?1:(b>nt?nt:b));
      for (i in 0..(List.Count(n0)==1?0:(List.Count(n0)-1)))
      {
        resultp = [];
        resultr = [];
        for (j in 0..(List.Count(n1)==1?0:(List.Count(n1)-1)))
        {
          jj = n1[j]-1;
          pts1 = r1[0].PointAtParameter(n0[i]==1?[0.5]:
            (0..1..#n0[i]));
          pts2 = r1[1].PointAtParameter(n1[j]<=3?[0.5]:
            (1/jj..1-1/jj..#jj-1));
          pts3 = r1[2].PointAtParameter(n0[i]==1?[0.5]:
            (0..1..#n0[i]));
          pts4 = r1[3].PointAtParameter(n1[j]<=3?[0.5]:
            (1/jj..1-1/jj..#jj-1));
          vc1 = r1[0].NormalAtParameter(0.5);
          vc2 = r1[1].NormalAtParameter(0.5);
          vc3 = r1[2].NormalAtParameter(0.5);
          vc4 = r1[3].NormalAtParameter(0.5);
          pln1 = Plane.ByOriginNormalXAxis(pts1,z,vc1);
          pln2 = Plane.ByOriginNormalXAxis(pts2,z,vc2);
          pln3 = Plane.ByOriginNormalXAxis(pts3,z,vc3);
          pln4 = Plane.ByOriginNormalXAxis(pts4,z,vc4);
          rc1 = Rectangle.ByWidthLength(pln1,small_t,small_t);
          rc2 = Rectangle.ByWidthLength(pln2,small_t,small_t);
          rc3 = Rectangle.ByWidthLength(pln3,small_t,small_t);
          rc4 = Rectangle.ByWidthLength(pln4,small_t,small_t);
          resultp[j] = [pts1,pts2,pts3,pts4];
          resultr[j] = [rc1,rc2,rc3,rc4];
        }
        pts[h][i] = resultp;
        recs[h][i] = resultr;
      }
    }
  }
  return = [r0,List.Flatten(pts,2),List.Flatten(recs,2)];
}
};
```

Figure 77: Non-uniform pattern script in Dynamo 2.0.3 (author)



#### 4.2.2.3.8.2.3 The produced non-uniform rectangular patterns

The algorithm had produced for the non-uniform rectangular pattern 409 different patterns using the following light dimensions: 500x500, 300x300, 600x600mm in addition to adding points as well as a spotlight reference. Figure 78 shows some of the outputs.



Figure 78: Non-uniform rectangular pattern outputs generated in Dynamo 2.0.3 (author)

#### 4.2.2.3.8.3 Uniform circular and polygon patterns

The circular pattern will start from a point taken from the intersection of x and y axis from the grid. And a circle will be drawn around this point. Since the circle doesn't have a length and a width, the algorithm will treat it as one line which will be divided by the input number of lights that will place points around the circle, and on those points the different light shapes will be placed.

Due to the fact that the imaginary outline is a circle, the algorithm is able to create various polygon shapes inside the circle as seen in Figure 79; where it starts from a circle to a triangle, and as the end user starts adding more sides using a given slider, the shape develops to a quadrilateral, pentagon, hexagon, heptagon, all the way to decagon.

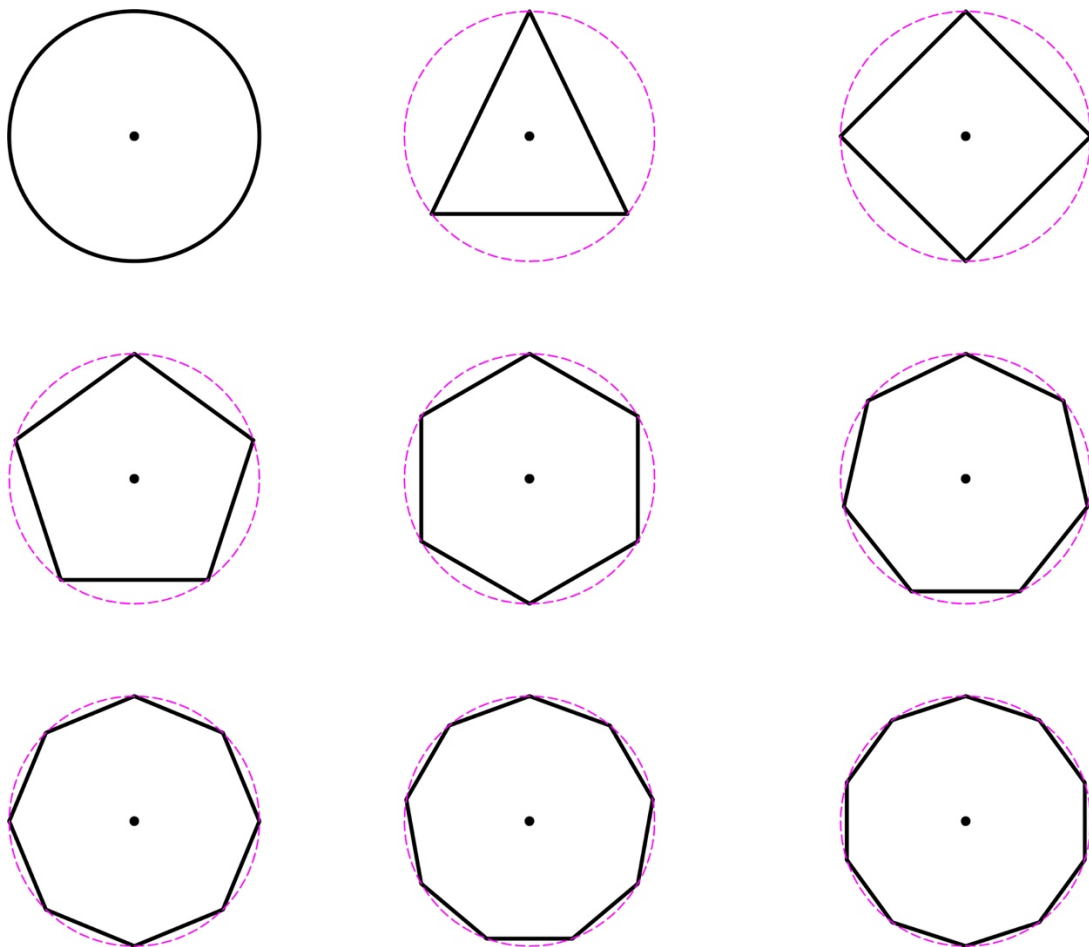


Figure 79: The different shapes the polygon algorithm develops into

#### 4.2.2.3.8.3.1 Uniform circular and polygon script explanation

For the algorithm to work; the user has to input the radius of the circle and the number of sides the polygon shape will have. In addition to the points that will be taken from the grid intersection and the light variables that are used for all the patterns.

As seen in Figure 80; **circ** highlighted in yellow created the circle using a plane that was created using the point on the grid in **pln0**. The end user has the ability to change the number of sides using a slider that's shown further in the section *The light patterns controllers*; the polygon shapes starts once the slider reaches to 3, as 3 sides creates a triangle, but between 0 and two in the slider, the end user will only get a circle. In order to do that, an if statement was created that's shown in in Figure 80 under **s0** definition that's highlighted in red; where a it states the following: if the number of sides are less or equal to two, always run the algorithm to creating a circle, otherwise the algorithm will create a polygon which uses two variables ( the created circle and the number of sides) as inputs.

The three definitions of **small1**, **small2** and **small\_t** highlighted in green are for the light inputs. To allow different orientations for the lights. The same lists were discussed in detail in creating uniform lighting patterns.

**s1** highlighted in blue will offset the shape, so if it's a circle, the algorithm will offset the circle, and if it is a polygon it will offset it by the width given and then explode it. Since the circle is treated as one line, only one calculation is required which is the same as the other calculations in the previous patterns creations, where **s1** is measured and divided by **C** - that calculates the longer dimension of the light- and multiplied by 1.5. After that the algorithm checks if it is more than 3, then it will return to 3 as a limitation parameter that is applied and discussed in creating uniform lighting. Finally, the result is rounded down to avoid any fractions output.

**n0** highlighted in orange in Figure 80 is also the list of counts that can be found in the previous patterns creations, allowing the different divisions for the polygon to create the different variety of patterns. **i** is the start of a different for-loop; if the count is 1, the division output will be one point only, so if it's a circle, the output will only be one light, whilst if it's a triangle; it will have three lights because of its sides, and so on.

```

UNIFORM CIRCULAR & POLYGON PATTERNS

def pat3(p0,rad,sides,small:var[[]..[]])
{
  z = Vector.ZAxis();
  pln0 = Plane.ByOriginNormal(p0,z);
  circ = Circle.ByPlaneRadius(pln0,rad);
  s0 = sides<=2?circ:Polygon.RegularPolygon(circ,sides);
  small11 = List.Reverse((small)@L2<1>);
  small12 = List.Flatten(List.Transpose([small,small11]),1);
  small_t = List.Transpose(List.UniqueItems(small12));

  return = [Imperative]
  {
    pts = [];
    recs = [];
    for (h in 0..List.Count(small)-1)
    {
      wid1 = List.MaximumItem(small[h]);
      s1 = s0.Offset(wid1).Explode();
      c = small_t[0][h]>small_t[1][h]?small_t[0][h]:small_t[1][h];
      a = Math.Floor(s1[0].Length/(c*1.5))>3?3:Math.Floor(s1[0].Length/(c*1.5));
      n0 = 1..(a<=1?1:a);
      for (i in 0..(List.Count(n0)==1?0:(List.Count(n0)-1)))
      {
        resultp = [];
        resultr = [];
        for (j in 0..(List.Count(s1)<=2?0:(List.Count(s1)-1)))
        {
          pts1 = s1[j].PointAtParameter(n0[i]==1?[0.5]:
            (1/n0[i]..1..#n0[i]));
          vc1 = s1[j].NormalAtParameter(0.5);
          pln1 = Plane.ByOriginNormalXAxis(pts1,z,vc1);
          rc1 = Rectangle.ByWidthLength(pln1,small_t[0][h],small_t[1][h]);
          resultp[j] = pts1;
          resultr[j] = rc1;
        }
        pts[h][i] = resultp;
        recs[h][i] = resultr;
      }
    }
    return = [s0,List.Flatten(pts,1),List.Flatten(recs,1)];
  }
};

```

Figure 80: Uniform circular and polygon patterns script in Dynamo 2.0.3 (author)

Another level of for-loop is created – highlighted in gray- which goes through all the edges of the polygon and divide every side of the shape. So, if it is a circle the for-loop will run once, if it is a triangle it will run three times and so on. In other terms, this for-loop counts the number of lines that creates the polygon after exploding it, and if the number of edges is less than 2, the algorithm will always have 1 point. Otherwise then it will deal with the division based on the numbers of the number of the lines creating it.

Furthermore; **pts1** definition that is highlighted in purple in Figure 80 is added to prevent the over lapping of the light points in the corners of the polygon shapes. Where the division of every line does not skips placing a point at the beginning while placing a point at the end. The rest of the script creates a Normal Vector, a plane and draws the light rectangles using index [h][0] as per the uniform lighting algorithm that discussed these definitions in depth.

#### 4.2.2.3.8.3.2 The produced uniform circular and polygon patterns

The produced patterns were around 530 patterns for all the polygon shapes. The variety refers to two main parameters, the lights inputs, and the division method. If the light dimensions are increase, the outputs will increase coherently. Figure 81 shows a minor part from the circle pattern results and Figure 82 and Figure 83 displays some of the patterns for the different polygon shapes.

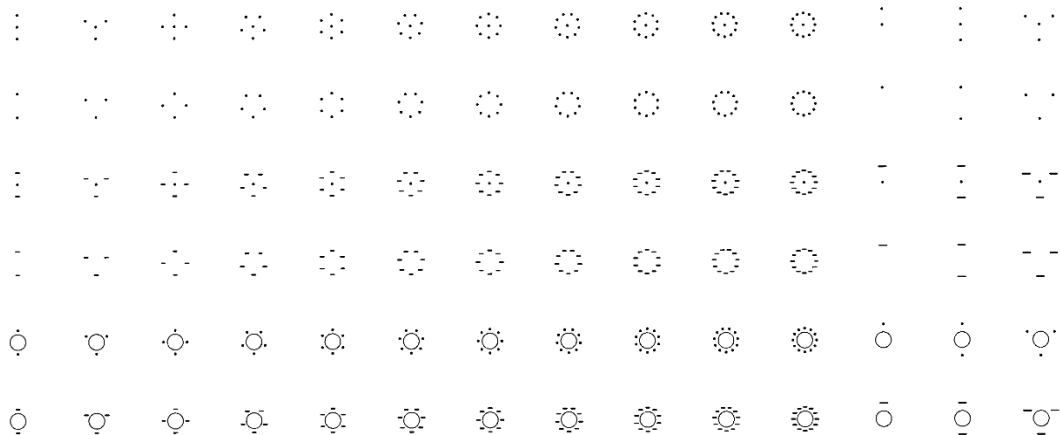


Figure 81: Uniform circular patterns outputs generated in Dynamo 2.0.3 (author)

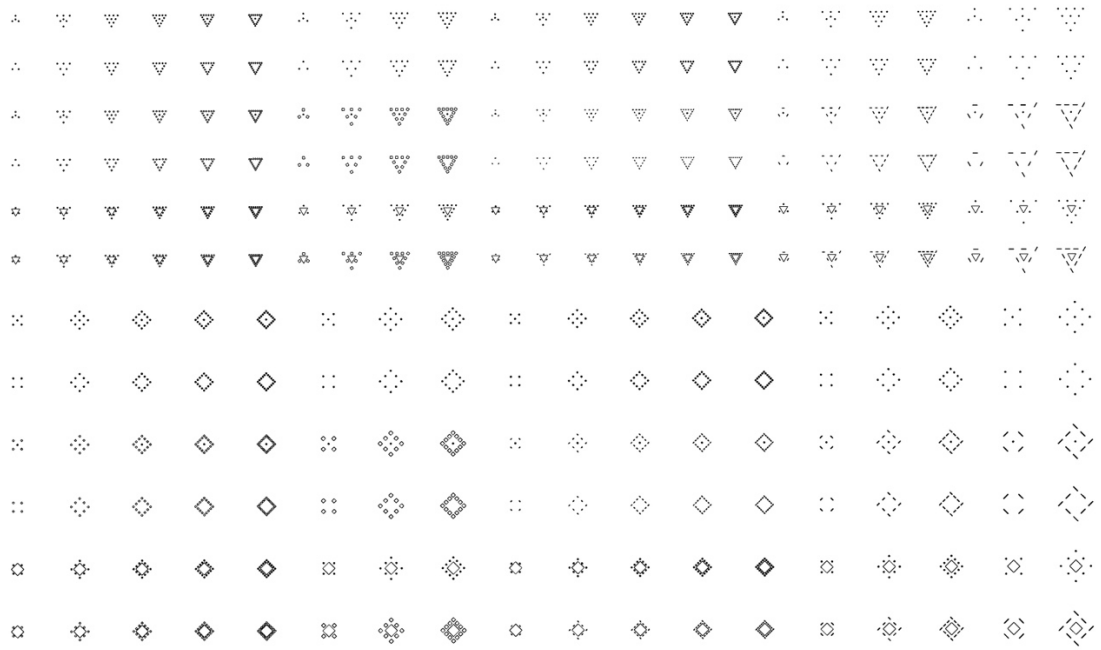


Figure 82: Triangle and quad uniform patterns outputs generated in Dynamo 2.0.3 (author)

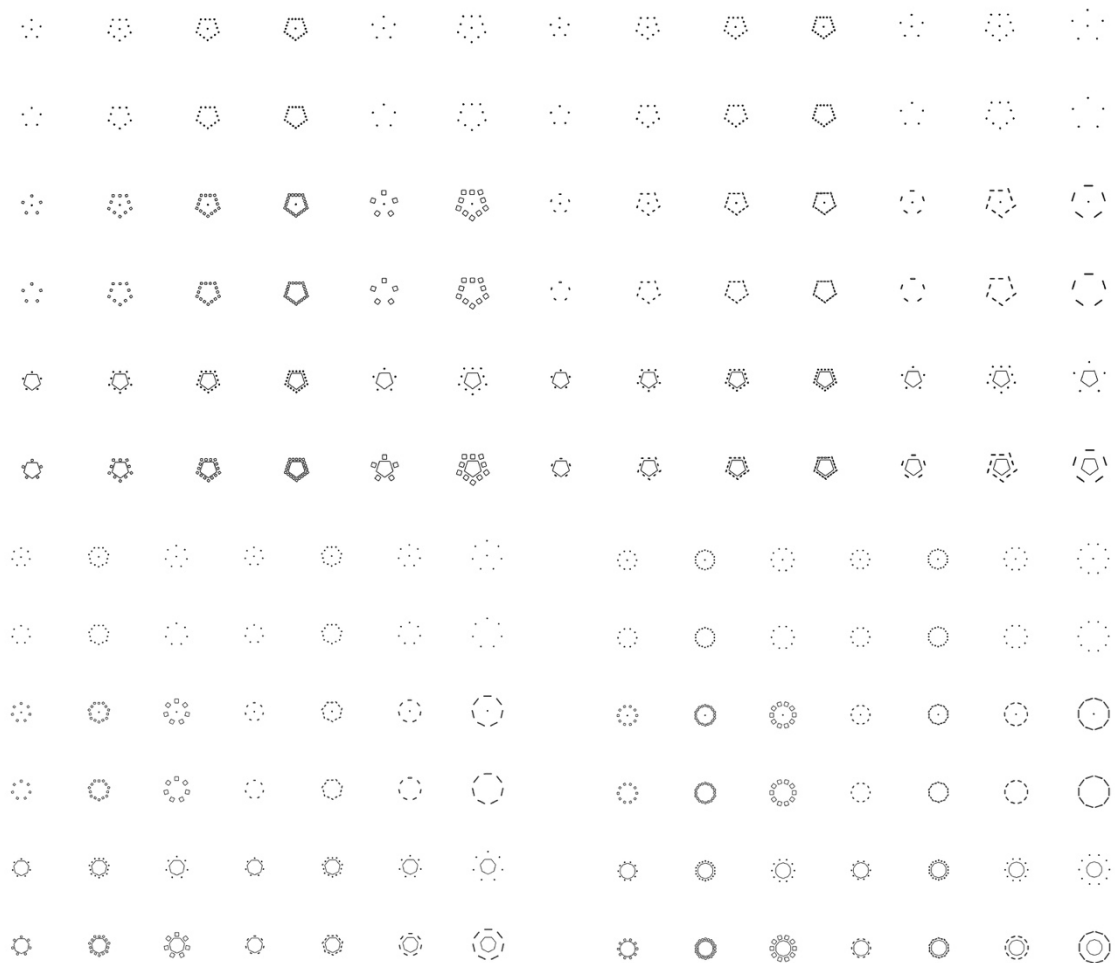


Figure 83: Pentagon, hexagon, and decagon uniform pattern outputs generated in Dynamo 2.0.3 (author)

#### 4.2.2.3.8.4 Non-uniform circular and polygon patterns

The transition between the uniform and the non-uniform circular and polygon patterns obtains the same transition characteristics that is explained previously in the transition between the uniform and the non-uniform rectangular patterns. In addition, it will only use the input square lights that are taken from the non-uniform lights variables and apply it on the different polygon shapes to create different patterns.

##### 4.2.2.3.8.4.1 Non-uniform circular and polygon script explanation

Figure 84 is showing the final script for the non-uniform circular and polygon patterns. Where creating a vector, a plane and a circle is as the same as the uniform script, then the different lists were removed that reversed the lights in order to insure all the orientations due to the fact that the square lights are only used in this approach.

S0 is still the same; highlighted in red in Figure 84; in order to develop the shapes from a circle to the different types of polygons. Similar to the uniform circular pattern, where an if statement is written **s0 = sides<=2?circ:Polygon.RegularPolygon(circ,sides);** the script basically states that if the sides are less or equal to two, then the algorithm will create a circle, otherwise create a polygon that takes two variables which are ( the circle radius and the number of sides for the shape). These two variables are taken from two sliders that are discussed further in *The light patterns controllers*.

After that, the second for-loop starts where **nt** that's highlighted in yellow in Figure 84 takes the number of lights from the end-user, to feed it to the created index under the name of **[h]**; which allows the algorithm to running through all the different sizes of lights, one value at a time, in order to divide the shape and place the lights.

The definition of **a** is highlighted in blue in Figure 84; showing the calculations for the divisions where **a=Math.Floor(s1[0].Length/(c\*1.5))>nt?nt:Math.Floor(s1[0].Length/(c\*1.5))**. As

the length of one of the lines that forms the shape is taken (**s1**) and divided by the current light input the algorithm is dealing with taken from the [h] index and multiplied by 1.5.

Then the rest of the statement is checking if in case the algorithm did the division and the result produced a bigger number than the actual value of the lights input, the result will just produce points that don't have any shapes because the algorithm ran out of lights input. So in order to eliminate this error. The rest of the definition **>nt?nt:Math.Floor(s1[0].Length/(c\*1.5))** states if it is more than the number of lights (**nt**) the algorithm will force the division to be the same number as the number of lights.

```

NON-UNIFORM CIRCULAR & POLYGON PATTERNS

def pat4(p0,rad,sides,small1:var[])
{
  z = Vector.ZAxis();
  pln0 = Plane.ByOriginNormal(p0,z);
  circ = Circle.ByPlaneRadius(pln0,rad);
  s0 = sides<=2?circ:Polygon.RegularPolygon(circ,sides);
  small_t = small1;

  return = [Imperative]
  {
    pts = [];
    recs = [];
    nt = List.Count(small_t);
    for (h in 0..nt-1)
    {
      c = small_t[h];
      s1 = s0.Offset(c).Explode();
      a = Math.Floor(s1[0].Length/(c*1.5))>nt?nt:Math.Floor(s1[0].Length/(c*1.5));
      n0 = 1..(a<=1?1:a);
      for (i in 0..(List.Count(n0)==1?0:(List.Count(n0)-1)))
      {
        resultp = [];
        resultr = [];
        for (j in 0..(List.Count(s1)<=2?0:(List.Count(s1)-1)))
        {
          pts1 = s1[j].PointAtParameter(n0[i]==1?[0.5]:
            (1/n0[i]..1..#n0[i]));
          vc1 = s1[j].NormalAtParameter(0.5);
          pln1 = Plane.ByOriginNormalXAxis(pts1,z,vc1);
          rc1 = Rectangle.ByWidthLength(pln1,small_t,small_t);
          resultp[j] = pts1;
          resultr[j] = rc1;
        }
        pts[h][i] = resultp;
        recs[h][i] = resultr;
      }
    }
    return = [s0,List.Flatten(pts,1),List.Flatten(recs,1)];
  }
};

```

Figure 84: Non-uniform circular and polygon patterns script in Dynamo 2.0.3 (author)



#### 4.2.2.3.8.4.2 The produced non-uniform circular and polygon patterns

The algorithm had produced around 200 non-uniform circular patterns that can be used as light patterns applied on a grid or as center lights. Figure 85 shows some of the non-uniform circular patterns while Figure 86 shows the some of the non-uniform hexagonal patterns results.

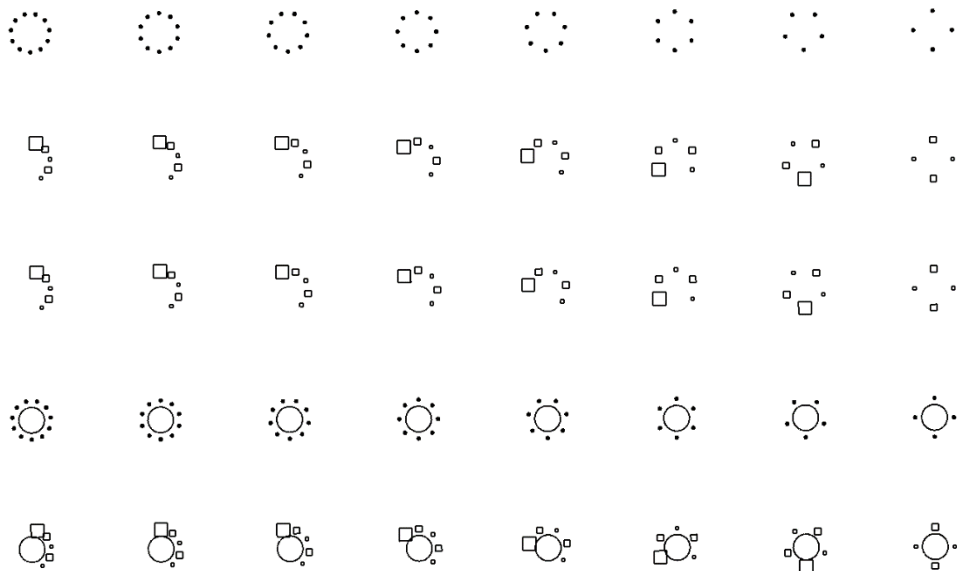


Figure 85: Non-uniform circular patterns outputs generated in Dynamo 2.0.3 (author)

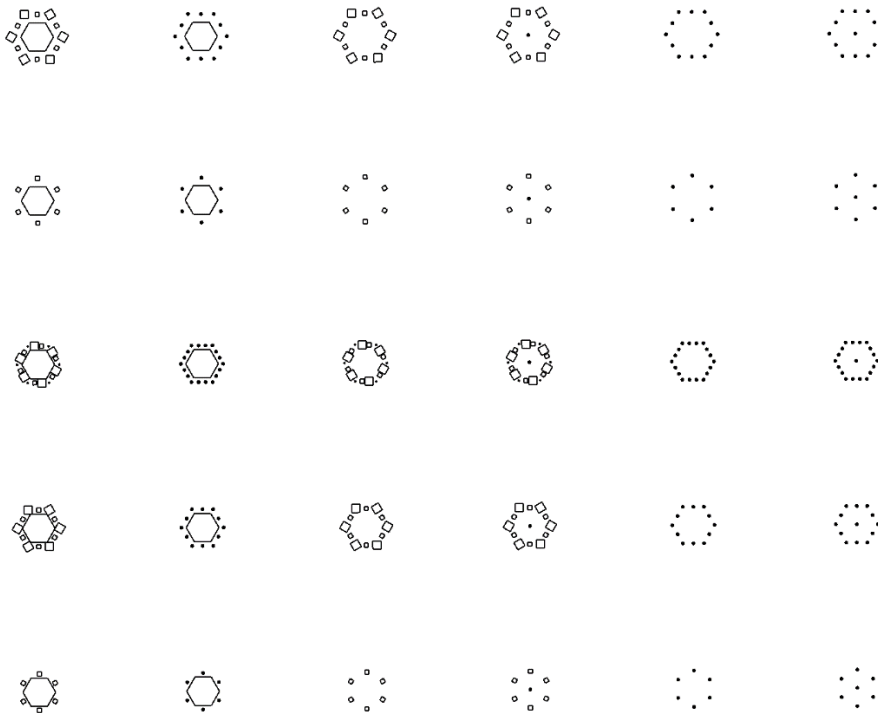


Figure 86: Non-uniform hexagon patterns outputs generated in Dynamo 2.0.3 (author)

The produced patterns develop and the spaces between the lights can be adjusted and increased based on the end user requirements. Figure 87 shows some of the outputs for triangle and quad non-uniform patterns.

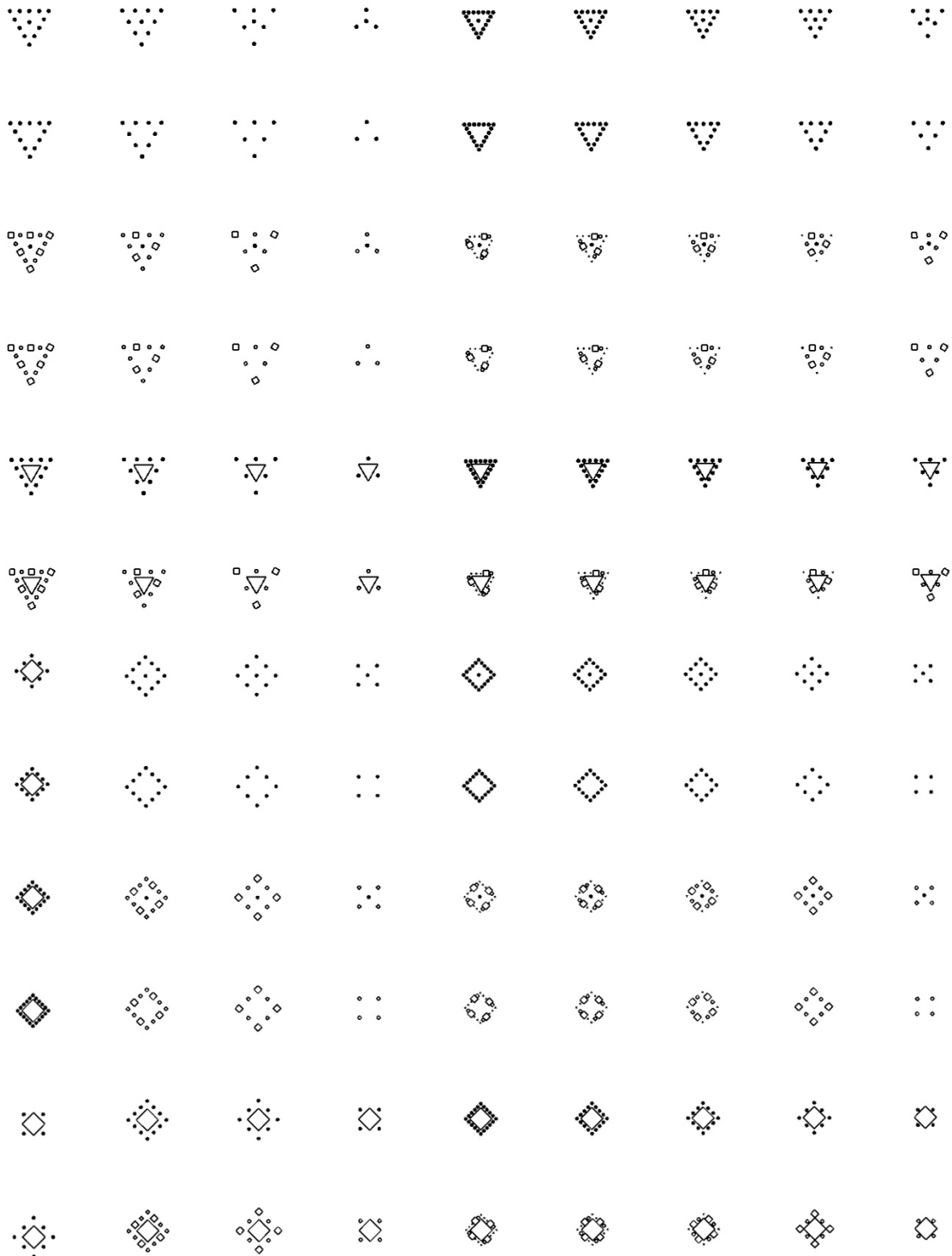


Figure 87: Non-uniform triangular and quad patterns outputs generated in Dynamo 2.0.3 (author)

#### 4.2.2.3.8.5 The light patterns controllers

In order to control the different variables for the four different produced patterns, different sliders were inserted under a group called Light Pattern Controls, seen in Figure 88.

The red outlined two sliders under the names of Pattern Scale 1 and Pattern Scale 2 allow the end user to change the imaginary rectangle outline, resulting in a change of the scale of the pattern for both uniform and non-uniform rectangular patterns, one slider changes the length and the other changes the width.

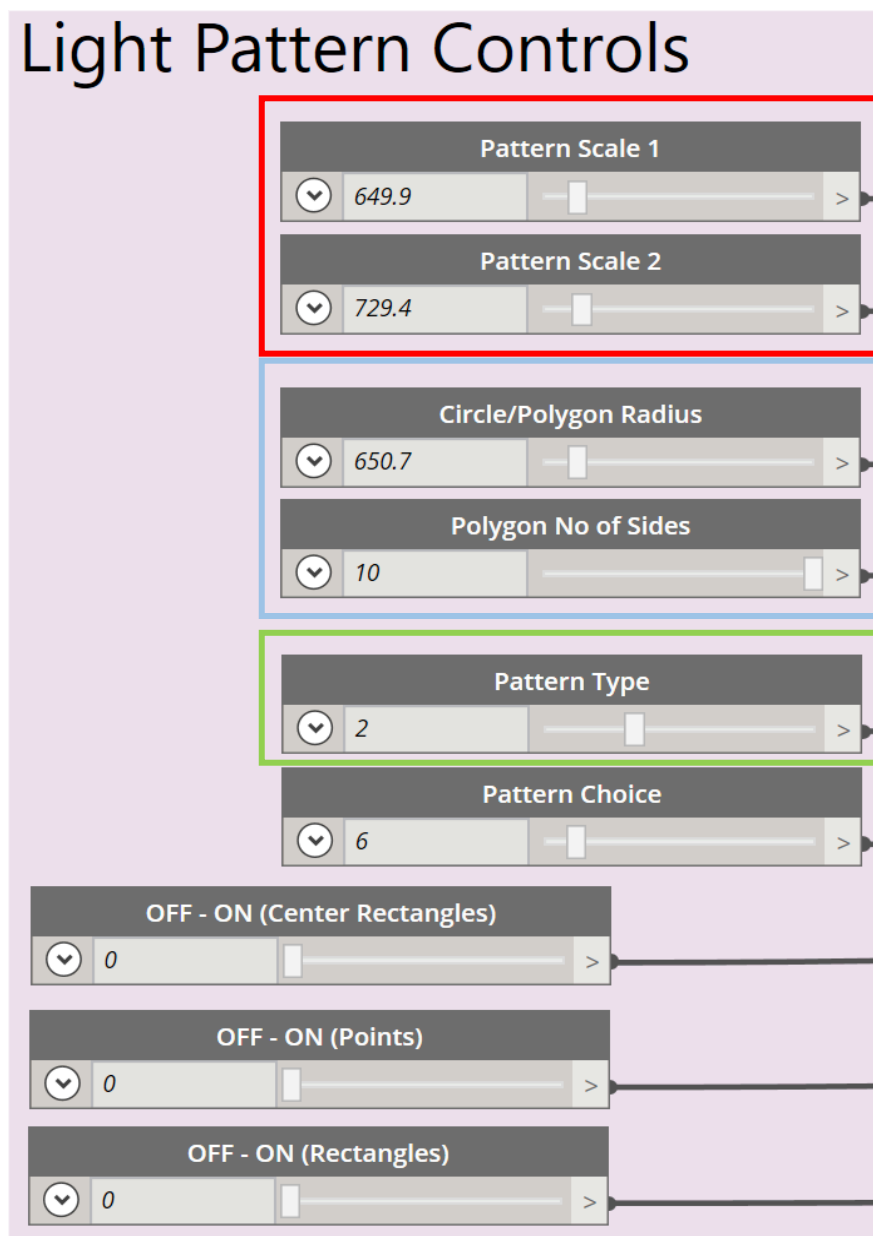


Figure 88: All the sliders for creating light patterns in Dynamo 2.0.3 (author)

Whereas the blue outlined sliders in Figure 88 control the circle and the polygons radius. And the Polygon Number of Sides slider determines the shape in the uniform and non-uniform circular and polygon patterns. If the number of the slider is on 1 or 2; all the produced patterns will be circular, otherwise the polygon shaped patterns will show once the slider is placed on three, creating triangular patterns, all the way to decagon patterns.

The Pattern Type slider, outlined in green in Figure 88, controls the four patterns. Table 6 shows the number of the slider and the lights patterns that will appear accordingly.

*Table 6: Pattern Type Slider Details (author)*

The input number in the slider	The pattern type
1	Uniform rectangular pattern
2	Non-uniform rectangular pattern
3	Uniform circular/polygon pattern
4	Non-uniform circular/polygon pattern

The Pattern Choice slider allows the end user to move the slider to view all the different produced designs under each type. While the three remaining sliders (Center Rectangles, Point and Rectangles) turns on and off different shapes within the light pattern itself.

For example, If the end user does not want to have any points that refers to spotlights within the produced patterns, he/she can switch off the point slider. The same command is created for the center rectangular which places a green colored center rectangle in the patterns, and the Rectangles slider that draws the outline of the inputted lights, it is usually colored in blue.

Figure 89 shows how turning on and off the three outlined controllers can impact the pattern. (1) is showing the full surface with the center rectangle, points and rectangular slider are switched on, while (2) shows how the same pattern will look with the center rectangle switched off. And (3) in Figure 89 is displaying where the center rectangle and points are switched off,

keeping only the light rectangle switched on. (4) is showing the result when all the three sliders are turned off.

The remaining green points that are showing in (4) Figure 89 are the points that were imported from the grid not the pattern. If the end user wishes to remove all the points on the ceiling, he/she needs to switch off the grid slider.



Figure 89: Explanation of the three on/off sliders for the light pattern controllers in Dynamo 2.0.3 (author)

### 4.2.3 Subsystem 02: Counting the generated light quantities

Some design methods were connected to a Node that counts the lists to produce numerical outputs indicating the quantity of lights used in the method. While other methods required adding a script to group the results, followed by connecting the groups.

The boundary lights script was connected to a **List.Count** seen in Figure 90, that counted the lights outputs and displays them in a Watch Node. In addition, the grid outputs were also connected using the same method.

The grid output points do not display the patterns count. As it presents the number of points that were made from x and y axis intersection. Those points are later fed to creating uniform/non-uniform light patterns.

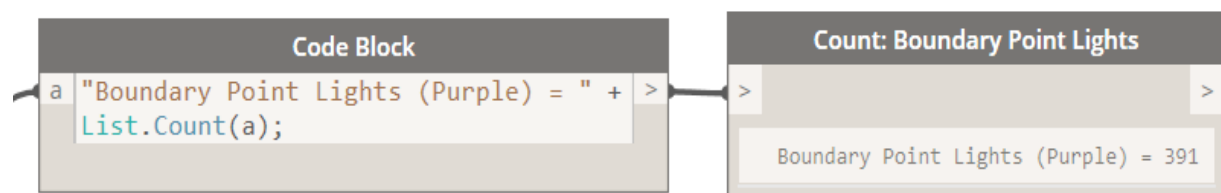


Figure 90: Boundary lights list count connected to a watch node to display the numbers in Dynamo 2.0.3 (author)

The outlined Code Block in Figure 91 counts the center point for every light pattern produced. Where every light pattern requires a center point to be created.

This count is beneficial in knowing the number of patterns placed on a surface. And can also be used to be the number of fixtures on the ceiling. The colors terms between brackets shown in Figure 91 allows the end user to identify each light category visually.

A minor issue occurred in counting the different light patterns. Since all the light patterns were created using the same point, the algorithm count could not differentiate between each pattern. A scripted algorithm was created to group the same light types together.

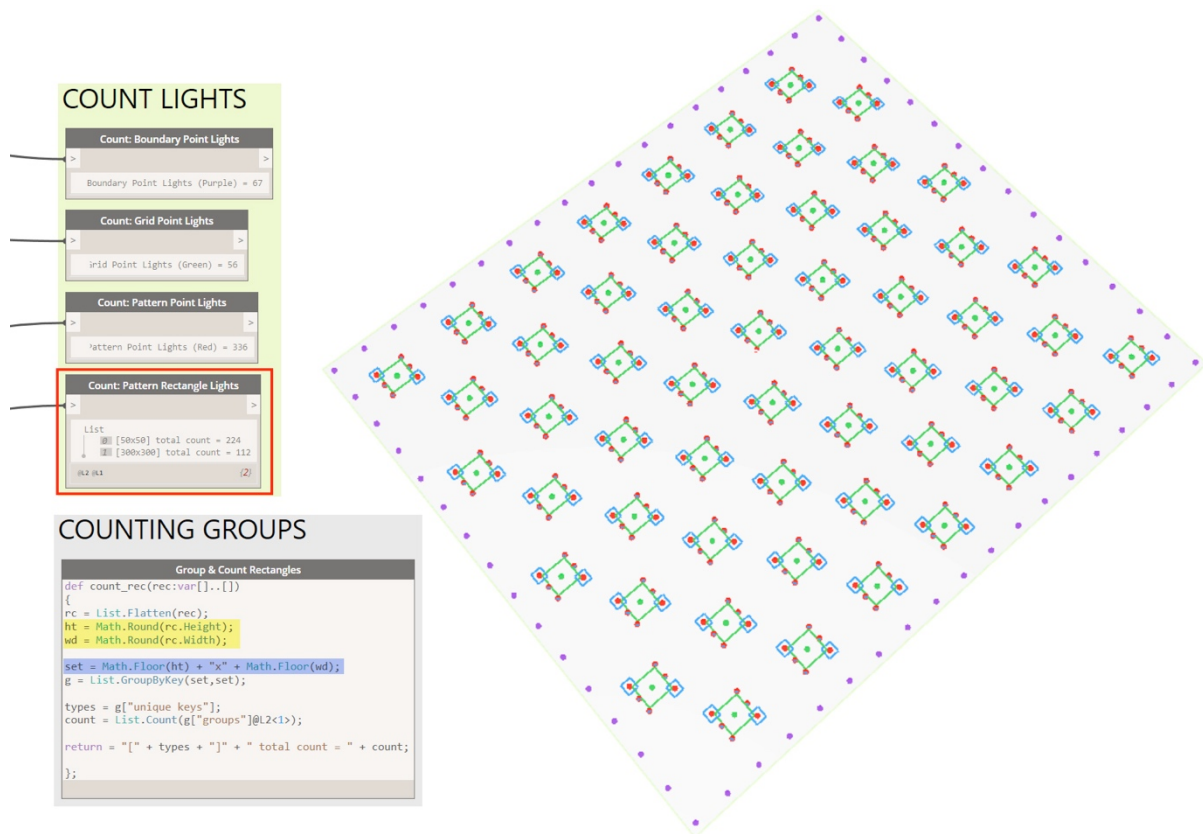


Figure 91: The light counts results and the scripted algorithm for the Pattern Rectangle Lights in Dynamo (author)

Figure 91 shows the script where all the light dimensions are fed to the counting script. The light types were obtained from the initial list that was used to run the algorithm in the beginning.

The yellow highlighted script inquires the length and the width of every light, and list them separately. Creating a length group and a width group. After that, the two separate groups carried to blue highlight script.

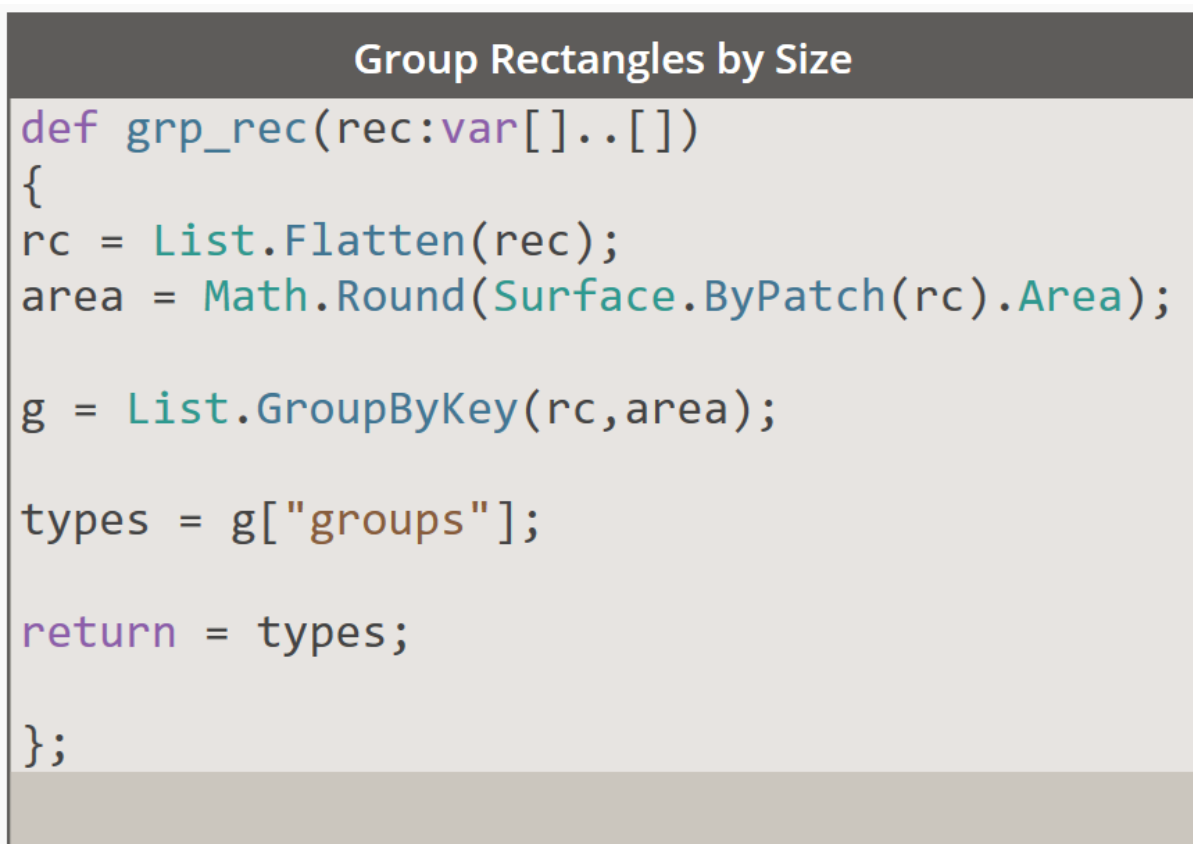
Where the blue highlighted script in Figure 91 creates the textural writing that is visible in the red outlined Node in Figure 91. Where the text indicates the [length list x width list].

After that, every similar type of lights that has the same dimensions is detected using **List.GroupByKey** command seen in Figure 91.

Then, every type of lights that has the same dimensions is grouped in its own list and counted to get the proper sum for every light type used in producing a pattern.

#### 4.2.4 Subsystem 03: Exporting the results back to Autodesk Revit

After the algorithm system was created and the various scripts were tested individually, the produced results in Dynamo required to be exported back into Revit in order for this system to be valuable for the end user. Where it will not only produce different ceiling designs in Dynamo, but will also be able to apply the results on Revit through connecting the produced lights to actual light families found in Revit in order to conduct any lighting calculations required. The results were transferred from Dynamo into Revit using the script shown in Figure 92.



```
def grp_rec(rec:var[]..[])
{
rc = List.Flatten(rec);
area = Math.Round(Surface.ByPatch(rc).Area);

g = List.GroupByKey(rc,area);

types = g["groups"];

return = types;

};
```

Figure 92: The Script to export the results in Dynamo back to Revit (Dynamo 2.0.3)

A definition for grouping the lights was done previously in *Subsystem 02: Counting the generated light quantities*, which produced textual results for the end user, indicating the produced quantities for every light. Another script was created as seen in Figure 92, where every two dimensional rectangle that indicated a light size was patched, converting the light shapes into surfaces. This step was conducted to calculate the area of each rectangle to ease the



grouping process. For example; a drawn rectangle that has 30x30cm dimensions will always produce an area of 900cm<sup>2</sup>, allowing the algorithm to group all the 900cm<sup>2</sup> results together, and so on.

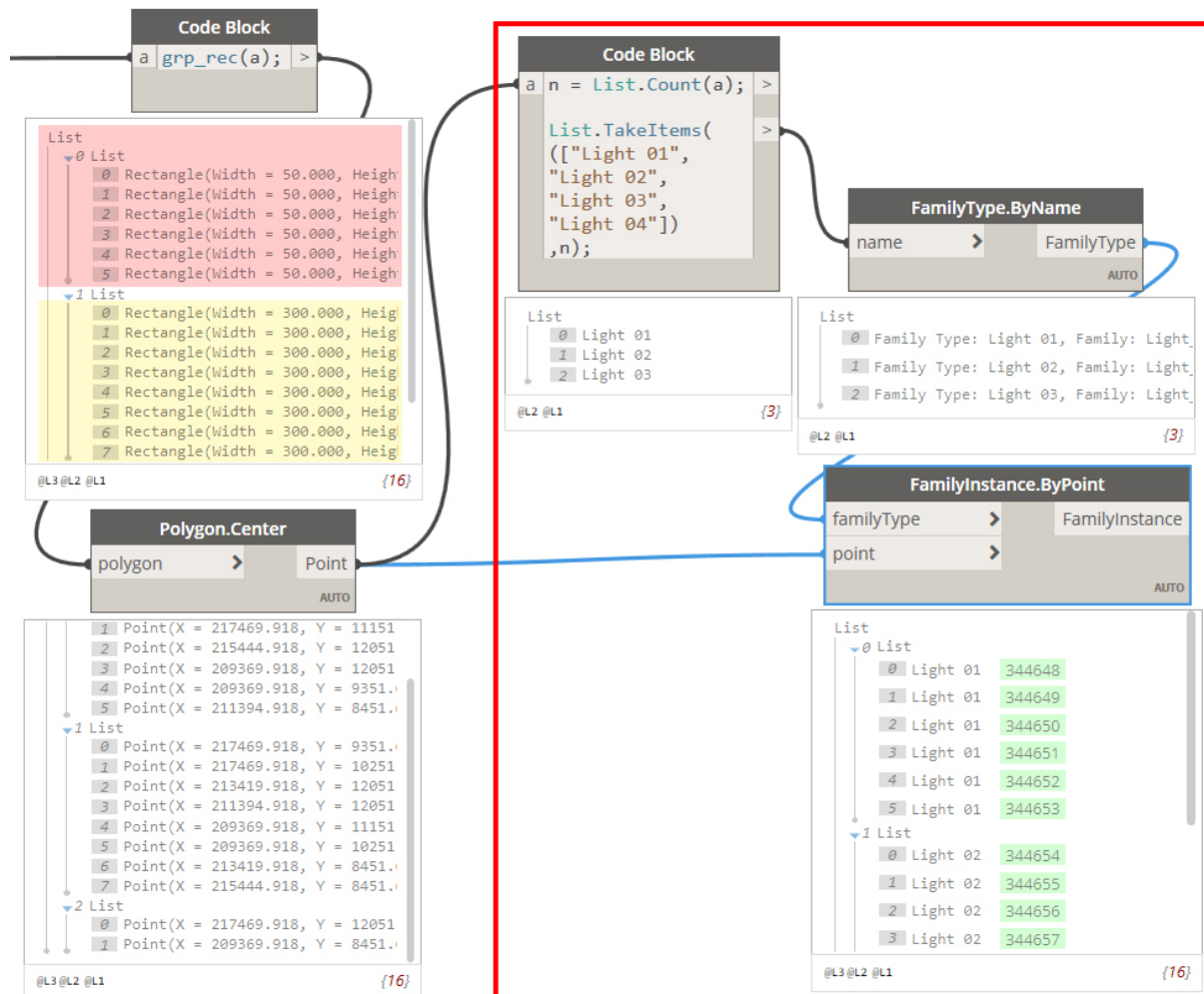


Figure 93: The grouping Nodes created in Dynamo 2.0.3 (author)

After every light category is grouped individually based on area, the lists that are highlighted in red and yellow in Figure 93 were produced. In addition, the center of each rectangle is acquired since Autodesk Revit only needs a point from Dynamo to connect a light family to.

The outlined Node in Figure 93 connects every group to a certain light in Revit. Where a light family is created in Revit that has several lights which correspond to the produced lights in Dynamo.

## 4.2.5 Arranging the generative system

### 4.2.5.1 Arranging the results horizontally and vertically

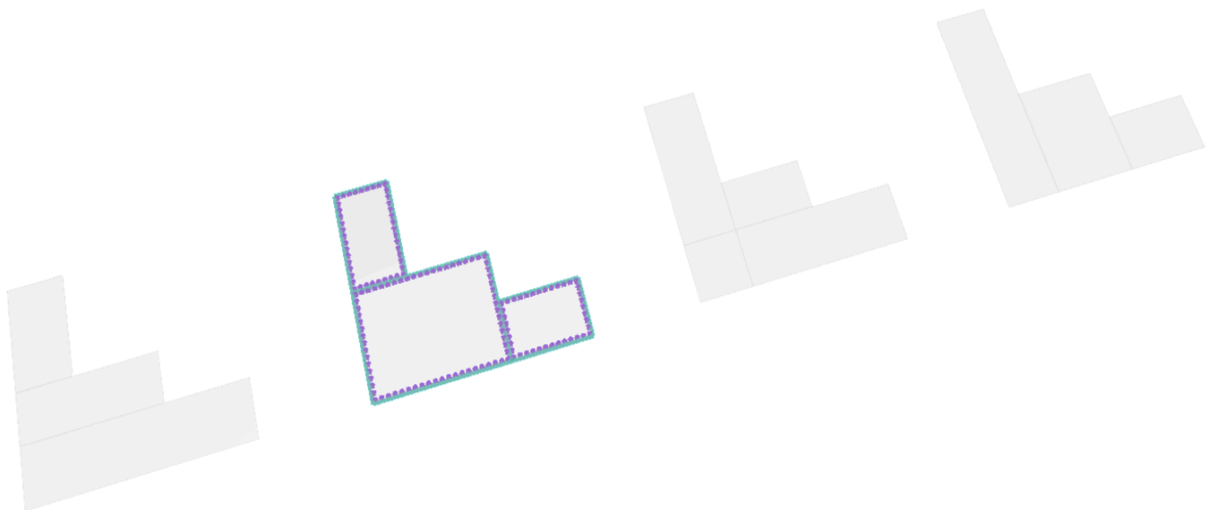
#### 4.2.5.1.1 Creating proper spacing between the generated ceilings

The distribution of the results is created through using a sequence of values given to the surfaces' outputs. Where the first value is zero, allowing the first output to be in the origin surface location.

The sequence is created using three elements; the original location of the surface, the quantity of values that indicates the surfaces outputs, and the spacing between each surface.

The length of the surface is measured and multiplied by 1.5 in the y axis direction, the multiplication is added to the scrip so the surfaces' results will not overlap. The entire sequences of results will move on the positive and the negative directions based on the end user selection from the slider that's seen in Figure 36 with the name of: Split Choice.

Figure 94 shows the clear distinguished origin ceiling placed next to the splitting results.



*Figure 94: Close up shot showing the outlined selected ceiling compared to the other ceiling options in Dynamo (author)*

#### 4.2.5.1.2 Arranging and placing the generated light patterns on the original ceiling

In addition to arranging split surfaces' results, a cross section arrangement for the light patterns was created, giving a visual representation in Dynamo's work space shown in Figure 95. Where both outputs are seen in the opposite axis, allowing the user to have an overall view of the results.

The end user is able to choose a pattern distribution through the Pattern Choice slider shown in Figure 95, as the light pattern changes coherently with the number in the slider. In addition to the ceiling split selection through using the Split Choice slider shown in Figure 95. In other terms, the end user is able to move the splitting options on y axis while moving the patterns on x axis, and the intersection point of both axis is the location of the original surface that is imported from Revit.

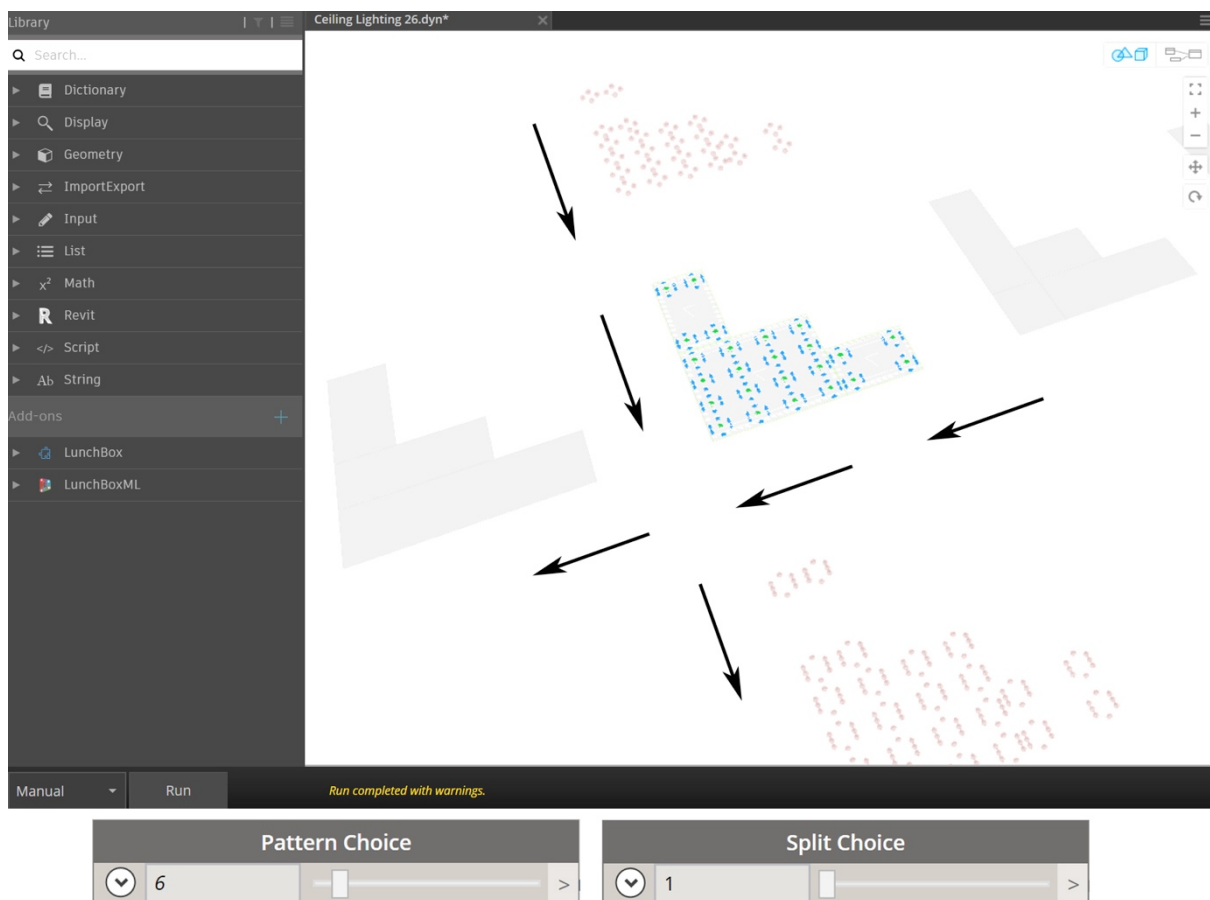


Figure 95: Dynamo workspace after the arrangement of the splits results and the patterns (author)

#### 4.2.5.2 Coloring the surfaces and the produced lights

Minor Nodes were added to enhance the overall presentation of the light inputs and produced results. Allowing the end user to distinguish the different light types produced clearly in Dynamo.

The first adjustment was created to the surfaces of the produced ceilings. As the selected ceiling will have different color opacity that the other ceilings as seen in Figure 96. Where the selected surface has an opacity of 50% with light green borders as seen, whereas the other ceilings have an opacity of 100% with gray borders.

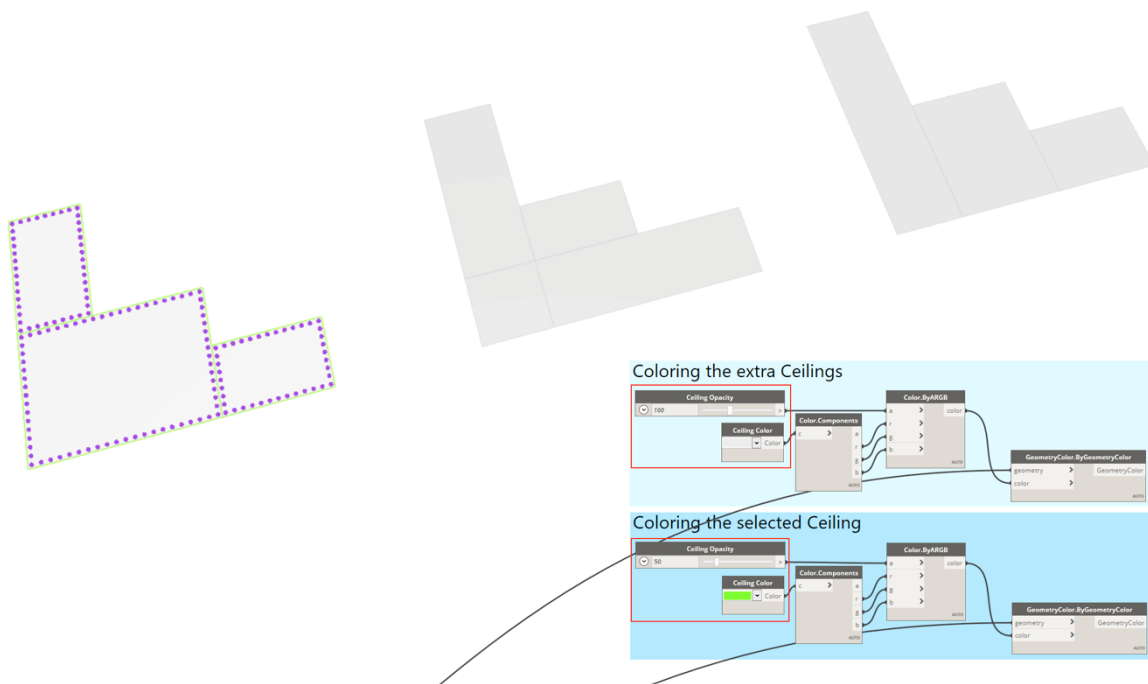


Figure 96: Changing the opacity and borders for the surfaces in Dynamo 2.0.3 (author)

Since Dynamo's graphics may not always show the best resolutions as the end user changes the angles of the view, so reducing the selected ceiling opacity ensures that the lighting results will always show.

In addition to changing the ceiling opacities, the patterns produced lights had to be color separated to ease the selection and elimination process for the end user. Figure 97 shows how

the lights are colored next to the Nodes that created the required coloring. The center rectangle lights were colored in green, while the points in red and the produced light dimensions in blue. If the end user decides to choose between any light category turning on and off certain sliders, the results will show as Figure 98.

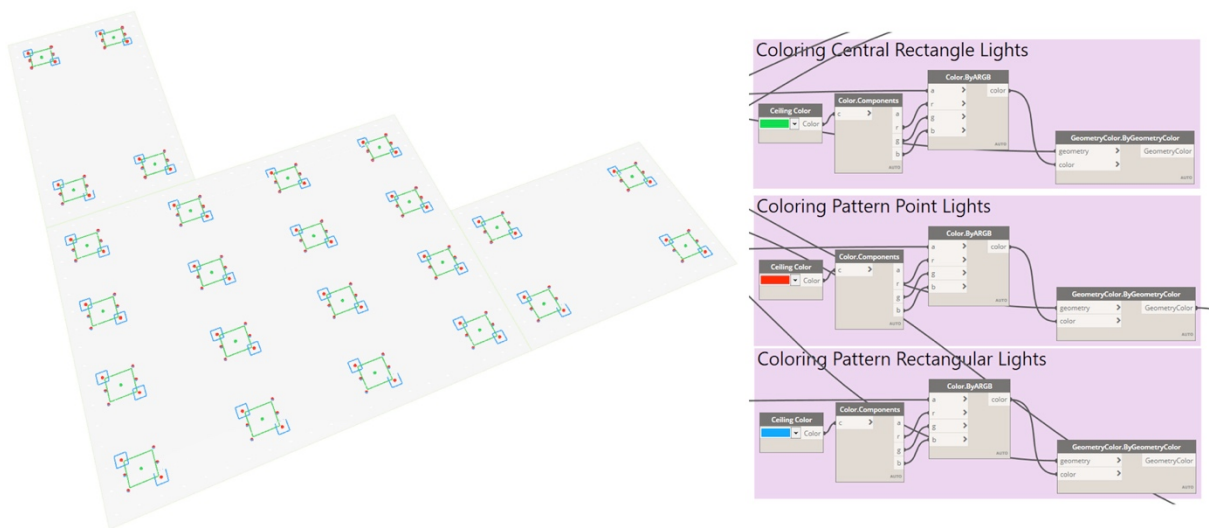


Figure 97: Lights patterns color display on a surface and the connected Nodes in Dynamo 2.0.3 (author)



Figure 98: Lights patterns color display on a surface in Dynamo 2.0.3 (author)

Furthermore, the boundary lights were colored in purple to separate it from the patterns lights, Figure 99 shows the boundary lights applied on a simple ceiling structure. And the grid points were also colored in green as seen in Figure 100, Where the diamond grid is showing, these points are the result of the intersection between x and y axis division that occurred in the grid script.

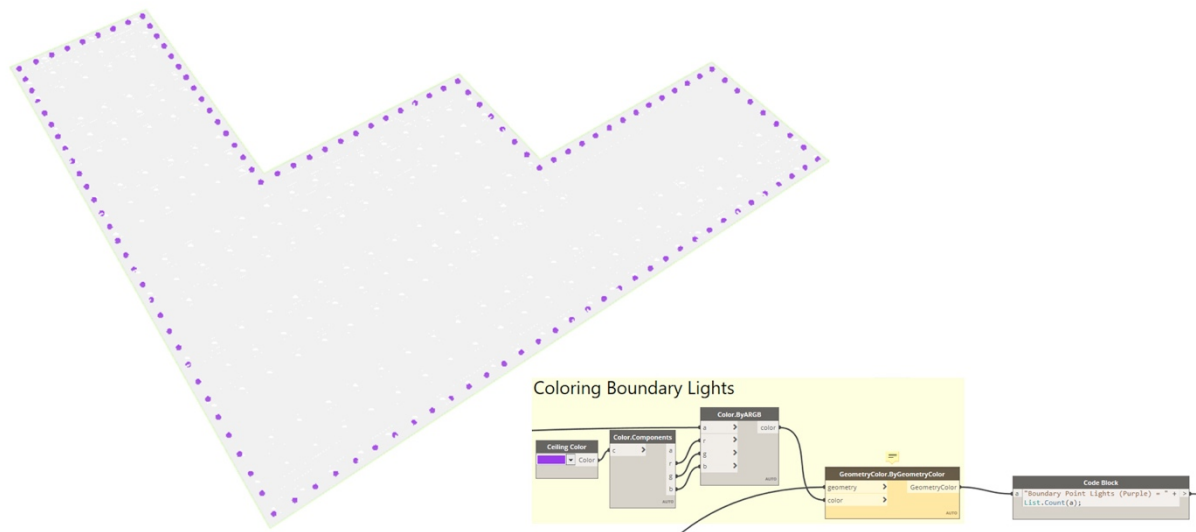


Figure 99: Boundary lights color display on a surface and connected Nodes in Dynamo 2.0.3 (author)

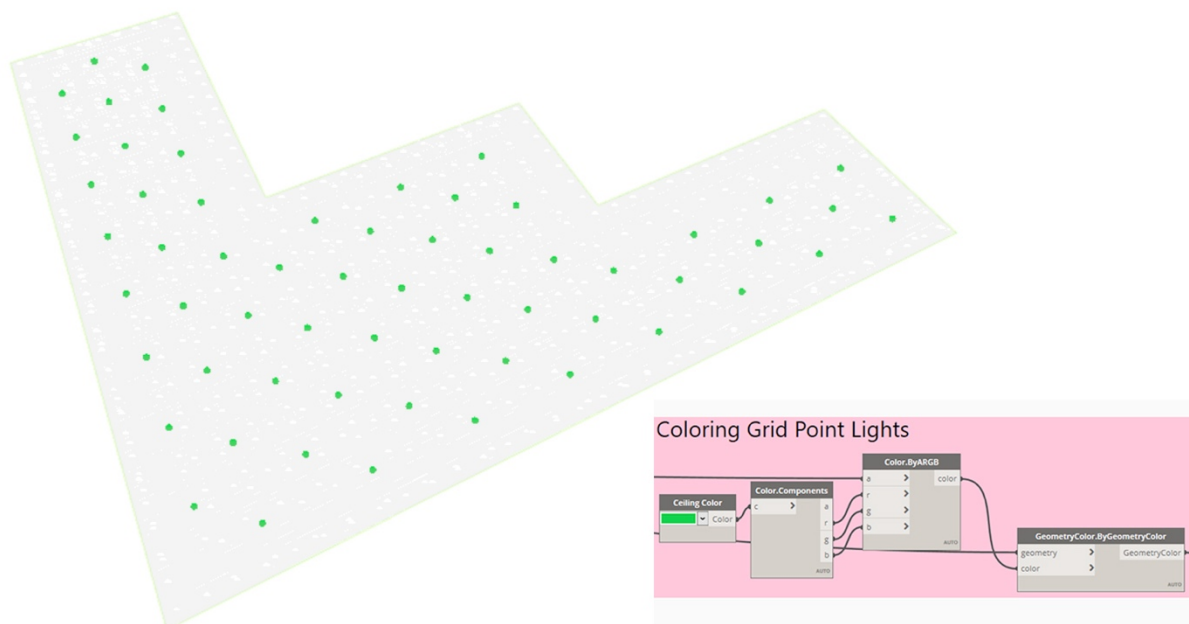


Figure 100: Grid lights color display on a surface and connected Nodes in Dynamo 2.0.3 (author)

## Chapter 5

Results and discussion: Testing the generative system

## 5 Chapter 5: Results and discussion - Testing the generative system

### 5.1 Introduction

The algorithm is an interactive system that generates results immediately based on reading inserted variables, some variables are inserted in a form of numbers, such as Figure 101 that shows the lighting sizes in millimeter. While other variables are in the form of sliders; allowing the designer to manipulate the values through moving the slider tap, to produce instant visual changes in the designs.

Every variable changed by the designer allows a different output to be generated. The limitations of the outputs are related directly to the end user's aim in that certain layout. Although the algorithm generates various options, it is up to the designer to change, alter and translate what every drawn line can be.

Thus, human's evaluation is essential in the algorithm's outputs, to assure producing the best required results. Although the scripted algorithm is an outline for a system that generates reflected ceilings designs that can be implemented directly on ceilings, it works better when the end user tries to manipulate the given sliders in Dynamo script. Since that exhibits all the possibilities that this algorithm is able to generate.

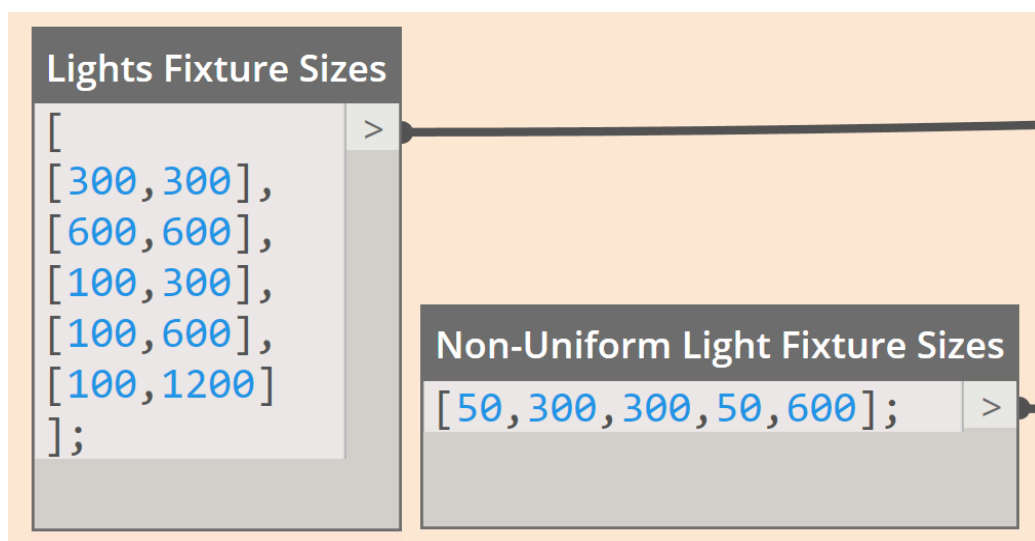


Figure 101: Inserted lighting sizes in the algorithm in Dynamo Nodes (author)



This chapter tests the scripted generative design system using different approaches that are identified in three phases: Phase 01, Phase 02, and Phase 03. Ensuring that the generative algorithm presents feasible outputs in terms of the diversity in the mechanisms for selection, limitation and mutation of the outputs and sustainability implementation.

Phase 01 highlights the diversity of the produced reflected ceiling layouts, the system's ability to produce relatively high quantities of reflected ceilings designs, and it sheds light on the system's flexibility in manipulating and altering results. It is conducted through testing the system using four case studies. The case studies were diverted into two sections: (1) Conceptual case-studies (2) Field case-studies.

Conceptual case studies target ceiling forms and asymmetrical layouts. In order to test the generative system ability and limitations in producing reflected ceiling layouts with untraditional spaces. The selected case-studies in this section are in different countries around the world. The tested case-studies are two offices, the first office is called The Green Room, located in Utrecht, Netherland, while the other office is Commerzbank headquarters, located in Frankfurt, Germany. The two case studies revealed interesting limitations that standard, more traditional ceilings might have not been able to reveal. Which provide an insight to the enhancement possibilities in the algorithm, for it to work more efficiently and to be used on a wider spectrum.

Field case-studies tested the algorithm on existing offices located in Dubai, United Arab Emirates. Where concerned authorities were contacted, permissions were obtained, and site visits were carried out. Field case-studies aimed to test the generative system efficiency in depth. Since designers might take on projects that had been previously designed and alter in it or might sign a project in its construction phase. The field case-studies showed both options, where two offices were selected in Dubai Design District area. Both offices were not occupied

by any company. The first office was in a base-build status, while the other office was recently cleared by the previous tenants; thus, interior elements remained. Results displayed a different aspect of the generative system and produced results that the conceptual case-studies might have not revealed.

Phase 02 highlights how the scripted algorithm is a comprehensive generative system that not only produces and alter unlimited design options, but also takes the results into practical applications. As the light variables in Dynamo -seen in Figure 101- were connected to actual lighting families in Autodesk Revit. To ensure that the light families are valid, lighting calculations were conducted on two different reflected ceiling layouts.

The two reflected ceiling layouts in Phase 02 were chosen from the generated results in the second field case-study in Phase 01. The selection of the two layouts aimed to highlight the diversity that the system can create. As the first selected ceiling used three different light families, while the second reflected ceiling used one lighting family. Furthermore, this phase had also tested another feature the algorithmic system has which is counting lights, as it automatically produces the quantity of lights any selected ceiling has. And in both cases, the system showed accurate counts. To perform artificial lighting calculations, a plug-in called Elum-tool was downloaded in Autodesk Revit, which produced accurate results that can predict the luminance levels in any tested area.

Phase 01 and Phase 02 in testing the scripted algorithm shows how the generative system creates a full cycle that starts from generating various and diverse reflected ceiling designs, altering them automatically and giving the end user the ability to change the design based on their desires, all the way to producing numerical data for lighting counts, and artificial lighting calculations, all within the same software.

This full cycle does not require the end user to use multiple software nor to perform different exporting approaches to transfer the designs from Dynamo to Revit, as the scripted algorithm does that automatically. Nor does it require to export the design to run lighting calculations. Furthermore, all the tools in Revit can be applied on the final design, which includes exporting the design into any suitable format the designer desire, to use it in AutoCAD or any other software.

Phase 03 in testing the generative system approaches it from a sustainable aspect. As it proves sustainability using two approaches; the first approach analyses the scripted algorithm using literature review that validates the generative system's sustainability with-in itself. Showcasing various parameters, the algorithm utilizes to maximize the positive affect and minimize negative affect.

The second approach fosters sustainability through two key parameters: (a) Increasing benefits, and (b) reducing costs. Numerical data were obtained using cross-sectional study conducted on 61 interior designers and architects. Increasing benefits parameter was achieved through evaluating and comparing the beneficial outcomes that had been proven in the generative system individually in Phase 01 and Phase 02 against the participants' responses.

Reducing cost was attained through a simple time-cost analysis, which utilized the data obtained from the survey, analyzed the saved time and financially priced it. Presenting sustainability through creating more benefits at low costs, as results showed an astonishing reduction of cost which reached 99%.

Figure 102 outlines the three discussed phases and the steps are performed under each phase. Showcasing the framework that was conducted to test the generative system.

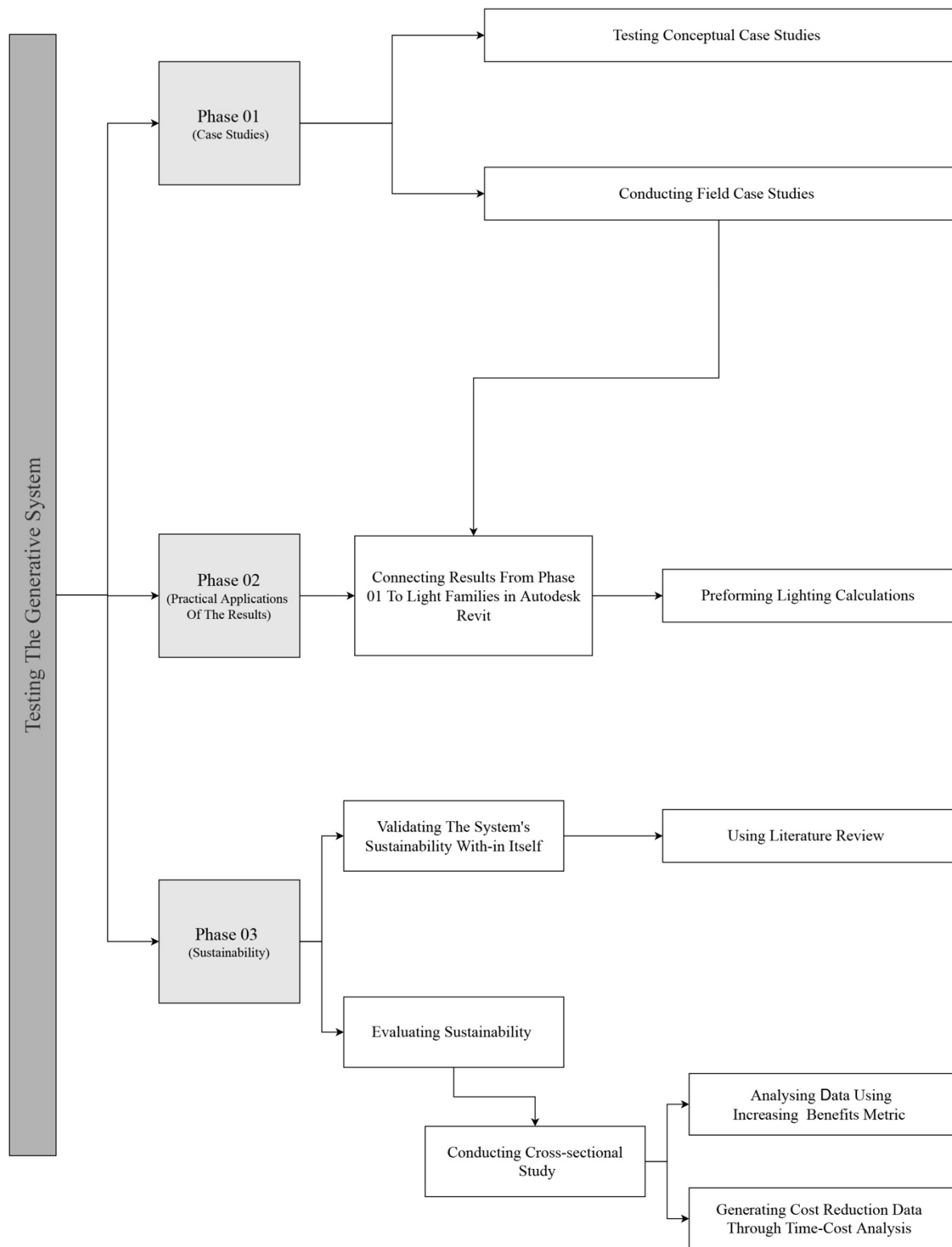


Figure 102: The framework outline for Results and discussion Chapter - Testing the generative system (author)

## 5.2 Phase 01: Case studies

### 5.2.1 Conceptual case studies

#### 5.2.1.1 Office 01: The green room - Utrecht, Netherlands

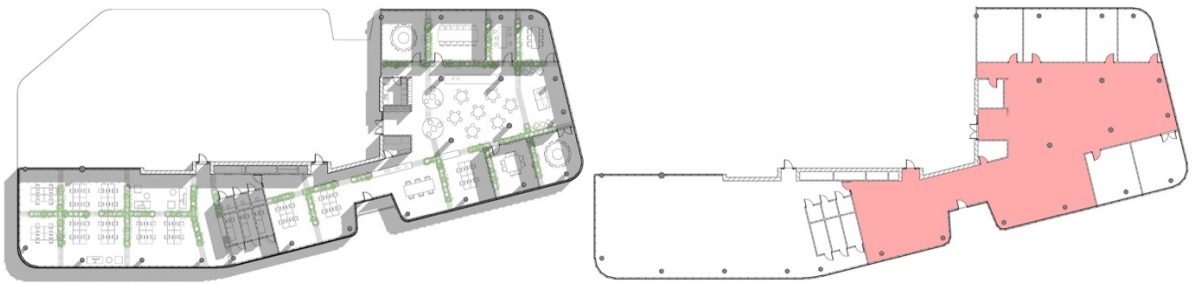
The first tested layout is shown in Figure 103; it's an office located in the 14<sup>th</sup> floor of a tower in Utrecht, Netherlands. The office space is designed by the Dutch studio: Space Encounters, featuring leafy partitions designed as an alternative to the standard claustrophobic boxed-in workrooms (Levy 2019).



Figure 103: The green room layout (Levy 2019)

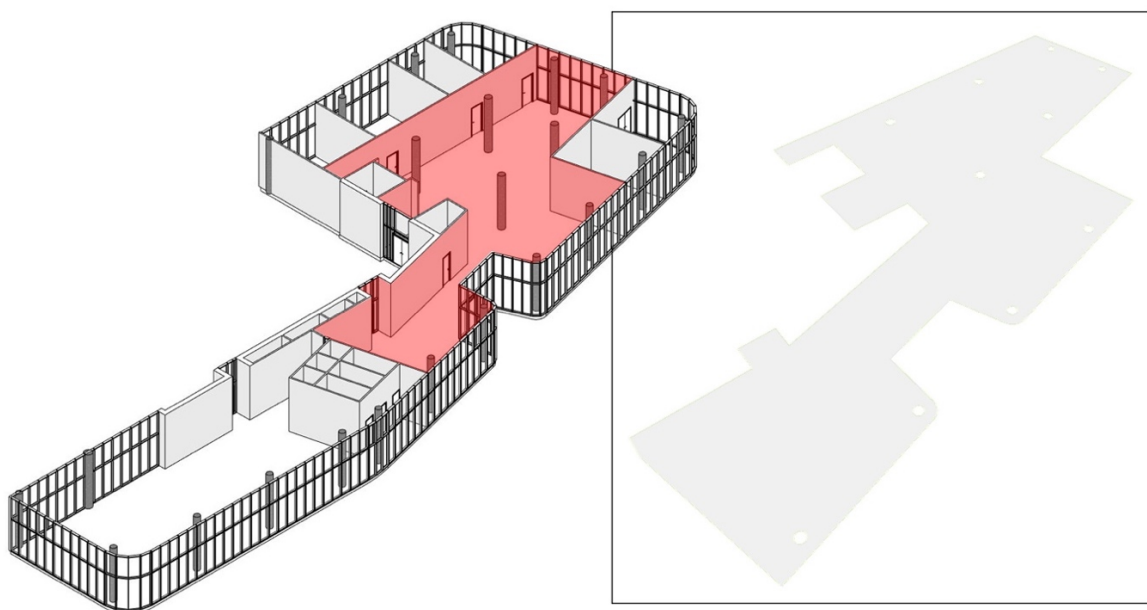
This layout was selected due to its shape. The curved corners in the layout and the asymmetrical spaces should result in a new assessment approach for the algorithm. The columns in the layout provide a new challenge for the algorithm, while the various edges with different angles shows great potential in using the splitting option in the algorithm.

The floor plan was modeled in Revit as seen in Figure 104. The highlighted area in red was selected and transferred to Dynamo for testing the algorithm. While scripting the algorithm; every method was tested on a relatively simple design, thus testing it on the selected area should provide results that identify the strong aspects of the algorithm and the fields that it requires improvement in, providing a better scope the system's advantages and limitations.



*Figure 104: The green room layout after modeling in Autodesk Revit (author)*

The algorithm is constructed in an approach that allows the end user to only select one ceiling at a time. In order for a designer to be able to create a reflected ceiling design for a large area that's divided by partitions or walls, the ceilings need to be joined in Revit and converted into one surface only to be inserted in Dynamo. However, in this case study the red area was tested only.



*Figure 105: The modeled layout in Autodesk Revit and the selected ceiling to export to Dynamo (author)*

Figure 105 shows the selected ceiling that was transferred from Revit to Dynamo for testing the algorithm. As seen, the ceiling has different holes that were created from the columns, in addition, the ceiling has curved angles in certain corners and perpendicular angles in other corners. Furthermore, the ceiling is asymmetrical, having parts that fail to correspond to one another in shape. All these elements will investigate the algorithm ability to generate reflective ceiling layouts in asymmetrical ceilings.

### 5.2.1.1.1 Office 01 limitations and shortfalls

#### 5.2.1.1.1.1 Unable to identify ceilings with openings

The first limitation the algorithm showed is its inability to identify openings in the inserted ceiling. The openings are caused by the structural columns that are visible in Figure 105. Results showed that the algorithm was not able to identify the surface, it resulted in an error and wasn't able to generate any lighting option.

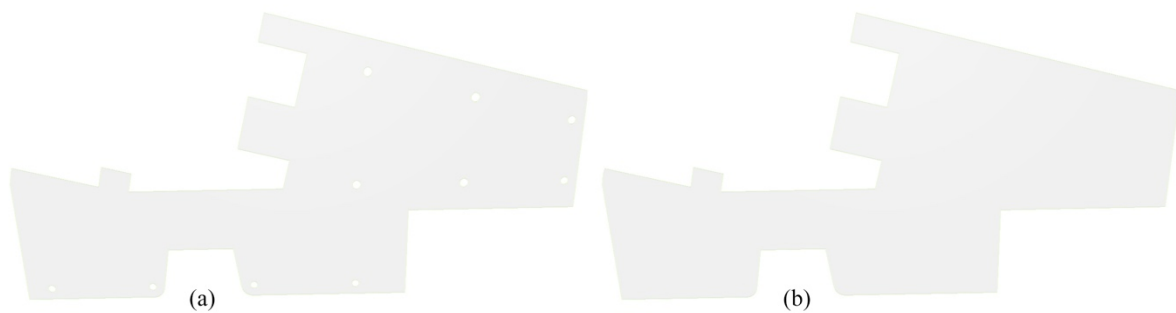
Figure 106 shows the initial results; as the outlined (null) list indicates an error list, which means the algorithm did not find the ceiling to apply and lighting methods on.



Figure 106: The error list that appeared from the ceiling shape in Dynamo 2.0.3 Nodes (author)

The reason the algorithm was not able to identify the inserted ceiling was because its scripted structure. It is built on extracting the boundaries of the surface to identify any inserted ceiling. In this case, the boundaries of the openings were also extracted. This situation created an error in the algorithm, since the ceiling openings had produced areas that are not connected to the outer boundaries to the surface.

For the algorithm to identify the surface and run the calculations correctly, the openings were patched in Revit; allowing the algorithm to apply a polygon line only on the boundaries of the surface. Figure 107 shows ceiling (a) that resulted in an error result, and ceiling (b) that produced proper results after the openings in were patched in Revit.



*Figure 107: The inserted ceiling before and after patching from Dynamo 2.0.3 workspace (author)*

#### **5.2.1.1.1.2 Limiting the generated results due to laptop capacity**

Figure 108 shows the produced number of the generated ceilings by the algorithm. As it was able to produce 512 reflected ceilings designs options. The list in Figure 108 shows 511 results due to the fact that Dynamo's index starts from (0) not (1).

Since the algorithm generated 512 options for this ceiling shape, the calculations took about 15 minutes to predict all the possibilities produced from permutations and combinations in splitting the ceiling. In addition, Dynamo had crashed twice while calculating the produced ceilings. It seemed that the used laptop with 8 GB memory was not able to produce results as fast as required.



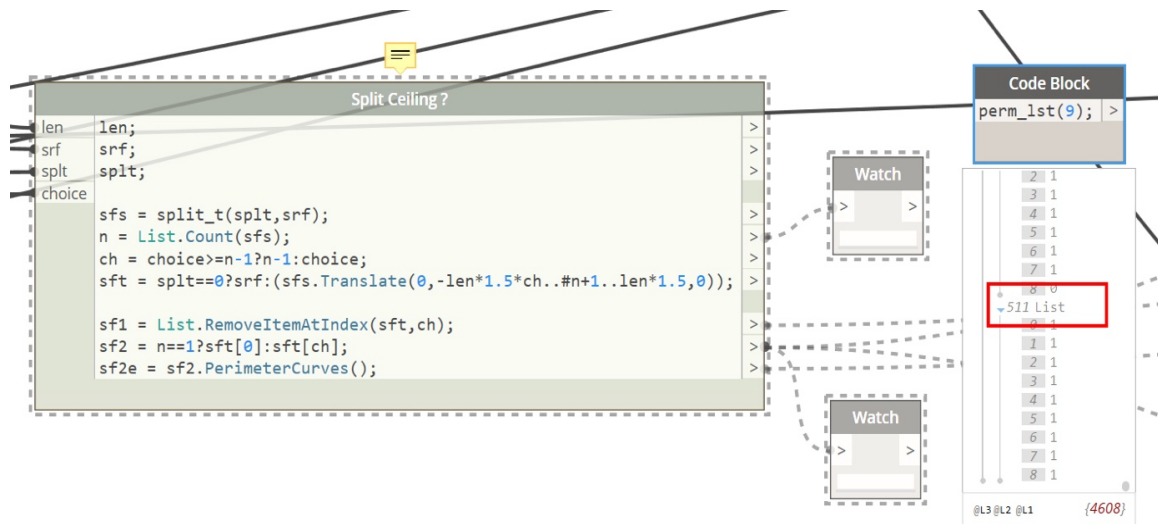


Figure 108: The algorithmic generated results on splitting the ceiling exhibited in Dynamo 2.0.3 (author)

A simple limiting script was added to limit the generated results to only 40 options in one design approach, which divides the ceiling. In doing so, the mathematical procedure for permutations and combinations in order to split the ceiling was reduced significantly. The algorithm took only 1 minute to produce more than 100 ceiling options. This solution is not necessary if the used laptop's capacity is higher.

The end user will not be able to perform this solution unless if he/she has some knowledge about visual programming. The best solution in this case is to wait 15 to 20 minutes for the algorithm to calculate all the possible scenarios. Figure 109 shows the original script and the limiting script, as the green outline is the original script and the red outline shows the new limitation as it selects 40 produced layouts only.

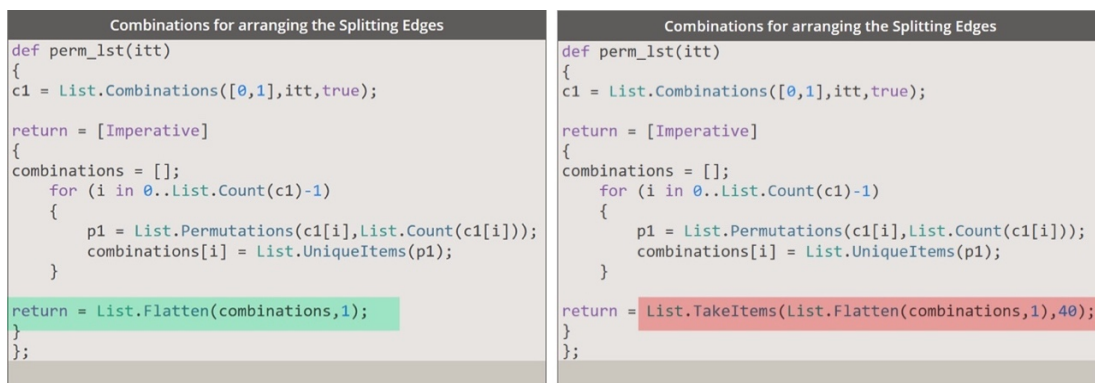


Figure 109: Limiting results script created in Dynamo 2.0.3 (author)

### 5.2.1.1.1.3 Curved corners shortfall

The corners and the narrow corridors were considered in creating the algorithm. However, a condition for curved corners was not added. Which resulted in a division error when applying one design method which is boundary lighting. Figure 110 shows two generated options, where the boundary lighting is applied in both, the boundary lighting is shown in purple dots.

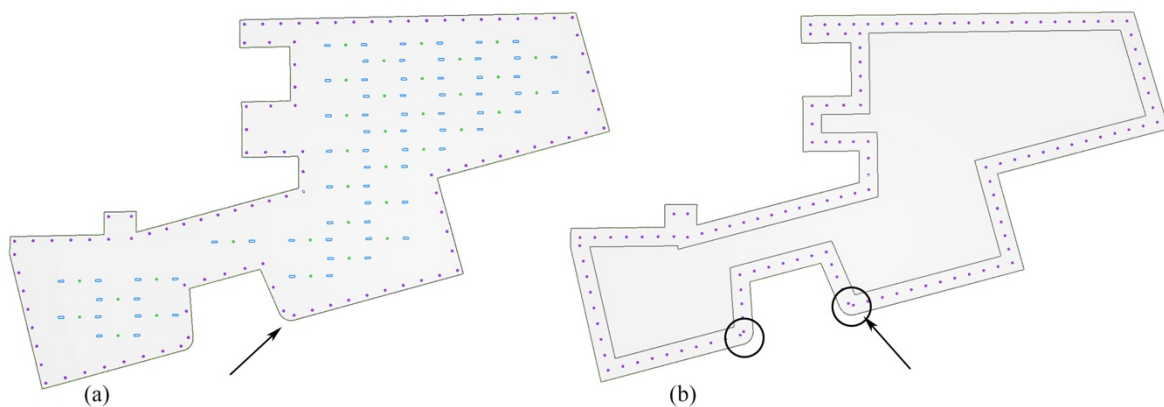


Figure 110: Curved corners shortfalls for Office 01 generated from the algorithmic system in Dynamo 2.0.3 (author)

(a) in Figure 110 displays boundary lighting and a grid option. With the spacing between the boundary lighting being 1500mm. the error is not noticeable visually, however it is clear that the space between the two light coordinates in the corner is different that the spacing between the rest of the light coordinates, creating inaccurate results.

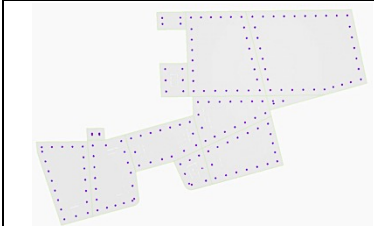
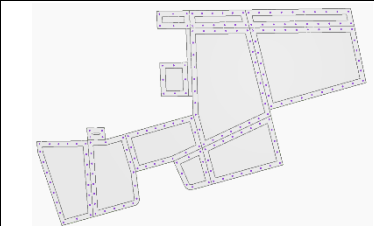
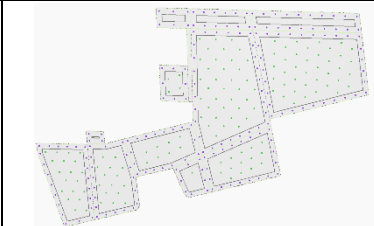
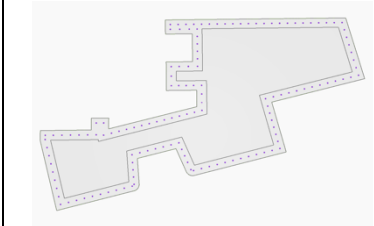
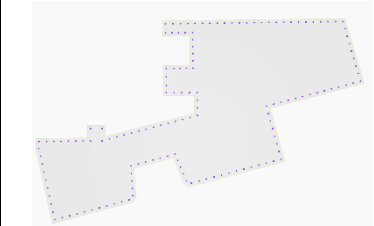
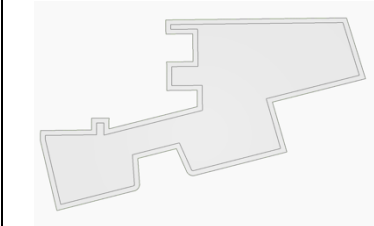
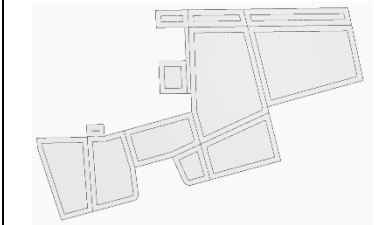
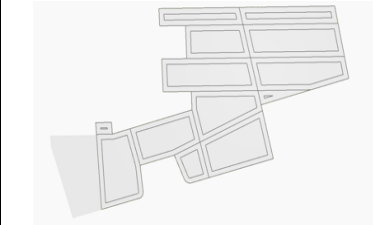
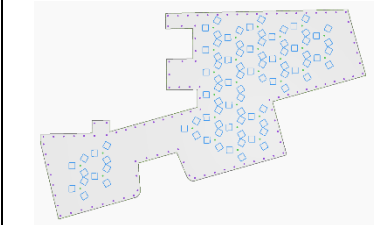
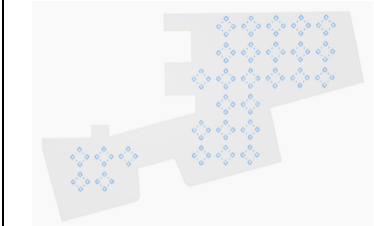
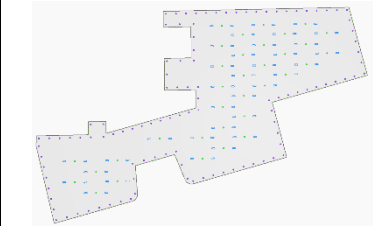
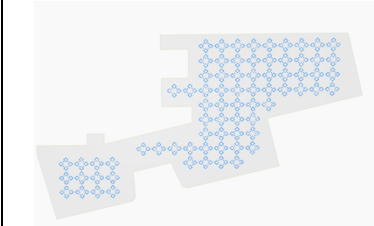
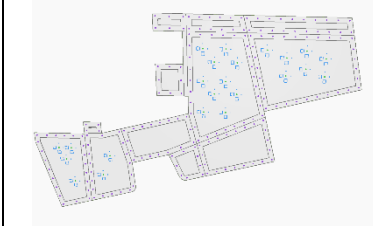
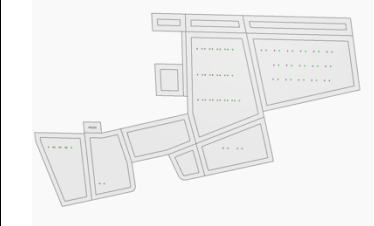
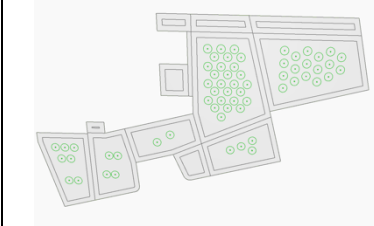
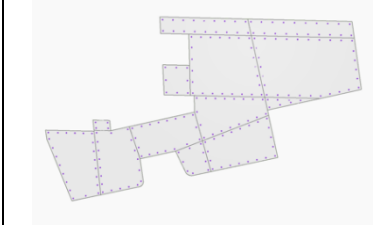
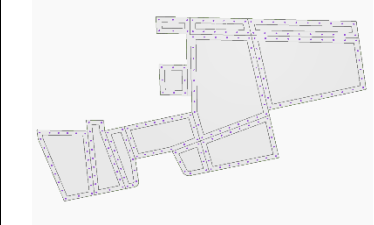
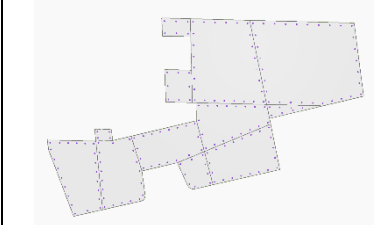
While the division shortfall in (b) in Figure 110 is more visually prominent, as the ceiling showcased the drop down ceiling option merged with the boundary lighting with 1000mm spacing. The light coordinates are divided based on the curved arch that forms the corner of the ceiling, and not based on the continuous line of the polygon that formed the ceiling.

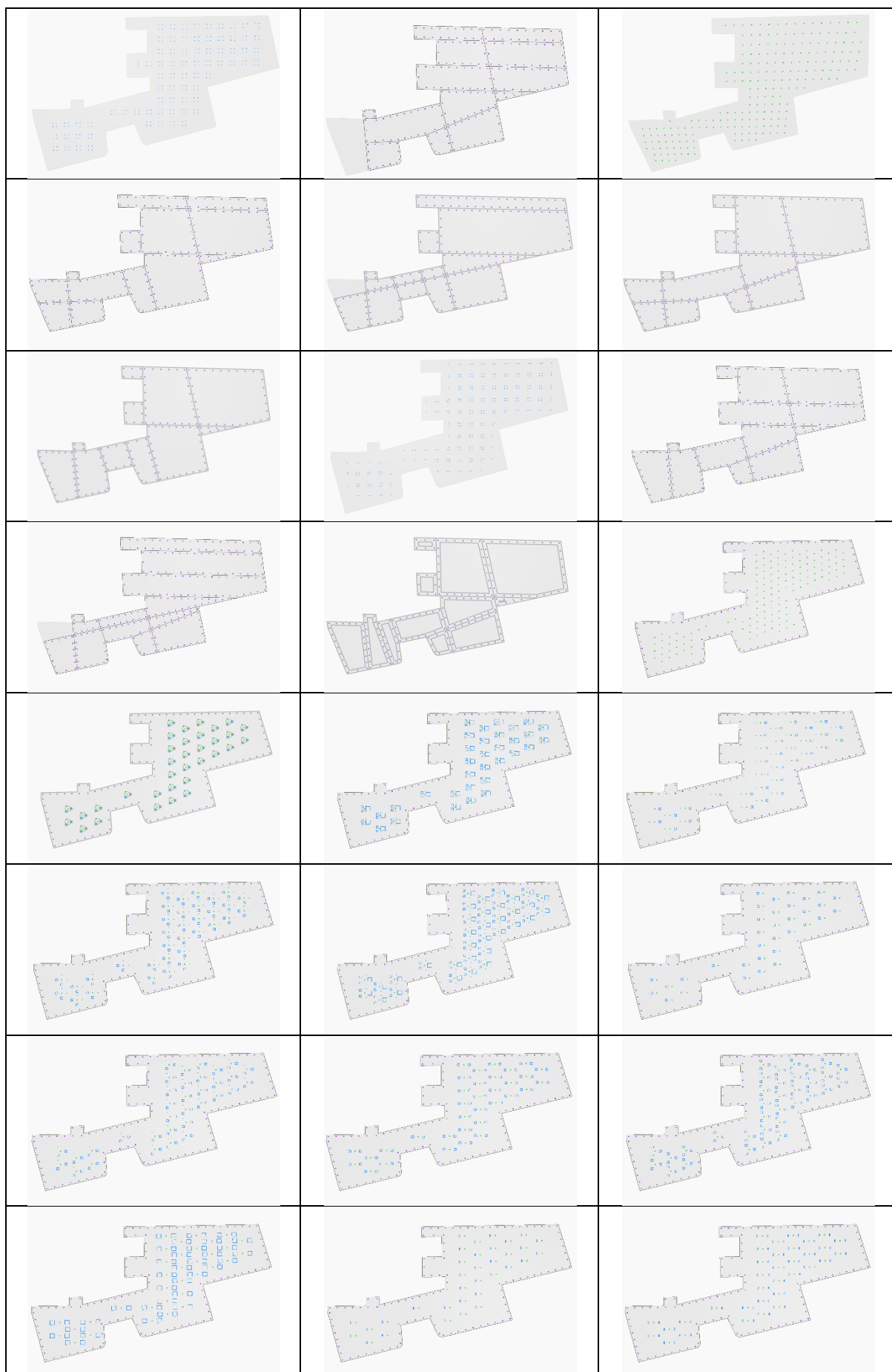
To fix this issue, an if statement or an individual script needs to be added in the algorithm. And then tested, producing proper division regardless of the arch diameter.

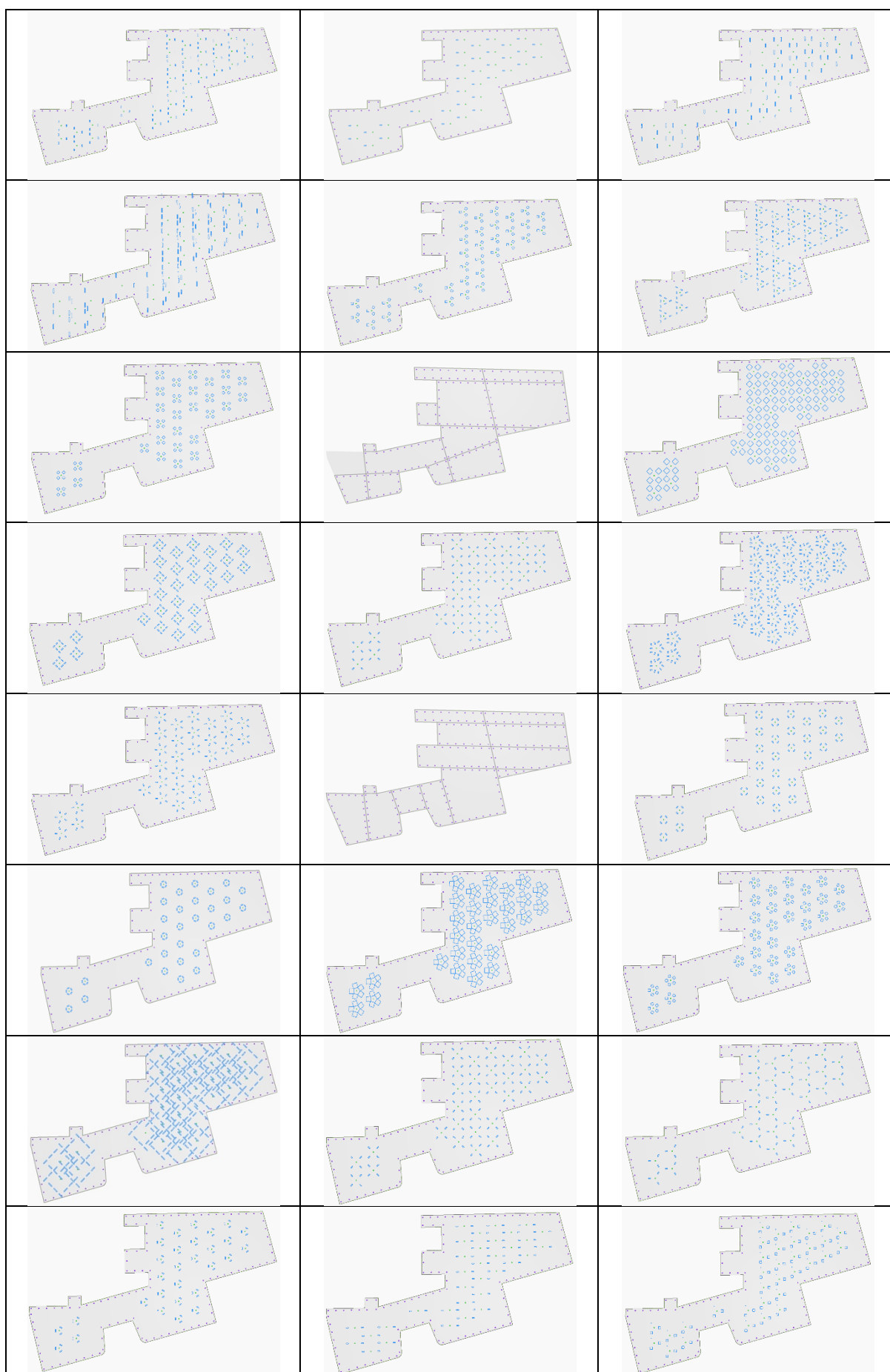
5.2.1.1.2 Office 01 results

Table 7 shows some of the produced ceilings results, where design options showed different approaches that the end user can choose from. The results vary; starting from a simple boundary lighting around the ceiling, all the way to multiple approaches combined; showing the ceiling divisions with boundary lighting and different lighting patterns.

Table 7: Office 01 generated ceilings designs results from the algorithmic system in Dynamo 2.0.3 (author)

Generated Ceiling Designs		
		
		
		
		
		
		





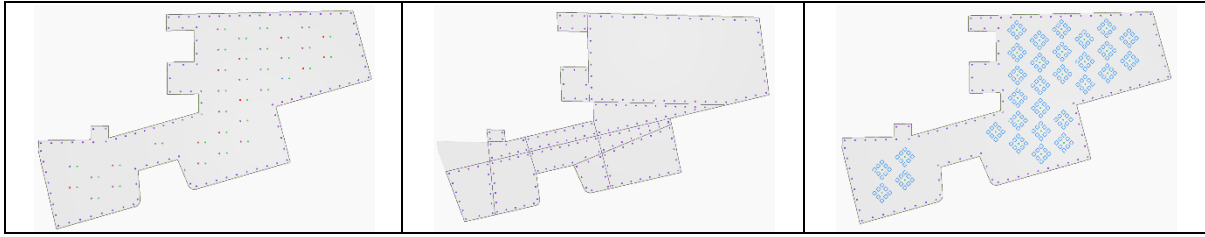


Table 7 shows the algorithm diversity in generating different reflected ceiling layouts immediately. Producing 69 ceiling designs in a matter of 1 minute. Reducing significant design time. The algorithm also provides for the designer the ability to select the ceiling he/she likes, and adjust it based on what suits their needs or design vision. The designer can view multiple options of different lighting arrangements instantly.

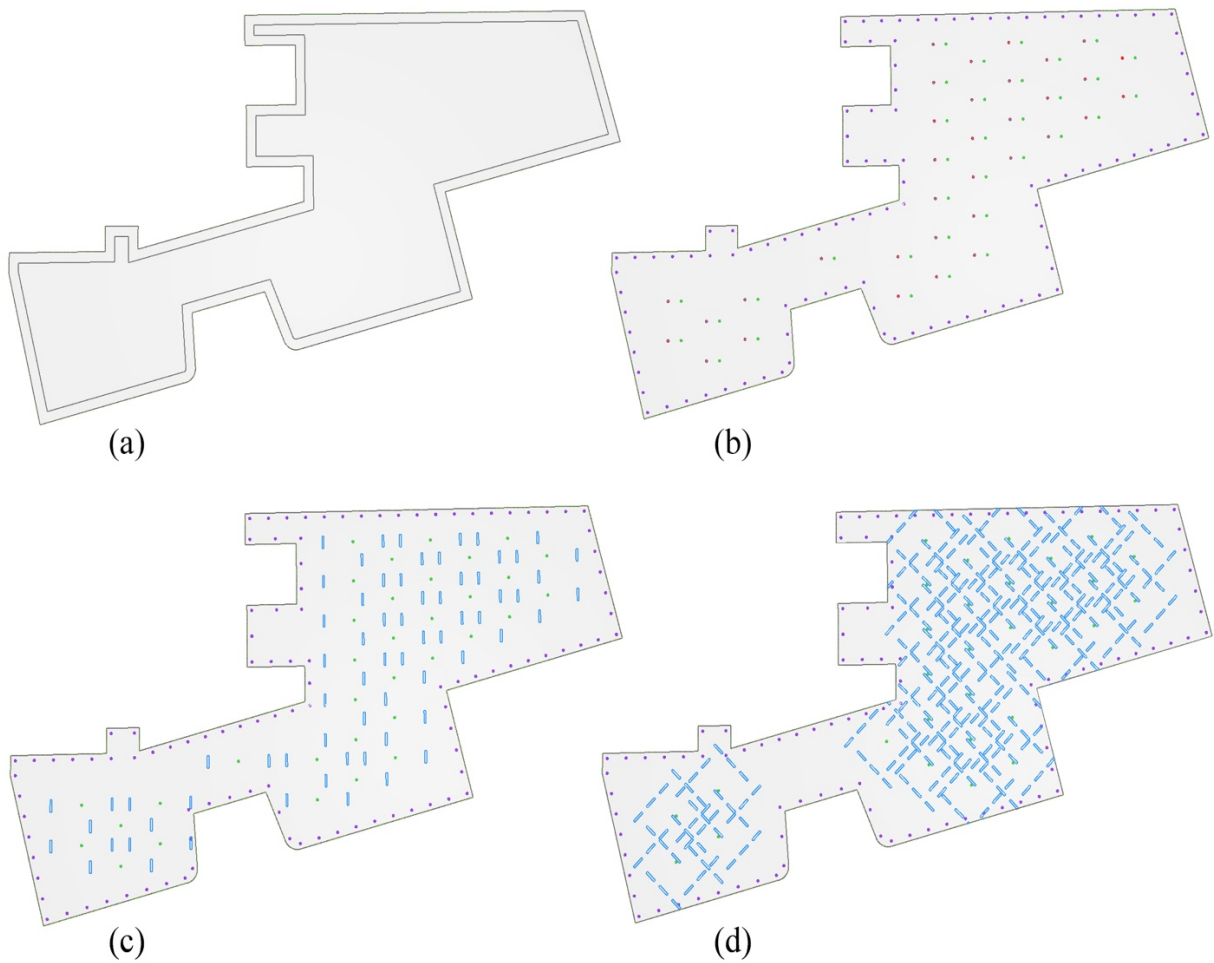


Figure 111: The diversity example of the produced layouts for Office 01 generated in Dynamo 2.0.3 (author)

Figure 111 is showcasing a small example of the diversity in the generated layouts the algorithm can produce. As (a) option is showing a simple drop ceiling layout, where the designer can set the proper distance, he/she requires from the wall.

While (b) in Figure 111 shows boundary lighting method and a simple spot lights grid created in a relatively different design than the standard grid applied on ceilings.

(c) option in Figure 111 is showing a bit more complex layout that mixes between spotlights and 100x600mm lights along with boundary lighting all around the layout. The inserted dimensions of the lights are also within the hands of the end user. As they are inserted through a list called Light Sizing.

While (d) in Figure 111 displays an overly complex irregular lighting distribution for 60x100mm lights. Although the quantity of lights produced in (d) option is not required in this area and might not be used by designer. It shows the capacity of the algorithm as it produces all the possibilities of lighting patterns regardless of its efficiency to the space.

Filtering results is left for the designer's hands. Since the immediate results are organized through writing a script that allows the end user to see the results visually and choose between the produced options in the workspace of Dynamo. Section *Arranging the generative system* discusses in detail how the designer can see the produced results.

The algorithm interactive system not only generates for the designer various options of ceiling layouts based on selected inputs. But also involves the end user through the progress. Whether it is by inserting certain variables and lighting dimensions that produces specific layouts, or through altering the produced outputs to what suits the design's vision and the interior theme. The capacity of the algorithm is constantly tested throughout the interaction between the designer and the algorithm.



Figure 112 shows the flexibility that creates an interactive system between the designer and the algorithm. The interactive levels in the algorithm are various and comprehensive. Starting from a simple approach as Figure 112, that shows the increasing or decreasing of the spacing between light coordinates in boundary lighting method.

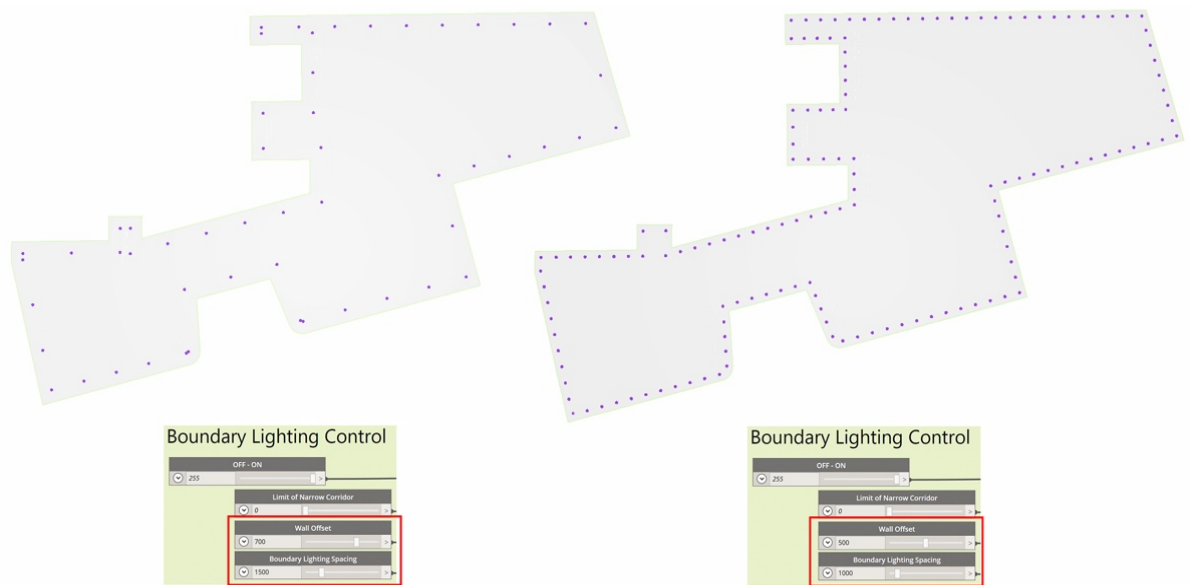


Figure 112: Comparison between boundary lighting design output for Office 01 by changing variables in Dynamo (author)

The visual results for the algorithm are immediate, producing altered results in seconds, which provides for the designer different and fast possibilities of alterations and fixing. If the same alteration is conducted in AutoCAD or Autodesk Revit, it will take around 5 to 7 minutes based on the designer's speed. However, the algorithm produces instant results as the end user simply moves the slider to whatever value he desire.

In addition, instant results of various lighting design patterns were created. Figure 113 shows four produced ceiling designs where the Pattern Type slider is the only variable that is changing, resulting in four different options generated instantly. The Pattern Type slider is discussed in the previous chapter which controls the four different families of patterns the algorithm produces: The uniform rectangular pattern, the non-uniform rectangular pattern, the uniform circular/polygon pattern and the non-uniform circular/polygon pattern.



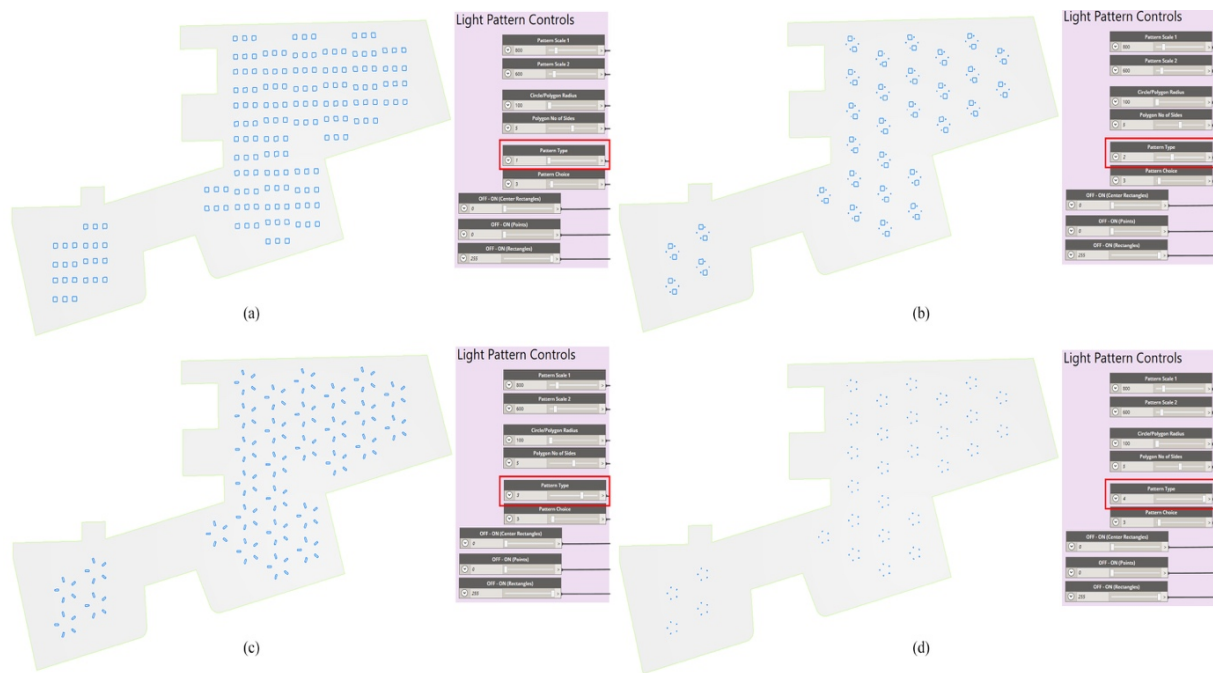


Figure 113: Different ceiling patterns for Office 01, generated solely by changing the pattern type in Dynamo 2.0.3 (author)

The produced results of the algorithm are controlled by every slider. As every minor change in any value in these sliders produce different ceiling designs. And all the sliders are controlled by the designer.

Table 8 summarizes the limitations and the results found in this case study. The limitations shown in the schedule had emerged after testing the algorithm on this ceiling, as the ceilings conditions of structural columns and curved corners were not considered earlier in building the algorithm. While the advantages confirmed the potential of the algorithm and its ability to save time, produce a high number of reflected ceilings, and its capacity to alter the results instantly based on the designer's request.

Testing the algorithm on various case studies reveals a lot of limitations that might not have been considered while scripting it. The recommendations points added to Table 8 provide an insight to the enhancement possibilities in the algorithm, in order for it to work more efficiently and to be used on a wider spectrum in the future.

Table 8: Office 01 limitations and advantages (author)

Office 01: The green room results	
Limitations	Advantages
1- The algorithm is not able to identify ceilings with structural holes in them. Or any openings in the center of the ceiling. The inserted ceilings must be one continuous surface for the algorithm to draw a polygon line around the shape and identify it.	1- The algorithm initially generated 512 reflected ceiling design options
	2- The algorithm generated more than 100 ceiling designs in a matter of 1 minute after limiting the splitting ceilings method to only 40 options.
2- The generated reflected ceilings designs took around 15 minutes, which is considered relatively slow in generative design to calculate all the possibilities which resulted in 512 options, this error refers to the used laptop capacity and not the algorithm itself.	3- The visual results for the algorithm are immediate, producing altered results in seconds, which provides for the designer different and fast possibilities of alterations and fixing.
	4- Various lighting design patterns were created
3- The curved corners of the ceiling affected the division error when applying boundary lighting method. As it was divided based on the curved arches that formed the corners of the ceiling and not based on the continuous line of the polygon that formed the ceiling.	5- An interactive system between the designer and the algorithm is established, as the produced results of the algorithm are controlled by every variable inserted in the algorithm's sliders that are controlled by the designer.
Future recommendations	
1- Increase the original script to accommodate the condition of ceilings' openings.	
2- Insert a script or an if statement for curved corners to sustain the division correctly for boundary lighting method.	
3- Add a slider that allows the algorithm to limit the generated produced ceilings to stop any possibility of crashing Dynamo while running.	

### 5.2.1.2 Office 02: Commerzbank headquarters - Frankfurt, Germany

The second case study is considered the world's first ecological office tower in Europe, the fifty-three-story building is designed by Forster and Partners in 1997 (foster and partners 2018).

With a triangular plan, the tower relies on natural systems of lighting and ventilation. As the offices are astonishingly naturally ventilated for 85% of the year, reducing energy consumption up to 50% in comparison to an equivalent air-conditioned office. Furthermore, the offices are designed to be lit naturally in addition to having open-able windows, which allows the occupants to manage and control their desired environment (foster and partners 2018).

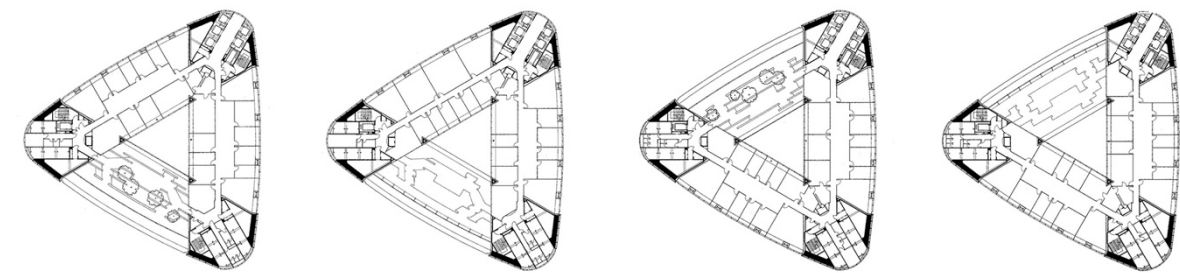
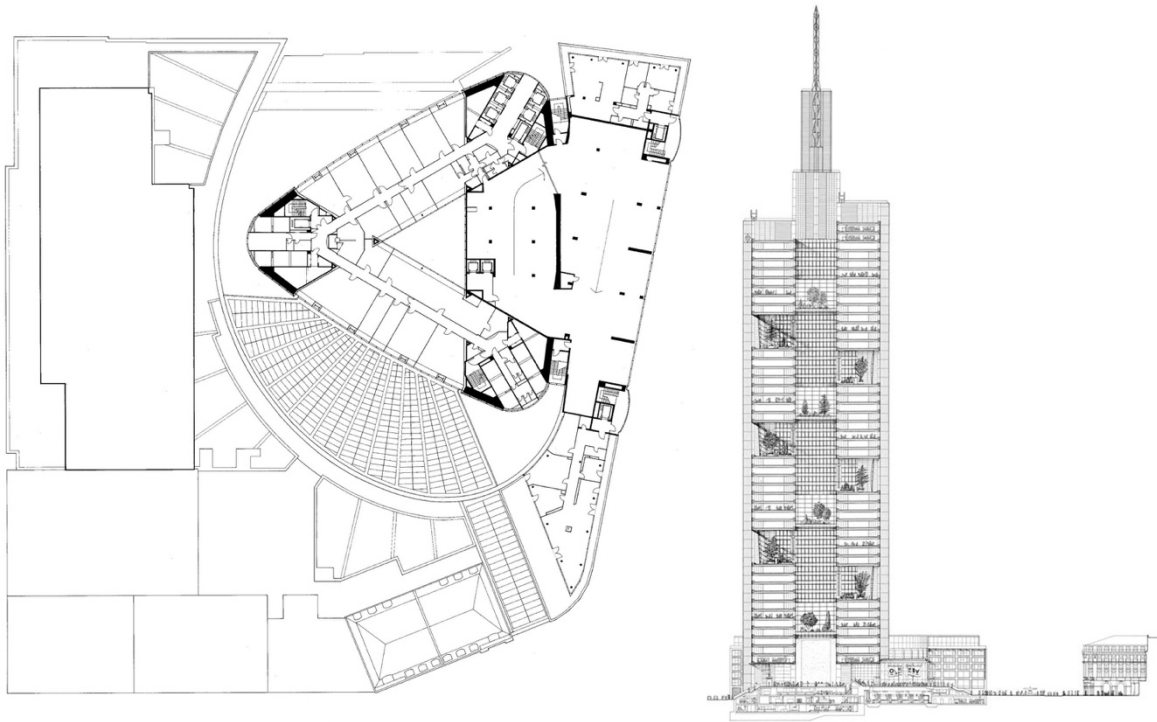


Figure 114: Commerzbank Headquarters - Frankfurt, Germany (foster and partners 2018)

The design concept of the offices tower is inspired from the anatomy of a flower, where the offices areas represent three petals, while the stem is created by the full height central atrium. Creating a triangular plan that is shown in Figure 115.

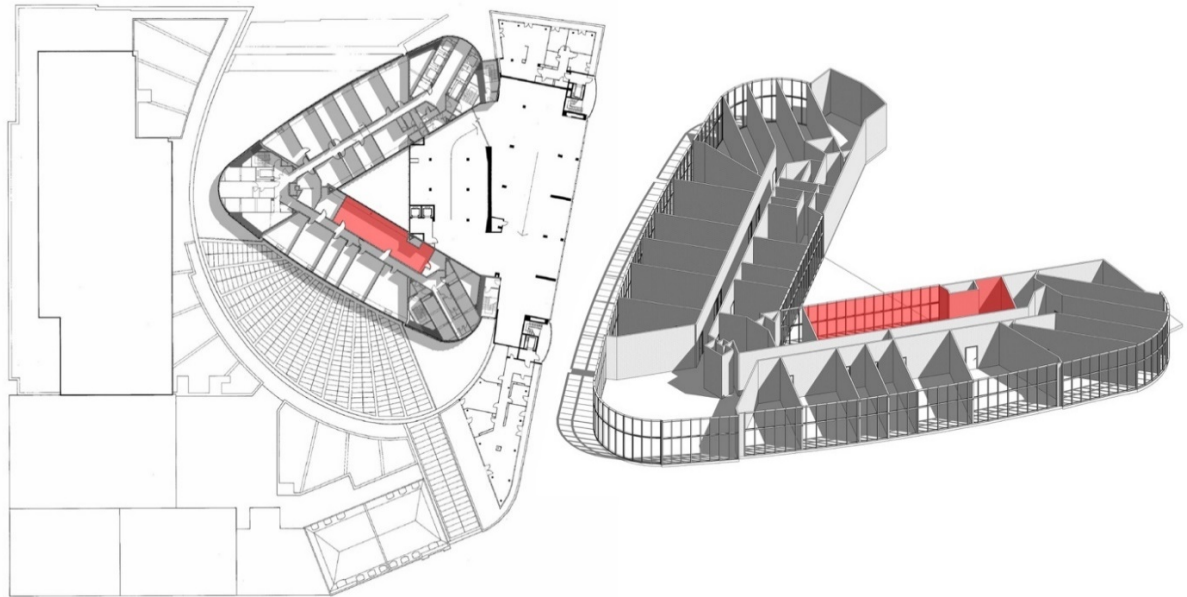


*Figure 115: Commerzbank Headquarters layout and elevation (foster and partners 2018)*

This case study is chosen due to its unique floor plan and shape. While creating the algorithm, a lot of simple ceiling shapes were tested to ensure the efficiency of it, revealing errors that were fixed while developing the proper script for the algorithm. Those errors are discussed in the previous chapter. However, unique conceptual layouts were never tested throughout the scripting of the algorithm, thus this unique layout should reveal some limitations and highlight some advantages for the algorithm that a regular symmetrical floor plan might have not reveal.

In order to test the algorithm, the layout was modeled in Autodesk Revit as seen in Figure 116, The division of the space internally is mostly standards rectangular rooms. The algorithm is able to only select one ceiling at a time, thus; the red highlighted ceiling in Figure 116 was

tested as it showed the possibility of revealing untapped potential and limitations in the algorithm.



*Figure 116: Commerzbank Headquarters modeled layout in Revit 2018 (author)*

Figure 117 shows the imported ceiling to Dynamo. Since the other rooms are regular rooms, which will produce regular results, this room was selected. The room layout will allow the algorithm to investigate the chamfered edge that's shown by the arrow in Figure 117 and the asymmetrical area of it, which might produce unexpected results.



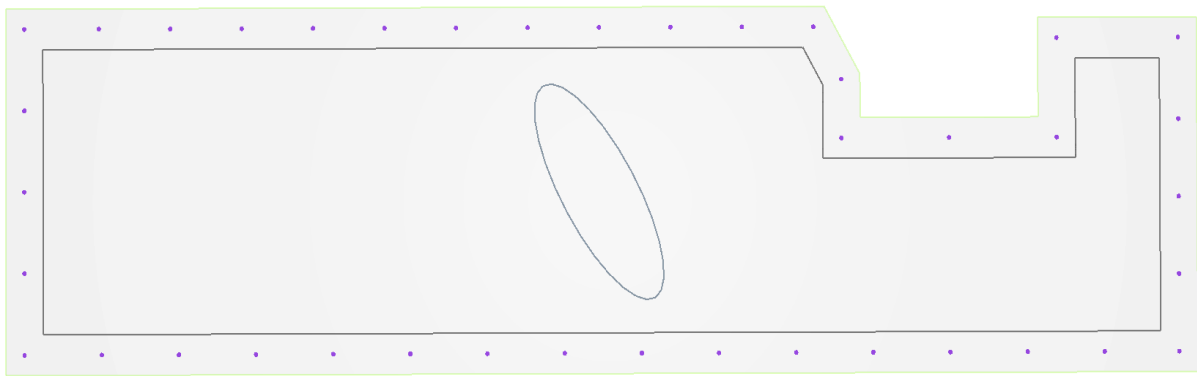
*Figure 117: The imported ceiling in Dynamo for Office 02 (author)*



### 5.2.1.2.1 Office 02 limitations and shortfalls

#### 5.2.1.2.1.1 Producing unparallelled shapes with the surface outline

The room had produced an odd result that was researched to identify the reason beyond it. After the ceiling was imported to Dynamo, one of the methods that were implement was the polygon drop ceiling method, where the approach creates different polygon shapes selected by the designer as a drop-down ceiling.



*Figure 118: Bad drop-down ceiling results for Office 2 generated in Dynamo 2.0.3 (author)*

Figure 118 shows the result when turning on the ellipse controller. The ellipse is not aligned with the surface boundaries nor any angle in it. This result should not occur as the algorithm script should produce parallel designing with the ceiling direction.

To apply certain light methods such as implementing two different types of grids and adding polygon shapes in the center of the surface. The algorithm needs to create an imaginary bounding box around the inserted surface, then intersects the shape of the surface with the created bounding box.

This limitation was revealed in this case study because of the plan shape and orientation in Revit. If the surface is inserted from Revit, the bounding box will be created based on the World Axis in Autodesk Revit, regardless of the orientation of the surface in Dynamo. This case is

presented in Figure 119; where the bounding box is shown in pink around the surface in Dynamo workspace next to the Revit model.

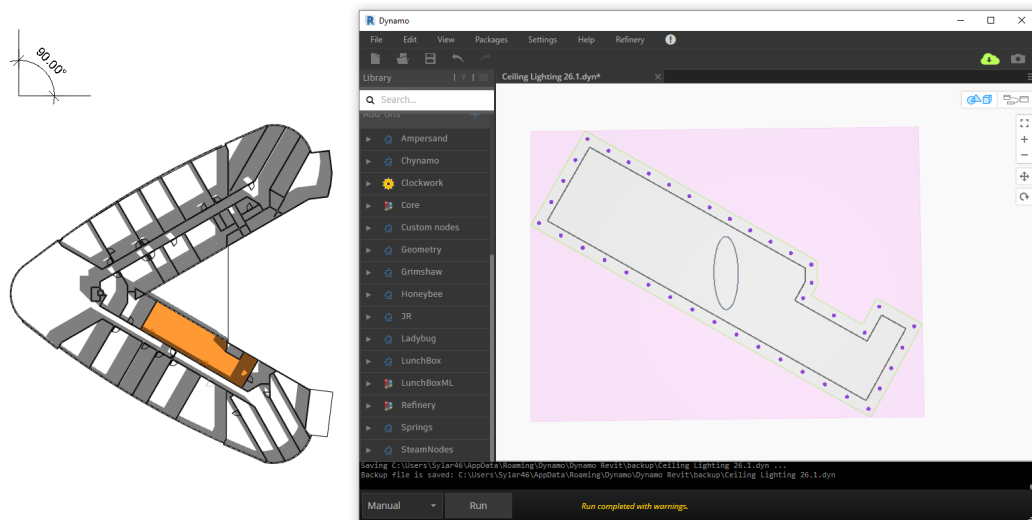


Figure 119: The modeled ceiling in Revit and the bounding box around the ceiling in Dynamo workspace (author)

The bounding box is forced to be aligned with the x, y and z axes in Revit where the original model is created. As the algorithm will fit the created ellipse from the polygon drop down ceiling method to the same axes of Revit as seen in Figure 120. This shortfall indicates that the bounding box might not always be the best approach, in addition to the importance of the model's orientation. This shortfall will impact all the generated designs methods since all of designs approaches will follow the World Axis in Revit.

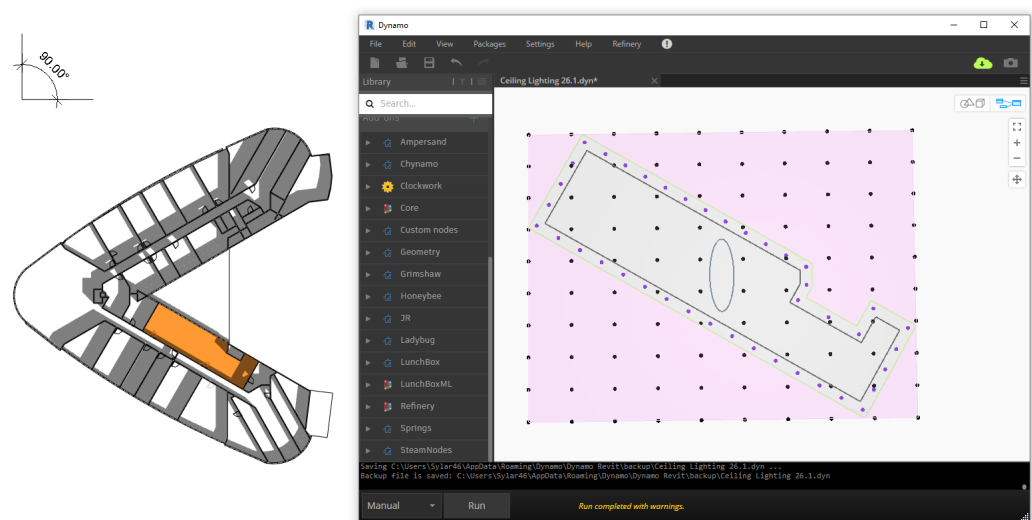


Figure 120: The applied grid using the boundary box on Revit axes in Dynamo 2.0.3 against the Revit model (author)

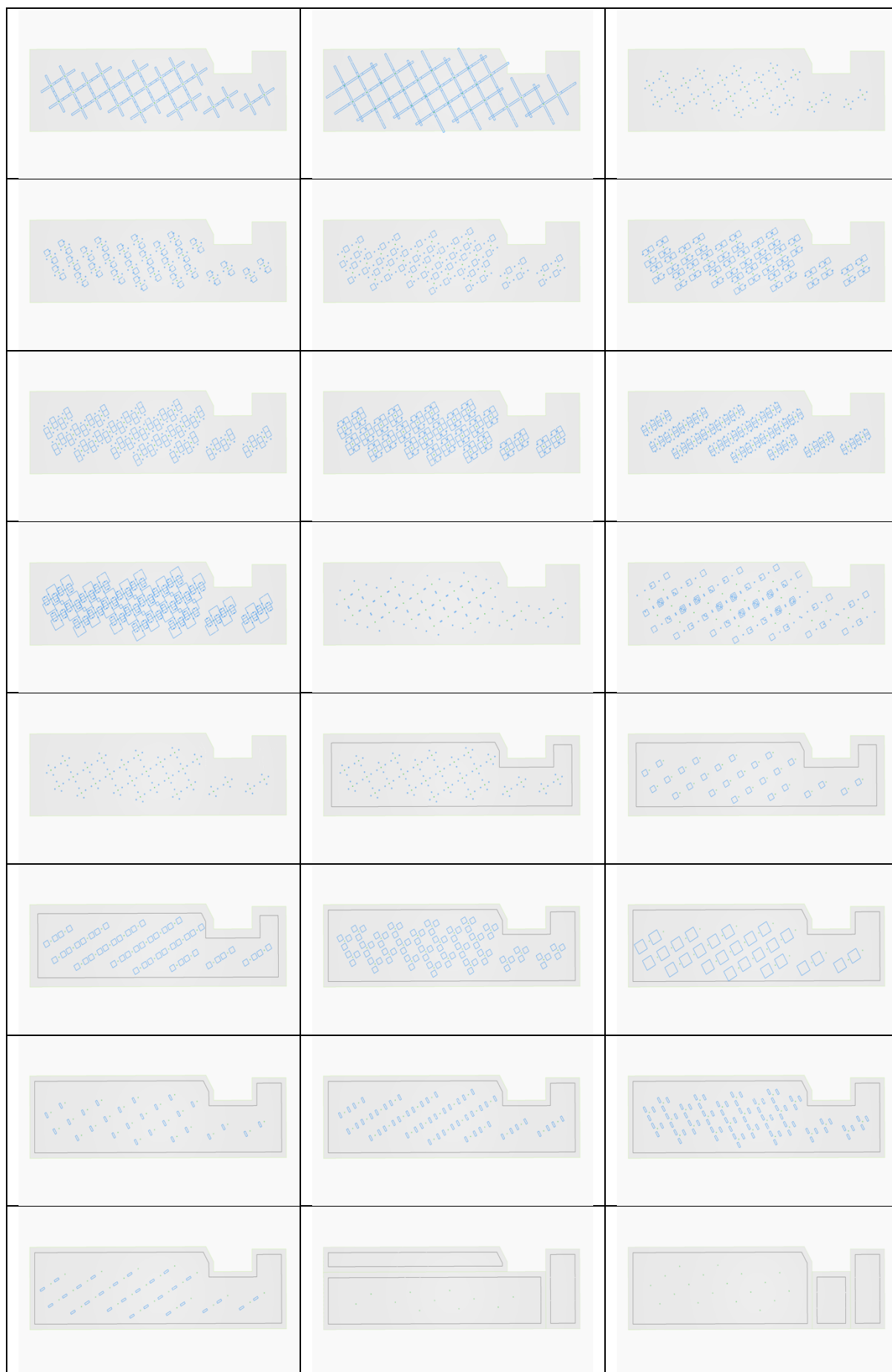
5.2.1.2.2 Office 2 results

Due to the previously discussed shortfall, all the produced light patterns were on a non-vertical slop direction, instead of the traditional horizontal or vertical directions to produce patterns. Nonetheless, the generated patterns were unique and different from the standard patterns that an individual might design. Table 9 shows some of the generated results, showcasing 78 different ceiling designs for the selected room in 1 a minute time span.

Table 9: Office 02 generated ceilings designs results from the algorithmic system in Dynamo 2.0.3 (author)

Generated Ceiling Designs		
		
		
		
		
		









As shown in Table 9, the algorithm can generate numerous results, showcasing various generated layouts that display several lighting quantities in each. Some options display only one simple approach while other generated options display more than one method merged.

One of the advantages seen in this case study is the different generated light patterns; demonstrating the diversities the algorithm can create in lighting quantities. Figure 121 shows three produced ceilings that visually show the development of the same pattern.

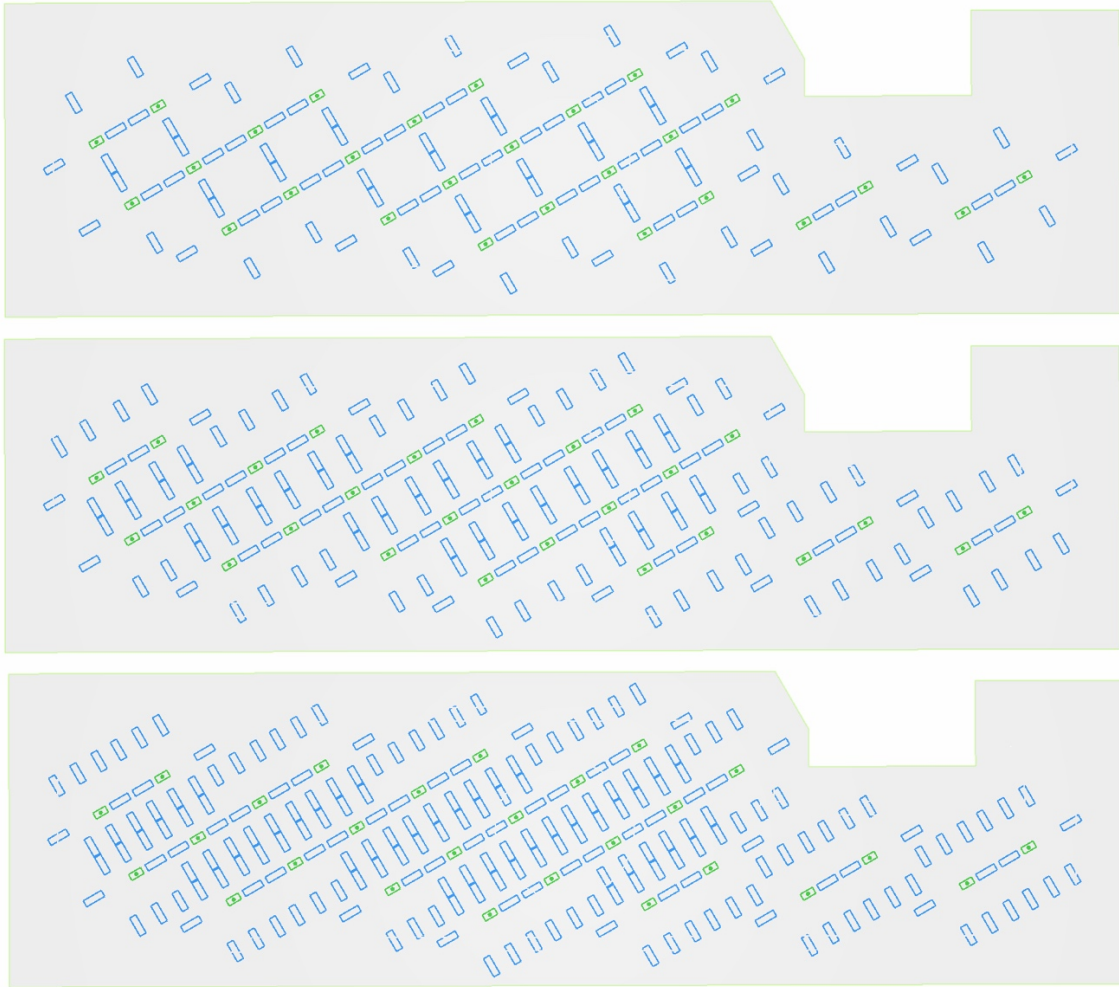
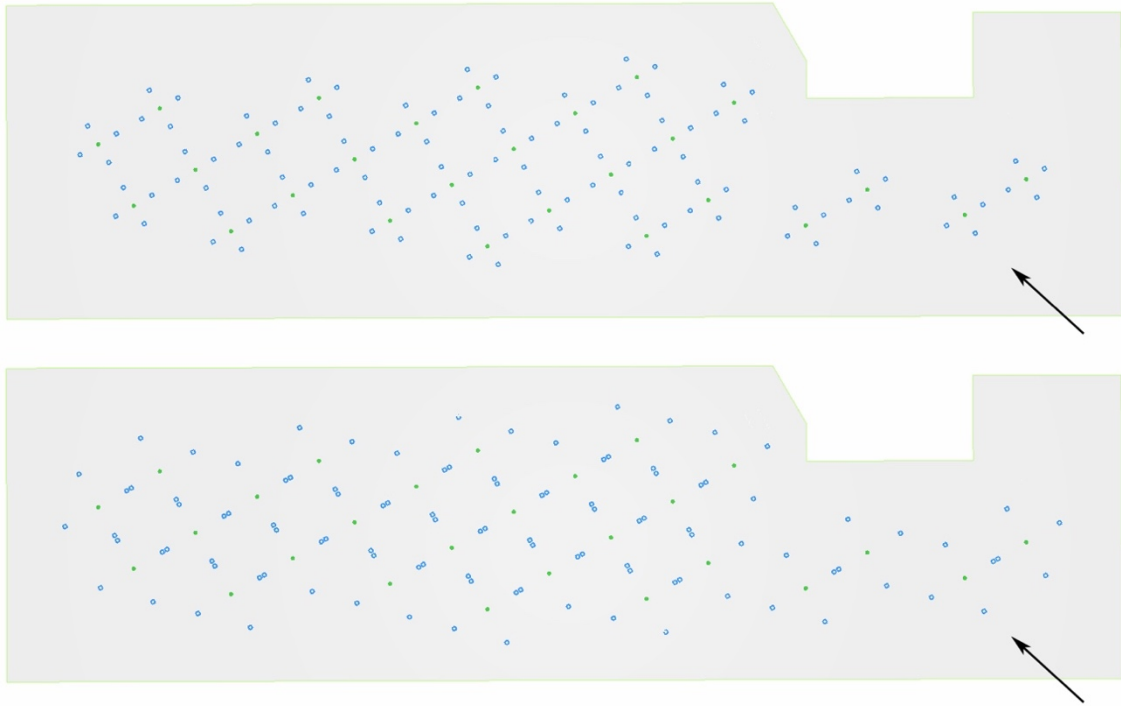


Figure 121: The development of the same pattern in Room 01 in Office 02 generated in Dynamo 2.0.3 (author)

The pattern development showcased in Figure 121 is due to the fact that the algorithm generates these patterns using mathematical divisions in order to distribute the selected lights. The mathematical calculations repeat itself for around 10 times in a matter of seconds, ensuring that every division possibility is created. As every mathematical value generated produces a pattern.

In addition to creating various layouts with different quantities of lights fixtures, the end user is able to manipulate the scale of the pattern and the quantity of the lights that created the pattern while still sustaining the pattern shape and outline. Furthermore, the designer can choose the lighting type they desire for that selected pattern through turning on and off the lights using different sliders, the sliders details are discussed in depth in *The light patterns controllers*.



*Figure 122: Generated results for Room 01 in Office 02 from Dynamo 2.0.3 (author)*

Figure 122 displays two produced ceilings from this case study, the light patterns forms beautifully in the wider space, while it space out and reduce in the narrower areas - where the black arrow is pointing in Figure 122 - creating this asymmetrical look for the ceiling, which fits the surface's shape.

These patterns are originally created by a symmetrical grid that places an imaginary boundary box on the ceiling surface, dividing it equally on both axes, and then intersecting the surface's shape with the division. In this case, the division in the narrow part is different than the wider part.

This irregular division occurred due to increasing the value in the slider that controls the grid limitation. If the value is reduced; the grid will space out more creating more uniformity in the patterns. However, the results in Figure 122 creates a proper division that flows with the surface's shape.

Figure 123 demonstrates different results generated for the selected ceiling. It visually presents how the algorithm has the capacity to combine or separate different light methods that were scripted individually earlier.

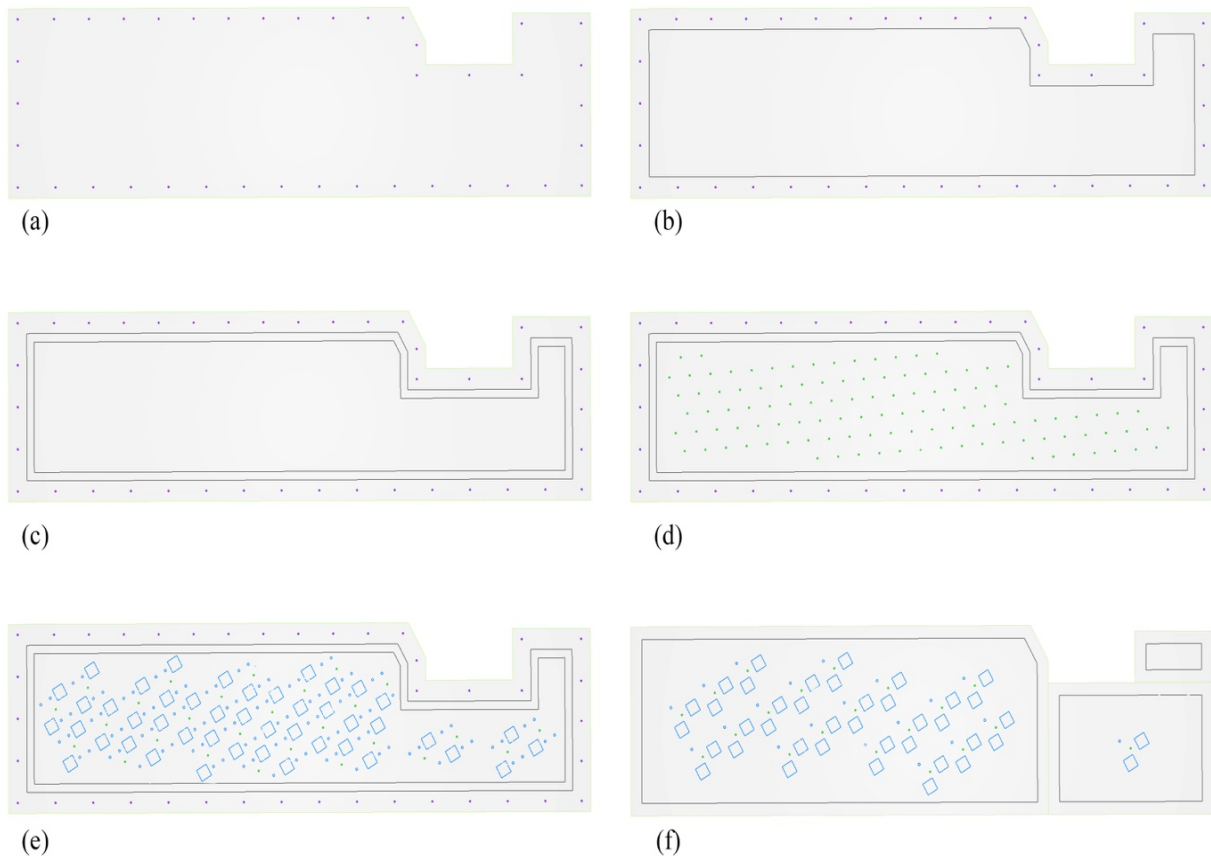


Figure 123: The development of different lighting layouts for Office 02 generated in Dynamo 2.0.3 (author)

These results show how the algorithm works, creating unlimited options that are controlled by the end user. As the algorithm is capable to produce ceilings' layouts with one design parameter: generating multiple outputs. And able to combine the different design parameters together in one ceiling at the same time. Which is produced in a matter of minutes.

(a) in Figure 123 is showing boundary lighting; a simple approach to generate lighting around the surface of the ceiling. While (b) added the drop-down ceiling method to the boundary lighting. Whereas (c) option had doubled the drop-down ceilings while sustaining the previous methods.

(d) in Figure 123 had inserted the grid to the inner surface that was created from the double drop-down ceiling. (e) option in Figure 123 had transformed the grid into a pattern light. And (f) had applied splitting the ceiling option while removing the boundary lighting and the second drop down ceiling. Which can be added or removed using the different controllers shown at Dynamo's workspace.

Table 10 concludes the results that appeared in this case study, showcasing the limitations and the advantages the algorithm produced for this ceiling, in addition, the recommendations that might help to upgrade the algorithm.

Table 10: Office 02 limitations and advantages (author)

Office 02: Commerzbank headquarters	
Limitations	Advantages
1- The lights methods were forced to be aligned with the model's orientation, in Autodesk Revit, regardless of the orientation of the surface in Dynamo.	1- The light patterns were not aligned with the surface boundary which created visually appealing designs.
	2- the algorithm generated 78 results in a matter of 1 minute.
	3- The results showcased diverse lighting quantities in each produced layout.
	4- The algorithm distributed the light patterns asymmetrically, which fitted the surface's shape. Creating proper grid division that flows with the surface's shape.
	5- The capacity to producing various ceilings' layouts using one design parameter only. While producing layouts that combine the different designs parameters in one ceiling at the same time.
Future recommendations	
1- Add an option that changes the light patterns directions to include positive and negative non vertical slopes.	
2- Investigate the possibility to change the axes that the algorithm relies on instead of Revit orientation to Dynamo's orientation. Or allows the end user to change the orientation which will immediately impact all the design methods.	

### 5.2.1.3 Summary of the results from both conceptual case studies

Table 11 shows the shortfalls besides the advantages of testing the algorithm on two conceptual case studies that had asymmetrical ceilings. Furthermore, recommendations are added to enhance the practicality of the algorithm, making it more comprehensive.

Table 11: The summary of the results from the conceptual case studies (author)

Limitations	Advantages
1. The algorithm is not able to identify ceilings with structural holes or openings in the center of the ceiling.	1. The algorithm's quantity of produced ceiling diverse based on the ceiling's shape.
2. The laptop's capacity plays a direct role on the required time to produce results.	2. A high number of ceiling designs were generated in a matter of 1-3 minutes in both cases.
3. The algorithm is not able to apply boundary lighting method appropriately on curved corners	3. The algorithm provides an interactive system for the designer that can alter and fix the visual results.
4. The orientation of the surface in Revit impacts the direction of the lighting methods in Dynamo	4. The results showcased various lighting quantities in the generated options
	5. The algorithm can create proper grid division that flows with the surface's shape.
	6. Various ceilings' layouts using one design parameter only were produced. In addition to layouts that combine the different designs parameters in one ceiling at the same time.
Future recommendations	
1- Add an option that changes the grid direction to include non-vertical slopes.	
2- Investigate the possibility to change the axes that the algorithm relies on instead of Revit orientation to Dynamo's orientation. Or add an option that allows the end user to change the orientation which will immediately impact all the design methods.	
3- Increase the original script to accommodate the condition of ceilings' openings.	
4- Insert a script or an if statement for curved corners to sustain the division correctly for boundary lighting method.	
5- Add a slider that allows the algorithm to limit the generated produced ceilings to stop any possibility of crashing Dynamo while running.	



## 5.2.2 Field case studies

### 5.2.2.1 Office A101 - Building 5 - Dubai Design District

The first field case study is an office located in Building 5, Dubai Design District (d3) area in Dubai, UAE. D3 is considered a creative ecosystem that nurtures designers from different fields. It is the home for the region's most famous architectural and interior design companies.



Figure 124: Building 5 floor plan - The office A101 floor plan (Tecom Group)

Figure 124 is showing the office location in Building 5 floor plan with a close-up image. The space's total area is 161.881 square meters with one main entrance only. The office is in a base build status since it is not occupied by any company now. There are not any lighting fixtures placed at the ceiling except for the emergency lights as seen in Figure 125. And currently lit naturally by the two full windows on both sides of the space. Since there are not any partitions or walls in the space, the entire ceiling was tested at once.



*Figure 125: Site images of Office A101 (author)*

This case study will allow the algorithm to be tested on an actual area that is still not designed nor occupied. Showcasing the capacity of the algorithm when tested on a complete surface and highlighting its importunacy in the primary status of design.

In order to run the algorithm on the selected office, the space was modeled in Autodesk Revit as seen in Figure 126. The shape of the ceiling is relatively simple; thus no limitations were expected in this case study.

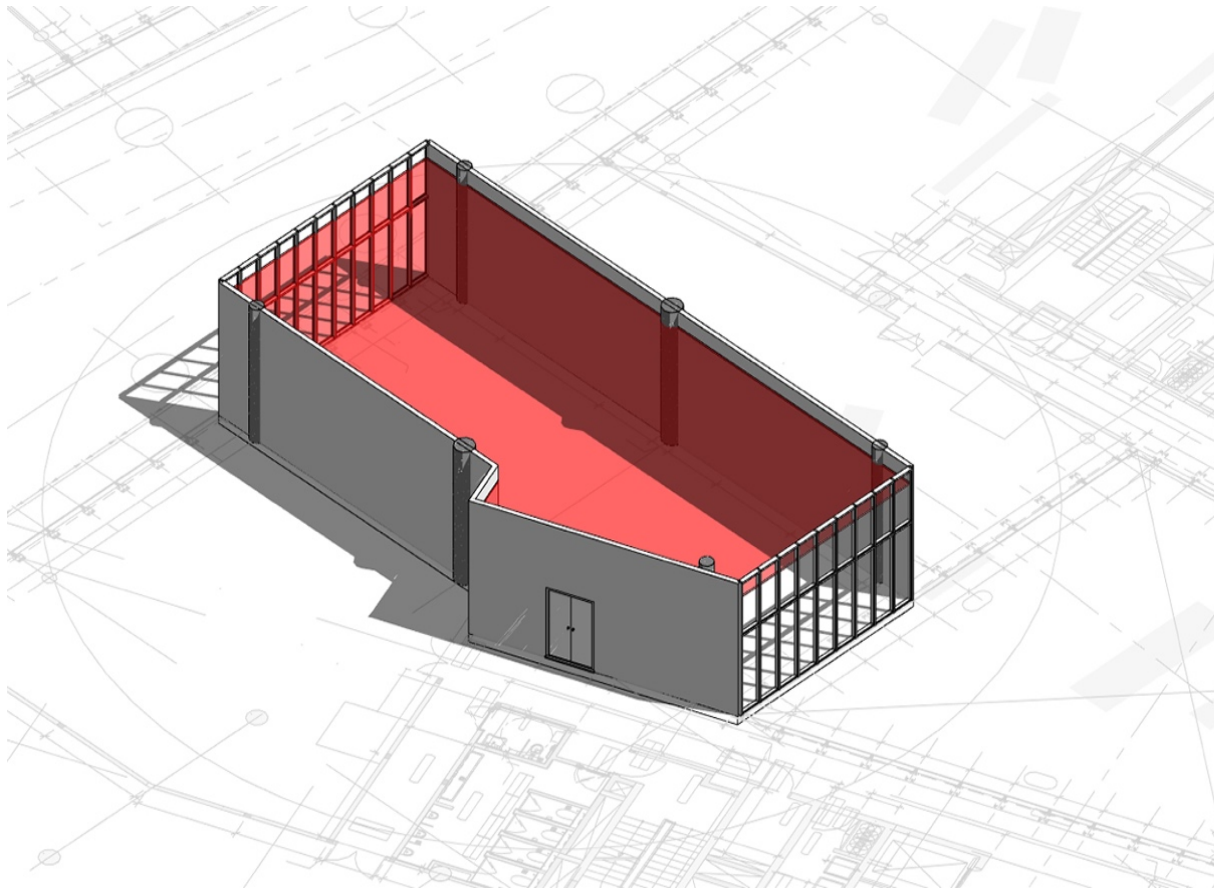


Figure 126: Office A101 modeled in Autodesk Revit 2018 (author)

An additional step was created in this case study to avoid the shortfall that appeared in *Office 02 limitations and shortfalls*. The office was oriented in Revit to become parallel to the horizontal axes, so when inserted in Dynamo; the applied lighting strategies will follow the same horizontal orientation.

This small step can have drastic changes to the generated outcome. If the end user wishes to have the lights methods applied in an angular direction, the model in Revit needs to be oriented towards the desired angle, since any changes in the orientation in Dynamo will not impact the outputs.


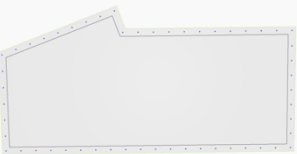
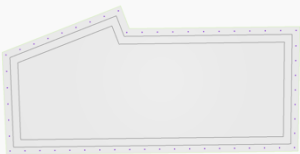
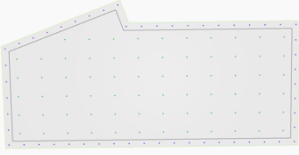
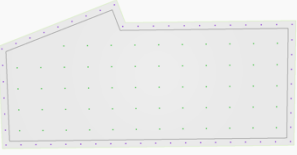











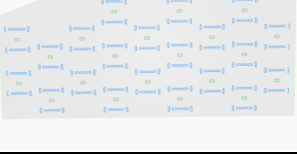



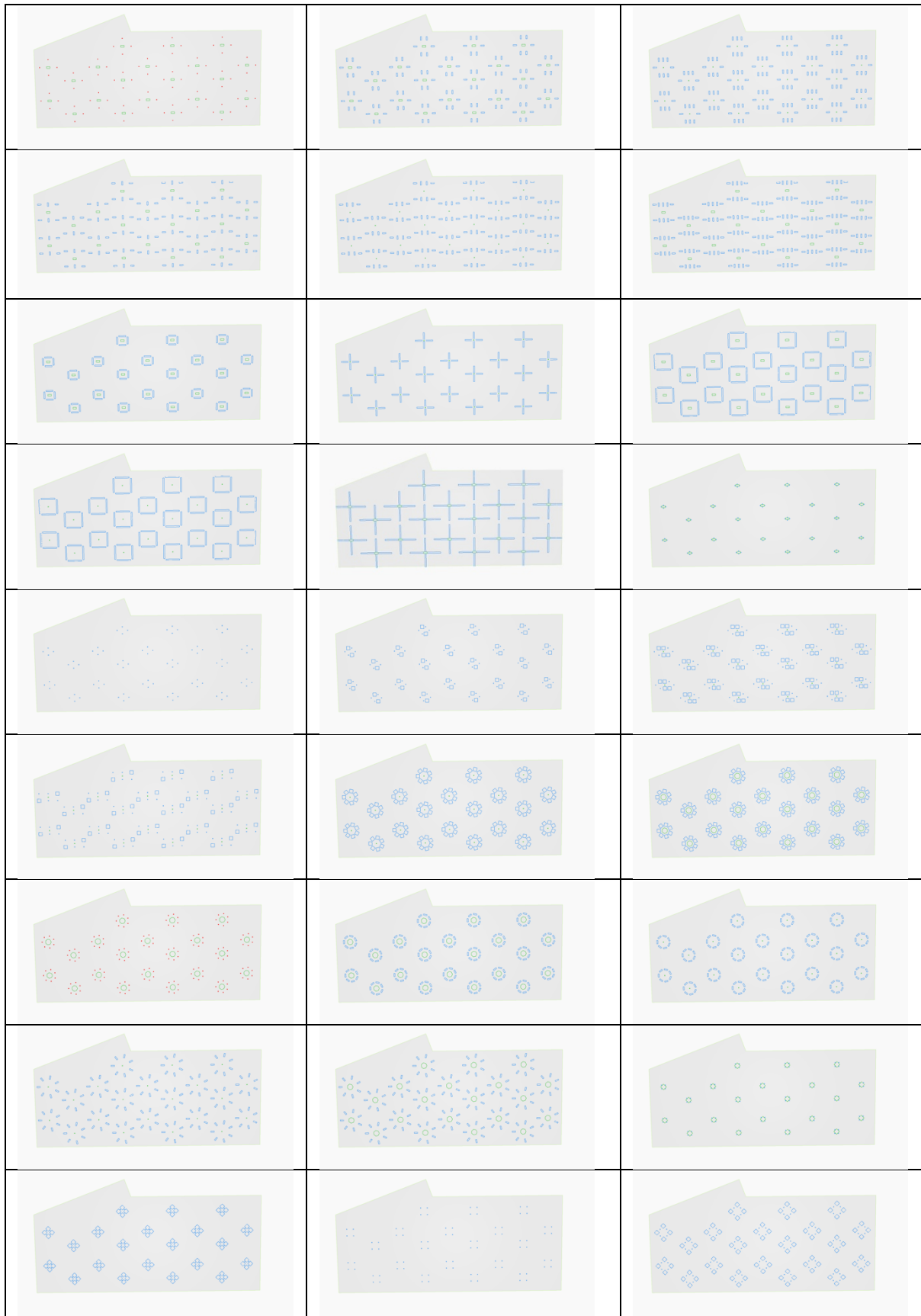
5.2.2.1.1 Office A101 results

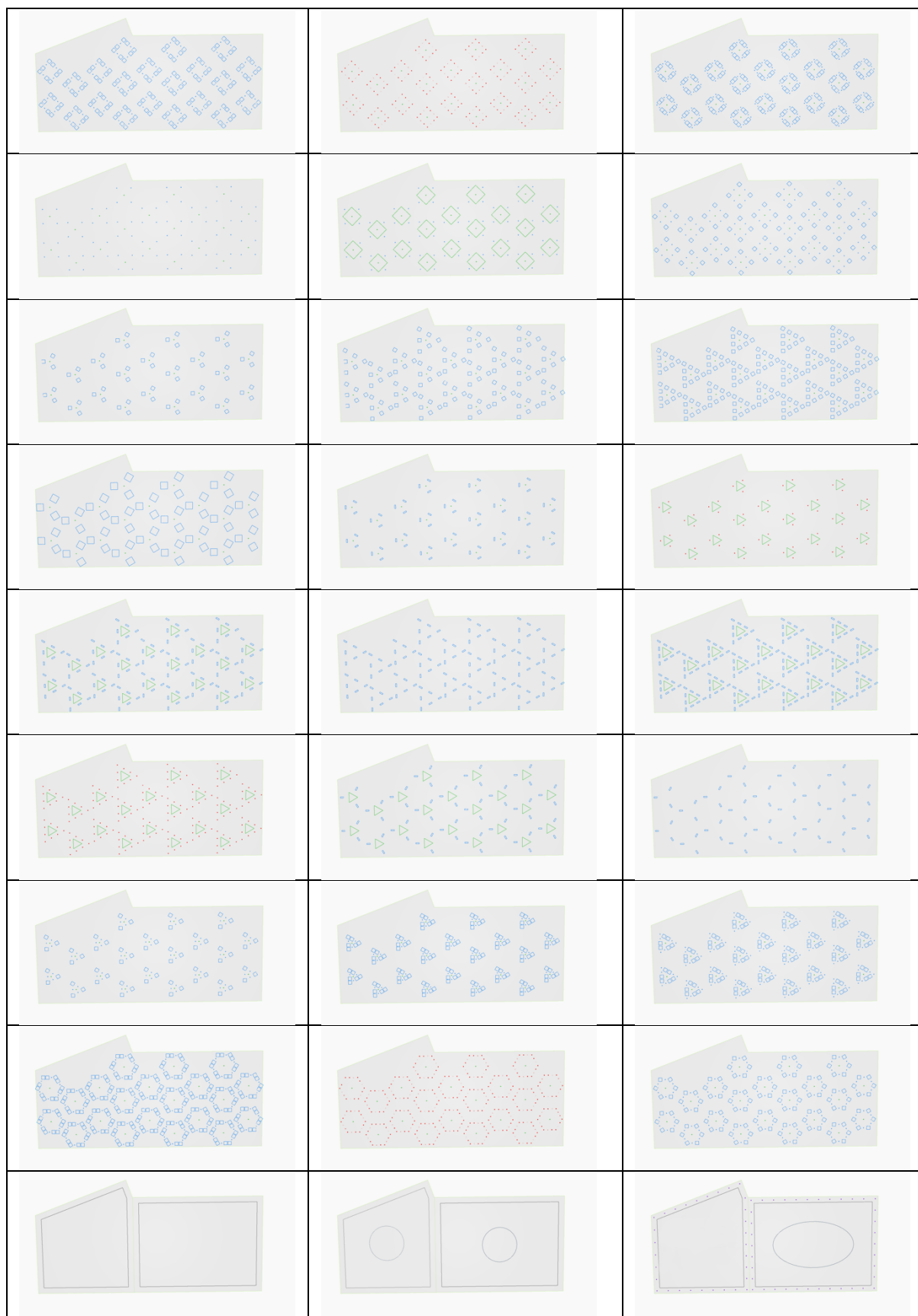
The algorithm had generated more than 100 various ceiling designs for this case study, 103 results are shown in Table 12. The designer can generate up to 500 option, through playing and manipulating the different variables that are created for every design method.

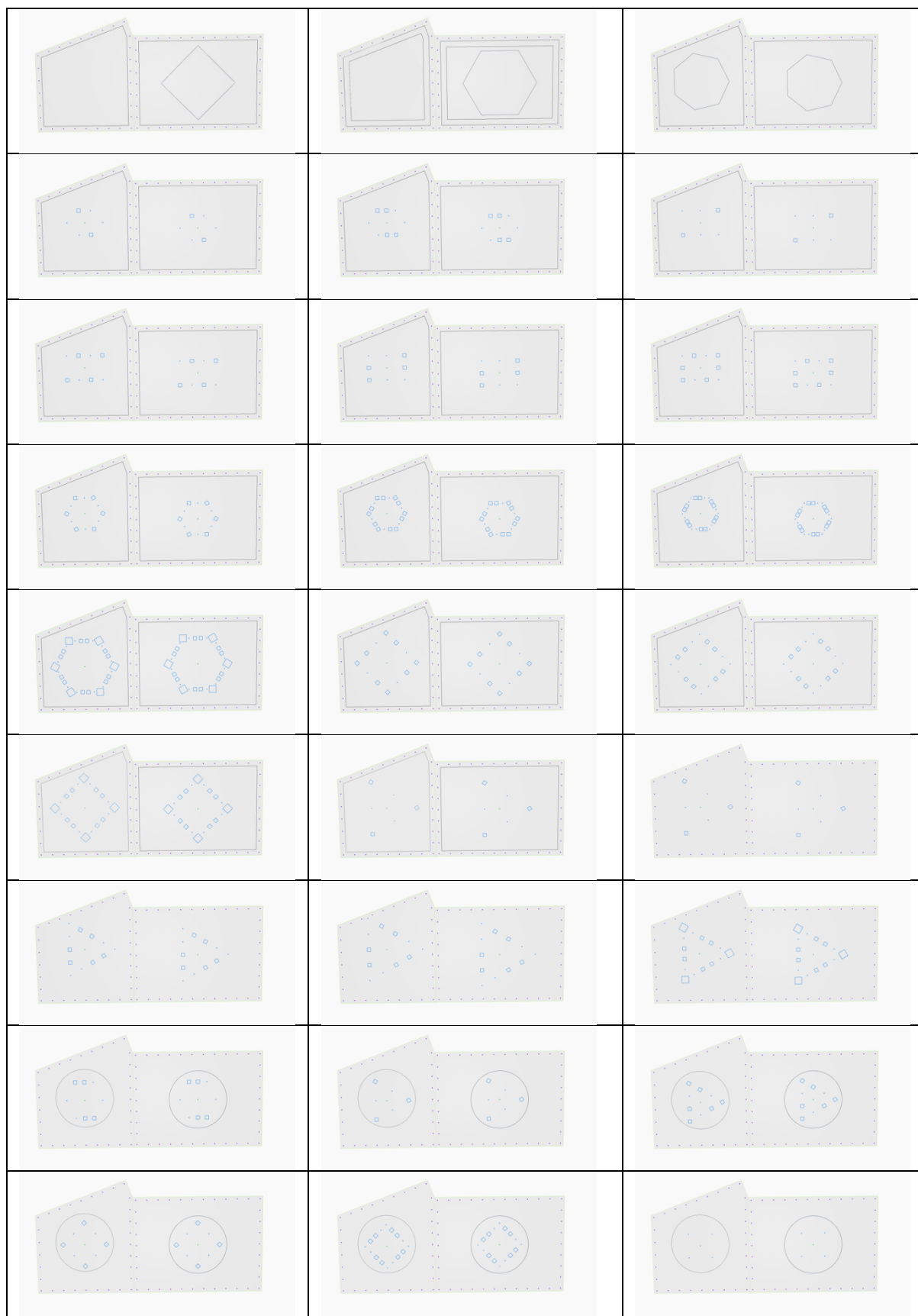
One of the main advantages of the algorithm is its flexibility, allowing it to produce individual ceilings using only one design parameter and generate ceilings that showcase more than one design method.

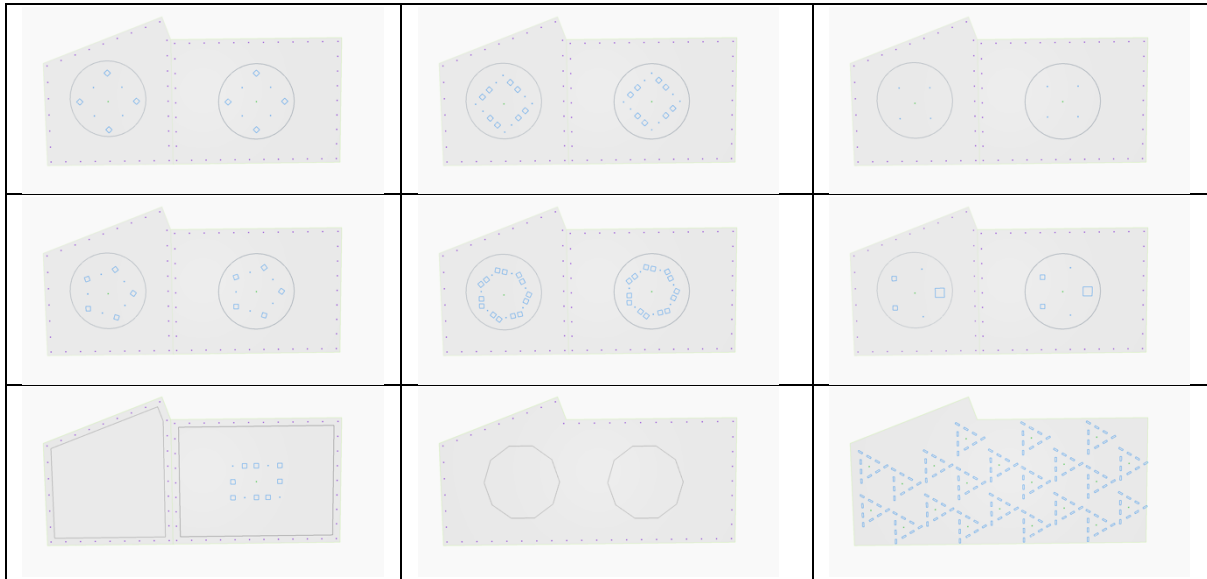
Table 12: Office A101 produced options from the generated ceilings designs in Dynamo 2.0.3 (author)

Generated Ceiling Designs		
		
		
		
		
		
		









Results showed the strong aspect of the algorithm for generating reflective ceilings designs; it produces a diverse number of ceilings in a matter of 2 to 3 minutes. Which saves great time to the designer and provides a variety of innovative options.

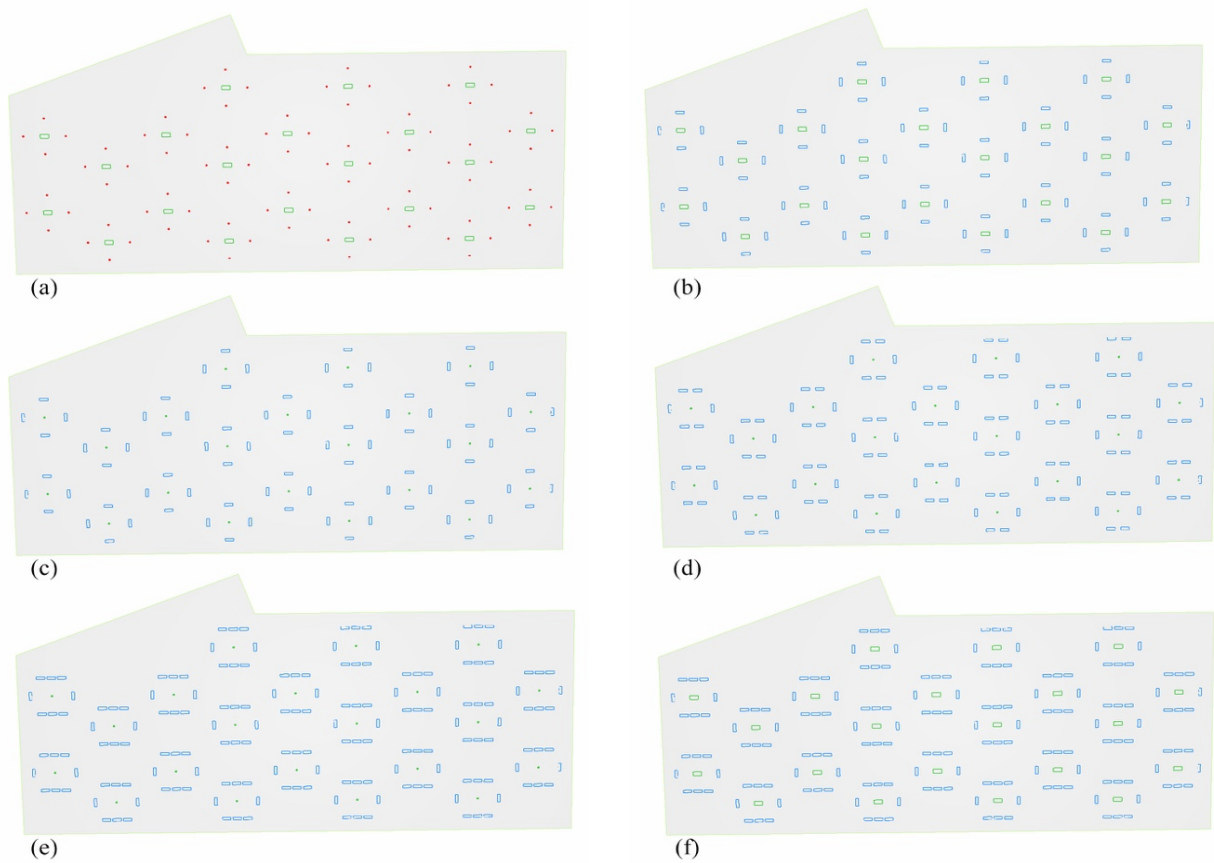


Figure 127: Development of the produced patterns in Dynamo 2.0.3 for Office A101 (author)



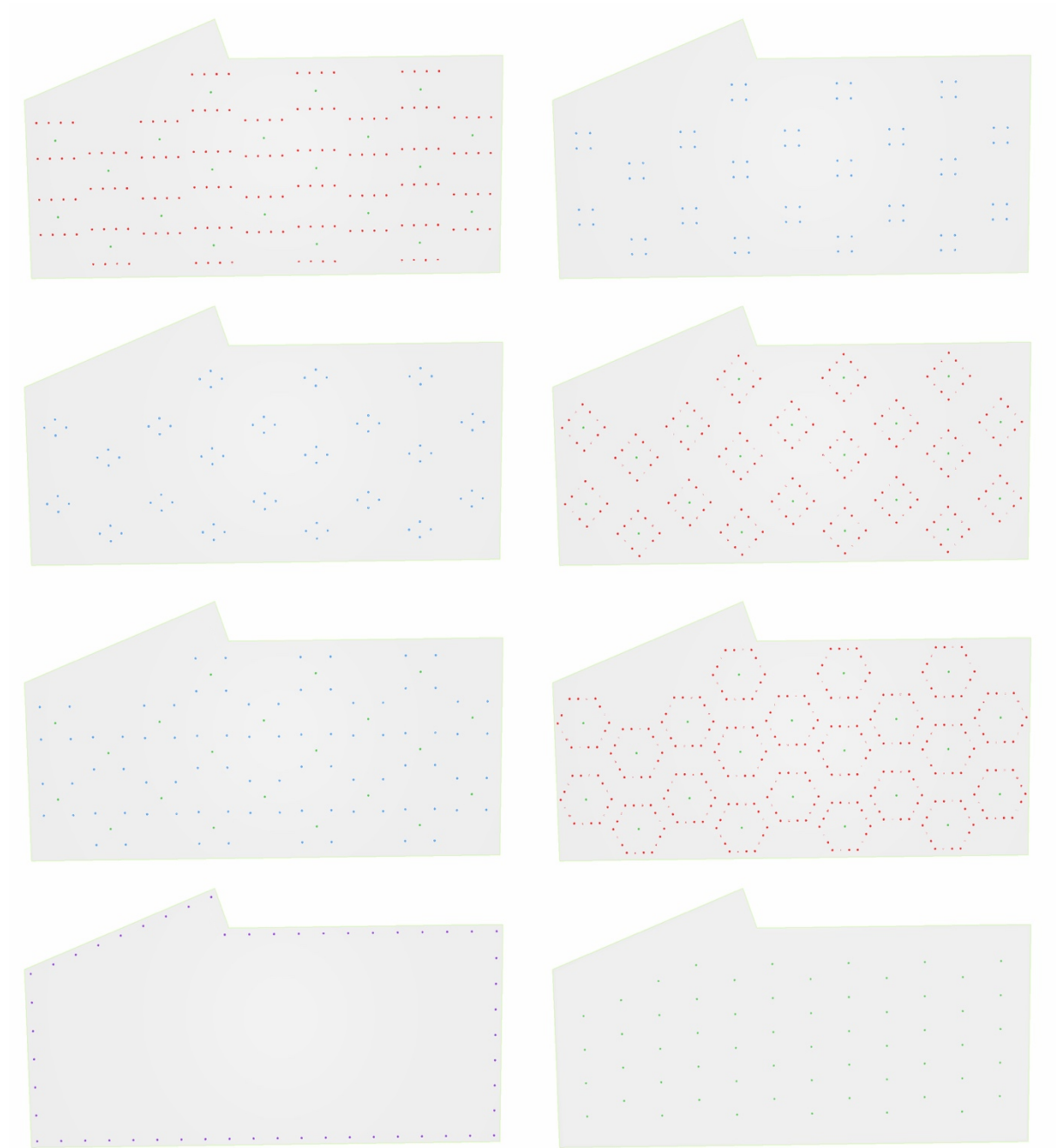
Figure 127 shows two key approaches the algorithm provides for the end user; The first approach is shown in (a), (b) and (c), as it produces different patterns from the same outline. These patterns are extracted from the same pattern but with different design variables turned on and off - an example for these variables is discussed earlier in Figure 89 -. This approach relies heavily on the interaction between the system and the designer. As he/she can select what to switch on and off through controlling the given variables in the controller sliders.

The second approach is generated automatically by the algorithm, where (c), (d), and (e) in Figure 127 shows the same pattern but with different division, giving the designer the diversity to select the number of light fixtures required while sustaining the same pattern. While (f) in Figure 127 displays the ability of the designer to change the lights' variables even if a certain pattern with specific division is selected.

The first approach in Figure 127 shows (a) with the center rectangular and the points around it that indicates spot lights, while (b) displays the same center rectangular lighting with 600x100mm lights turned on instead of the points. However, the points are actually located in the center of these lights. (c) had replaced the center rectangle to a point and kept 600x100mm lights. This shows how one outline can easily produce different patterns.

While (c), (d) and (e) in Figure 127 shows the second approach the algorithm uses to produce patterns, where (c) shows a division that resulted in 2 lights; producing only one 600x100mm light on every side. Whereas (d) had increased the division to result in 3 lights, and (e) to 4 lights. The algorithm will repeat the same division vertically as well.

Every pattern will only be divided up to 3 times and then the next pattern will be divided vertically and horizontally and so on. In addition, the algorithm uses 4 different scripts to generate the patterns, that falls under 4 different categories that are discussed previously in detail in *Grid sub-approach: Creating Light patterns*.



*Figure 128: Different generated ceilings using spotlights in Dynamo 2.0.3 for Office A101 (author)*

Figure 128 highlights the different results the algorithm produced using only spotlights. This shows the ability of the algorithm to not only produce diversity in lighting patterns, but also in arranging spotlights to create different ceiling reflective designs. Varying from a simple grid or boundary lighting to more complex designs that forms polygon shapes. The spaces between every spotlight and the radius of the shapes in the showcased designs can be increased or decreased based on the designer's desire.

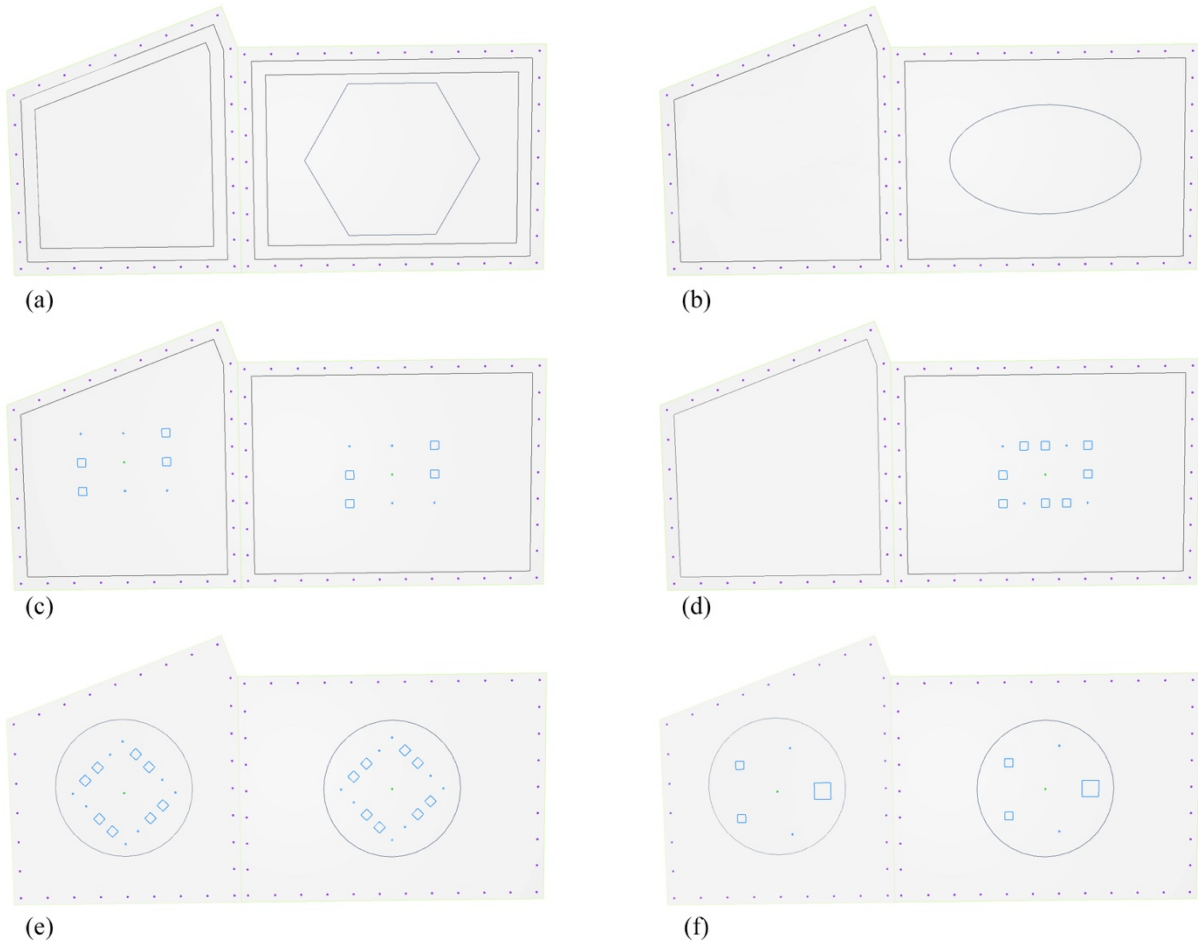


Figure 129: Generated mixed lighting design methods in Dynamo 2.0.3 for Office A101 (author)

Figure 129 displays other produced options generated by the algorithm, where different design approaches are mixed. This shows that this system does not only depend on producing designs using one design approach. But has the capacity to generate different designs through mixing different approaches together.

For example; (a) in Figure 129 shows that the algorithm had applied splitting ceilings approach, creating 2 areas for the ceiling, and applied boundary lighting, double drop down ceiling and a polygon drop down ceiling in one of the split areas. The levels of the ceilings are left entirely to be determined by the designer's desire as these information depends on the ceiling height, and the algorithm is not fed the ceiling's height since it only uses two axes: y and x axes to

create the surface shapes. (b) shows a different option where an ellipse polygon drop-down ceiling is applied on only of the split ceilings.

(c) and (d) in Figure 129 show a different possibility after applying the splitting method. Where a designer can place only one pattern in the center of each ceiling. The generative system will treat every ceiling individually after the splitting, giving the designer the flexibility to pick and choose what to apply for each.

While (e) and (f) create a different option where the pattern is applied in the center of the drop-down ceiling. Any selected pattern will always fall in the center of any selected polygon shaped drop ceiling because both approaches use the same center point for the surface.

Table 13 highlights the advantages that were seen in testing the algorithm on this ceiling. No limitations had appeared due to the simplicity of the ceiling shape.

*Table 13: Office A101 limitations and advantages (author)*

Field case study 01- Office A101	
Limitations	Advantages
None were found in this case study	1- Producing over 100 different ceiling designs in 2-3 minutes.
	2- Great deal of diversity in light patterns, design approaches and lighting fixtures quantities
	3- The algorithm showed its capacity as a functional interactive system between the designer and producing designs.

### 5.2.2.2 Office A202 - Building 7 - Dubai Design District

The second field case study is also located in the same area; Dubai Design District (d3) in Dubai, UAE. The office is located on the second floor in Building 7. Its space is larger than the first field case study, where the total area is 341.704 square meter. Figure 130 shows the location of the office in building 7, in addition to the office layout.



Figure 130: 24: Building 7 floor plan and the office A202 floor plan (Tecom Group)

The previous tenants had recently cleared the office; thus, the glass partitions and the floor finishing remain. The office has one entrance and it is divided into 4 areas. Two rooms with an area of 61 square meter each, and an area of 186 square meter for the employees' desks. In addition; a small storage room with and area of 24 square meter.

The space is currently lit naturally through the windows seen in Figure 131, since the ceiling only has the emergency lights.

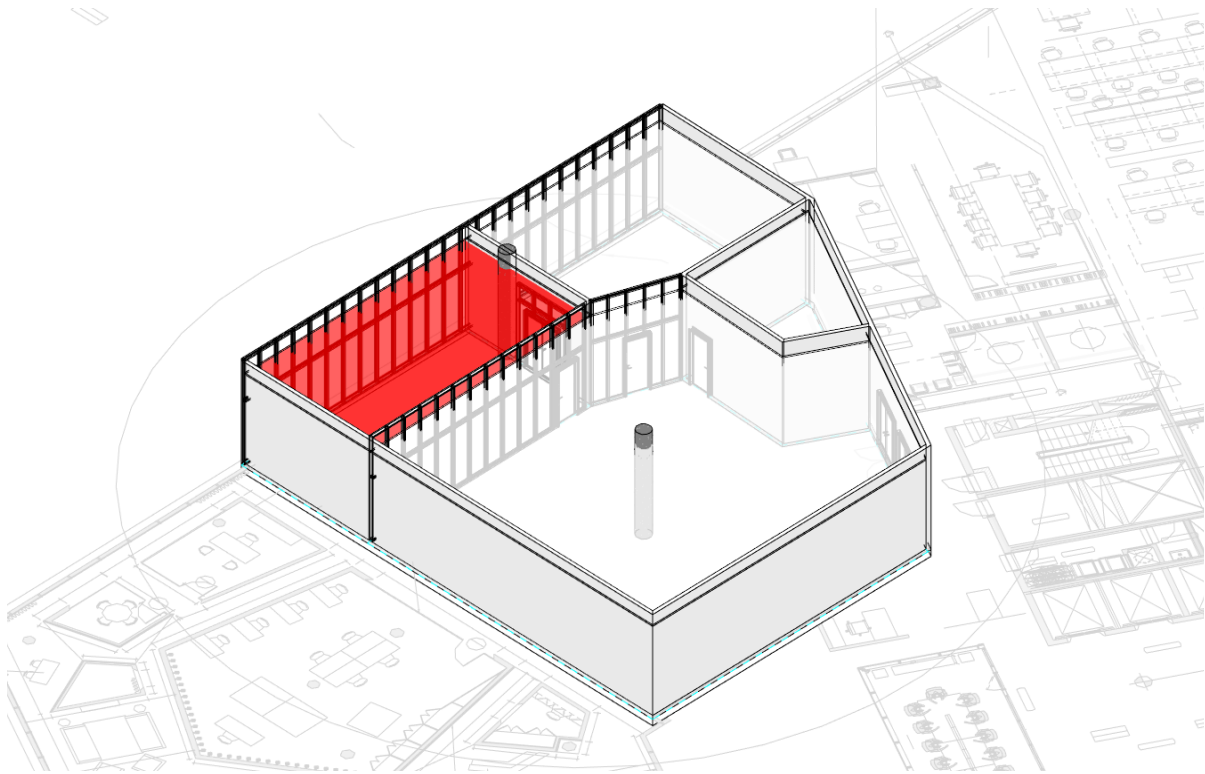




Figure 131: Site images of Office A202 against its location in the layout (author)

Since the office has 4 areas, only one area is selected. Which is the meeting room highlighted in red in Figure 132. The meeting room holds an area of 61 square meter, and since no rectangular room was tested in the previous case studies, this room should display the optimized resulted of the algorithm, where all the different lighting methods should work without any limitations.

As all the previous case studies, the office was modeled in Autodesk Revit as seen in Figure 132. To select the chosen ceiling and insert it to Dynamo to start the calculations.



*Figure 132: Office A202 modeled in Revit 2018 (author)*

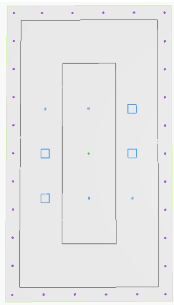

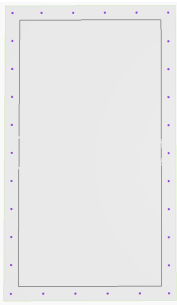
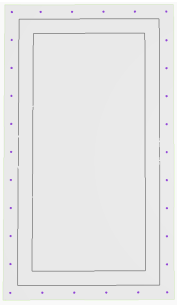
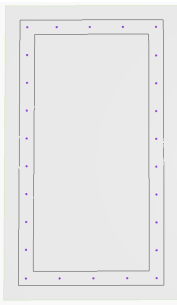
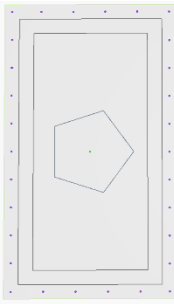
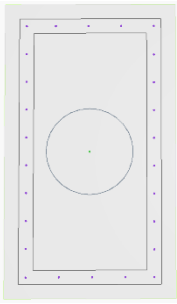
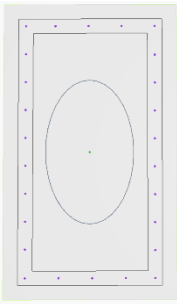
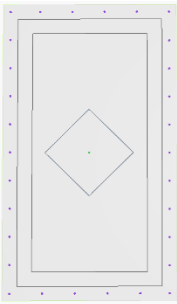
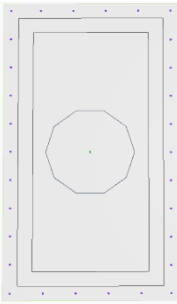
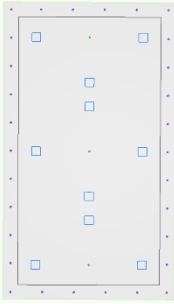
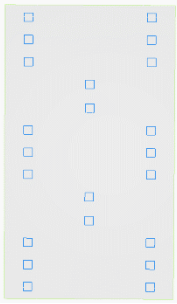
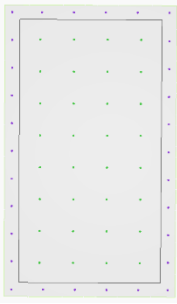
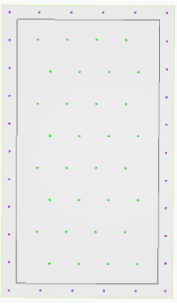
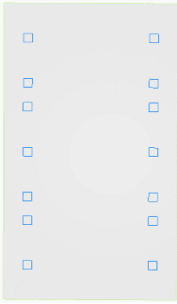
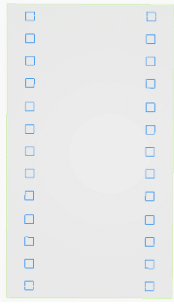
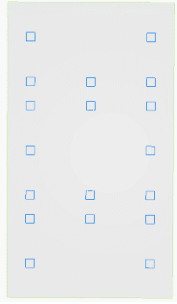
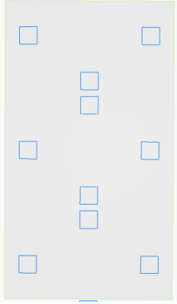
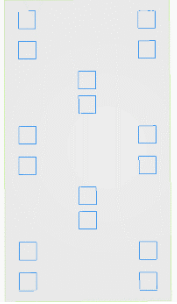
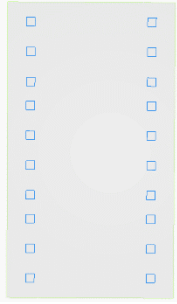
Like the previous field case study, the orientation of the model was adjusted in Revit to ensure that the desired orientation of the lighting methods will appear in Dynamo.

An additional phase will be added to this case study only, where 2 designs from the generated ceiling designs will be chosen to run the lighting analysis in Autodesk Revit 2018 in *Phase 02: Connecting the results to Autodesk Revit*.

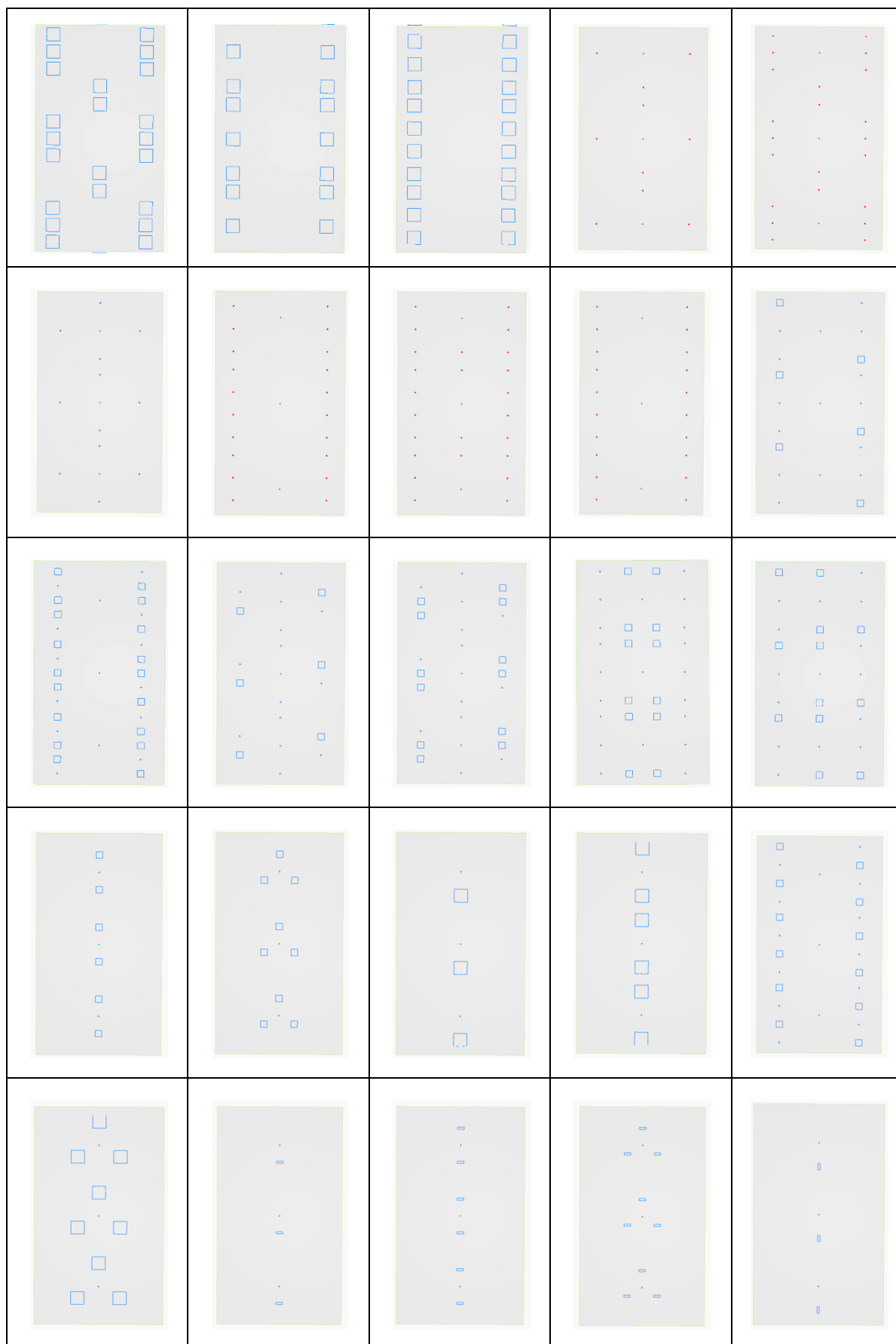
5.2.2.2.1 Office A101 results

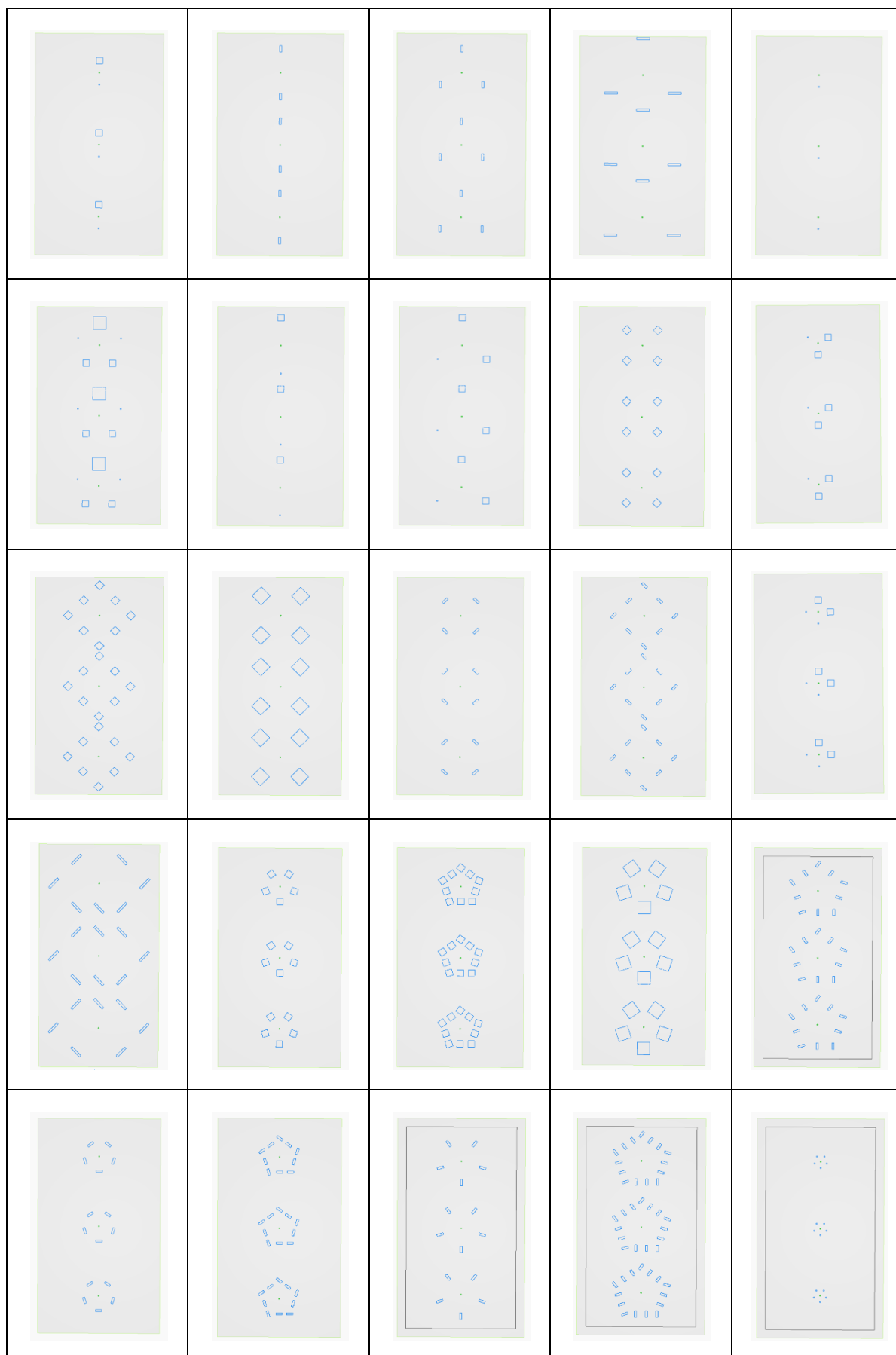
Table 14 shows the results generated by the algorithm for this case study, as it was able to produce 110 ceiling designs.

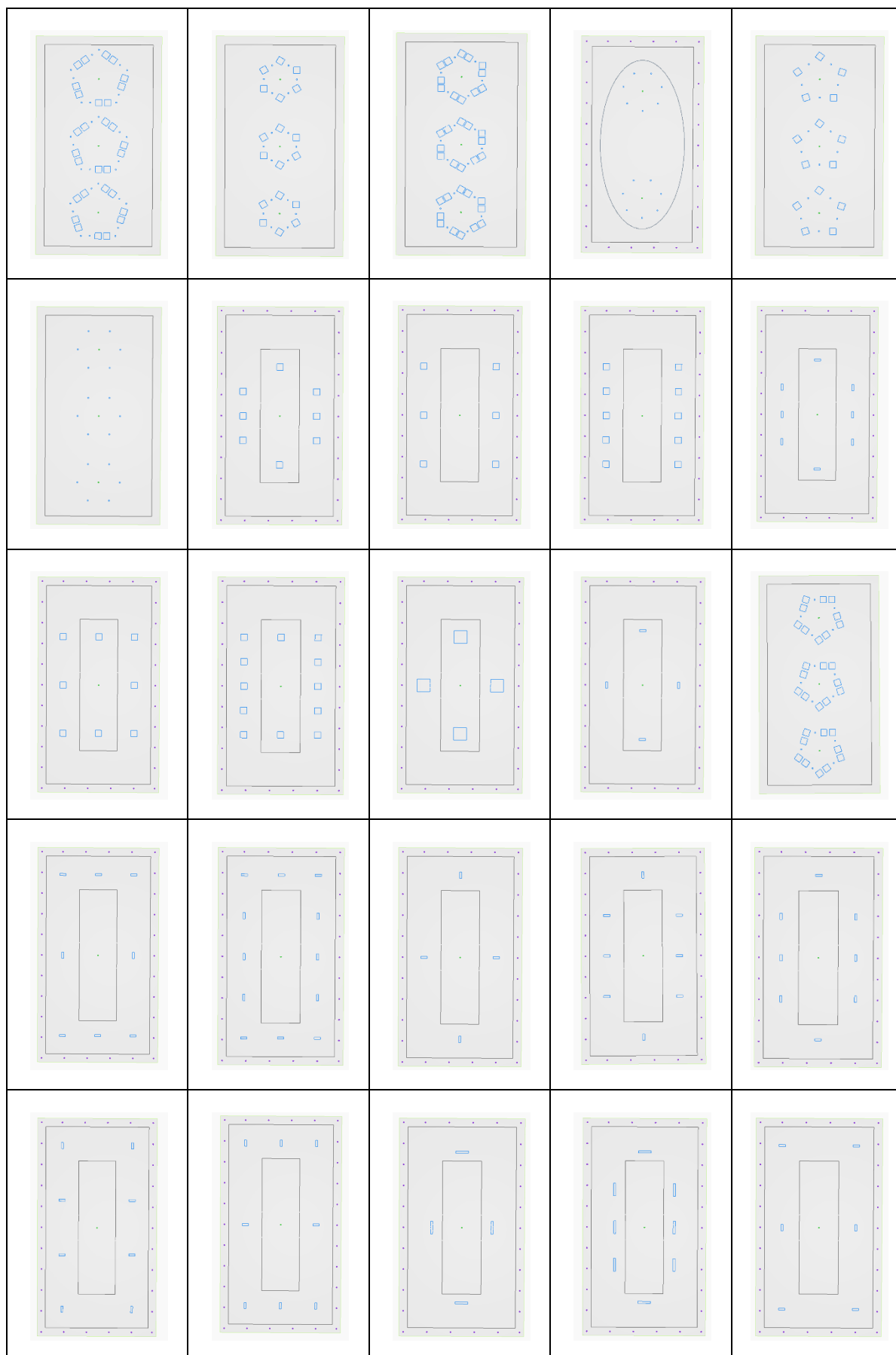
Table 14: Office A202 produced options from the generated ceilings designs from Dynamo 2.0.3 (author)

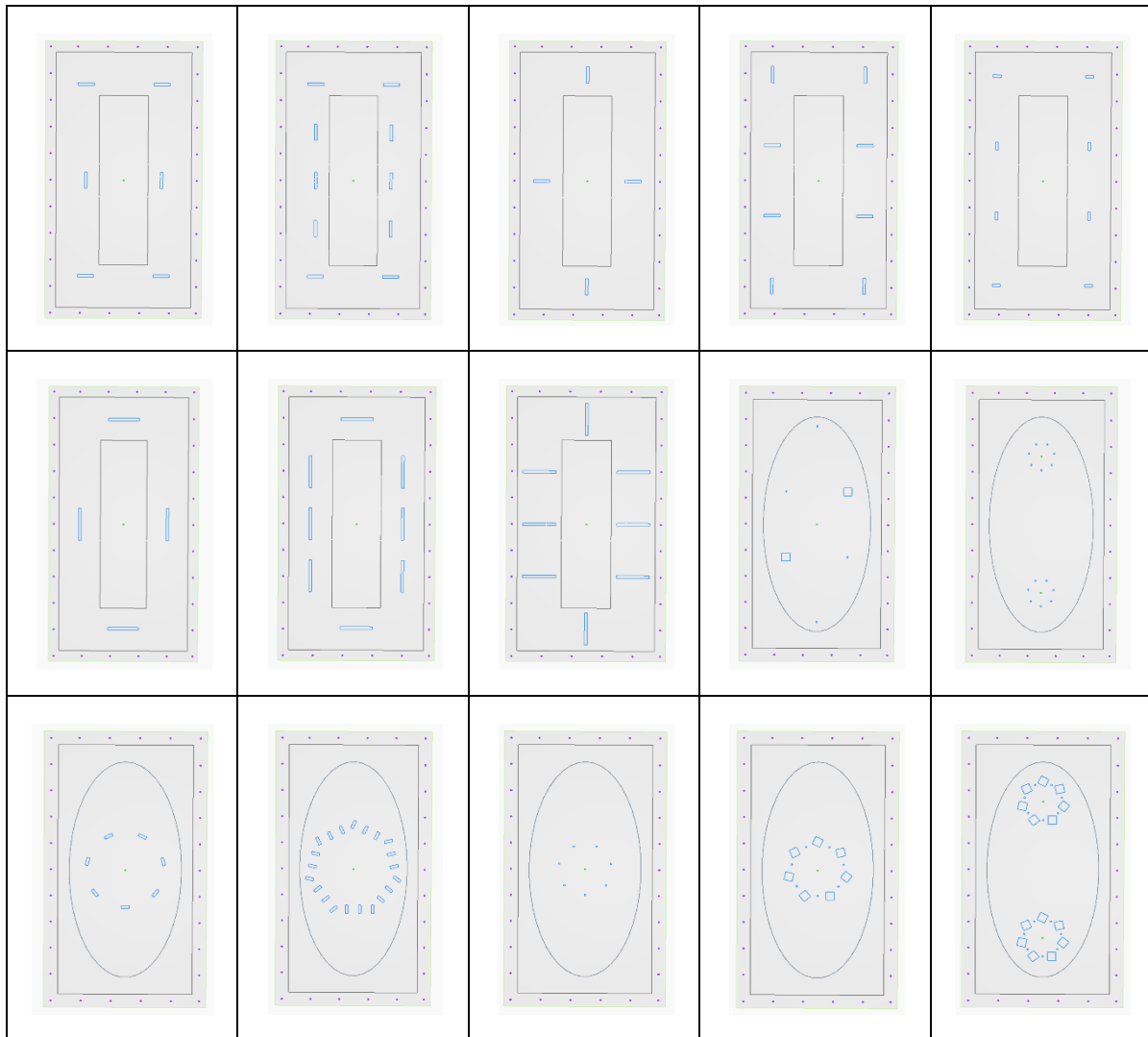
Generated Ceiling Designs				
				
				
				
				











As per the previous case studies outputs, the generated results insure the variety and the elasticity of the algorithm in a small amount of time. Running the algorithm only took one minute to produce the showcased results.

Figure 133 shows the same pattern applied using different methods, this flexibility is what generates endless amounts of reflective ceiling designs. The created pattern is a mixture of 300x300mm square lights, and points that can indicate spotlights or suspended lights from the ceiling. They are arranged in a circular outline to produce the displayed pattern.

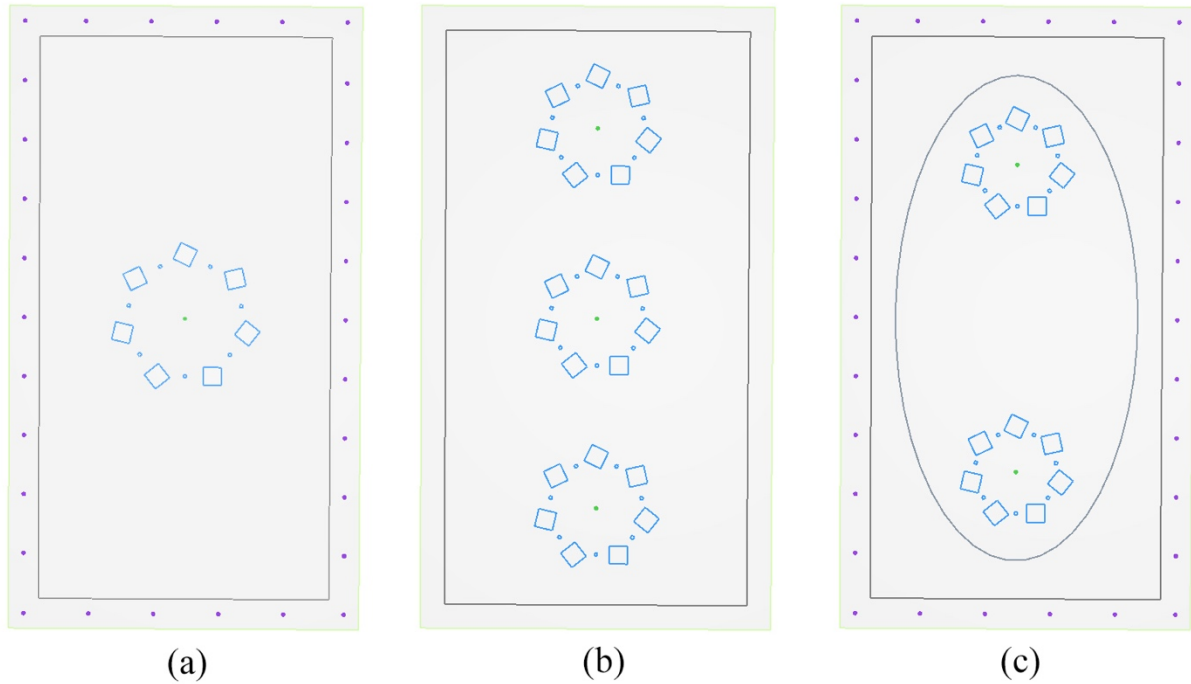


Figure 133: Same pattern merged with different pattern generated in Dynamo 2.0.3 for Office A202 (author)

(a) in Figure 133 shows the pattern in the center of the room with a simple boundary lighting around the ceiling. While (b) shows the same pattern multiplied 3 times.

(c) is showing the same pattern repeated twice and merged with a drop-down ellipse ceiling Figure 133. Although the pattern radius had changed in the three cases, the dimensions of the light remains the same. The three options can fit the tested meeting room with no issues.

Figure 134 shows different results of arranging 300x300mm lights in different methods. These results were produced automatically without the need for the designer to change any variables as they are based solely on division. The produced ceilings show diversity in arrangement and in quantities of lights.

The 300x300mm can be changed into any suitable dimensions that the end user desire. The algorithm is scripted in an approach that allows all these patterns to revert to Revit and apply the lights on the actual ceiling in the model.

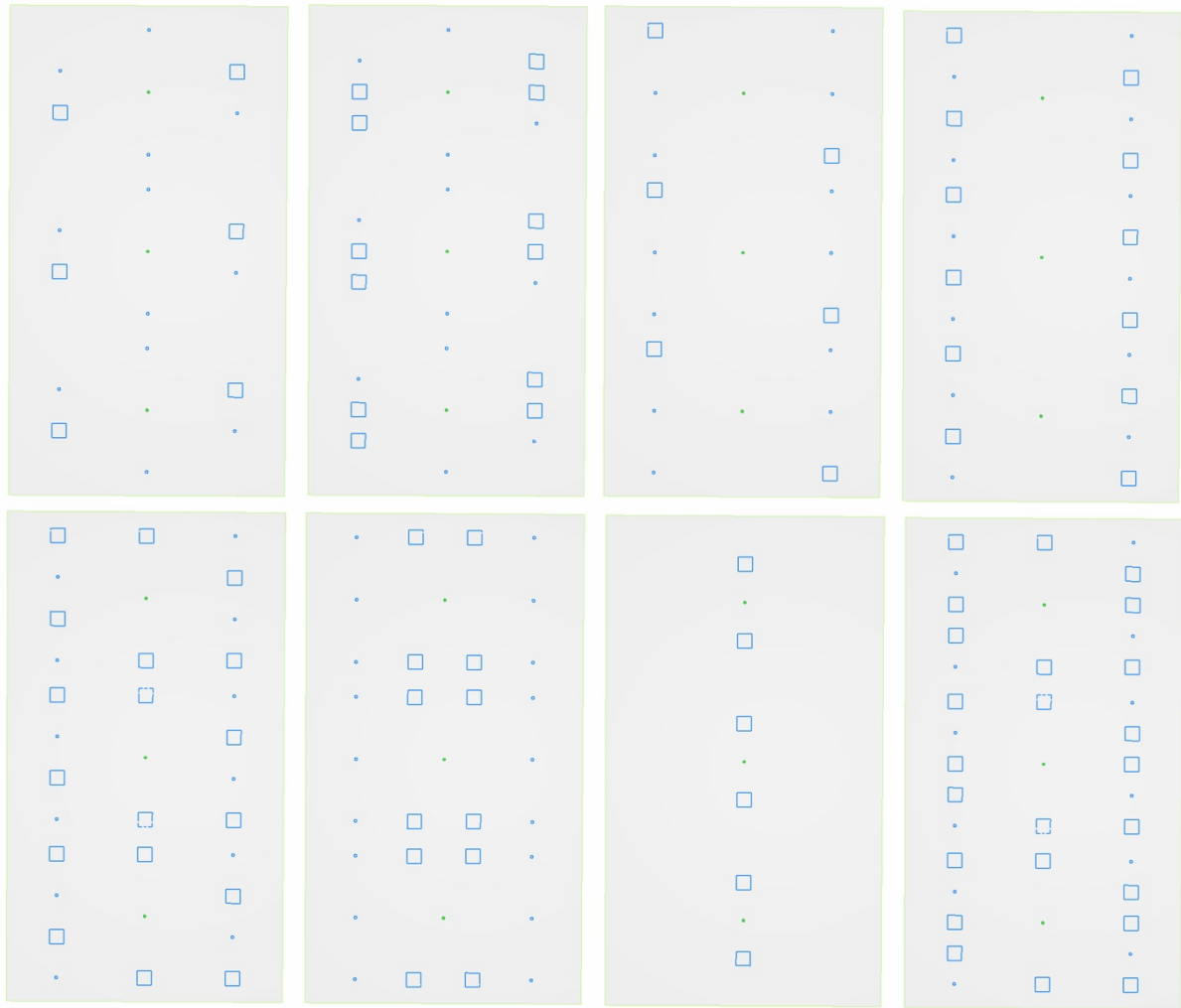


Figure 134: Different types of designs using the same light generated in Dynamo 2.0.3 for Office A202 (author)

Figure 135 shows with arrows how the mathematical division happens in the algorithm resulting in creating all the possibilities of the same pattern but with different quantities. Where the horizontal arrow shows how the same pattern develop vertically, and the vertical arrow displays how the same pattern develops horizontally. Every pattern will only be divided 3 times only. The initial scripted algorithm did not have the 3 times limit, where every limit would depend on the pattern scale itself. In other words, the pattern will keep dividing itself as long as the lights fit along the shape and did not overlap. But this approach had produced over 4000 patterns, which is a large quantity of patterns that makes it inefficient for the designer to go through. Hence, the limitation of 3 times were added as seen in Figure 135.

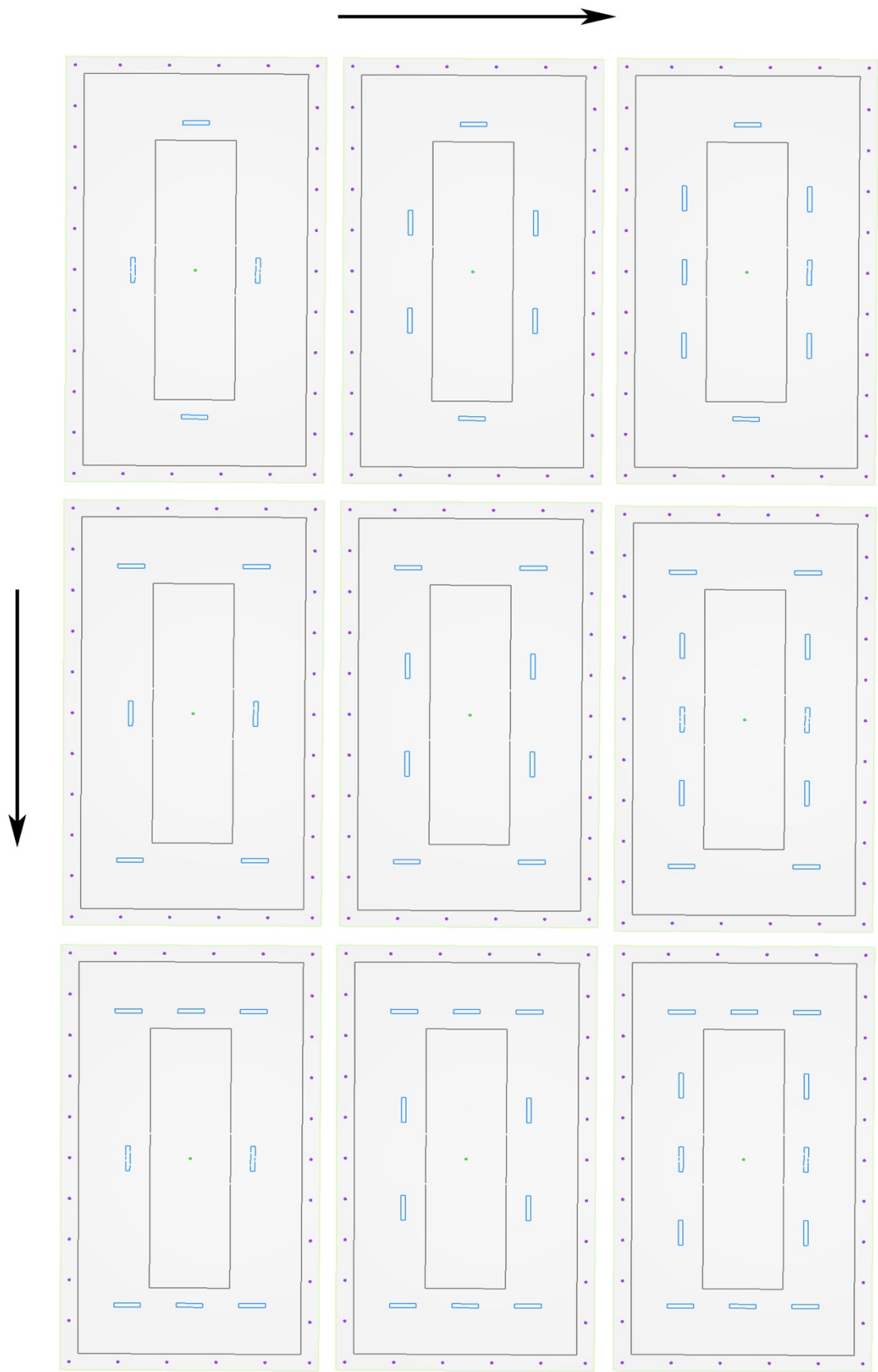


Figure 135: Division / combination example for both directions generated in Dynamo 2.0.3 for Office A202 (author)

Table 15 highlights the advantages and the limitations found in the two field case studies that were conducted on two different offices in Dubai Design District area.

*Table 15: Office A101 and Office A202 limitations and advantages (author)*

Field case study - Office A101& A202	
Limitations	Advantages
None were found in this case study	1- Producing over 100 different ceiling designs in 2-3 minutes for both case studies.
	2- The algorithm provided options with lighting fixtures quantities
	3- Produced numerous options in designing light patterns
	4- The algorithm showed its capacity as a functional interactive system between the designer and producing designs.

### 5.2.3 Summary of both conceptual and field case studies

Testing the algorithm through two different aspects: conceptual case studies and field case studies was essential to evaluate the efficiency of it and document the generated results. This evaluation had revealed the limitations that need to be investigated further to enhance the algorithm's capacity. To make it a comprehensive system that works with every ceiling condition that might occur.

During the scripting process of the algorithm, a lot of variables and cases were considered, and the system was contentiously tested to identify errors and fix them. However, most of the testing at that time the was on different dimensional shapes, to ensure that the algorithm is able to take all the possible ceiling layouts, and work effectively on the majority of shapes that forms any ceiling. Small conditions like having a curved corner in parts of the layout or the orientation of the model was not considered in the scripting process.



The conceptual case studies had exposed new limitations the algorithm did not inspect. Providing proper feedback that aims to develop the system and upgrade it. Unfortunately, the field case studies did not reveal new limitations due to the simplicity of the ceiling shapes, as these shapes were considered in the scripting process of the algorithm.

The advantaged were similar in the conducted case studies; both conceptual and field case studies proved the algorithm's capacity in four main parameters:

- 1- Generating many ceiling designs in a short time; 2 to 3 minutes.
- 2- Providing diversity in the generated designs in three design aspects: light patterns, used design approaches and lighting fixtures quantities
- 3- Showcased its ability as an interactive system in two aspects: the designer's ability to alter and fix the visual results immediately, and the algorithm capacity to turn on and off different approaches that allows the designer to pick and choose the suitable designs he/she desires.
- 4- The capacity of the algorithm to produce various ceiling layouts using one design parameter only. And produce layouts that combine the different designs parameters in one ceiling at the same time.

The major limitation the case studies had revealed, which needed proper investigation to identify the reason behind it; was the orientation of the model in Revit. This limitation emerged in the second conceptual case study that investigated Commerzbank headquarters. Which showed the connection between Dynamo and Autodesk Revit, where the model's orientation in Revit has a direct impact on all the lighting methodologies in Dynamo. Forcing all the lighting methods to match the same orientation as the building in Autodesk Revit regardless of the ceiling's orientation in Dynamo. This can be considered as an advantage or a disadvantage. In this case study, this orientation had produced creative results, while in other case studies it

might produce inefficient results. Therefore, the orientation of the model always needs to be considered.

### 5.3 Phase 02: Connecting the results to Autodesk Revit

The next phase of testing the algorithm is through connecting the results back to Autodesk Revit and running lighting calculations on the selected designs. This phase will be applied using the fourth field case study: Office A202, located in Building 5, Dubai Design District.

#### 5.3.1 Exporting the lights from Dynamo and connecting it to Revit light families

Section *Subsystem 03: Exporting the results back to Autodesk Revit* in the previous chapter had discussed in detail how the algorithm exports back the results from Dynamo into Revit. Which is done through grouping every light category into different groups, to connect the produced lights in Dynamo to actual light families found in Revit.

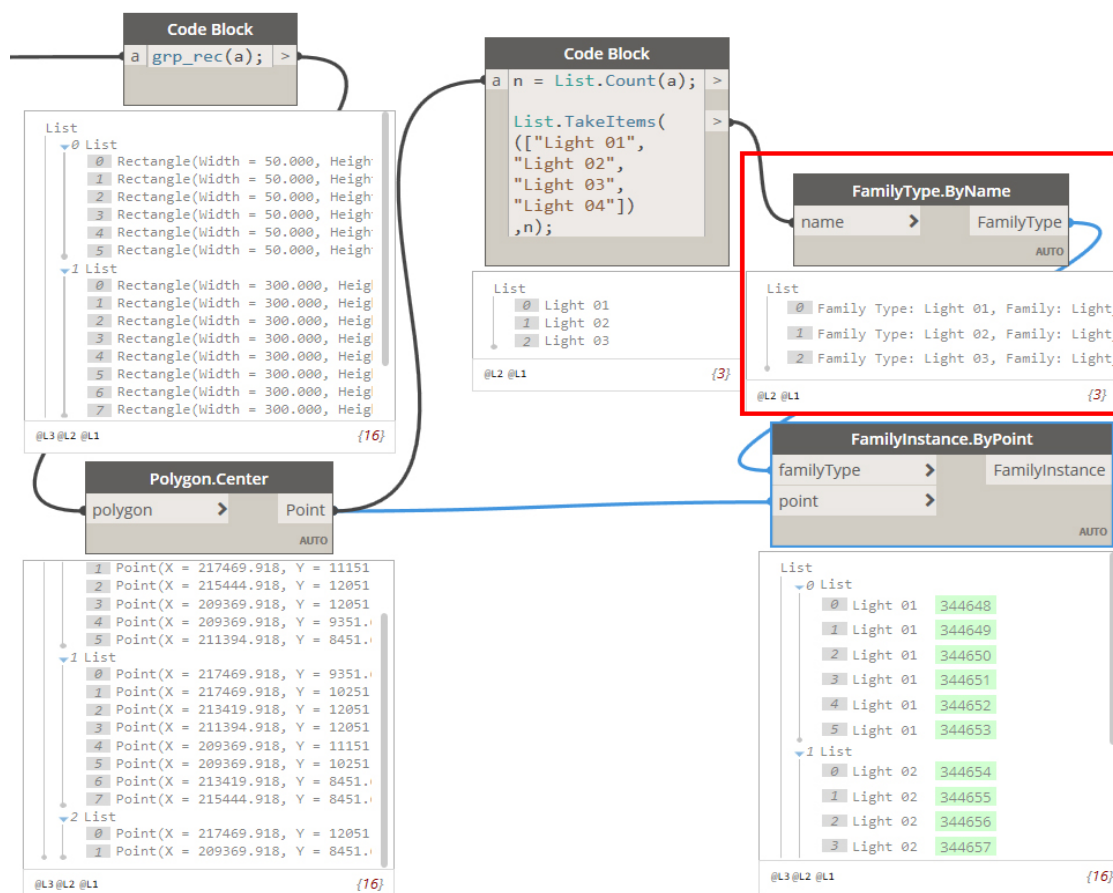


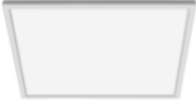



Figure 136: Nodes that creates light lists to connect to Revit in Dynamo 2.0.3 (author)

Figure 136 shows the Nodes that create the connection between Dynamo and Revit. Where the grouped lighting lists in Dynamo are connect to the created light families in Revit that's seen in the red outlined list.

The light families in Autodesk Revit where obtained from Acuity Brands Lighting, Inc. The company is in Atlanta, Georgia (Acuity Brands Lighting 2020).

The chosen lights are seen in Table 16; which have the same dimensions as the inserted light dimensions in Dynamo that generated all the different ceilings designs. The company provides in its website (IES. Lighting formats) that can be connected to Revit lighting families immediately.

Table 16: The selected lights from Acuity Brands Lighting (author)

Lighting Source: Acuity Brands Lighting, Inc						
Light Series	Image	Width and Length	CRI	Color Temperature	Lumens	Wattage
Epanl LED Flat Panel		600x600mm	80	35k	2000	17
JSFQ Slim Form Surface Mount Downlight Square		300x300mm	90	35k	1300	15
Wf6 Thin LED Downlight		Dia:50mm	90	30k	Low Lumen	10
Slot 1 Recessed		600x75mm	90	30k	1000	37.82
		1200x75mm	90	35k	1000	37.82

### 5.3.2 Lighting calculations in Autodesk Revit

Once the algorithm connects the produced lights in Dynamo to a light family in Autodesk Revit, the results of any selected ceiling will automatically show on the original model in Autodesk Revit. In this case; the Revit lighting families shown in Table 16 were downloaded from Acuity Brands Lighting website in an IES format that allowed Revit to connect them to the same families identified in Dynamo.

For the lighting calculations, two ceilings were chosen from the fourth field case study: Office A202. As seen in Figure 137 (a) shows another generated ceiling using three different lights families: Epanl-LED Flat Panel, JSFQ -Slim Form Surface Mount Downlight Square and Wf6-Thin LED Downlight. While (b) shows a generated result using only one light family, which is Epanl-LED Flat Panel.

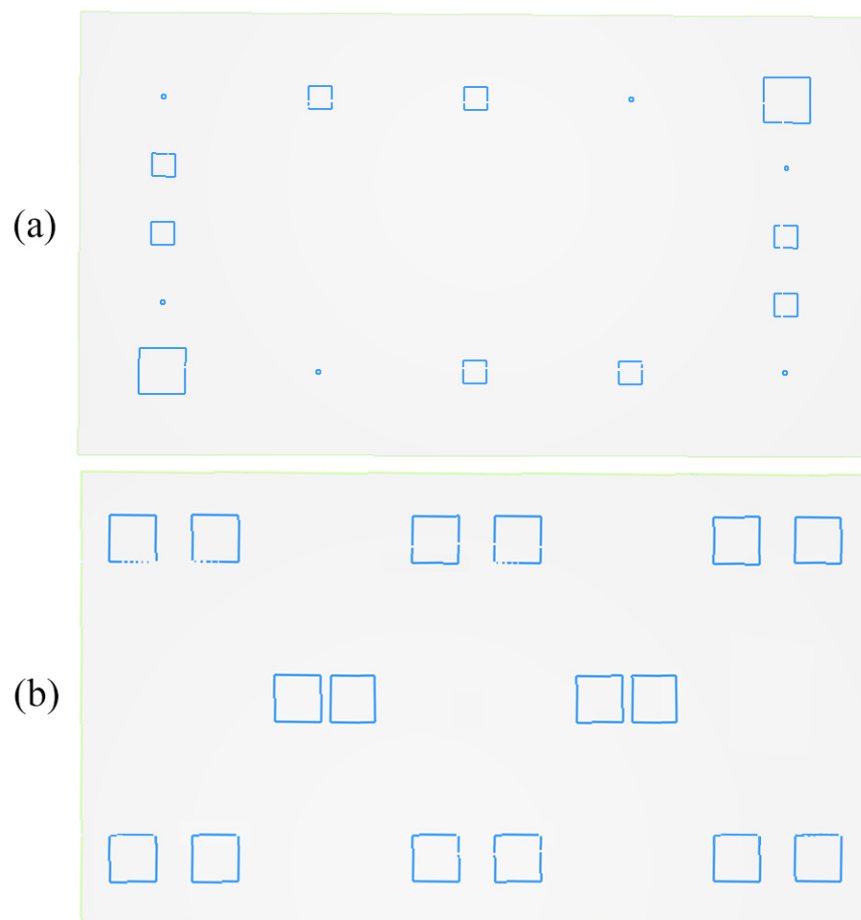


Figure 137: The chosen ceilings for the lighting calculations generated in Dynamo 2.0.3 (author)

In addition to producing numerous lighting designs using one or multiple lighting families. The algorithm also provides the accurate counting for any selected ceiling. Figure 138 shows the counting for ceiling (a) and (c) in the red outlined area.

Although the other methods are turned off; the counting for all the lighting methods will show as seen in the Nodes in Figure 138, if the end user turned them on; the results will match the showed counting. The details about the counting method script is discussed in detail *Subsystem 02: Counting the generated light quantities* in the previous chapter.

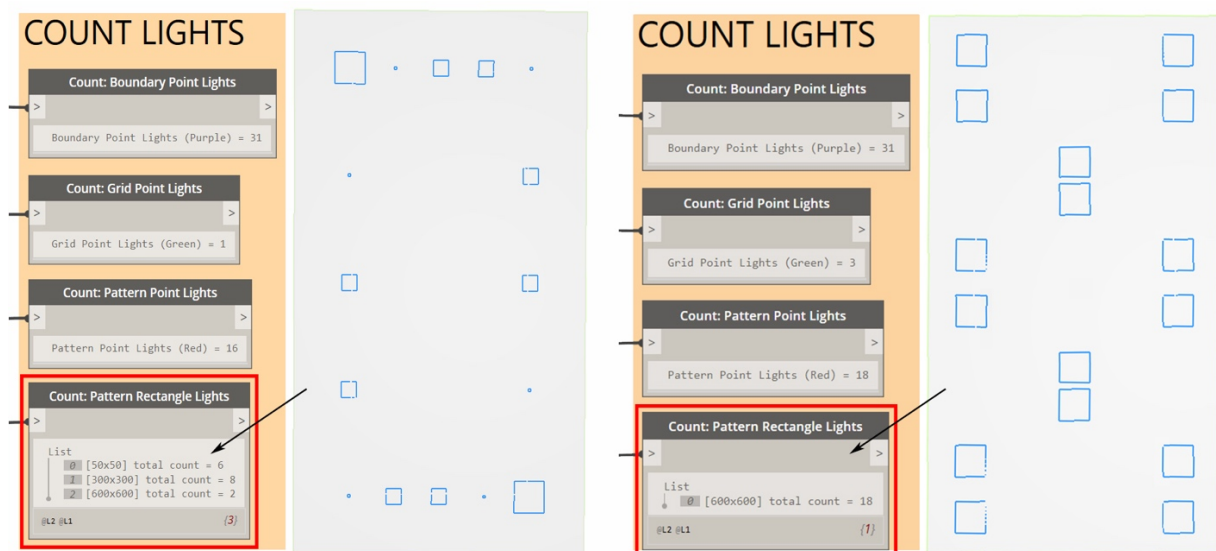


Figure 138: The counting for ceiling (a) and for ceiling (b) from Dynamo 2.0.3 (author)

To run the lighting calculations; an integrated lighting calculation plug-in was downloaded into Auto desk Revit. ElumTools add-in can calculate the luminance on any surface in Revit through identifying the lighting fixture families (Lighting Analysts 2020).

This plug-in was chosen due to its ability to provide the desired lighting calculations. In addition to having the flexibility to insert the selected lighting families shown in Table 16. Which were connected to Dynamo and Autodesk Revit previously.

Once the tool is downloaded, its own ribbon toolbar will appear as displayed in Figure 139 for the end user to access the different analysis tools.

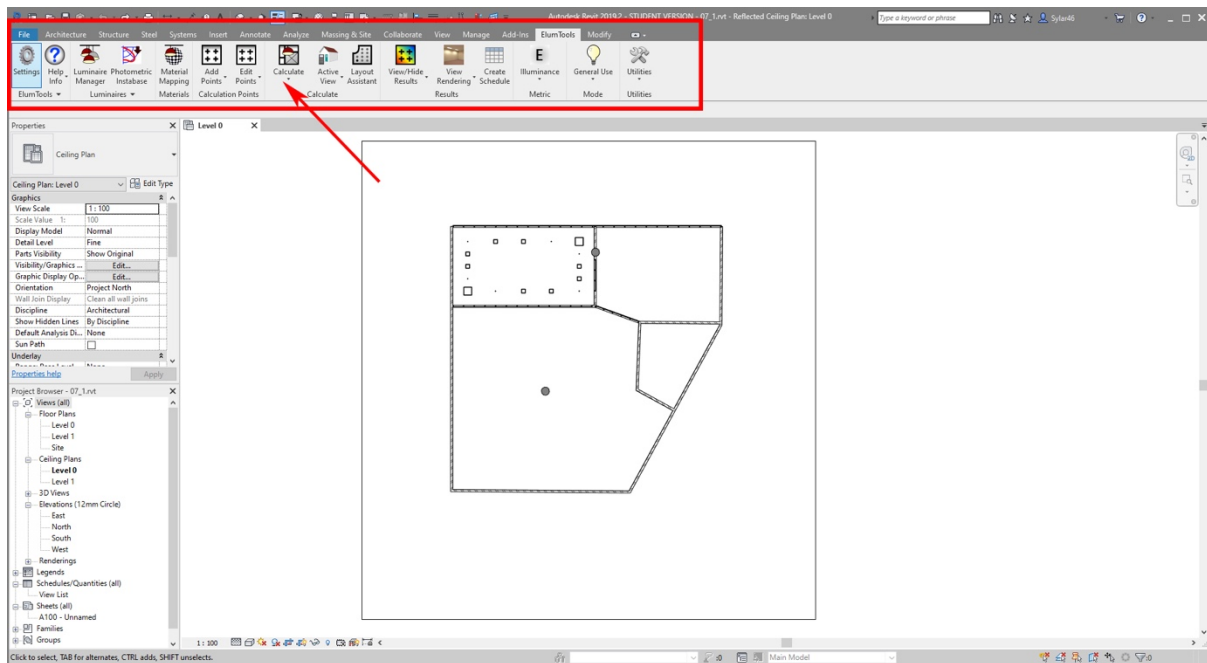


Figure 139: ElumTools Plug-in in Revit Workspace (Autodesk Revit 2018)

### 5.3.2.1 Lighting calculations for ceiling (a)

Figure 140 shows the generated design for ceiling (a) in Dynamo and the immediate results in Autodesk Revit. Since the algorithm scripts in Dynamo connects the generated lights to Revit lighting families; the results of any selected ceiling will automatically show on the original model in Autodesk Revit.

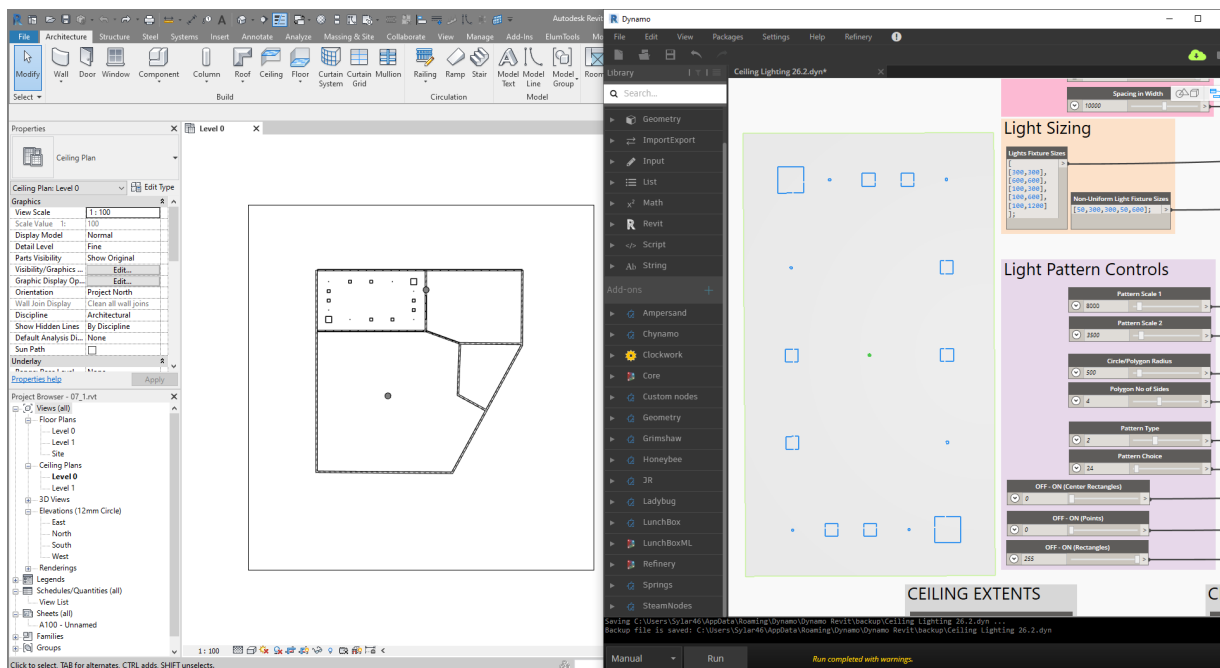


Figure 140: Dynamo produced results for ceiling (a) and application of them in Revit workspace

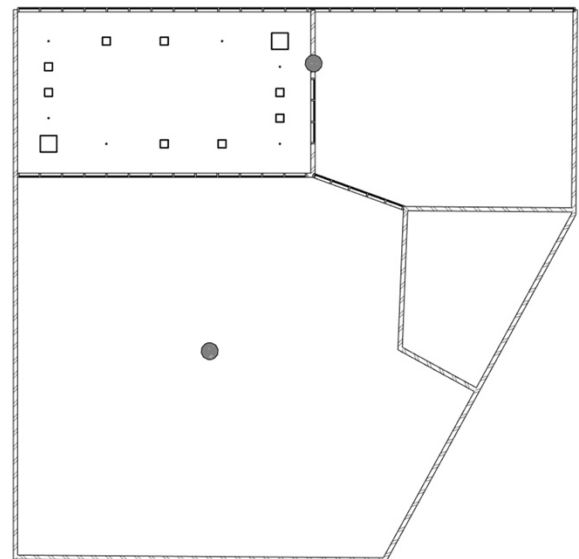
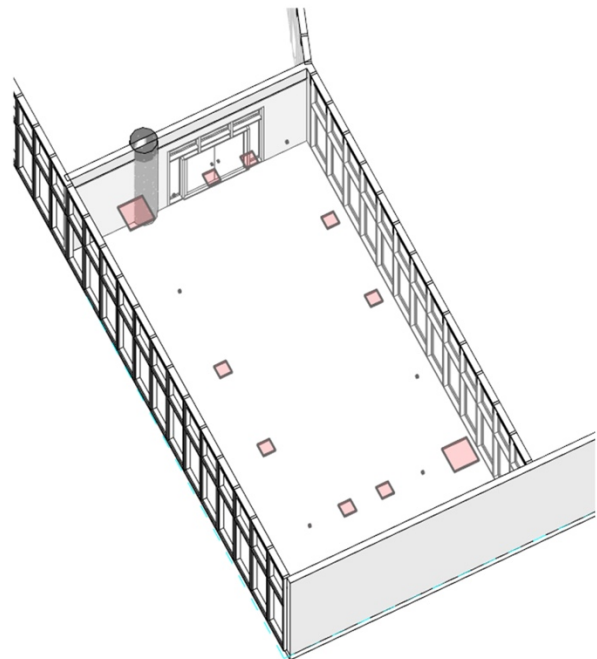
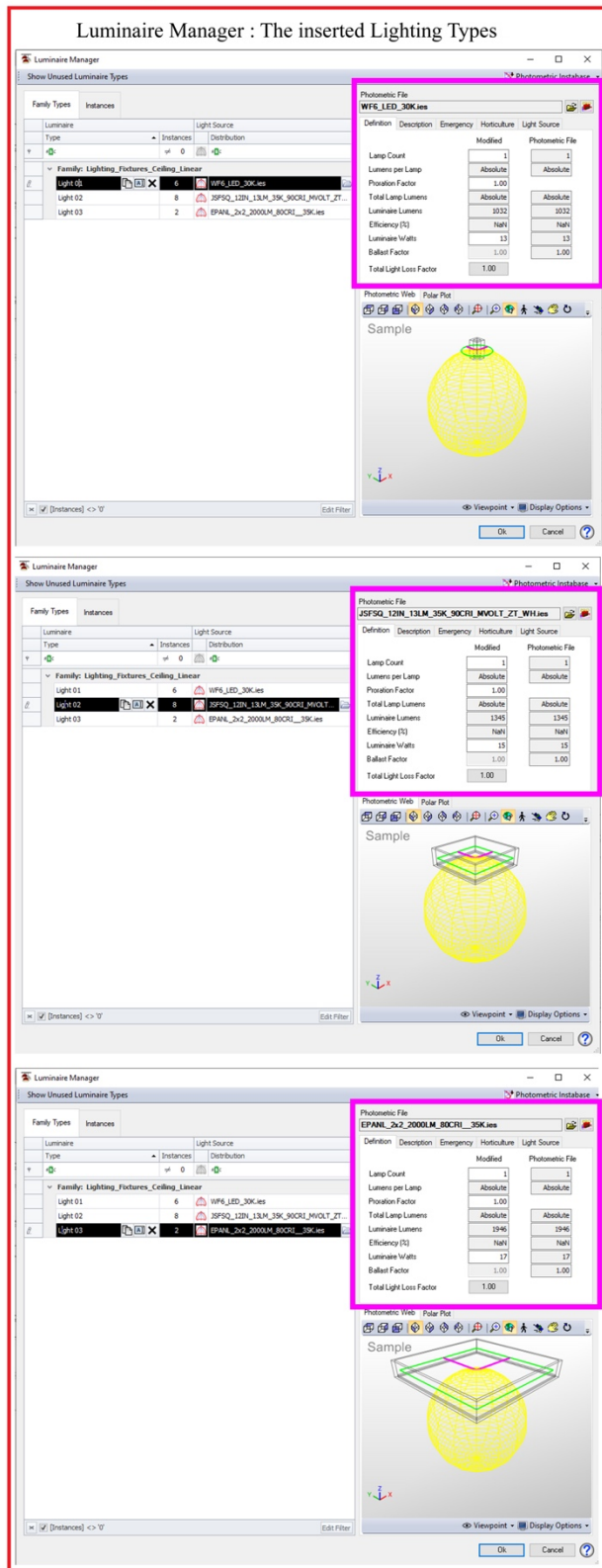
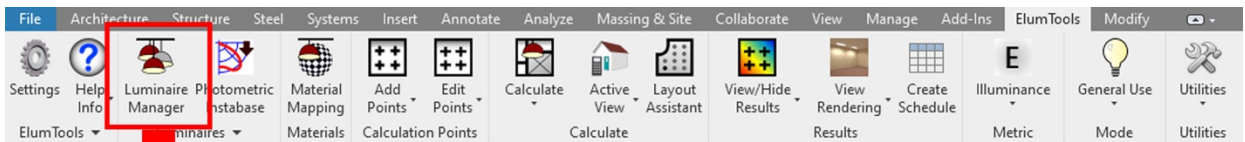


Figure 141: The inserted lights and the Revit model with the layout model (Autodesk Revit 2018)

Figure 141 is showing the Luminaire Manager tool in the ribbon toolbar along with the three light families presented under that tool. Although Table 16 shows 5 different types of lights, 3 are only connected in this case study since the produced ceiling layout is using three different light families only as seen in the shown Revit model in Figure 141.

The Luminaire Manager tool provides the standard specifications for every light family used in Revit, which is outlined in purple in Figure 141. This gives the flexibility for the designer to always revert to this list to turn on/turn off certain lights or change the luminaire Watts if desired.

Since the displayed light families in Figure 141 are downloaded as an IES format from Acuity Brands Lighting, Inc website; there was no need to alter nor change any light specifications to match the standards of the existing light that's being sold from the company. After uploading the light families in the Luminaire Manager tool, the calculation points were added using Add Points Tool shown in Figure 142. This step allows the designer to choose the desired quantity and space between every calculating point to produce lighting calculations results based on these points.

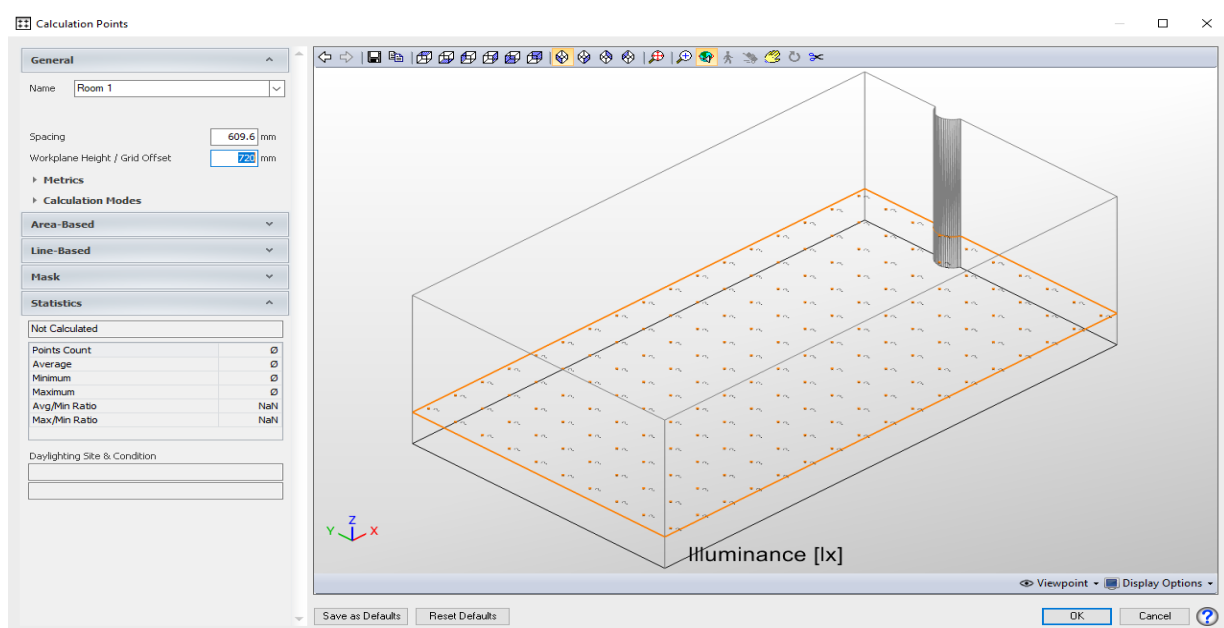
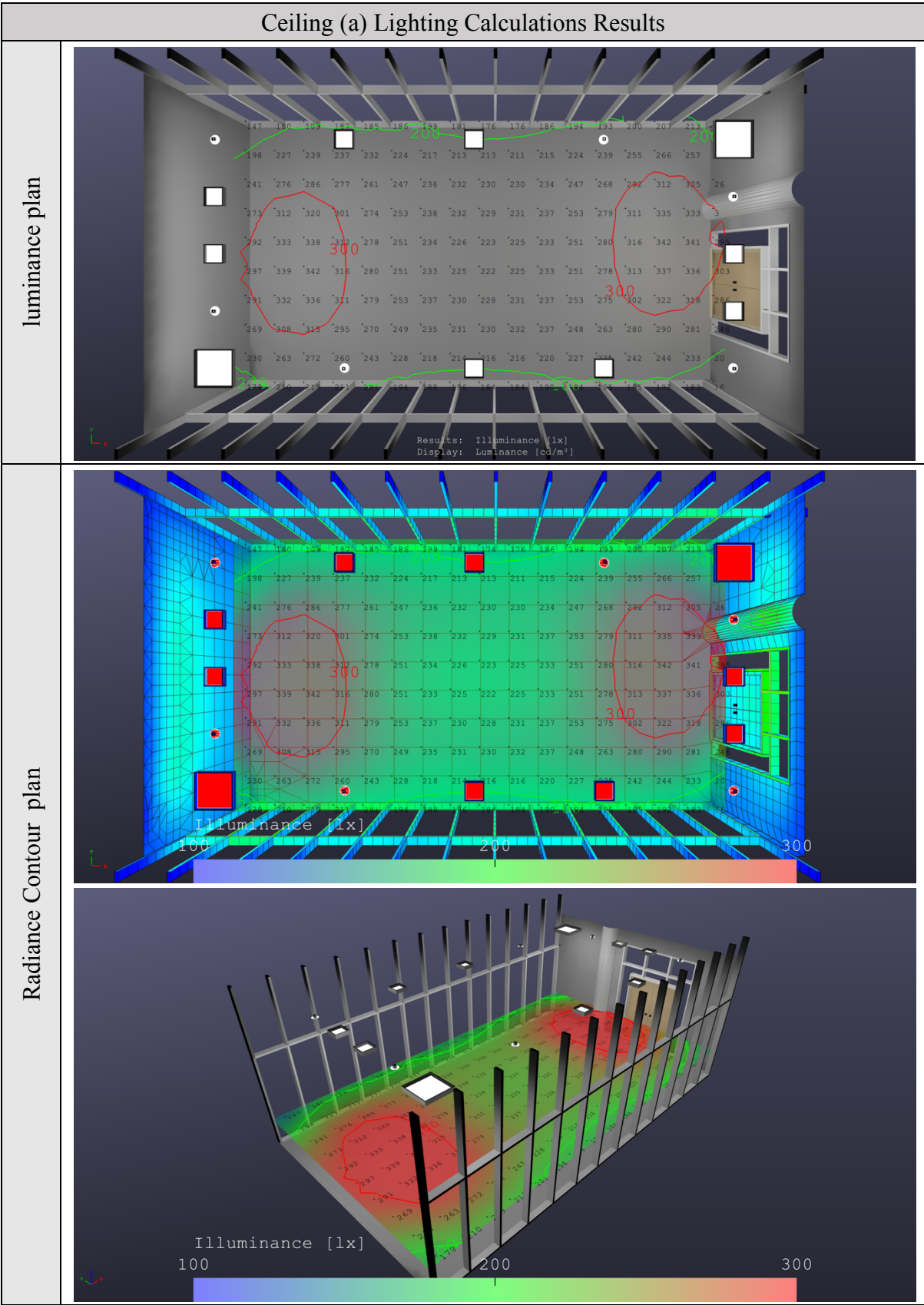


Figure 142: Adding Calculations Points (Autodesk Revit 2018)



After running the lighting calculations, Table 17 displays the produced results from ceiling (a).

Table 17: Ceiling (a) Lighting Calculations Results (Autodesk Revit 2018)



As seen in Table 17; the highest luminance levels were noticed on the sides on the room, with a maximum luminance of 342 lx, this is due to the 5 lights placed close to each other on the sides of the room.

However, if the designer wishes to change the spacing between the lights or reduce the light quantity to minimize the lx levels. He/she will simply alter the dimensions through changing the value in the slider Pattern Scale, which will automatically alter the space of the lights in Revit. The Pattern Scale slider is presented in Figure 140. This shows the flexibility of the scripted system as any alteration in Dynamo will automatically alter the light families in Autodesk Revit and can be modified even after choosing the pattern and running lighting calculations without presenting any glitches or errors.

Another method to minimize the lx levels is through changing the luminaire Watts for the light family - outlined in purple in Figure 141- to match another luminaire produced by the supplier in order to reduce lower levels of Watts.

In addition, the minimum luminance is noticed in top left of the room with 147 lx which is illuminated by Wf6-Thin LED Downlight. While results showed 248 lx as the luminance average of the room. The luminance results are summarized in Table 18.

*Table 18: Ceiling (a) luminance Generated Results from Autodesk Revit 2018 (author)*

luminance Generated Results					
luminance Calculation Points Name	luminance Average	luminance Maximum	luminance Minimum	luminance (Avg/Min)	luminance (Max/Min)
Ceiling (a)	248 lx	342 lx	147 lx	1.7	2.3

### 5.3.2.2 Lighting calculations for ceiling (b)

Ceiling (b) is generated using the scripted algorithm in Dynamo as seen in Figure 143. This ceiling was chosen because it only utilizes one light family in Autodesk Revit, while ceiling (a) had utilized three different lighting families. Showcasing the diversity of generated options the algorithmic system can produce.

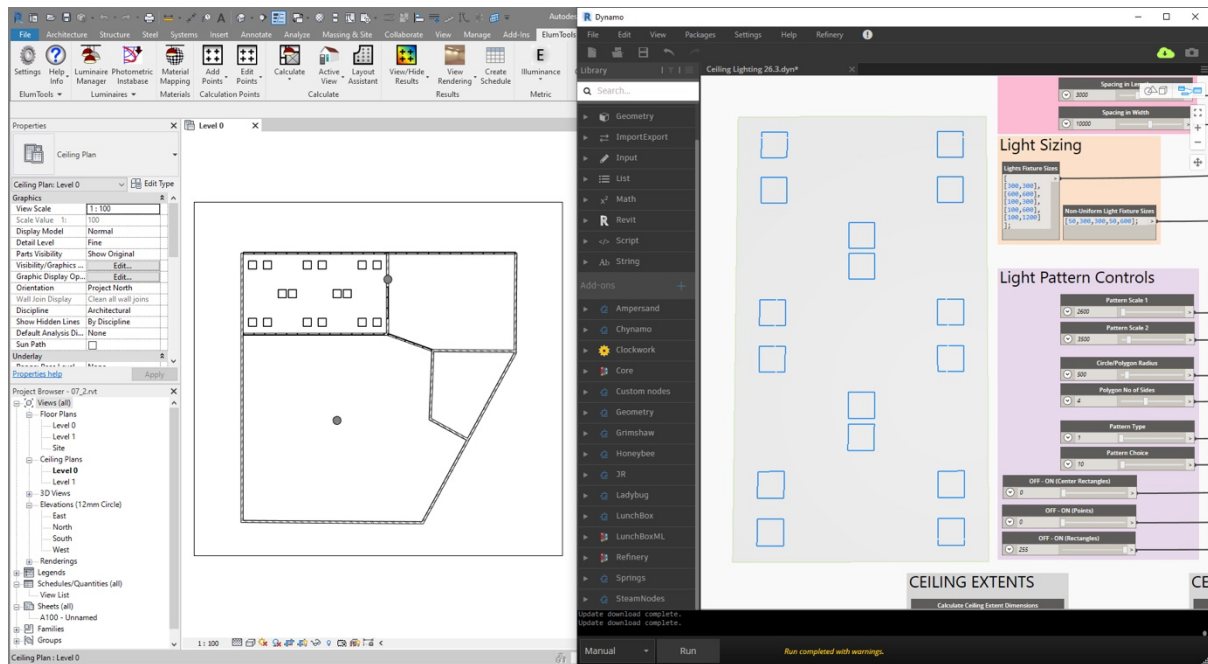


Figure 143: Dynamo produced results for ceiling (b) and application of them in Revit workspace

The next step is to identify the selected light family in ElumTools plug-in using the Luminaire Manager tool. Figure 144 shows the model in Autodesk Revit and the selected lighting family for the design, which is downloaded with its specifications from Acuity Brands Lighting, Inc. The selected light family is Epanl-LED Flat Panel.

The designer is always able to change the light family to any desired option. As long as the naming of the new light families is altered to match the original naming for the previous light families, which are: Light 01, Light 02, and up to Light 05.

If the naming of the light families is changed by the end user in Autodesk Revit. Dynamo will not be able to connect the produced results to the light families in Autodesk Revit, which will

lead in an error in displaying the results in Autodesk Revit workspace. This issue occurs because the algorithm is creating the connection through the similarities of names in both software. For example: the algorithm understands that group A in Dynamo is connected to Light 01 family in Autodesk Revit. If Light 01 family in Autodesk Revit was removed, the connection will fail. The Nodes displayed earlier in Figure 136 shows the connection clearly where the red outline shows the names of the family types in Dynamo.

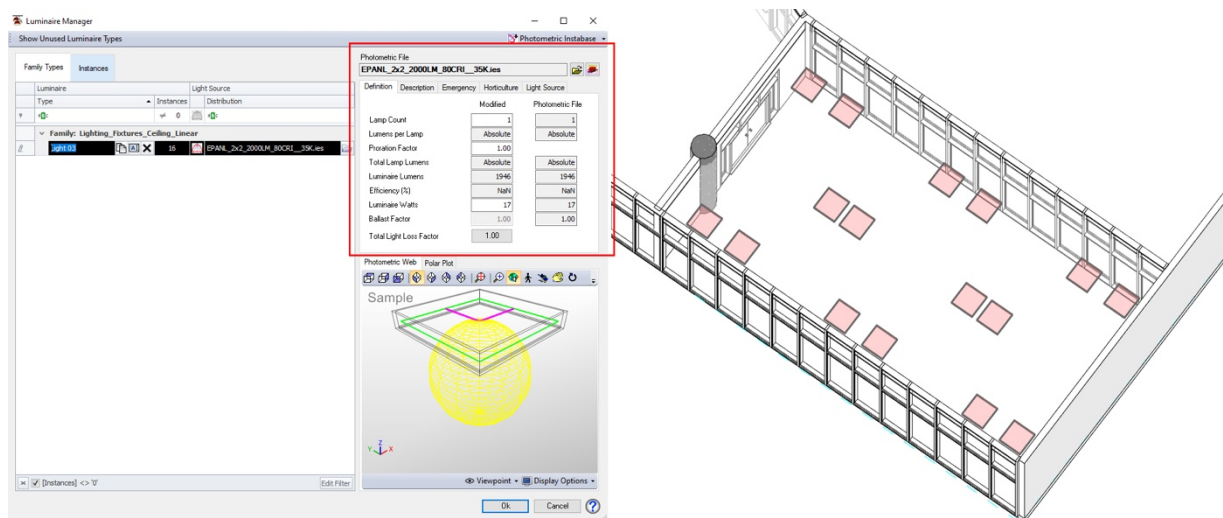


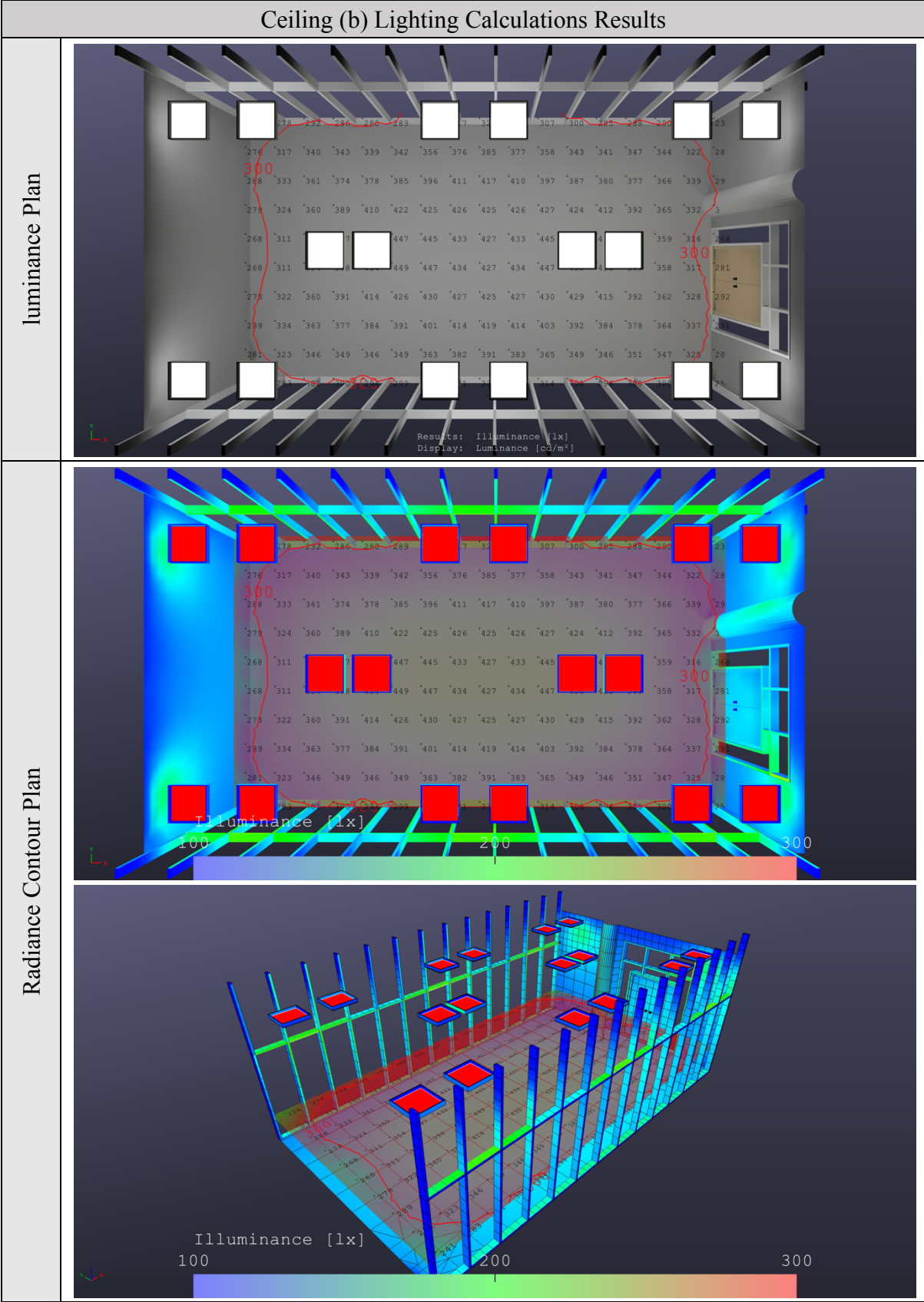
Figure 144: The light specifications for the produced ceiling (Autodesk Revit 2018)

The same procedure is repeated, where the next step is to select the calculation points and place them in the room with the desired height, for the lighting calculations to use these points as a base level to perform the lighting calculations and display the results. In both ceiling (a) and (b) the calculation points were set on a height of 720mm, matching a meeting table height.

After that, the lighting calculations are conducted to investigate the efficiency of the produced ceiling. Table 19 shows the produced results from using ElumTools plug-in.

Results showed great difference between both ceilings. Since (a) had a total of 16 lights that diversified in lumens, while ceiling (b) used 18 lights with the same lumens. The difference of lighting lumens and quantities is between both ceilings had created 30.5% difference in the luminance average.

Table 19: Ceiling (b) Lighting Calculations Results (Autodesk Revit 2018)



The produced values show a maximum luminance level of 450 lx. The highest luminance levels were found in the centered area of the room due to the minor spacing between every two lights in the middle space. While the luminance minimum had a value of 228 lx, emerged in the left top corner of the room. The luminance average of the room was 357 lx, which is notably higher than ceiling (a). The generated results are summarized in Table 20.

*Table 20: Ceiling (b) luminance Generated Results from Autodesk Revit 2018 (author)*

luminance Generated Results					
luminance Calculation Points Name	luminance Average	luminance Maximum	luminance Minimum	luminance (Avg/Min)	luminance (Max/Min)
Ceiling (a)	357 lx	450 lx	228 lx	1.6	2

### 5.3.3 Phase 02 out-turns

Figure 145 presents a visual comparison between both cases, where ceiling (b) had produced higher luminance levels than ceiling (a). Ceiling (b) showed an increase of 30.5% for the luminance average. And a rise of 24% for luminance maximum values, and 35.5% increase in the luminance minimum value.

These results show that the algorithm creates diverse options for the designer to use from, using one light family to generate ceilings that are mathematically produced to ensure every potential arrangement for the lights. Or producing ceilings that utilize every light family to create all the design possibilities. In other words, the algorithm starts producing all the possibilities using one light only, then develops to generate all the potential options using two lights, followed by three lights and so on. Until it reaches into using 5 different types of lights in one ceiling.



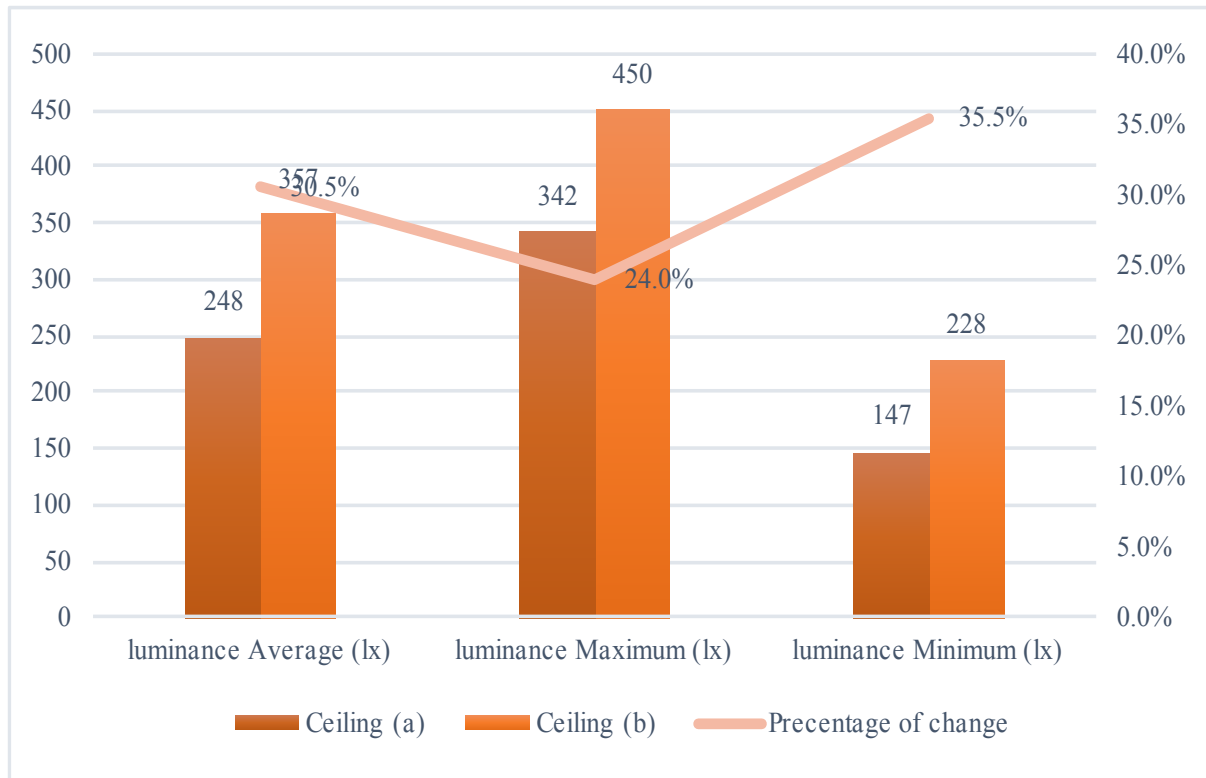


Figure 145: Ceiling (a) and ceiling (b) comparison (author)

The results also display the capacity of the scripted generative system; as it is able to take the produced designs from Dynamo to a further step after, and applies them automatically in Autodesk Revit, as well as connecting the produced outputs to actual light families taken from lighting suppliers. Producing accurate results that can predict the luminance levels in any tested area.

In addition, different levels of flexibilities in the generative system are shown through the different phases. Lighting calculations had shown the given flexibility to the designer regarding choosing the desired lighting families with different light specifications. As the light specifications are not fixed and can be altered while sustaining the same family or changed completely through adding a new lighting family that has an IES format. In addition, the actual design of the ceiling can be altered, manipulated, or changed completely in Dynamo and it will automatically reflect the new design in Autodesk Revit.

### 5.3.4 Lighting power density calculations

Litrature had showed previous generative systems that were designs soly for performing lighting calculations, such as (Plebe & Pavone 2017) and (Seyedolhosseini et al. 2020). Lighting power density (LPD) was calculated manually in this dissertation, since selecting light families is assigned in Autodesk Revit and not in Dynamo. If the lighting power density is calculated in Dynamo, the flexibility of selecting and changing the light families in Autodesk Revit will not be possible. As the Wattage of every inserted light will be detrmained in Dynamo to perform the calculations and not in Autodesk Revit.

The light counts shown earlier in Figure 138 highlights the sizes of each used family which is connected to the inserted light families shown in Table 16. The total area of the selected room is 61m<sup>2</sup> as showcased previously in Figure 131, which is around 657 square foot (sF).

Ceiling (a) had three different types of lights: Epanl-LED Flat Panel, JSFQ -Slim Form Surface Mount Downlight Square and Wf6-Thin LED Downlight. While Ceiling (b) was generated using only one light family, which is Epanl-LED Flat Panel. Table 21 exhibits the details of every light. Along with the Lighting Power Density for each ceiling. Where LPD for ceiling (a) upon calculating it is 0.54 W/sF, while LPD for ceiling (b) is 0.47 W/sF, as  $(17 \text{ Watts} \times 18 \text{ pieces}) / 657 \text{ SF} = 0.47 \text{ W/sF}$ .

Table 21: Lighting fixtures details for Ceiling (a) and (b) and LPD calculations (author)

	Fixture Description	Watts Per Fixture	Quantity	Total Fixture Wattage	Lighting Power Density (W/sF)
Ceiling (a)	Epanl-LED Flat Panel	17	2	359	0.54
	JSFQ -Slim Form Surface Mount Downlight Square	15	15		
	Wf6-Thin LED Downlight	10	10		
Ceiling (b)	Epanl-LED Flat Panel	17	18	306	0.47



#### 5.4 Phase 03: Sustainability elements of the proposed generative design sysetm

(Cohen & Reich 2017) had proposed an appropriate framework that analyzes sustainability in biological systems. They have proved that this framework is applicable for general design as well. Furthermore; the book they've wrote provided a checklist to be implemented as a design principal that fosters economical, enviromental, social and ethical sustanability.

This framework was created through connecting ideality to sustainability. The connection was created through the similarities of definitions, where (Cohen & Reich 2017) affirmed Charter and Tischner definition of sustanibility as all products, services, hybirds and systems that decrease the negative effect and increase the positivie effect of the four key parameters of sustainability - economical, enviromental, social and ethical - (Charter & Tischner 2001). And connected the definition to the concept of ideality defined by the Theory of Inventive Problem Solving (TRIZ). Defining ideality as the benefits-cost systems ratio; where all systems aim to develop the ratio of their functional abilities to the costs throughout their life cycle (Orloff 2017). Yael Cohen and Yoram Reich discussed that ideality is accomplished through maximizing positive affect and minimizing negative affect, and that's how the bridge is created for both notions. Attaining sustainability through ideal systems that create more benefits at low costs (Cohen & Reich 2017). Table 22 shows the sustainability framework proposed by (Cohen & Reich 2017).

*Table 22: ideality strategies proposed as a framework that fosters sustainability (Cohen & Reich 2017)*

Ideality- sustainability strategies	
Increase benefits	Adding functions to the existing working parts
	Improve the performance of some functions
Reduce costs	Exclude auxiliary functions that support the main function but may be removed without affecting the performance of the main function
	Combine subsystems of several functions into a single system
	Transferring some functions to a super system
	Utilizing internal and external resources that already exist and are available
	Improving the conductivity of energy through the system

#### 5.4.1 Proving the system's sustainability with-in itself

All the points in Table 22 were utilized in creating the generative system. To prove that the system with-in itself addresses sustainability individually. And demonstrate that the created generative system is a valid sustainable system based on (Cohen & Reich 2017) definition. Table 23 shows the same points presented in Table 22 with related points showcasing the created generative system, connected to every ideality-sustainability strategy to a function in the scripted algorithmic system.

Table 23: The proposed sustainable framework with the algorithm's sustainable ability for each point (Cohen & Reich 2017)

Ideality- sustainability strategies		The generative system sustainability
Increase benefits	Adding functions to the existing working parts	The algorithm is scripted in Dynamo, which is a visual generative program for Autodesk Revit. That is widely used in the design and construction field. The produced results are automatically applied in Autodesk Revit which adds function to existing working parts.
	Improve the performance of some functions	The generative system provides unlimited, innovative options of reflected ceilings layouts in minutes. Which assists and improves the current performance of individuals in producing ceiling layouts.
Reduce costs	Exclude auxiliary functions that support the main function but may be removed without affecting the performance of the main function	The algorithm within itself allows the end user to turn on and off design approaches, without affecting the main function, as it will still generate a high number of generative reflected ceilings layouts regardless.
	Combine subsystems of several functions into a single system	Different Dynamo scripts in the algorithm perform different functions in producing various design approaches individually, which are combined to produce a full efficient system

	Transferring some functions to a super system	<p>Not only the system produces different designs, a script is created to automatically connect the results to actual light families in Autodesk Revit, allowing the same generated design to be transferred to Autodesk Revit for the designer to use and to preform lighting calculations on the design.</p> <p>In addition, the system provides lighting counts for every light input, saving efficient time for the designer that is equivalent to money.</p>
	Utilizing internal and external resources that already exist and are available	<p>Dynamo, which is a visual programming software that's usually used to create forms and generate commands in Autodesk Revit, was utilized to script algorithms to generate reflected ceilings layouts.</p> <p>In addition, an external plug-in called (Elum tools) was used in Autodesk Revit to perform the lighting calculations without the need to export the results into any other software.</p>
	Improving the conductivity of energy through the system	<p>The generative system allows the end user to perform lighting calculations for every selected design. Which can coherently allow the end user to conduct energy efficiency calculations.</p> <p>Furthermore, the speed in; producing reflected ceiling layouts, altering the output results, and providing the lighting counts saves focus-time and energy for designers which can be utilized in performing other tasks.</p>

(Cohen & Reich 2017) framework for sustainability validates that the generative system itself is a sustainable system that utilizes almost all sustainability strategies.

#### **5.4.2 Proving the system's sustainability by utilizing results from a cross-sectional study**

Furthermore, following (Cohen & Reich 2017) framework that fostered sustainability, it is addressed through two key parameters: (a) Increasing benefits, and (b) reducing costs.

Numerical data were attained through a cross-sectional study conducted on 72 interior designers and architects, that showcased various parameters the generative system utilizes to maximize the positive affect and minimize negative affect. In addition, a simple time-cost analysis is conducted, to present sustainability through creating more benefits at low costs.

The survey was tailored to evaluate and compare the parameters that had been proven in the generative system individually in Phase 01 and Phase 02. Which can be summarized to the following: (a) The diversity and the quantity of produced ceilings. (b) The flexibility in two aspects; creating reflected ceiling layouts, and the flexibility in altering the proposed reflected ceiling layouts (c) Required time to produce and modify reflected ceiling layouts, (d) The average cumulative hours in order to get the counts of lights for the bill of quantity (B.O.Q.). Furthermore, the survey allows to create time-cost analyses, which is one of the key targets for conducting it.

##### **5.4.2.1 Survey participants**

The cross-sectional study was conducted on a sample of 72 interior designers and architects. 11 surveys were excluded to avoid inaccurate results as some designers are currently working in none-related design careers.

The targeted participants included 35 females and 26 males, between ages of 25 to over 50 years old. 59.02% were between 25-30 years old, while 26.23% ticked the category of 31-40 years old. Whilst 13.11% were 41 to 50 years old. And only 1.64% were over 50 years old.

The age category between 25-30 years old in the tested sample showed 80% of interior designers, 60% of architects and 18.2% of designers that are currently working in contracting and fit-outs companies. The demography of architects showed higher percentages in age categories between 31-50 years old than interior designers. While the 54.5% of designers who work in contracting and fit-outs are 31- 40 years old. Figure 146 shows each career category with its age participant age percentages.

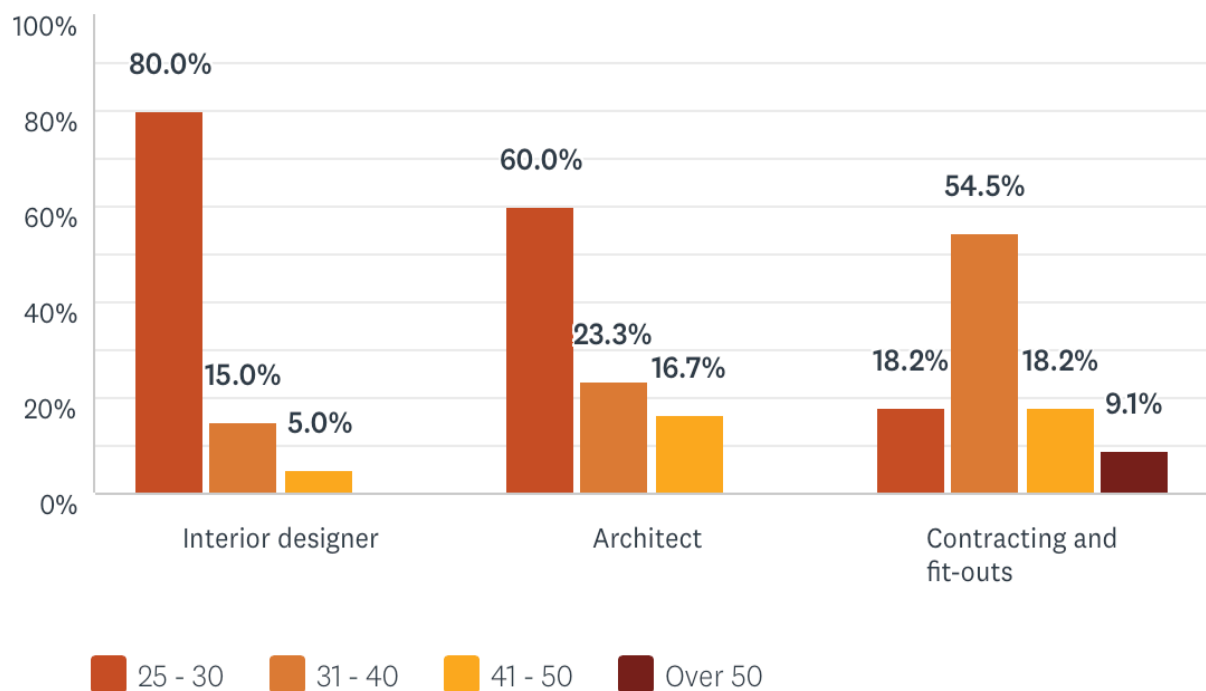


Figure 146: The survey participants demographic (author)

The variation of the designers' demographic was essential to showcase the coherent relationship between the level of expertise in design with age, which will reflect accordingly on the quantity of the produced reflected ceiling layouts, and the designer's speed to do so.

In addition, it will highlight the willingness to adopt and use generative systems in design. Which address sustainability through increasing benefits. Furthermore, the diverse demographic allows to investigate the relationship between expertise and the cost of the designer's time and value per hour. Which is related to sustainability through reducing costs.

#### **5.4.2.2 Survey questions outline**

The survey was prepared using an online survey platform. It presented eleven multiple-choice questions and one close-ended question and one open-ended question.

Questions (1-2) obtains general knowledge about the participant gender, age, while questions (3-4) aim to determine the level of exposure in designing reflected ceiling layouts, through identifying the field of expertise and the years of experience; which creates an evaluation metric that consider or neglect the survey results.

**Q1: Please specify your gender:** Female / Male

**Q2: Please specify your age:** 25-30 / 31-40 / 41-50 / Over 50 years old

**Q3: Please specify your field of expertise in design:** Interior Designer - product designer / Architect / Contracting and fit-outs / Sales-marketing and business development / Other

**Q4: How many years of working experience do you have?** 1-5 years / 6-10 years /11-15 years /16 -20 years / Over 20 years

Question 5 is a closed-ended question, that separates the participants based on the answer to obtain accurate results. The question asks if the designer had worked on or altered reflected ceiling layouts in their field.

If the participant answered positively, the survey would propose question 6, while the negative answer will lead the participant to question 10.

**Q5: Have you designed or altered reflected ceiling layouts/ lighting layouts in your field?**

Yes / No

Questions 6 aims to get the average quantity a designer would usually propose in a project for reflected ceiling layouts. Which enables to compare the diversity and quantity of the produced ceilings from the generative system to a human's level of quantity and diversity.

**Q6: How many reflected ceiling layouts do you usually propose in a project?**

1-3 layouts / 4-6 layouts / 7-10 layouts / 11-15 layouts

While questions (7-9) investigates the cumulative hours required from the designer to: design a reflected ceiling layout, alter it, and count the lights in it. The answers produce numerical data that is compared with the generative system in: (a) the needed time in producing layouts, and in altering results, (b) the saved cost upon connecting it with saved time.

**Q7: How many hours do you usually take to alter the ceiling's proposed designs after the client's feedback?** 1-3 hours / 4-6 hours / 7-10 hours / 11-15 hours / Above 15 hours

**Q8: How many hours would you spend on creating a ceiling layout for a 7x7m meeting room?** 1-3 hours / 4-6 hours / 7-10 hours / 11-15 hours / Above 15 hours

**Q9: Based on your experience, how many cumulative hours do you take in getting the counts of lights while doing a B.O.Q for large ceilings such, e.g.: theaters, dining halls, places of workshops, office towers, museums ... etc.?** 1-3 hours / 4-6 hours / 7-10 hours / 11-15 hours / Above 15 hours

Question (10-13) produce results that connects cost with time to perform the cost-benefit analysis. Furthermore, the questions investigate the level of exposure and usage of Autodesk Revit, and any lighting calculations software. To have a better perspective of the generated system's practicality in the design field.

**Q10: How many cumulative hours do you work daily?** Less than 6 hours / 7-10 hours / 11-15 hours / More than 15 hours

**Q11: Choose an expected salary rate for a designer with 5years of experience?**

Less than 4,000 AED / 4,000 - 8000 AED / 8,000 - 15,000 AED / 15,000 - 20,000 AED / Above 20,000 AED

**Q12: Are you familiar with Autodesk Revit software? And have you used it before?**

Yes, I use it all the time / Yes, I use it only when necessary-required / Yes, I've heard about it but I never used it before / No, I've never heard about it or used it before.

**Q13: Do you use any lighting design or lighting calculation software?** Yes / No

The questions survey was distributed to a network of designers and architects through a web link, as it was created through an online survey platform, providing flexibility and accessibility for participants.

The online survey link was only given to trusted designers and architects, in order to assure attaining accurate data without bias.

And information was gathered through a time span of a week, where participants from the selected design fields gave their answers throughout the day.

Practical experience was a key element in gathering data since it impacted the accuracy of the answers for the remaining questions. Thus, Question 3 was the main question that filtered the results, where surveys that selected: Sales-marketing and business development / Other, from the multiple-choice answers were not considered and removed from the total data gathering.



### 5.4.2.3 Survey results

Results showed that 82.9% of the participants will propose 1-3 reflected ceiling layouts in a project, as seen in Figure 147. While based on the case studies conducted in Phase 01 in this chapter; the generative system can produce a minimum of 50 reflected ceiling layouts. Accounting that the designer will propose 3 layouts; the algorithm is able to produce 94% more layouts than a designer. Creating significantly higher results, that will surely include greater diversity in design than only 3 layouts. Proving one of the many benefits in generative design.

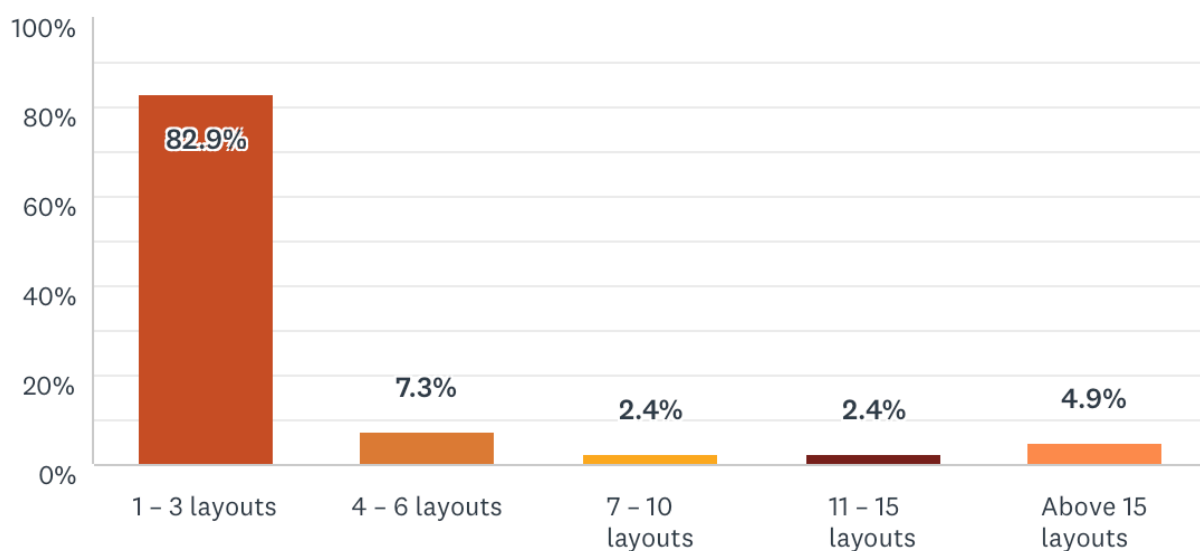


Figure 147: Question 6 results/ How many reflected ceiling layouts do you usually propose in a project? (author)

Question 8 had asked the designers: How many hours would you spend on creating a ceiling layout for a 7x7m meeting room? Results showed that 70.7% had selected from 1 to 3 hours, while 19.5% stated 4 to 6 hours, while 7.3% had surprisingly stated 7 to 10 hours as seen in Figure 148. Calculating the average for the highest percentage only, a designer will take around 2 hours to produce layouts for 7x7 meeting room. In comparison with the generative system that produced various layouts in Phase 01. The algorithm takes an average of 3 minutes to run and to produce results. Saving remarkable time of 97.5%. This highlights the amount of time the generative system can save, which can be utilized by the designer to complete other design tasks, especially in cases of big projects and deadlines that requires extra working hours.

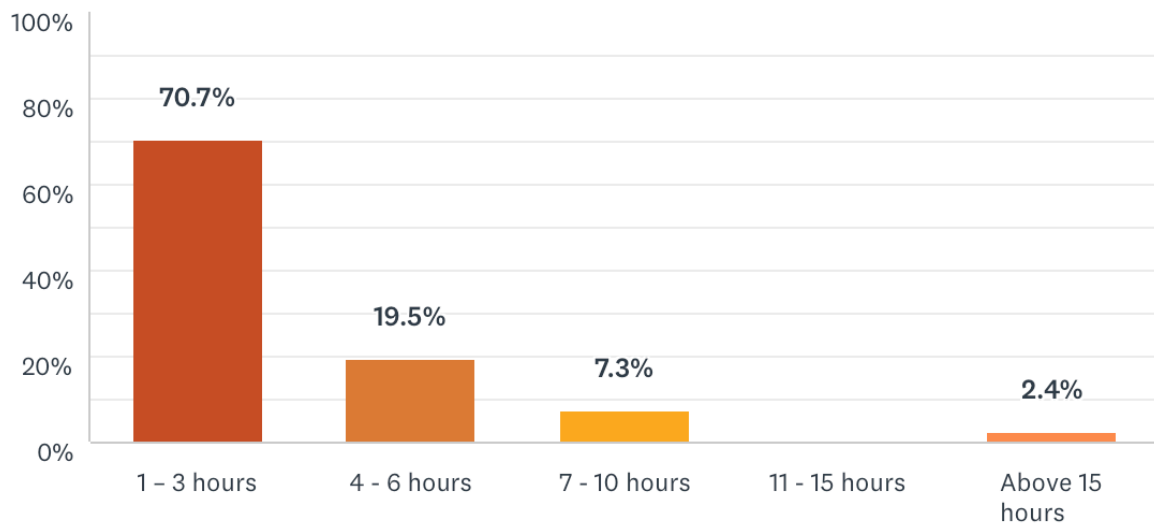


Figure 148: Question 8 results /How many hours would you spend on creating a ceiling layout for a 7x7m meeting room?  
(author)

Time is also addressed in question 7 and 9. Where 51.2% of the designers stated in question 7 that it takes 1-3 hours to alter the ceiling's proposed designs after the client's feedback. However, the algorithm can alter the results automatically in a mouse click. As the designer changes the required variables using the given different sliders. The results are not only altered in Dynamo, but automatically fixed in Autodesk Revit and connected on the spot to the light families in Revit. Figure 149 shows the different responses for question 7.

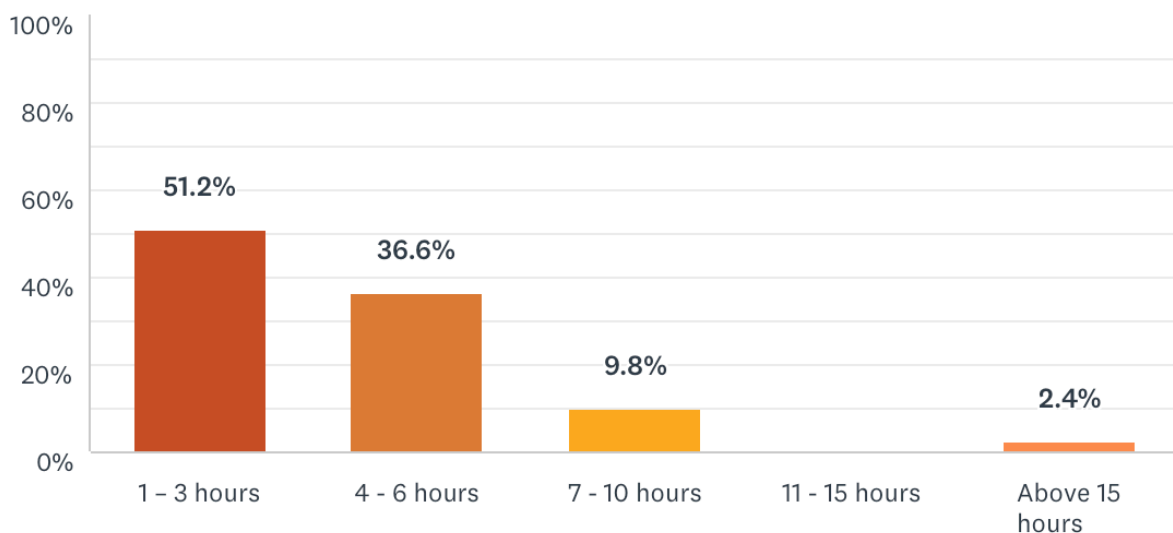


Figure 149: Question 7 results / How many hours do you usually take to alter the ceiling's proposed designs after the client's feedback? (author)

While question 9 showed diverse results as seen in Figure 150. As the highest percentage was 31.7% for 4- 6 cumulative hours that a designer would take to calculate the counts of lights while doing a B.O.Q for large ceilings such as: theaters, dining halls, places of workshops, office towers, museums ...etc. Whereas the generative system automatically produces the counts of the lights with the produced reflected ceilings, regardless of the ceiling size or shape as seen in Phase 02 in this chapter. It only requires the end user to run it the algorithm once which takes an average of 3 minutes to do so. Which saves an astonishing 99% of the time if compared with the average of the highest percentage from question 9. The algorithm can decrease significant amount of time that is wasted in counting the lights.

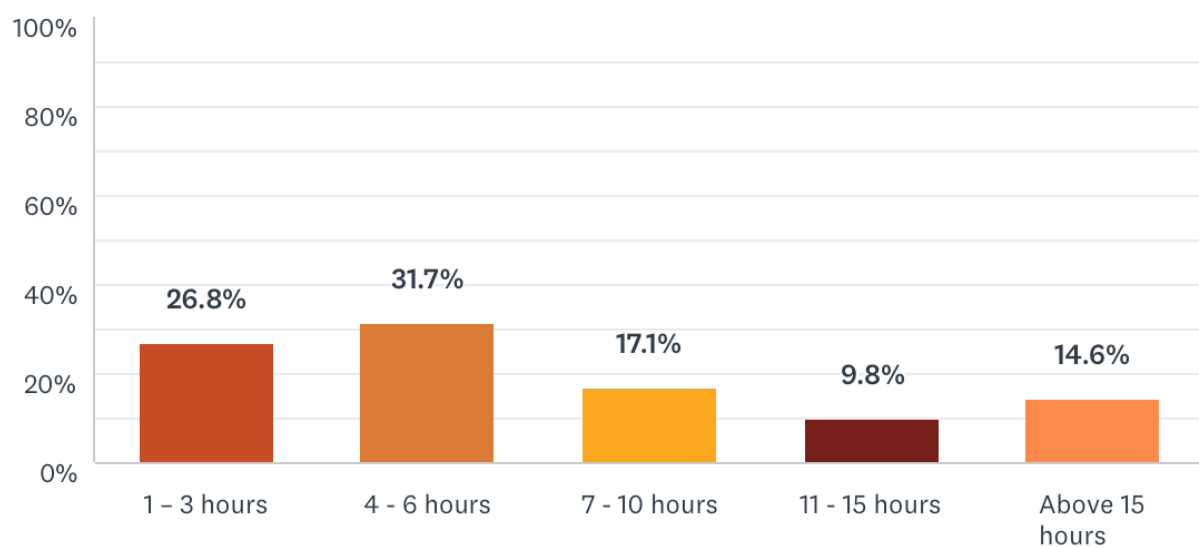


Figure 150: Question 9 results / Based on your experience; how many cumulative hours do you take in getting the counts of lights while doing a B.O.Q for large ceilings such, e.g.: theaters, dining halls, places of workshops, office towers, museums ... etc.? (author)

Table 24 shows the comparison between the designer's ability and the generative system ability. Summarizing the average results from questions (6-9) that are created to obtain quantitative data.

The percentages of change column in Table 24 presented the saved time the generative system can achieve. Showing sticking results that addresses the diversity and the quantity of produced

ceilings, the flexibility in time in order to produce and modify reflected ceiling layouts, furthermore, the time to alter the proposed reflected ceiling layouts, and the average cumulative hours in order to get the counts of lights for the bill of quantity (B.O.Q.)

Table 24: Comparison between the designer's ability and the generative system ability - Questions (6-9)

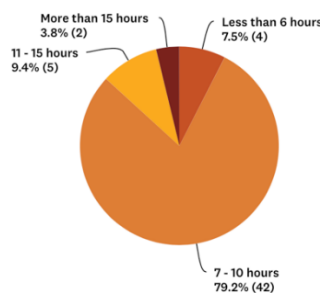
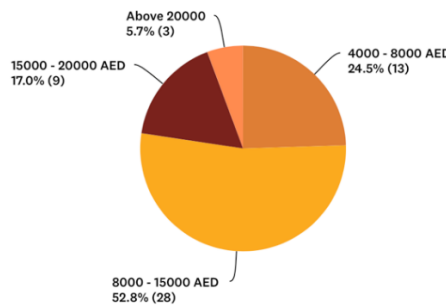
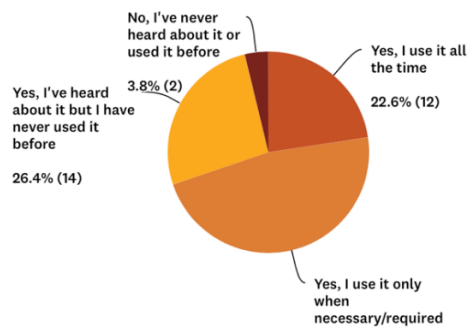
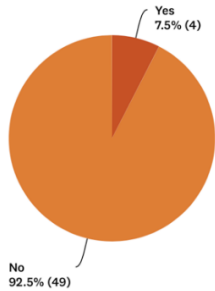
Questions	The designer's ability (Average)	The generative system ability (Average)	The percentage of change
Q6: Number of produced reflected ceiling plans	2 plans	50 plans	96%
Q7: Time required to alter the ceiling's proposed designs after the client's feedback	2 hours	1 minute	99%
Q8: Time required to create a ceiling layout	2 hours	3 minutes	98%
Q9: Time required to calculate the counts of lights while doing a B.O.Q for large ceilings	5 hours	3 minutes	99%

Table 25 presents the different percentages for questions (10-12). The obtained results for questions 10 and 11 are discussed in detail in the following section *Time-Cost analysis*

Question 12 in Table 25 showed the level of exposure and usage of Autodesk Revit, providing results that illustrates the practicality of the generated system in the design field. Where only 3.8% of the sample participants never heard about Autodesk Revit. While 47.2% have a background knowledge of it and use it when required. Whereas 22.6 % use it all the time. And 26.4% know about the software but do not utilize it. This shows the market's potential of the generative system, where 69.8% of designers use Autodesk Revit. In addition, the newest version of Revit 2020 is including Dynamo as a tool in it, which increases the feasibility of this system significantly.

While question 13 showed that 92.5% of the targeted sample didn't use any lighting calculations software before. Indicating two assumptions: (1) Designers will only perform lighting calculations when required by a certain sector, which can be justified using the second assumption (2) Using various software that requires exporting and importing specific formats, in order to design and calculate different lights makes designers avoid running lighting calculations. The hassle of doing so wastes time and is not a priority if not required.

Table 25: Results for questions (10-12)

The Questions	The highest precentage of responses															
Q10: How many cumulative hours do you work daily?	 <table><tr><th>Hours</th><th>Percentage</th><th>Count</th></tr><tr><td>7 - 10 hours</td><td>79.2%</td><td>(42)</td></tr><tr><td>11 - 15 hours</td><td>9.4%</td><td>(5)</td></tr><tr><td>More than 15 hours</td><td>3.8%</td><td>(2)</td></tr><tr><td>Less than 6 hours</td><td>7.5%</td><td>(4)</td></tr></table>	Hours	Percentage	Count	7 - 10 hours	79.2%	(42)	11 - 15 hours	9.4%	(5)	More than 15 hours	3.8%	(2)	Less than 6 hours	7.5%	(4)
Hours	Percentage	Count														
7 - 10 hours	79.2%	(42)														
11 - 15 hours	9.4%	(5)														
More than 15 hours	3.8%	(2)														
Less than 6 hours	7.5%	(4)														
Q11: Choose an expected salary rate for a designer with 5 years of experience?	 <table><tr><th>Salary Range (AED)</th><th>Percentage</th><th>Count</th></tr><tr><td>8000 - 15000 AED</td><td>52.8%</td><td>(28)</td></tr><tr><td>4000 - 8000 AED</td><td>24.5%</td><td>(13)</td></tr><tr><td>15000 - 20000 AED</td><td>17.0%</td><td>(9)</td></tr><tr><td>Above 20000</td><td>5.7%</td><td>(3)</td></tr></table>	Salary Range (AED)	Percentage	Count	8000 - 15000 AED	52.8%	(28)	4000 - 8000 AED	24.5%	(13)	15000 - 20000 AED	17.0%	(9)	Above 20000	5.7%	(3)
Salary Range (AED)	Percentage	Count														
8000 - 15000 AED	52.8%	(28)														
4000 - 8000 AED	24.5%	(13)														
15000 - 20000 AED	17.0%	(9)														
Above 20000	5.7%	(3)														
Q12: Are you familiar with Autodesk Revit software? And have you used it before?	 <table><tr><th>Response</th><th>Percentage</th><th>Count</th></tr><tr><td>Yes, I use it only when necessary/required</td><td>26.4%</td><td>(14)</td></tr><tr><td>Yes, I use it all the time</td><td>22.6%</td><td>(12)</td></tr><tr><td>No, I've never heard about it or used it before</td><td>3.8%</td><td>(2)</td></tr><tr><td>Yes, I've heard about it but I have never used it before</td><td>26.4%</td><td>(14)</td></tr></table>	Response	Percentage	Count	Yes, I use it only when necessary/required	26.4%	(14)	Yes, I use it all the time	22.6%	(12)	No, I've never heard about it or used it before	3.8%	(2)	Yes, I've heard about it but I have never used it before	26.4%	(14)
Response	Percentage	Count														
Yes, I use it only when necessary/required	26.4%	(14)														
Yes, I use it all the time	22.6%	(12)														
No, I've never heard about it or used it before	3.8%	(2)														
Yes, I've heard about it but I have never used it before	26.4%	(14)														
Q13: Do you use any lighting design or lighting calculation software?	 <table><tr><th>Response</th><th>Percentage</th><th>Count</th></tr><tr><td>No</td><td>92.5%</td><td>(49)</td></tr><tr><td>Yes</td><td>7.5%</td><td>(4)</td></tr></table>	Response	Percentage	Count	No	92.5%	(49)	Yes	7.5%	(4)						
Response	Percentage	Count														
No	92.5%	(49)														
Yes	7.5%	(4)														

#### 5.4.2.4 Time-Cost analysis

The impotency of saving time has a coherent relationship with the cost an individual or a company can save. Which has a direct relationship with sustainability as the scripted generative structure is a system that produces various benefits in the lowest costs.

To connect time with money, participants were asked in question 11; the expected salary rate for a designer with 5 years of experience. 52.8% had answered from 8000 to 15,00 AED. While question 10 had asked how many cumulative hours do the participants work daily, where 79.2% had answered 8 to 10 hours daily, as seen in Table 25. If the average of both results were calculated. A designer with 5 years of experience will work 8.5 hours daily with a monthly salary of 11,500 AED. In that case, the designer works 42.5 hours in 5 days, which is 170 hours monthly. That means that the designer is being paid around 68 dirhams per hour, an approximate of 1 dirham and 13 fils per minute.

Following the results of question 8 discussed earlier; a designer will take an average of 2 hours to produce reflected ceiling layouts for a 7x7 meeting room. Which costs 136 dirhams. While the generative system takes 3 minutes. Costing 3 dirhams and 39 fils. That means the generative system saves approximately 133 AED. Which saves 98% of the cost.

In question 7; designers selected an average of 2 hours to alter the ceiling's proposed designs after the client's feedback. Which is costs 136 dirhams. While the algorithm automatically alters the results as the designer changes the value in the given slider, which takes an average of 1 minute. Saving 99% of the cost.

As seen in Figure 150 earlier, 31.7% of the participants stated that counting lights for a relatively large spaces takes around 4-6 hours. Which means 5 hours on an average. Costing 340 Dirhams. While the generative system produces the count of the lights automatically with the generated designs. With an average of 3 minutes. Saving 99% of the cost.

Figure 151 shows the cost comparison between designer's cost and the generative system cost. In three different aspects; (a) generating reflected ceiling layouts, (b) altering the ceiling's proposed designs after the client's feedback, and (c) counting lights for a relatively large space. Percentages show astonishing results where the generative system saves 98%-99% of the cost.

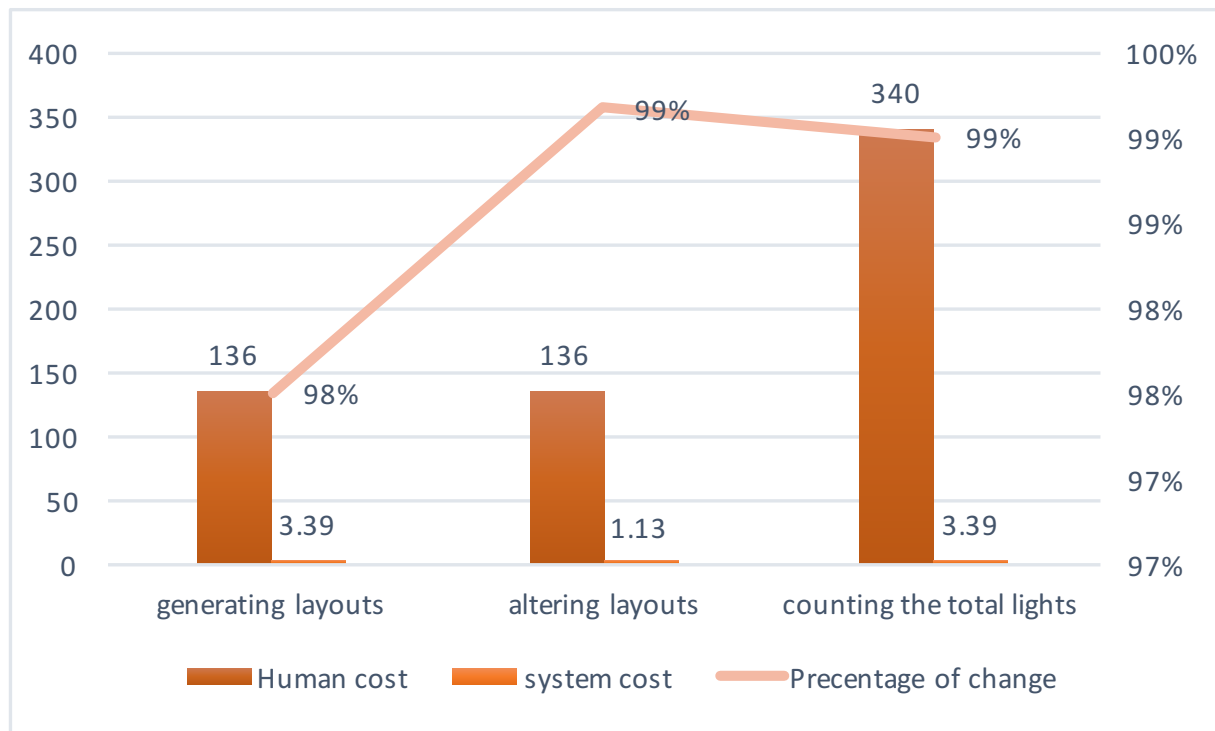


Figure 151: Human cost and the generated system cost comparison

The reduced cost was calculated based on the reduced time, which was priced based on the crossed-sectional study. The results of testing the algorithmic system exhibits sustainability through providing numerous benefits that were discussed and proven in both Phase 01 and Phase 02 in this chapter. In addition, the crossed-sectional study provides numerical data and percentages that showcased the reduced cost and time the generative system can do.

Using (Cohen & Reich 2017) definition of sustainability; the generative system had proved its sustainability through maximizing positive affect and minimizing negative affect, creating more benefits at low costs.

## Chapter 6

### Conclusions



## **6 Chapter 6: Conclusions**

### **6.1 Introduction**

This chapter presents an overview of the work presented in this dissertation, defining the created generative algorithmic system, and presenting a comprehensive briefing for its structure and summarizing the different phases selected to test the system along with the presented results. In addition, it underlines the key contribution of the research in this field. Along with the faced limitations while scripting and testing the system and proposes further work that is founded based on this research.

### **6.2 Summary**

This research presented a novel design approach that generates reflected ceiling layouts. Presenting an algorithmic system scripted utilizing Dynamo software - a programming application hosted by Autodesk Revit - to create a generative system that employs various design approaches through transferring design concepts into mathematical scripts to generate suitable diverse two-dimensional lighting layouts.

The structure of the algorithmic system can be simplified into the following: three key foundations for the generative system to identify a ceiling shape. And various sub-approaches developed based on the three key foundations. The different scripted approaches were incorporated in a direction that reduced calculations time for each script and allowed each approach to be used individually as tools, or as a full system to generate various reflected ceiling designs outputs.

The three key foundations in the algorithmic system are: splitting the surface into smaller surfaces, forming boundaries around the surface, and creating two types of grids, enabling the placement of lights coordinates on the surface. They were created based on key design characteristics that are similar in all the ceilings designs regardless of the shape. The

characteristics are the surface's overall boundaries, the degree, and the quantity of angles it has.

The different sub-approaches were formulated as an extension of the key approaches, as the scripting of two different types of grids resulted in creating lighting patterns sub-approach. As two primary types of lighting patterns using specified lighting dimensions were scripted, the rectangle-based patterns, and the polygon-based patterns. Each type contains two approaches that generates patterns: uniformed and non-uniformed. While the formation of boundaries resulted in designing boundary lighting sub-approach, which diverted into two aspects: boundary lighting for full ceilings, and boundary lighting for narrow corridors. Furthermore, it produced drop-down ceilings sub-approach, that is split into two categories regarding the geometry of the ceiling. The first category will produce a drop-down ceiling that replicates the same shape of the ceiling, while the other category will create different polygon shapes in the center of the ceiling, developing from a circle all the way to a decagon.

Arranging the algorithmic system was created through utilizing different interactive means. The default status of the generative system combines all the several approaches and sub-approaches into a single system that produces more than 500 different reflected ceiling layouts. Nevertheless, the end user can exclude auxiliary functions that support the main function, by turning them off, without affecting the performance of the main function. This approach was created to give the flexibility to target certain designs approaches that meets the designers' desire.

The created generative system prevents the need to use multiple software or to perform different exporting approaches to transfer the designs from Dynamo to Revit, as the scripted algorithm does that automatically. In addition, the exhibited results eliminated the need to export the design from Autodesk Revit to any other format to generate lighting calculations.

Furthermore, the generative system allows applying all the tools in Revit on the final design, which includes exporting the design into any suitable format the designer desire, to use it in AutoCAD or any other software.

Results had validated the system's capacity in various perspectives. The algorithmic system was tested through three different phases, each phase had successfully attained targets that highlighted the system's capacity in different perspectives.

Phase 01 and Phase 02 prove the generative system as a comprehensive system that can perform a full design cycle. Starting from generating various and diverse reflected ceiling designs, altering them automatically and giving the end user the ability to change the design based on their desires, all the way to producing numerical data for lighting counts and artificial lighting calculations, all within the same software.

Phase 01 had performed two types of case studies: conceptual case studies and actual case studies. Results showcased the algorithm diversity in generating different reflected ceiling layouts immediately. Producing more than 100 ceiling designs in a matter of 1 minute. Produced ceiling layouts showed diversity in in three design aspects: light patterns, used design approaches and lighting fixtures quantities.

In addition, results exhibited the algorithm interactive system that involves the end user through the progress and altering the produced outputs. As the designer inserts certain variables and lighting dimensions to produce specific layouts, while manipulating the generated results by utilizing different controllers to manipulate the scale of the pattern and the quantity of the lights. Phase 01 results had also showed the algorithmic system flexibility in producing ceilings' layouts with one design parameter, generating multiple outputs. And it is ability to combine different design approaches together in one ceiling at the same time as well.

Phase 02 of testing the generative system was through taking two generated ceiling layouts from the produced results of a case-study from Phase 01 and conduct lighting calculations on them by a plug-in called Elum-tool. Since the algorithmic system connects the two-dimensional geometrical shapes of lights in Dynamo to actual light families in Autodesk Revit.

This phase tested the comprehensiveness of the generative system to prove its ability to take the generated results into practical applications. In addition, it also had tested another feature the algorithmic system provides, which is counting lights. As it automatically exhibits the quantity of lights that any selected ceiling produce. Results showed accurate counts for both selected ceiling designs and delivered great outputs that predicted the illuminance levels in the tested areas.

Phase 03 exhibited the systems sustainability through two approaches, the first approach had discussed the system sustainability within itself through (Cohen & Reich 2017) proposed framework that analyzes sustainability for systems, while the second approach had proved sustainability through two parameters: maximizing benefit and decreasing cost. As numerical data were attained through a cross-sectional study conducted on 61 interior designers and architects, including 35 females and 26 males, between ages of 25 to over 50 years old.

Increasing benefits parameter was achieved through evaluating and comparing the beneficial outcomes that had been proven in the generative system individually in Phase 01 and Phase 02 against the participants' responses. While Reducing cost was attained through a simple time-cost analysis, which utilized the data obtained from the survey, analyzed the saved time, and financially priced it. Presenting sustainability through creating more benefits at low costs, as results showed an astonishing reduction of cost which reached 99%.

### 6.3 Conclusions

The essential contribution of this dissertation is the creation of a comprehensive, interactive generative system that produces relatively high quantities of lighting layouts for interior ceilings. The research goals that were introduced in Chapter 1 were achieved through the various results exhibited in Chapter 5, Phase 01, Phase 02, and Phase 03.

**1- To propose a computational generative design in producing various solutions for reflected ceiling layouts designs.** The tested case-studies presented in Chapter 5; Phase 01 exhibited the ability of the scripted generative system to provide various solutions for each tested case-study. As Conceptual case studies produced 69 results in Office 01 and 78 ceiling designs for Office 02. While the Field studies; tested on actual offices in Dubai Design District produced 108 results for Office A101 and 110 designs for Office A202.

**2- To exhibit a methodological framework that presents an outline for a comprehensive, interactive generative system that produces relatively high quantities of lighting layouts for interior ceilings, performing a full design cycle. Starting from generating various and diverse reflected ceiling designs, all the way to producing numerical data for lighting counts and artificial lighting calculations, all within the same software.**

Results in Chapter 5, Phase 01 with Phase 02 had achieved this target, where the generated options displayed in Table 7, Table 9, Table 12 and Table 14 from Office 01, Office 02, Office A101, and Office A202 presented the system's ability to generate large quantities of reflected ceiling designs while highlighting the diversity and the flexibility in the generated results.

While results in Phase 02 proved the system's comprehensive and automatic approach to transfer the generated results from Dynamo into Autodesk Revit in section 5.3.1,

without the need to export results. The generative system had connected the geometrical 2-D designs from Dynamo to lighting families in Autodesk Revit presented in Table 16. It was able to provide accurate lighting counts showcased in Figure 138 to two selected lighting layouts - Ceiling (a) and Ceiling (b), where Ceiling (a) showcased three different lighting families and Ceiling (b) showcased one lighting family - and performed proper lighting calculations presented in section 5.3.2.

Lighting calculations results had validated the efficiency of the system as the numerical outcomes were suitable to each selected design. Ceiling (b) had produced higher luminance levels than ceiling (a). Ceiling (b) showed an increase of 30.5% for the luminance average. And a rise of 24% for luminance maximum values, and 35.5% increase in the luminance minimum value compared to Ceiling (a). Which can be traced back to the selected design and the quantity and the type of the lights used in Ceiling (a) and Ceiling (b).

**3- *Demonstrate different aspects in sustainability in generative systems, such as saving time, cutting cost, increasing diversity, providing flexibility, delivering innovative designs, creating various design tools and much more.*** The results of Chapter 5; Phase 01, Phase 02, and Phase 03 had achieved these various aims.

The comparison between the numerical data attained from the cross-sectional study conducted and the results that were generated from the conducted case-studies in Chapter 5; Phase 01 and Phase 02. The comparison percentages displayed in 5.4.2.3 had validated the different aspects of sustainability in the scripted generative system. Where results had shown that time was decreased around 98%, with an increasement of 96% in the generated quantity of ceiling layouts, as the average time for a designer will be around 2 hours to propose an average of 2-3 lighting layouts, while the average

time of the generative system is 3 minutes to produce an average of 50 layouts. And a reduction of 99% of the cost was proved through identifying the average cost of hiring a designer per hour - which is around 68 dirhams per hour - and connecting it through the time required to produce results by the generative system and by a designer.

Results had also validated sticking comparison results in diversity, as the algorithmic system is able to produce 94% more layouts than a designer, creating significantly higher results, that will surely include greater diversity in design. In addition, section 5.4.2.3 in Chapter 5 had presented visible results that validated the flexibility and simplicity of altering generated results, section-study results had stated the average of 2 hours to alter the ceiling's proposed designs after the client's feedback, costing 136 dirhams based on the average pay for a designer per hour. While the algorithmic system alters it instantly through changing the value input in the given slider. Saved 98% of the time, which coherently saved 99% of the cost.

Furthermore, participants had selected an average of 5 cumulative hours for a designer to calculate the counts of lights while doing a B.O.Q for large ceilings such as: theaters, dining halls, places of workshops, office towers, museums ...etc. Whereas Phase 02, section 5.3.2. in Chapter 5 had proved the system's ability to automatically produces the counts of the lights along with the generated design results. Which saves an astonishing 99% of the time, which automatically saves 99% of the cost.

In addition, results in Chapter 5; Phase 01 had proven the scripted algorithmic system ability to work as design tool that can utilize one design parameter only - such as the exhibited generated results in Figure 127, Figure 121, Figure 134, Figure 135 - and utilizing various design parameters at once, presented in Table 7, Table 9, Table 12 and Table 14. And highlighted in Figure 111, Figure 123, Figure 129, Figure 133.

**4- Targeting sustainability through cost reduction and benefits increasement, through statistically proving the usage of the software to produce results is more beneficial compared to a designer.** Section 5.4.2.4 in Chapter 5 had achieved this goal. Where time-cost analyses had been conducted through calculating the reduced cost based on the reduced time, which was priced based on the crossed-sectional study, resulting in an astonishing reduction of 99% of the cost. In addition, the created generative system had proved sustainability in its own structure, through utilizing (Cohen & Reich 2017) framework of sustainability in generative system; by maximizing the positive affect and minimizing negative affect that -which is highlighted and compared in Table 23-

**5- To integrates generative lighting design with a BIM module.** This was confirmed through connecting Dynamo to Autodesk Revit, through importing a ceiling layout from Revit to Dynamo; addressed in Chapter 3 in section 3.2.1 and in depth in Chapter 4, section 0. And exporting the generated results from Dynamo to Autodesk Revit discussed in detail in Chapter 5, Phase 02, section 5.3.1.

Exporting happens automatically. The scripted generative system created in Dynamo will always have one center layout that is distinguished in green boundaries, located in the intersection point between x and y axes of the displayed results, as explained in Chapter 4, section 4.2.5.1.2. This location indicates the original ceiling location that was imported from Autodesk Revit. Hence, when the end user chooses a design that they wish to connect to Autodesk Revit upon viewing all the displayed results in Dynamo. He/she would move the chosen design to the intersection point location using a given slider, which will automatically connect lighting families to the selected choice as seen in Chapter 5, Phase 02.



This research had provided various contributions to the literature and the design field. suggesting an approach to automated lighting layout system in indoor space and proposes a way that generates alternatives of lighting design through using compensation of mathematical and algorithmic scripts.

It validates the interpretation of conceptual designs into mathematical approaches and digital languages that targets interior design and achieve proper design results. Contributing into the usage of generative design approaches in interior artificial lighting. The mathematical scripts are explained in detail in Chapter 4, where every design approach had its primary Dynamo scripts. For example, a mixture of two mathematical approaches - permutations and combinations - were created to provide all the possibilities of splitting a ceiling design, which is discussed in details in sections 4.2.2.1.2.14.2.2.1.2.2, 4.2.2.1.2.3. And the creation of Quad and Diamond shaped grids was created using simple mathematical calculations that allows the generative system to place light coordinates in the shape of a grid on any inserted ceiling surface, and much more.

In addition, it introduces the use of algorithmic design for indoor artificial lighting design in interior offices. Increasing the productivity with hundreds of outputs delivered in a short amount of time, freeing relatively expensive human resources from somewhat repetitive tasks. And increasing creativity as the scripted algorithms generate non-traditional designs.

Furthermore, individual algorithmic scripts that generate each approach and sub-approach can be copied and utilized as design tools to help designers in distributing lights or perform simple light design approaches, such as creating a grid, apply boundary lighting and so on. Instead of running the entire generative system to develop lighting layouts.

## 6.4 Research limitations

The main limitations of this research are Dynamo software limitations. Which can be divided into two categories: limitations faced through scripting the algorithmic system, and the limitations faced when testing the system. The following points highlights the limitations faced through scripting the algorithmic system:

The graphics tool in Dynamo is not as developed as the other Adobe software. It does not have a good tool to draw lines that indicates splitting the surface in different directions at the same time. it can however split an angle towards one direction only, but not all the angles at once consecutively. To fix this issue; new Nodes from a plug-in called Lunch Box were added to the script. This plug-in is created by a third party called The Proving Ground. It is used as a tool for managing data, and it allows geometry behaviors such as generative form making (Nathan 2013).

The current available tools in Dynamo are not able to achieve the goals of the algorithmic system, thus, a lot of individual tools were translated from design concepts to logic and scripted mathematically to a certain goal. A lot of errors and limitations occur during this process. Constant testing and continuous development were attained to ensure that the written scripts and definitions do perform the required goal.

The software does not allow switching and altering between choices, which became an issue as the algorithmic system started growing, as an efficient method was required to allow the exchange between design approach. Hence, a definition was scripted in this dissertation that works as a switch, allowing easy transition and selection between different approaches.

Although Dynamo generates different outputs, the software places all the options in the same location, causing all the results to overlap. That is why a method was investigated and scripted

to distribute all the generated options while considering appropriate spaces between each option, which depended on the dimensions of the inserted ceiling.

One of the major limitations was creating a system that present the results in a legible way. Creating an appropriate method to display all the results in Dynamo workspace without repetition of the generated designs, since the scripted algorithms interfere with each other, producing results that cannot be displayed using the spacing approach. This issue was solved through various testing and resulted in creating a complex script that allowed a vertical and a horizontal intersective movement, that guarantees displaying all the results that gives the end user a proper visual feedback, displaying all the possibilities, while using a tool that allows a smooth transition between every result.

While designing the lighting counts approach, limitations appeared due to Dynamo's attributes. Since the software cannot provide a counting tool that counts different lighting types individually, calculating all the values inside a list regardless of the type of light. This limitation caused various failures in the counting approach. In the end, a scripted approach was designed to group the different lights individually based on area and count them accordingly.

The following points highlights the limitations faced when testing the algorithmic system:

(a)The algorithm is not able to identify ceilings with structural holes in them. Or any openings in the center of the ceiling. (b) Suitable computer's qualifications are important to avoid the standard computational issues (c) The algorithm is not able to apply boundary lighting method appropriately on curved corners. (d) The orientation of the surface in Revit impacts the direction of the lighting methods in Dynamo.

## 6.5 Further work

This dissertation can be considered as a basepoint for developing a complete generative system that can become an independent software or as a plug-in tool in Autodesk Revit to help designers generate all the possible reflected ceiling layout designs possible. hence, future work includes the enhancement of the existing algorithmic system and the continuous development of the scripts.

One of the most immediate needs in terms of future work is to develop evaluation metrics for the generative system. The current system relies heavily on the visual interaction between the designer and the generative system. Since the algorithmic results will go through evaluation metrics from the human mind, as designers select the results not the algorithm. The proposed work will filter the results through an evaluation metric. Providing for the end user various metrics to choose from.

This research recommends utilizing Project Refinery Beta as a tool for optimizing only one approach. Project Refinery is generative design tool that is hosted by Autodesk which can filter and optimize Dynamo results based on selected inputs. Project Refinery is an Autodesk generative design beta for different fields of designs that gives users the power to quickly explore and optimize their Dynamo designs. In the case of the exhibited algorithmic system, one algorithmic script for creating light patterns can be taken into Refinery and apply suitable evaluation metrics that allows the results to be filtered accordingly. However, due to the complexity of the algorithmic system, and the level of interactions of the scripted algorithms together, the entire algorithmic system cannot be taken into Refinery at once.

That is why the research is recommending scripting various evaluation metrics in the form of logical questions and definitions that allow suitable assessment that will make the generative system outputs easily usable by architects, researches, and students.

The evaluation type addresses evaluating the generated results from a sustainable aspect. It includes four metrics, evaluating the generated ceiling layouts through (a) the used quantity of the lights in every ceiling layout, (b) the level of diversity in the generated ceiling layouts, and (c) the lighting power density based on the design (d) the cost of the proposed designs

The used quantity of lights is selected as a sustainable metric as it has a direct impact of the energy levels of the space. This evaluation metric can be scripted and connected to the created light groups that were created to perform the lighting calculations. The script will perform a limitation approach along with a logical statement for this metric to be successful in the algorithmic system, in addition it will need to be connected to a slider that allows the end user to select a certain value that represents the amount of lights they desire in a ceiling layout. For example, an if statement can be scripted stating that if the slider value is on number 5, then the algorithmic system will showcase only the lighting layouts that has 5 lighting fixtures in them, and so on. Defining the script might be slightly complicated to do, as a clear approach needs to be created for the algorithmic system to include the counting regardless of the type of used lights. This approach would require the algorithmic system to produce results and investigate the quantities of each generated ceiling. limitation might occur since the built generative system perform counting for all the selected design approaches regardless if the end user had turned on or off the design.

The level of diversity in the generated ceiling layouts is the second proposed metric for the sustainability evaluation type. This metric has a direct relationship with the third metric, as every lighting type has a certain characteristics and watts, impacting the lighting power densities. This metric arranges the generated designs based on the number of light families used. The minimum light family used will be 1, indicating only one type of lights in the generated lighting layout, while the maximum will be 5 based on the inserted lighting types utilized in this generative system, allowing the end user to select how many lighting types they

desire to use. If this evaluation metric will be applied. The lighting input variables that is added by the designer in the beginning of the algorithmic system should not be flexible anymore. As this evaluation metric will be built on numerical data that is taken from the lighting input which describes the dimensions of the lights and connects different dimensions to different Revit lighting families. Any alteration in the lighting input might result in errors, as there is a chance that the algorithmic system might not be able to identify any new lighting inputs that are not connected to Revit lighting families.

The third evaluation metric in sustainability will arrange the generated ceiling layouts based on the lighting power densities of each design. This indicates creating a mathematical algorithmic script that performs lighting power densities for each generated design. This evaluation metric can be constructed as its own as an independent algorithmic system, since there is a possibility for generative system to crash if the calculations for every generated design were merged along with the different calculations performed by the system to generate designs. This research recommends exploring this evaluation metric as its own, recognizing its ability to produce powerful results that is built on this generative system.

The fourth recommended evaluation metric will filter the proposed layouts based on the cost of the lights used in the generated design. Where the end user will be able to filter and arrange the results from low to high or vice versa, allowing the designer to easily point out the cheapest and the most expensive cost for the variety of designs.

The proposed approach will remove the fixability in inserting any type of dimensions in the lighting lists displayed earlier as the Lighting Sizing in Figure 11 in Dynamo. As this approach will only provide certain dimensions that are connected to specific light families in Autodesk Revit. Those light families will be attained from specific suppliers in the United Arab Emirates in an IES format along with the cost of each light type. The scripting approach will be similar

to the approach that was created to produce the counts for the lights discussed in section 3.2.3, where every light type is grouped separately and paired with its cost, as a script is required in order to do so. In addition, a definition script that provides mathematical calculation is essential in order to produce the cost of every design.

For example; if one result of the generated ceiling designs showed three different types of lights - A, B and C - where the design had 5 bulbs of type A, 3 of type B and 4 of type C. Assuming that Type A costs 25AED, Type B costs 12AED and type C costs 35AED. First, the algorithmic script will have to identify the three different types, this identification can be similar to the same approach as the counting approach in section 3.2.3, where the area of every type was identified in order for the algorithmic script to identify the different types. Then, the counting of every time is created, in this stage the algorithmic system understands the quantity and the types of each light in the design. Finally, a definition script that multiplies the cost of every type with the quantity found in the produced design. In this example; the script can be an if-script where it states: if the light found is type A, then the algorithm will multiply 25 by the quantity of the lights for type A. The script will also have a second condition that states; if the found light is type B, then multiply 12 by the quantity found for type B, or else multiply the quantity of Type C with 35. Upon that statement, the cost in this example for type A will be 125 AED, 36 AED for type B, and 140 AED for type C.

In addition, another definition script will be connected to the if-script, which adds up all the results found in the if-script in one list. Creating a total cost of lights in the given example as 301 AED. This procedure will be repeated to all the generated ceilings, and then listed in lists that can be connected to another script in order to filter from high to low and vice versa.

This research recommends updating the current algorithmic generative system based on the tested case-studies investigated earlier. Since some case-studies had showed minor limitations that were not considered in the scripting process.

The first update is adding a slider that allows the end user to rotate the grid direction 360 degree instead of having it applied on the same orientation as the model in Autodesk Revit. This approach will investigate the possibility to change the axes that the algorithm relies on instead of Revit orientation to Dynamo's orientation. Giving the end user the opportunity to change the orientation which will immediately impact all the design methods.

The second update addresses the boundary approach in identifying ceilings. Where another script should be added to accommodate the condition of ceilings' openings. Since the current algorithmic system will only work on ceilings that do not exhibit any holes in them or any structural openings in different areas of them.

The third recommended update is to include a script that addresses curved corners in a ceiling, as testing the generative system showed minor division errors in the boundary lighting approach when applied on curved corners.

Finally, this research recommends adding a slider that allows the end user to limit the amount of produced generate ceiling layouts required. Since some ceiling shapes may result in creating a lot of designs that require some time to produce. Hence, a limitation approach will help in decreasing the calculations time and stop any possibility of crashing Dynamo while running.

The algorithmic generative system had been investigated and tested frequently through the scripting process and through the testing process, however every case study might reveal new limitations, thus, it is recommended to test the algorithmic system on at least 100 different ceiling designs to ensure that no new limitations might occur.



Upon finalizing testing the generative system in a more comprehensive approach, adjusting discovered errors and applying various filters. It is recommended to create another version of the generative algorithmic scripts to match different software. This dissertation recommends investigating creating a similar version through grass-hopper; a visual programming software for Rhino software.

The design concepts will be similar, however the used language for creating the scripts and the definitions might differ slightly. Various investigations are required to see the suitability of grass-hopper in delivering the same results as Dynamo.

The optimal case is re-writing the scripts in a coding language, Python for example. Where any programming software would be able to read the scripts and definitions, generating suitable results in multiple platforms. A part of the new scripts will aim to develop this concept into a plug-in for different software. That would require multiple coding scripts that allows the generative system to work on any inserted software, not to mention translating all the design approaches to suitable scripts. This requires a lot of sources and an in-depth knowledge in coding and scripting, along with extensive testing on the different software platforms.

## 7 References

- Abdulgaki, Z., Abdulbaqi, H. & Mohialden, Y. (2018). A Novel Interior Space Planning Design Based on MDB-FA Method. *International Journal of Civil Engineering and Technology (IJCIET)*, vol. 9 (10), pp. 641–646.
- Acuity Brands Lighting. (2020). "About Us | Acuity Brands". *Acuity Brands* [online]. [Accessed 1 July 2020]. Available at: <https://www.acuitybrands.com/en/about-us>
- Agkathidis, A. (2015). Generative Design Methods - Implementing Computational Techniques in Undergraduate Architectural Education. *eCAADe*, vol. 2, pp. 48-54.
- American Medical Association. (2016). "AMA adopts guidance to reduce harm from high intensity street lights". *American Medical Association* [online]. [Accessed 9 April 2020]. Available at: <https://www.ama-assn.org/press-center/press-releases/ama-adopts-guidance-reduce-harm-high-intensity-street-lights>
- Anderson, C., Bailey, C., Heumann, A. & Davis, D. (2018). Augmented space planning: Using procedural generation to automate desk layouts. *International Journal of Architectural Computing*, vol. 16 (2), pp. 164-177.
- Badino, E., Shtrepi, L. & Astolfi, A. (2020). Acoustic Performance-Based Design: A Brief Overview of the Opportunities and Limits in Current Practice. *Acoustics*, vol. 2 (2), pp. 246-278.
- Baloch, A., Shaikh, P., Shaikh, F., Leghari, Z., Mirjat, N. & Uqaili, M. (2018). Simulation tools application for artificial lighting in buildings. *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 3007-3026.
- Bangali, J. (2018). Discomfort Glare Evaluation using DIALux Lighting Simulation software and using developed Python Program model. *International Journal of Sustainable Lighting*, vol. 20 (2), pp. 44-50.
- Barreto, G. (2016). *Generative Design for Building Information Modeling*. Master of Science Degree in Information Systems and Computer Engineering. Técnico Lisboa.

Basta, A., Serror, M. & Marzouk, M. (2020). A BIM-based framework for quantitative assessment of steel structure deconstructability. *Automation in Construction*, vol. 111, p. 103064.

Boyce, P. (2016). Editorial: Exploring human-centric lighting. *Lighting Research & Technology*, vol. 48 (2), pp. 101-101.

Bukhari, F. (2011). *A Hierarchical Evolutionary Algorithmic Design (HEAD) System for Generating and Evolving Building Design Models*. PHD. Queensland University of Technology.

Caetano, I. & Leitão, A. (2016). Using Processing with Architectural 3D Modelling. *Complexity & Simplicity - Proceedings of the 34th eCAADe Conference*. University of Oulu. Oulu, Finland.

Caetano, I. & Leitão, A. (2018). Integration of an algorithmic BIM approach in a traditional architecture studio. *Journal of Computational Design and Engineering*, vol. 6 (3), pp. 327-336.

Caetano, I., Santos, L. & Leitão, A. (2020). Computational design in architecture: Defining parametric, generative, and algorithmic design. *Frontiers of Architectural Research*, vol. 9 (2), pp. 287-300.

CALDAS, L. & ROCHA, J. (2001). A Generative Design System Applied to Size's School Of Architecture At Oporto. *The 6th Conference on Computer-Aided Architectural Design Research in Asia*. Sydney, Australia. University of Sydney:Sydney, N.S.W.

Camenzind Evolution. (2019). "Google Office,Stockholm". *Camenzindevolution.com* [online]. [Accessed 18 April 2020]. Available at:  
<https://www.camenzindevolution.com/Office/Google/Google-Office-Stockholm>

Cameron, R. (2018). Data, Dynamo, and design iteration. *Building Design & Construction*.

Charter, M. & Tischner, U. (2001). *Sustainable Solutions : Developing Products and Services for the Future*. Taylor & Francis Group:Sheffield. Viewed 3 July 2020. <https://ebookcentral-proquest-com.aus.idm.oclc.org/lib/aus-ebooks/detail.action?docID=1741735#>

Chen, Q., Oh, S. & Burhan, M. (2020). Design and optimization of a novel electrowetting-driven solar-indoor lighting system. *Applied Energy*, vol. 269, p. 115128.

Chien, S. & Flemming, U. (2002). Design space navigation in generative design systems. *Automation in Construction*, vol. 11 (1), pp. 1-22.

Coelho, P., Amaral, J. & Guimarães, K. (2018). Use of Genetic Algorithm for Spatial Layout of Indoor Light Sources. *20th International Conference on Enterprise Information Systems (ICEIS 2018)*. SCITEPRESS – Science and Technology Publications.

Cohen, Y. & Reich, Y. (2017). *Biomimetic design method for innovation and sustainability*. Switzerland:SPRINGER.

DİNO, İ. (2012). Creative Design Exploration By Parametric Generative Systems In Architecture. *Metu Journal Of The Faculty Of Architecture*.

Doulos, L., Kontadakis, A., Madias, E., Sinou, M. & Tsangrassoulis, A. (2019). Minimizing energy consumption for artificial lighting in a typical classroom of a Hellenic public school aiming for near Zero Energy Building using LED DC luminaires and daylight harvesting systems. *Energy and Buildings*, vol. 194, pp. 201-217.

Du, T., Jansen, S., Turrin, M. & van den Dobbelsteen, A. (2019). Impact of Space Layout On Energy Performance Of Office Buildings Coupling Daylight With Thermal Simulation. *E3S Web of Conferences*, vol. 111, p. 03077.

Dunn, N. (2012). *Digital fabrication in architecture*. London:Laurence King.

Duplák, D., Flimel, M., Duplák, J., Hatala, M., Radchenko, S. & Botko, F. (2019). Ergonomic rationalization of lighting in the working environment. Part I.: Proposal of rationalization algorithm for lighting redesign. *International Journal of Industrial Ergonomics*, vol. 71, pp. 92-102.

Dynamobim. (2014). "Q&A about Dynamo | Dynamo BIM". *Dynamobim.org* [online]. [Accessed 14 April 2020]. Available at: <https://dynamobim.org/qa-about-dynamo/>

El-khaldi, M. (2007). *Mapping Boundaries of Generative Systems for Design Synthesis*. Master Of Science In Architecture Studies. American University Of Sharjah.

Feist, S., Barreto, G., Ferreira, B. & Leitao, A. (2016). Portable Generative Design For Building Information Modelling. *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA 2016*, pp. 147–156.

Fernandes, R. (2013). *Generative Design: a new stage in the design process*. Thesis to obtain the Master of Science Degree in Architecture. Técnico Lisboa.

foster and partners. (2018). "Commerzbank Headquarters | Foster + Partners". *Fosterandpartners.com* [online]. [Accessed 26 June 2020]. Available at: <https://www.fosterandpartners.com/projects/commerzbank-headquarters/>

FRAZER, J., FRAZER, J., Xiyu, L., Mingxi, T. & JANSSEN, P. (2002). Generative and Evolutionary Techniques for Building Envelope Design. *Generative Art 2002, 5th International Conference GA2002*. Milan, Italy. Generative Design Lab:Milan.

Gandhi, P. & Brager, G. (2016). Commercial office plug load energy consumption trends and the role of occupant behavior. *Energy and Buildings*, vol. 125, pp. 1-8.

Gao, H., Zhang, L., Koch, C. & Wu, Y. (2019). BIM-based real time building energy simulation and optimization in early design stage. *IOP Conference Series: Materials Science and Engineering*, vol. 556, p. 012064.

Gunagama, M. (2018). Generative Algorithms in Alternative Design Exploration. *SHS Web of Conferences*, vol. 41, p. 05003.

Gunay, H., O'Brien, W., Beausoleil-Morrison, I. & Gilani, S. (2017). Development and implementation of an adaptive lighting and blinds control algorithm. *Building and Environment*, vol. 113, pp. 185-199.

Henriques, G., Bueno, E., lima, D. & Sardenberg, V. (2019). Generative Systems: Intertwining Physical, Digital and Biological Processes, a case study. *Conference: Architecture in the Age*

*of the 4th Industrial Revolution - Proceedings of the 37th eCAADe and 23rd SIGraDi Conference*, vol. 1. [Accessed 18 July 2020].

Ilbeigi, M., Ghomeishi, M. & Dehghanbanadaki, A. (2020). Prediction and optimization of energy consumption in an office building using artificial neural network and a genetic algorithm. *Sustainable Cities and Society*, vol. 61, p. 102325.

Jabi, W. (2013). *Parametric design for architecture*. London:L. King.

Karlen, M., Benya, J. & Spangler, C. (2017). *Lighting Design Basics*. Newark:John Wiley & Sons.

Khan, S. & Awan, M. (2018). A generative design technique for exploring shape variations. *Advanced Engineering Informatics*, vol. 38, pp. 712-724.

Kim, H., Huang, J., Hwang, Y. & Lee, J. (2016). Auto-layout of Lighting Objects to Support Lighting Design in the Early Phase of Design. *33rd International Symposium on Automation and Robotics in Construction (ISARC 2016)*. Auburn, Alabama, USA. Seoul, Republic of Korea.

Kolarevic, B. (2003). *Architecture in the digital age*. New York, N.Y.:Spon Press.

Kotnik, T. (2010). Digital Architectural Design as Exploration of Computable Functions. *International Journal of Architectural Computing*, vol. 8 (1), pp. 1-16.

Krish, S. (2011). A practical generative design method. *Computer-Aided Design*, vol. 43 (1), pp. 88-100.

Larsen, O., Jensen, R., Antonsen, T. & Strømberg, I. (2017). Estimation methodology for the electricity consumption with daylight- and occupancy-controlled artificial lighting. *Energy Procedia*, vol. 122, pp. 733-738.

Lee, Y. & Kim, S. (2016). Algorithmic Design Paradigm Utilizing Cellular Automata for the Han-ok. *Nexus Network Journal*, vol. 18 (2), pp. 481-503.

Levy, N. (2019). "Space Encounters uses planted partitions to divide Synchroon's Utrecht office". *Dezeen* [online]. [Accessed 23 June 2020]. Available at: <https://www.dezeen.com/2019/04/24/space-encounters-synchroon-utrecht-office/>

Lighting Analysts. (2020). "ElumTools – Lighting Add-in for Revit | Lighting Analysts". *Lightinganalysts.com* [online]. [Accessed 1 July 2020]. Available at: <https://lightinganalysts.com/software-products/elumtools/overview/>

Lim, Y., Seghier, T., Harun, M., Ahmad, M., Samah, A. & Majid, H. (2019). Computational BIM for Building Envelope Sustainability Optimization. *MATEC Web of Conferences*, vol. 278, p. 04001.

Liu, H., Lin, Z., Xu, Y., Chen, Y. & Pu, X. (2019). Coverage uniformity with improved genetic simulated annealing algorithm for indoor Visible Light Communications. *Optics Communications*, vol. 439, pp. 156-163.

Liu, H., Wang, X., Chen, Y., Kong, D. & Xia, P. (2017). Optimization lighting layout based on gene density improved genetic algorithm for indoor visible light communications. *Optics Communications*, vol. 390, pp. 76-81.

Lowry, G. (2016). Energy saving claims for lighting controls in commercial buildings. *Energy and Buildings*, vol. 133, pp. 489-497.

Madias, E., Kontaxis, P. & Topalis, F. (2016). Application of multi-objective genetic algorithms to interior lighting optimization. *Energy and Buildings*, vol. 125, pp. 66-74.

Magna, R., Gabler, M., Reichert, S., Schwinn, T., Waimer, F., Menges, A. & Knippers, J. (2013). From Nature to Fabrication: Biomimetic Design Principles for the Production of Complex Spatial Structures. *International Journal of Space Structures*, vol. 28 (1), pp. 27-39.

McCormack, J., Dorin, A. & Innocent, T. (2004). Generative design: a paradigm for design research. *Futureground, Design Research Society International Conference*. Monash University, Melbourne, Australia. Monash University Press:Melbourne, Australia.

Mendes, L., Freire, R., Coelho, L. & Moraes, A. (2017). Minimizing computational cost and energy demand of building lighting systems: A real time experiment using a modified competition over resources algorithm. *Energy and Buildings*, vol. 139, pp. 108-123.

Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D. & Benjamin, D. (2017). Project Discover: An Application of Generative Design for Architectural Space Planning. *Society for Modeling & Simulation International (SCS)*,. [Accessed 18 July 2020].

Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D. & Benjamin, D. (2017). Project Discover: An Application of Generative Design for Architectural Space Planning. *SimAUD 2017 Conference proceedings*. Symposium on Simulation for Architecture and Urban Design.

Nastasi, J., May, E., Snell, C. & Barry, B. (2018). *SU+RE : sustainable + resilient design systems*. Oxford:John Wiley & Sons (Profile, no. 251).

Natephra, W., Motamedi, A., Fukuda, T. & Yabuki, N. (2017). Integrating building information modeling and virtual reality development engines for building indoor lighting design. *Visualization in Engineering*, vol. 5 (1).

Natephra, W., Motamedi, A., Fukuda, T. & Yabuki, N. (2017). Integrating building information modeling and virtual reality development engines for building indoor lighting design. *Visualization in Engineering*, vol. 5 (1).

Nathan, M. (2013). "LunchBox for Dynamo 2013.11.11". *Theprovingground.org* [online]. [Accessed 6 May 2020]. Available at: <http://www.theprovingground.org/2013/11/lunchbox-for-dynamo-20131111.html>

Neufert, E. & Herz, R. (1970). *Ernst Neufert Architects' data*. London:Crosby Lockwood Staples.

Orloff, M. (2017). *TRIZ Algorithms of Invention*. Switzerland:Springer, Cham. Viewed 3 July 2020. [https://doi-org.aus.idm.oclc.org/10.1007/978-3-319-29436-0\\_3](https://doi-org.aus.idm.oclc.org/10.1007/978-3-319-29436-0_3)

Oxman, R. (2017). Parametric design thinking. *Design Studies*, vol. 52, pp. 1-3.



Oxman, R. (2017). Thinking difference: Theories and models of parametric design thinking. *Design Studies*, vol. 52, pp. 4-39.

Peters, B. (2018). Defining Environments: Understanding Architectural Performance through Modelling, Simulation and Visualisation. *Architectural Design*, vol. 88 (1), pp. 82-91.

Phelan, N., Davis, D. & Anderson, C. (2017). Evaluating Architectural Layouts with Neural Networks. *SimAUD*. Toronto, Canada. 2017 Society for Modeling & Simulation International (SCS).

Plebe, A. & Pavone, M. (2017). "Multi-objective Genetic Algorithm for Interior Lighting Design", in G. Nicosia, P. Pardalos, G. Giuffrida and R. Umeton (ed.). *Machine Learning, Optimization, and Big Data*. Volterra, Italy:Springer, Cham, pp. 222-233.

Rumpf, M., Schein, M., Kuhnen, J. & Grohmann, M. (2018). "Aspects of Sound as Design Driver: Parametric Design of an Acoustic Ceiling", in K. Rycke, C. Gengnagel, O. Baverel, J. Burry, C. Mueller, M. Nguyen, P. Rahm and M. Thomsen (ed.). *Humanizing Digital Reality*. Springer Nature Singapore Pte Ltd.

Şahin, M., Oğuz, Y. & Büyüktümtürk, F. (2016). ANN-based estimation of time-dependent energy loss in lighting systems. *Energy and Buildings*, vol. 116, pp. 455-467.

Seyedolhosseini, A., Masoumi, N., Modarressi, M. & Karimian, N. (2020). Daylight adaptive smart indoor lighting control method using artificial neural networks. *Journal of Building Engineering*, vol. 29, p. 101141.

Shea, K., Aish, R. & Gourtovaia, M. (2005). Towards integrated performance-driven generative design tools. *Automation in Construction*, vol. 14 (2), pp. 253-264.

Sheikhhoshkar, M., Pour Rahimian, F., Kaveh, M., Hosseini, M. & Edwards, D. (2019). Automated planning of concrete joint layouts with 4D-BIM. *Automation in Construction*, vol. 107, p. 102943.

Shikder, S., Price, A. & Mourshed, M. (2009). EVALUATION OF FOUR ARTIFICIAL LIGHTING SIMULATION TOOLS WITH VIRTUAL BUILDING REFERENCE. *European*

*Simulation and Modelling Conference: Modelling and Simulation 2009*. Loughborough, United Kingdom. ESM 2009.

Singh, V. & Gu, N. (2012). Towards an integrated generative design framework. *Design Studies*, vol. 33 (2), pp. 185-207.

Skowranek, R. (2017). *Lighting design*. 1st edn. Basel:Birkhäuser.

Smith, L., Kron, Z., Ransom, L. & Smolker, D. (2020). "Generative Design in Revit now available - Revit Official Blog". *Revit Official Blog* [online]. [Accessed 22 July 2020]. Available at: <https://blogs.autodesk.com/revit/2020/04/08/generative-design-in-revit/>

Soddu, C. (1988). Argenia, a generative natural design. *Milan First International Conference Generative Art '98*. Italy. Milan Polytechnic University:Milan.

Soddu, C. (1989). *Città aleatorie*. Milano [ecc.]:Masson.

Soddu, C. (1991). Simulation Tools for The Learning Approach To Dynamic Evolution Of Town Shape, Architecture And Industrial Design. *Calisce '91, International Conference On Computer Aided Learning And Instruction In Science And Engineering*. EPFL, Lausanne, Switzerland. Press Polytechniques et Universitaires Romandes:LAUSANNE.

Soddu, C. (1993). Human Machine Interaction in Design Processes. *Computer Science, Communications and Society*. NEUCHATEL. NEUCHATEL.

Soori, P. & Vishwas, M. (2013). Lighting control strategy for energy efficient office lighting system design. *Energy and Buildings*, vol. 66, pp. 329-337.

Stiny, G. & Mitchell, W. (1978). The Palladian grammar. *Environment and Planning B: Planning and Design*, vol. 5 (1), pp. 5-18.

Svoboda, L., Novák, J., Kurilla, L. & Zeman, J. (2014). A framework for integrated design of algorithmic architectural forms. *Advances in Engineering Software*, vol. 72, pp. 109-118.

Tachikawa, R. & Osana, Y. (2012). Office layout support system using genetic algorithm - generation of layout plans for polygonal space -. *International Conference on Systems, Man, and Cybernetics (SMC)*. Seoul. IEEE:Fukuoka.

Tagueu, M. & Ndzana, B. (2019). Lighting Optimisation Control Of Fluo/Led Systems Using Neural Network And Mathematical Model. *International Journal Of Electrical Engineering And Technology*, vol. 10 (4).

Terzidis, K. (2006). *Algorithmic architecture*. Oxford:Architectural Press.

The Dynamo Primer. (2019). "The Revit Connection | The Dynamo Primer". *Primer.dynamobim.org* [online]. [Accessed 14 April 2020]. Available at: [https://primer.dynamobim.org/08\\_Dynamo-for-Revit/8-1\\_The-Revit-Connection.html](https://primer.dynamobim.org/08_Dynamo-for-Revit/8-1_The-Revit-Connection.html)

Touloupaki, E. & Theodosiou, T. (2017). Performance Simulation Integrated in Parametric 3D Modeling as a Method for Early Stage Design Optimization—A Review. *Energies*, vol. 10 (5), p. 637.

Tsiamis, M., Oliva, A. & Calvano, M. (2017). Algorithmic Design and Analysis of Architectural Origami. *Nexus Network Journal*, vol. 20 (1), pp. 59-73.

Vedvik, R. (2019). Human-centric lighting explained. *Consulting - Specifying Engineer*, vol. 56 (7), pp. 26-30. [Accessed 9 April 2020].

Wagiman, K. & Abdullah, M. (2018). Intelligent Lighting Control System for Energy Savings in Office Building. *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 11 (1), p. 195.

Wagiman, K., Abdullah, M., Hassan, M., Mohammad Radzi, N., Abu Bakar, A. & Kwang, T. (2020). Lighting system control techniques in commercial buildings: Current trends and future directions. *Journal of Building Engineering*, vol. 31, p. 101342.

Wang, Z. & Tan, Y. (2013). Illumination control of LED systems based on neural network model and energy optimization algorithm. *Energy and Buildings*, vol. 62, pp. 514-521.

Wu, D. (2018). Analysis on Zero Energy Consumption Strategy for Office Buildings Lighting in Lianyungang Area. *IOP Conference Series: Earth and Environmental Science*, vol. 111, p. 012035.

Xu, L., Pan, Y., Yao, Y., Cai, D., Huang, Z. & Linder, N. (2017). Lighting energy efficiency in offices under different control strategies. *Energy and Buildings*, vol. 138, pp. 127-139.