Univerza
v Ljubljani
Fakulteta
za gradbeništvo
in geodezijo

LUKA GRADIŠAR

PRILOŽNOSTI GENERATIVNEGA PRISTOPA K NAČRTOVANJU STAVB

MAGISTRSKO DELO

MAGISTRSKI ŠTUDIJSKI PROGRAM
DRUGE STOPNJE GRADBENIŠTVO

Ljubljana, 2019

Hrbtna stran: **GRADIŠAR LUKA** 2019

Univerza
v Ljubljani

Fakulteta za
gradbeništvo in
geodezijo

Kandidat/-ka:

LUKA GRADIŠAR

PRILOŽNOSTI GENERATIVNEGA PRISTOPA K NAČRTOVANJU STAVB

GENERATIVE DESIGN OPPORTUNITIES TO THE BUILDING DESIGN

Mentor/-ica:
prof. dr. Žiga Turk

Predsednik komisije:

Somentor/-ica:
asist. dr. Robert Klinc

Član komisije:

2019

STRAN ZA POPRAVKE

Stran z napako	Vrstica z napako	Namesto	Naj bo

»Ta stran je namenoma prazna«

BIBLIOGRAFSKO – DOKUMENTACIJSKA STRAN IN IZVLEČEK

UDK:	**UDK: 004.42:69(497.4)(043.3)**
Avtor:	**Luka Gradišar, dipl. inž. grad. (UN)**
Mentor:	**prof. dr. Žiga Turk**
Somentor:	**asist. dr. Robert Klinc**
Naslov:	**Priložnosti generativnega pristopa k načrtovanju stavb**
Tip dokumenta:	**Magistrsko delo**
Obseg in oprema:	**51 str., 61 sl.**
Ključne besede:	**Generativni pristop, računalniško podprto modeliranje, optimizacija, genetski algoritmi, BIM, parametrično modeliranje, Dynamo, Project Refinery, vizualno programiranje**

Izvleček:

Gradbeništvo uporablja številne računalniške tehnologije. Pretežno gre za tehnologije, ki analizirajo že zamišljene rešitve in take, ki pomagajo pri predstavitvi in dokumentiranju zamisli. Nedavni razvoj umetne inteligence in sorodnih področij, pa odpira možnosti, da »računalnik« rešitve tudi bolj ali manj samostojno kreira. Eden od takih pristopov je metoda generativnega načrtovanja. V nalogi bodo predstavljene teoretične osnove te metode, potem pa bo aplicirana na primeru projektiranja elementov za senčenje stavbe v okolju programa Revit. Realen inženirski problem energetsko učinkovitega senčenja je bil poenostavljen in prilagojen v obliko zapisa za programsko okolje Dynamo. Predstavljen bo način vizualnega programiranja v tem okolju. Optimalna rešitev se kreira tako, da izberemo in definiramo vhodne spremenljivke ter kriterije, ki podajo numerično oceno alternativnim rešitvam. Po principu generativnega pristopa genetski algoritmi generirajo rešitve, ki jih testirajo in sortirajo ter se z vsako novo iteracijo bližajo optimalnim vrednostim. Generirajo in analizirajo prostor rešitev, da bi lahko potem avtonomno ali s posredovanjem inženirja izbrali končno rešitev. V nalogi postopek primerjamo s tradicionalnim načrtovalskim pristopom ter ocenimo prednosti in slabosti generativnega pristopa. Glavna ideja pristopa je sodelovanje projektanta z inteligentnim orodjem, ki ima komplementarne sposobnosti: pregled velikega števila podatkov, generiranje velikega števila alternativ, analiza rešitev, inkrementalno izboljšanje rešitev. Pristop je splošen in zato uporaben tudi kot pomoč pri načrtovanju na drugih področjih gradbeništva.

»Ta stran je namenoma prazna«

BIBLIOGRAPHIC– DOKUMENTALISTIC INFORMATION AND ABSTRACT

UDC:	**004.42:69(497.4)(043.3)**
Author:	**Luka Gradišar, dipl. inž. grad. (UN)**
Supervisor:	**Prof. Žiga Turk, Ph.D.**
Cosupervisor:	**Assist. Robert Klinc, Ph.D.**
Title:	**Generative design opportunities to the building design**
Document type:	**Master Thesis**
Scope and tools:	**51 p., 61 fig.**
Keywords:	**Generative design, computational design, optimization, genetic algorithms, BIM, parametric modeling, Dynamo, Project Refinery, visual programing**

Abstract:

Civil engineering field uses many different computer technologies. Mainly these are technologies for analysis of the already defined ideas and those which help with the presentation and documentation of said ideas. However, recent development in the artificial intelligence and related fields opens the possibility for the computer to independently generate solutions. One such approach is the generative design. The thesis outlines the theoretical basis of this method, which is then applied to the practical example of the shading elements for the building in Revit. Real engineering example of the energy efficient shading has been simplified and conformed to the Dynamo programming environment in which the visual programming will be presented. In order to find the optimal solution, it is necessary to select and define the input variables and objectives with fitness functions. In the generative design process, the solutions are generated, tested and sorted by genetic algorithms, and with each new iteration we get closer to the optimal solution. They generate and analyze the solution space, to find the final solution autonomously or with the help of an engineer. We compare this process with the design approach and outline the pros and cons of the generative approach. The main idea of the generative design is the collaboration between the designer and the machine, which has complementary capabilities, such as: sorting large amount of data, generating large number of alternatives, analysis of the solutions and iterative improvement of the solution. The approach is general and can be used to help with the designs in other areas of civil engineering.

»Ta stran je namenoma prazna«

ZAHVALA

Za pomoč in nasvete pri raziskovanju in izdelavi magistrskega dela ter podporo se iskreno zahvaljujem mentorju prof. dr. Žigi Turku in somentorju asist. dr. Robertu Klincu.

Zahvala gre tudi družini za pomoč in podporo v času študij in pisanja magistrskega dela.

Pri tem bi se tudi rad zahvalil vsem profesorjem, ki so mi spodbudili zanimanja za različna področja tekom študija in bili vedno pripravljeni odgovoriti na vsa vprašanja. Prav tako gre zahvala celotnemu osebju fakultete, brez katerega študij ne bi bil mogoč.

»Ta stran je namenoma prazna«

KAZALO VSEBINE

STRAN ZA POPRAVKE II

BIBLIOGRAFSKO – DOKUMENTACIJSKA STRAN IN IZVLEČEK IV

BIBLIOGRAPHIC– DOKUMENTALISTIC INFORMATION AND ABSTRACT VI

ZAHVALA VIII

KAZALO VSEBINE X

KAZALO SLIK XII

LIST OF FIGURES XVI

SLOVARČEK 1

1 UVOD 2

1.1 Opis problema 2

1.2 Namen in cilj 3

1.3 Metode dela 3

2 GENERATIVNI PRISTOP 4

3 PRISTOPI MODELIRANJA 5

3.1 Proceduralno modeliranje 5

3.2 Parametrično modeliranje 6

3.3 Računalniško podprto modeliranje 7

4 OPTIMIZACIJA 8

4.1 Genetski algoritmi 9

5 PROGRAMSKO ORODJE 12

5.1 Autodesk Revit 12

5.2 Dynamo 12

5.3 Project Refinery 12

6 ŠTUDIJA UPORABE GENERATIVNEGA PRISTOPA NA PRAKTIČNEM PRIMERU 13

6.1 Predstavitev problema 13

6.2 Metode reševanja 14

6.3 Opredelitve in poenostavitve modela ... 15
6.4 Izdelava modela ... 16
6.4.1 Določitev površine fasade ... 17
6.4.2 Koordinatni sistem ... 18
6.4.3 Zaporedje vodoravnih linij ... 19
6.4.4 Zaporedje točk vzdolž linij ... 21
6.4.5 Lega odprtin ... 22
6.4.6 Razdalja med zaporedjem točk in središčem odprtine ... 22
6.4.7 Enačba za odprtino ... 24
6.4.8 Smer premika točk ... 25
6.4.9 Premik točk za odprtino ... 27
6.4.10 Oblikovne krivulje ... 28
6.4.11 Enačbe širine elementov ... 29
6.4.12 Seznam vrednosti širin v posameznih točkah ... 30
6.4.13 Zamik točk za odprtino ... 31
6.4.14 Oblikovna površina ... 32
6.4.15 Nove krivulje, kot zunanji rob elementov za senčenje ... 33
6.4.16 Geometrija ... 34
6.5 Kriteriji ... 37
6.5.1 Količina materiala ... 37
6.5.2 Učinkovitost senčenja ... 37
6.6 Reševanje ... 41
6.7 Končna rešitev ... 46
7 ZAKLJUČEK ... 48
8 VIRI ... 50

KAZALO SLIK

Slika 2.1: Princip generativnega pristopa temelji na generiranju in testiranju alternativ. [1] 4
Slika 3.1: Enostavni primer proceduralnega modela, ki ga dobimo s funkcijo Sin(x)Sin(y) 5
Slika 3.2: Primer parametrične knjižnice v programu Revit. 6
Slika 3.3: Primer vizualnega programiranja. 7
Slika 4.1: Naključno generiranje začetne populacije. 9
Slika 4.2: Izbor deleža populacije z najboljšimi ocenami. 10
Slika 4.3: Naslednja iteracija. 11
Slika 4.4: Z vsako novo generacijo se zelo približamo optimalnim rezultatom 11
Slika 6.1: Rešitev uporabljena na mednarodnem projektu na univerzi Stanford. [11] 13
Slika 6.2: Obravnavani objekt na katerem želimo steklene površine pokriti z elementi za senčenje. .. 14
Slika 6.3: Iščemo optimalne vrednosti števila in širine elementov za senčenje, glede na porabo materiala ter glede na toplotne pribitke zaradi segrevanja v zimskih in poletnih mesecih. 15
Slika 6.4: Obravnava površina in njene lastnosti. Barve vektorjev rgb ustrezajo smerem xyz. 17
Slika 6.5: Izbira površine in določitev robnih točk. 17
Slika 6.6: Parametrične lastnost površine v programu Dynamo. 18
Slika 6.7: Določitev kartezičnega koordinatnega sistema za izbrano ploskev. 18
Slika 6.8: Prikaz zaporedja vodoravnih linij, ki predstavljajo število elementov. 19
Slika 6.9: Podajanje mejnega območja vhodne spremenljivke. 19
Slika 6.10: Določitev parametra za število elementov za senčenje. Število predstavlja na koliko delov se razdeli navpični rob površine iz katerih bomo generirali elemente. 20
Slika 6.11: Dolžinski parameter krivulje v programu Dynamo 20
Slika 6.12: Generiranje zaporedje vodoravnih linij vzdolž ploskve glede na zaporedje točk. 21
Slika 6.13: Prikaz generacije točk vzdolž vodoravnih linij. 21
Slika 6.14: Tvorba zaporedja točk vzdolž vodoravnih linij. 22
Slika 6.15: Lego odprtine definiramo z U, V koordinatami na površini. 22
Slika 6.16: Prikaz obravnavanih točk v odprtini, ki so pridobljene glede na oddaljenost od središča.. 23
Slika 6.17: Sortiranje dolžin med središčem in zaporedju točk. Dolžine, katerih velikost je manjša od radija ohranimo, ostalne nadomestimo z vrednostjo null, da ohranimo strukturo seznama. 23
Slika 6.18: Podajanje enačbe, ki podeli vrednost za koliko zamaknemo točke glede na njihovo oddaljenost od središča. Točkam, ki so izven radija dodelimo vrednost 0. To pomeni, da se te točke ne bodo zamaknile. 24
Slika 6.19: Grafični prikaz enačbe za premik točk za odprtine. 24
Slika 6.20: Prikaz premika točk v vertikalni smeri odvisno od odmaknjenosti od središča. 25
Slika 6.21: Poiščemo smerne vektorje med središčem in vsako točko v zaporedju. Smeri premikov točk predstavljajo komponente vektorjev v navpični smeri. 25

Slika 6.22: Prikaz različnih možnih kombinacij združevanja seznamov elementov v programu Dynamo. 26
Slika 6.23: Prikaz rezultata odmika točk, ki ponazarjajo odprtino. 27
Slika 6.24: Premik točk in tvorba krivulj skozi te točke. Točke zamaknemo za vrednost 27
Slika 6.25: Prikaz sečišč med zaporedjem vodoravnih linij in oblikovnimi krivuljami. 28
Slika 6.26: Tvorjenje točk na sečiščih oblikovnih krivulj in zaporedjem vodoravnih linij. 28
Slika 6.27: Tvorjenje seznama vrednosti, katere bodo tvorile spremenljivo širino elementov za senčenje. Vrednosti zapišemo parametrično, kar predstavlja tudi drugo vhodno spremenljivko, ki je definirana kot največja možna širina elementov. 29
Slika 6.28: Prikaz oblike, ki nam jo tvori enačba za določitev spreminjajoče širine. 29
Slika 6.29: Vrednosti spremenljivih širin dobljenih iz enačb združimo v novo zaporedje naraščujočih in padajočih vrednosti..... 30
Slika 6.30: Zamikanje točk v smeri normale glede na zaporedje naraščujočih in padajočih vrednosti. Pri tem vključimo v seznam robne točke, ki jih zamaknjemo z enakimi vrednostmi. 31
Slika 6.31: Površina, ki je generirana preko interpolacije med oblikovnimi krivuljami. 32
Slika 6.32: Prikaz zaznavanja sečišča med oblikovno provršino in pravokotnimi linjami glede na začetno površino. Preko tega dobimo točke, katere se bo povezavlo s krivuljami..... 32
Slika 6.33: Generiranje točk na sečiščih med pravokotnimi linijami glede na začetno ploskev in med odmaknjeno ploskev. Točkam na začetni površini poiščemo sorodne točke na odmaknjeni površini.. 33
Slika 6.34: Prikaz povezave točk v krivulje na odmaknjeni površini. Krivulje predstavljajo zunanji rob elementov za senčenje. 33
Slika 6.35: Tvorjenje krivulj na odmaknjeni ploskvi preko povezave točk, dobljenih iz sečišč. 33
Slika 6.36: Povezava krivulj v površine, katere odebelimo, da dobimo končno volumetrično geometerijo elementov za senčenje. 34
Slika 6.37: Generiranje površine, tako da povežemo krivulje na začetni površini in krivulje na odmaknjenji oblikovni površini. 35
Slika 6.38: Dodelitev debeline površinam elementov, katere pretvorimo v končno volumentrično geometrijo elementov za senčenje. 35
Slika 6.39: Prikaz geometrije elementov za senčenje v programskem okolju Dynamo. 36
Slika 6.40: Funkcija, ki nam izračuna volumen generirane geometrije elementov za senčenje, ki predstavlja izhodno spremenljivko oziroma prvi kriterij za optimizacijo. 37
Slika 6.41: Simuliranje sončnih žarkov z zaporedjem linij v smeri sončnega kota..... 38
Slika 6.42: Funkcija, ki nam izračuna število žarkov, ki jih elementi za senčenje zaustavijo. Število predstavlja izhodno spremenljivko oziroma kriterij za optimizacijo..... 39
Slika 6.43: Prikaz delovanja zaznavanja sečišča med sončnimi žarki in elementi za senčenje. Seznam elementov se mora zamakniti, tako da se vsaka linija povezuje z geometrijo nad njo. 40

Slika 6.44: Nastavitve v programu Project Refinery. Podati je potrebno vhodne in izhodne spremenljivke, ter nastavitve optimizacije. 42
Slika 6.45: 4D graf raztrosa rešitev, ki prikazuje volumen in število sončnih žarkov pozimi, ki jim je preprečen prehod v notranjost, v odvisnosti od števila in širine elementov (Večje točke predstavljajo večje vrednosti, manjše točke pa manjše vrednosti. Podobno z barvami, rdeča barva predstavlja manjše vrednosti, vijolična pa večje vrednosti). 43
Slika 6.46: Graf, ki primerja rešitve med kriterijema volumna in števila sončnih žarkov poleti, ki jim je preprečen prehod v notranjost. 44
Slika 6.47: Graf, ki primerja rešitve med kriterijema volumna in števila sončnih žarkov pozimi, ki jim je preprečen prehod v notranjost. 44
Slika 6.48: Sortiranje med optimalnimi rešitvami z omejevanjem vrednosti spremenljivk. 45
Slika 6.49: Izpis vrednosti spremenljivk od rezultatov dobljenih s sortiranjem. 45
Slika 6.50: Prikaz podobe elementov za senčenje na objektu. 46
Slika 6.51: Grafični prikaz prehoda sončnih žarkov skozi senčenje v zimskih mesecih pred optimizacijo. Rumena barva predstavlja žarke, ki preidejo skozi senčenje, siva barva predstavlja žarke, ki jih senčenje zaustavi. Za zimske mesece želimo povečati število žarkov, ki preide v notranjost, torej želimo videti več rumenih žarkov. 47
Slika 6.52: Grafični prikaz prehoda sončnih žarkov skozi senčenje v zimskih mesecih po optimizaciji. Rumena barva predstavlja žarke, ki preidejo skozi senčenje, siva barva predstavlja žarke, ki jih senčenje zaustavi. 47

»Ta stran je namenoma prazna«

LIST OF FIGURES

Figure 2.1: Basic idea of the generative design, which is based on generate and test principle. [1]..... 4
Figure 3.1: Simple example of procedural model, which is generated by function Sin(x)Sin(y) 5
Figure 3.2: Example of parametric family in Revit..... 6
Figure 3.3: Example of visual programming..... 7
Figure 4.1: Random generation of the initial population 9
Figure 4.2: Selection of the best-fit part of the population 10
Figure 4.3: Next iteration. 11
Figure 4.4: With each new generation we are closer to optimal results..... 11
Figure 6.1: Render of the solution used on the international project on Stanford University. [11] 13
Figure 6.2: Example building on which we want to cover glass surfaces with shading elements. 14
Figure 6.3: We want to find optimal values for the number and the width of the shading elements, based on the material needed and the solar gains during the winter and summer months. 15
Figure 6.4: Example surface and its parameters. Vector colors rgb correspond with the xyz directions. 17
Figure 6.5: Surface selection and defining border points. 17
Figure 6.6: Surface parameter properties in Dynamo. 18
Figure 6.7: Definition of the Cartesian coordinate system for the selected surface. 18
Figure 6.8: Illustration of the array of lines, which represent the number of the elements. 19
Figure 6.9: Defining the bounds of the input variales. 19
Figure 6.10: Input parameter for the number of elements for the shading. Input number is the number of divisions of a surface edge from which the elements will be modeled. 20
Figure 6.11: Distance parameter of a curve in Dynamo..... 20
Figure 6.12: Generating array of horizontal lines along the surface based on the array of points. 21
Figure 6.13: Illustration of the point generation along the horizontal lines. 21
Figure 6.14: Generating array of points along the horizontal lines. 22
Figure 6.15: Position of the opening is defined with U,V coordinates on the surface. 22
Figure 6.16: Illustration of the affected points, which are filtered based on the distance from the center point..... 23
Figure 6.17: Filtering the distances between the centerpoint and the array of points. Distances which are larger than the radius are kept, others are replaced with value null to preserve list structure. 23
Figure 6.18: Equation which assigns the values for the point translation based on the distance from the center point. Points which are beyond radius are attributed with value 0, which means they will not be translated. 24
Figure 6.19: Illustration of the offset equation for the openings. 24
Figure 6.20: Illustration of point translation based on the distance from the center point. 25

Figure 6.21: Direction vectors are found between the center point and each point in the array. Directions for the translation of each point are defined by the z-component of the vectors. 25
Figure 6.22: Illustration of different lacing methods for list combinations in Dynamo. 26
Figure 6.23: Illustration of the point translation result, which illustrates the opening. 27
Figure 6.24: Point translation and the generation of the curves connecting the points. 27
Figure 6.25: Illustration of the intersection between the array of lines and the shape curves. 28
Figure 6.26: Generating points on the intersections between shape curves and array of horizontal lines. 28
Figure 6.27: Creating a list of values for the variable width of the shading elements. Values are parametric, which is also a second input variable, defined as a maximum width of the elements. 29
Figure 6.28: Illustration of the shape, generated by the equation for the variable width. 29
Figure 6.29: Values of the variable widths are combined into a joint list of rising and falling values. 30
Figure 6.30: Point translation in the normal direction based on the array of the raising and falling values. Translated edge points are also added to the list of points. 31
Figure 6.31: Surface, which is generated by interpolating shape curves. 32
Figure 6.32: Illustration of the intersection detection between the shape surface and the lines perpendicular on the starting surface. Intersection returns points, which will be connected with curves. 32
Figure 6.33: Point generation on the intersection between the lines, perpendicular on the base surface and the offset surface. To points on the base surface we find associated points on the offset surface.. 33
Figure 6.34: Illustration of the curves connecting the points on the offset surface. The curves are outer edge of the shading elements. 33
Figure 6.35: Creating curves on the offset surface, by connecting the points from the intersection. 33
Figure 6.36: Connecting the curves into the surfaces to which we add thickness to create a solid geometry for the shading elements. 34
Figure 6.37: Generating surface by connecting the curves on the starting surface and the curves on the shape surface. 35
Figure 6.38: Applying thickness to the surfaces, which are transformed into the solid geometry of the shading elements. 35
Figure 6.39: Generated geoemetry of the shading elements in Dynamo. 36
Figure 6.40: Objective function which calculates the volume of the generated geometry. Volume is output variable or first objective for optimization process. 37
Figure 6.41: Simulating sunrays with an array of lines in the direction of the sun. 38
Figure 6.42: Objective function which calculates the number of sunrays blocked by the shading elements. Number is output variable or objective for optimization process. 39
Figure 6.43: Illustration how the intersection between the sunrays and shading elements works. List of elements need to be shifted so that the each individual line interacts with the geometry above it. 40

Figure 6.44: Settings in Project Refinery. Input and output variables need to be selected together with the optimization settings..... 42
Figure 6.45: 4D scatterplot of total volume and number of blocked sunrays in summer in realtion to number and width of the elements (larger values are displayed as larger points and smaller values are displayed as smaller points. Similarly with colors, larger values are displayed as purple colors and smaller values are displayed as red colors). 43
Figure 6.46: Scatterplot of solutions in comparison between the total volume and number of blocked sunrays in summer..... 44
Figure 6.47: Scatterplot of solutions in comparison between the total volume and number of blocked sunrays in winter. 44
Figure 6.48: Filtering optimal solutions by adding acceptable range of variable values. 45
Figure 6.49: Printout of the variable values from the filtered selection. 45
Figure 6.50: Visualization of how the shading elements look like when applied to the building. 46
Figure 6.51: Visualization of sunrays passing through the shading in winter months before the optimization process. Yellow color indicates the sunrays passing through the shading, grey color indicates sunrays blocked by the shading. For winter months we want to maximize the number of sunrays passing through, meaning we want to see more of yellow sunrays. 47
Figure 6.52: Visualization of sunrays passing through the shading in winter months after the optimization process. Yellow color indicates the sunrays passing through the shading, grey color indicates sunrays blocked by the shading. 47

SLOVARČEK

Generativni pristop	*Generative Design*
Proceduralno modeliranje	*Procedural modeling*
Parametrično modeliranje	*Parametric modeling*
Spremenljivke	*Variables*
Namenska/kriterialna funkcija	*Objective function*
Kriteriji	*Objectives*
Pogoji	*Conditions*
Meje	*Bounds/Boundaries*
Genetski algoritem	*Genetic algorithm*
Selekcija	*Selection*
Križanje	*Crossover*
Mutacija	*Mutation*
Strojno učenje	*Machine Learning*
Seznam elementov	*List of elements*
Množice elementov	*Array of elements*

1 UVOD

V zadnjih letih je opazen vedno hitrejši tehnološki razvoj. Računalniki postajajo zmogljivejši in pametnejši. Vedno pogosteje se srečujemo z izrazi kot so umetna inteligenca, strojno učenje, nevronske mreže, ipd. Tudi v gradbeništvu so bistveno napredovale možnosti uporabe različnih metod. To se izraža v vedno večji uporabi digitalnih orodij, ki jih vpeljujemo v gradbeništvo. Že do sedaj se je delo projektantov drastično spremenilo. Zaradi menjave svinčnika in papirja s CAD orodij se je proces dokumentiranja stavb precej poenostavil in pohitril. Prav tako, zaradi menjave CAD orodij z informacijskimi modeli se spreminja tudi sam proces načrtovanja. Vedno manjša je potreba po risarjih, načrtovanje se zliva z dokumentiranjem in je bistveno bolj celovito, razne stroke delujejo bolj povezano in koordinirano. Orodja za analizo so računalniško podprta, kar bistveno poenostavi in pospeši preverjanje rešitev.

Še vedno pa se mora rešitev, ki jih računalniki pomagajo analizirati in dokumentirati, spomniti inženir ali arhitekt. Ta magistrska naloga je o tem, kako bi računalniki pomagali tudi pri ustvarjanju inženirskih in arhitekturnih rešitev.

1.1 Opis problema

Trenutno se pri projektiranju večinoma še vedno uporablja »ročni« pristop.. Projektant mora razumeti problem, njegove kriterije in omejitve. Rešitve ustvari glede na svoje izkušnje, podobne pretekle primere, domišljijo, seveda v odvisnosti od vhodnih pogojev, omejitev in kriterijev. Omejitve določijo, katere rešitve so sploh dopustne, kriteriji pa omogočajo kvalitativne primerjave med različnimi dopustnimi rešitvami. V samem procesu je projektant tisti, ki (pred)postavi rešitev. Računalnik pomaga pri njeni predstavitvi (BIM) in pri njeni analizi (npr. številni programi za analizo konstrukcij). Ne pomaga pa pri ustvarjanju rešitev. To odpira možnosti iskanja novih pristopov.

Tak pristop je algoritmični, ki preusmeri osredotočenost na direktno sintezo rešitev na definiranje problema z jasnimi zakonitostmi. Zastavljeni problem lahko nato prepustimo v reševanje algoritmom ki so sposobni generirati in potem analizirati veliko število rešitev. Eden izmed teh pristopov je generativni pristop, ki temelji na konceptu *generiraj* in *testiraj*. Omenjeni pristop bomo spoznali v okviru magistrske naloge

1.2 Namen in cilj

Namen magistrske naloge je raziskati generativni pristop, pokazati uporabo le-tega na praktičnem primeru, oceniti prednosti in slabosti glede na tradicionalne pristope, opozoriti na priložnosti uporabe pristopa na drugih področjih in nakazati potrebe za nadaljnje raziskave.

Hipoteza naloge je, da z generativnim pristopom lahko projektant s pomočjo računalniške tehnologije v krajšem času preveri večje število alternativnih rešitev in da je končna rešitev zato lahko boljša kot bi bila brez uporabe te tehnologije. K odgovoru na hipotezo se bomo vrnili v poglavju »Zaključki«.

Pomen naloge je v tem, da je generativni pristop pomemben korak v smeri umetne inteligence, strojnega učenja, optimizacije in podobnega, kar se bo vse v kratkem dotaknilo tudi gradbeništva. Raziskovanje te tehnologije je spodbujeno tudi z dejstvom, da je vgrajena v nekatera orodja, ki jih inženirji široko uporabljajo in je zato raziskava lahko povsem realna in praktična.

1.3 Metode dela

Magistrsko delo dokumentira raziskovanje, ki je potekalo na osnovi reševanja praktičnega problema, kjer je bila uporabljena generativna metoda načrtovanja. Prek uporabe programskih orodij in procesov se bo s pomočjo literature in dokumentacije raziskovalo ozadje delovanja algoritmov in pogojev uporabljenega pristopa. Pri tem magistrska naloga predpostavlja znanja informacijskega modeliranja, računalniško podprtega modeliranja, algebre in vizualnega programiranja.

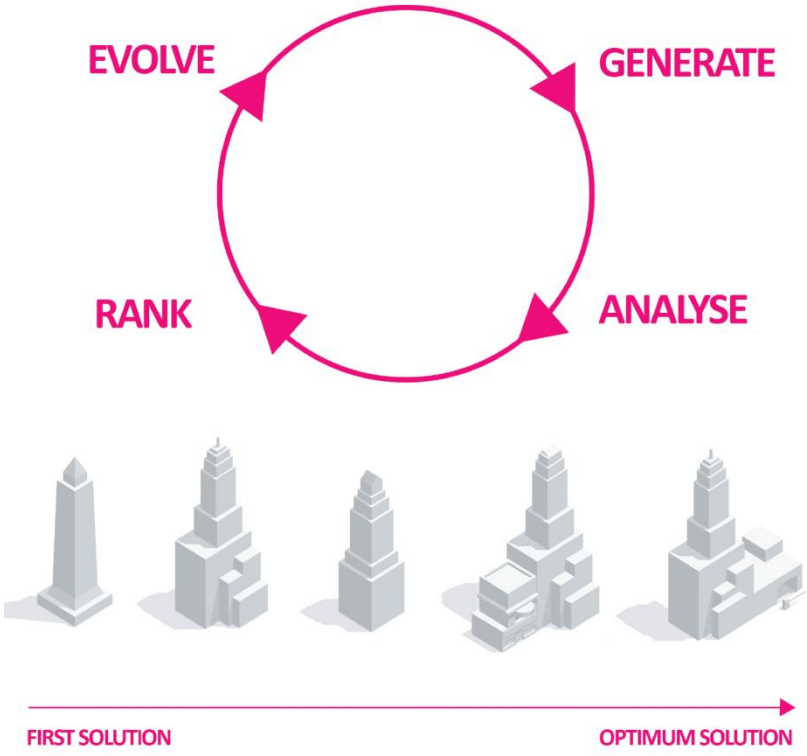
Praktičen problem se bo prevedlo in poenostavilo v obliko, ki je uporabna za orodja, ki so na voljo. Nastavitev problema predstavlja najbolj obsežen in zahteven problem, saj služi kot podlaga generativnega pristopa, zato bo v nalogi podrobno predstavljena. Problem se bo reševalo z generiranjem alternativ in iskanjem optimalnih rešitev s pomočjo genetskih algoritmov.

2 GENERATIVNI PRISTOP

Generativni pristop predstavlja sodelovanje projektanta z inteligentnim orodjem, ki lahko generira in analizira veliko število alternativ ter iterativno išče optimalno rešitev. V okviru načrtovanja projektanti definirajo in zastavijo problem za reševanje s pomočjo algoritmov, ki pomagajo pri iskanju optimalne rešitve ali raziskovanju prostora rešitev z namenom izboljšanja končnega rezultata. [1]

Zajema algoritme za generiranje rešitev, parametrično modeliranje in računalniško podprto modeliranje. Pri tem je posebnost, da algoritmi generirajo večje število rezultatov in iščejo optimalno rešitev glede na podane zahteve projektanta. Prednost pred ročnim pristopom je predvsem ta, da je računalnik sposoben generirati, analizirati in oceniti za nekaj velikostnih razredov večje število rešitev od človeka. [1]

Uporaba algoritmov nam pomaga pri reševanju in avtomatizaciji kompleksnega problema, ki ga je težko reševati direktno brez uporabe zmogljivih računalnikov. Pri tem je največ truda potrebno vložiti v poenostavitev in definiranje realnega problema v logični problem, ki ga algoritmi razumejo in znajo rešiti glede na podane zahteve projektanta. To predstavlja nov pristop pri načrtovanju zgradb, saj se preusmerimo iz ročnega reševanja problema na definiranje procesa, ki algoritmom pove, kako problem rešiti. [1]



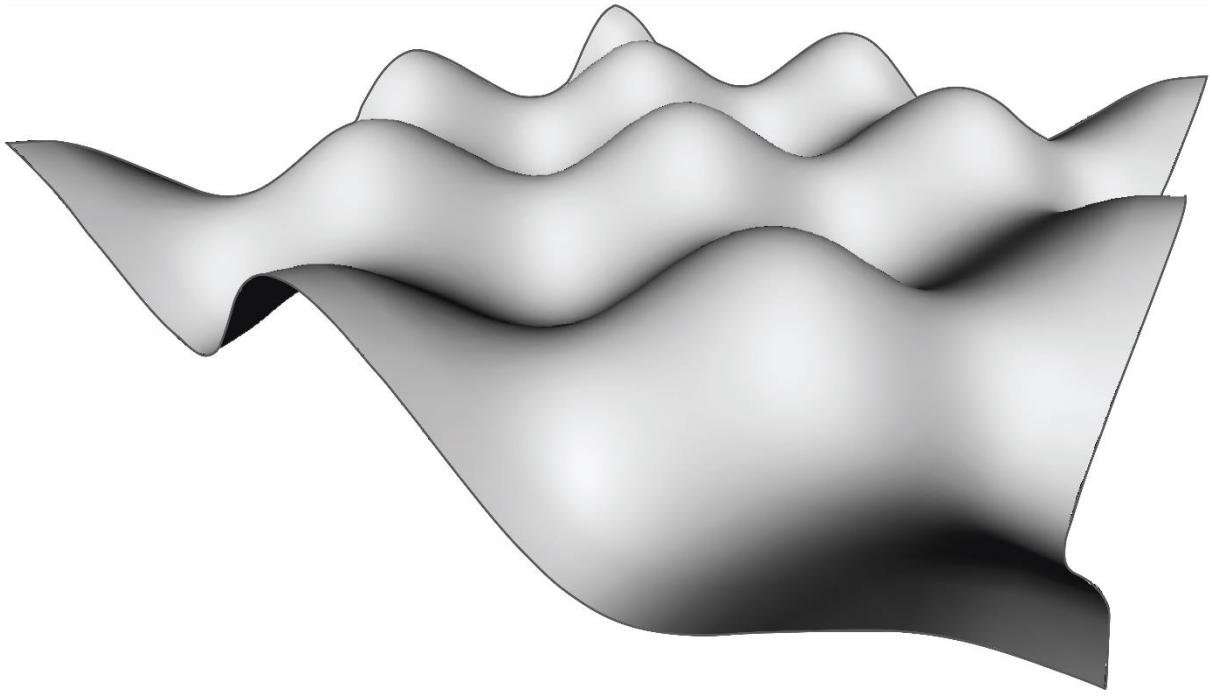
Slika 2.1: Princip generativnega pristopa temelji na generiranju in testiranju alternativ. [1]
Figure 2.1: Basic idea of the generative design, which is based on generate and test principle. [1]

3 PRISTOPI MODELIRANJA

Metode modeliranja, torej ustvarjanja rešitev, lahko razdelimo na dve zvrsti: metode prvega reda in metode drugega reda. Metode prvega reda tvorijo rezultate preko zaporedja ukazov modelarja oziroma avtorja načrta ali modela, torej lahko rečemo, da so to ročno narisani modeli ali risbe. Idejo, ki se je porodila v njegovi glavi, projektant prenese na papir ali v digitalni model. Pri metodah drugega reda ni več direktnega modeliranja, temveč gre za definiranje navodil, algoritmov ali pravil, ki tvorijo rešitve. Pristopi drugega reda običajno zahtevajo drugačen pogled na modeliranje in še bolj na inženirsko znanje, saj nas formalizacija, torej računalniško uporabna oblika problema in znanj, ki je potrebno za njegovo rešitev. Za uporabo teh metod je običajno potrebno znanje programiranja. Metodam drugega reda lahko rečemo tudi proceduralno modeliranje, saj modelirajo procedurno. [2]

3.1 Proceduralno modeliranje

To je širši pojem in zajema več tehnik modeliranja, med katerimi so tudi parametrično modeliranje in računalniško podprto modeliranje. Modeli se tvorijo preko eksplicitno podanih navodil ali algoritmov. Tvorjenje geometrije poteka samodejno glede na podana pravila ali funkcije. Kot primer lahko privzamemo enostavno funkcijo *Sin(x)Sin(y),* ki ustvari površino glede na spremenljivki *x* in *y*. [2]

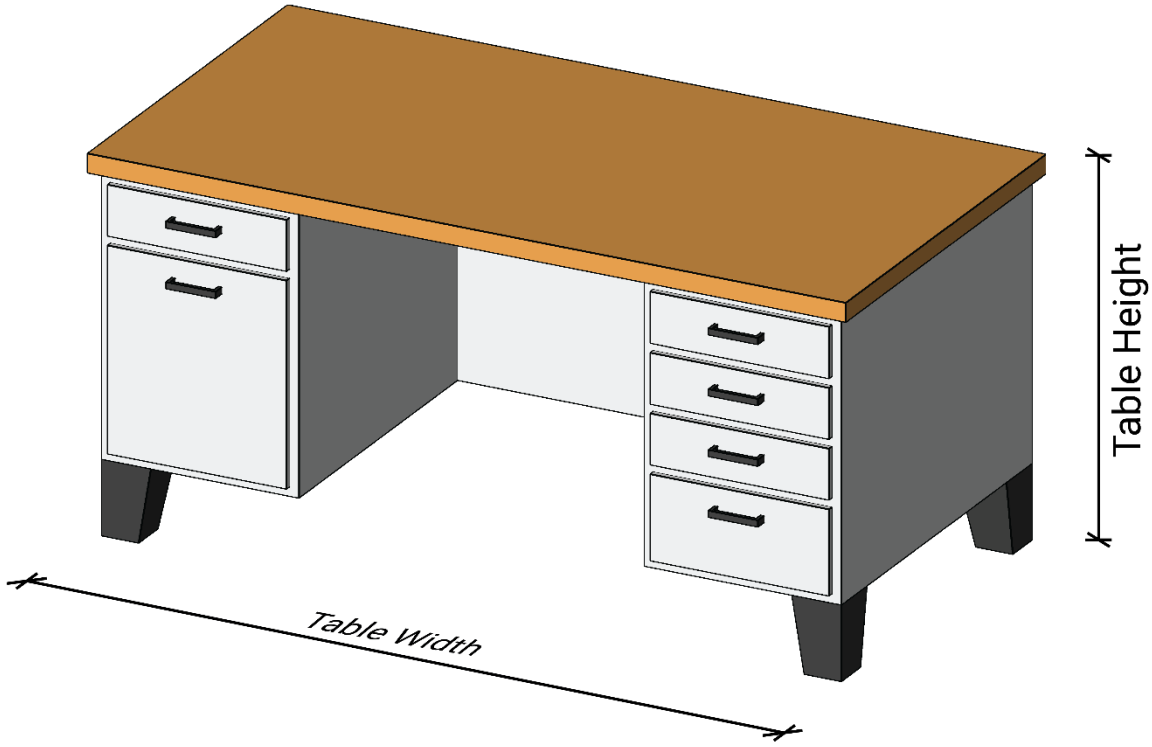


Slika 3.1: Enostavni primer proceduralnega modela, ki ga dobimo s funkcijo Sin(x)Sin(y)

Figure 3.1: Simple example of procedural model, which is generated by function Sin(x)Sin(y)

3.2 Parametrično modeliranje

V osnovi lahko pojem *"parametričen"* privzamemo kot spremenljivi opis lastnosti, pri čem s spreminjanjem parametrov vplivamo na obliko ali lastnosti, kot so npr. dolžina in širina. Primer so parametrične knjižnice v programu Revit. [3]



Slika 3.2: Primer parametrične knjižnice v programu Revit.

Figure 3.2: Example of parametric family in Revit.

V širšem pojmu parametrično modeliranje pomeni zaporedje enačb, ki tvorijo geometrijo ali model. Parametrični modeli so tako sestavljeni iz parametrov, zaporedja enačb in njihovih rezultatov. Parametri nam predstavljajo neodvisne spremenljivke, ki jih pretvorimo preko enačb v odvisne spremenljivke. To omogoča samodejno posodabljanje rezultatov glede na spreminjanje parametrov, kar predstavlja dinamični proces pri raziskovanju različnih rešitev. [2] [3]

3.3 Računalniško podprto modeliranje

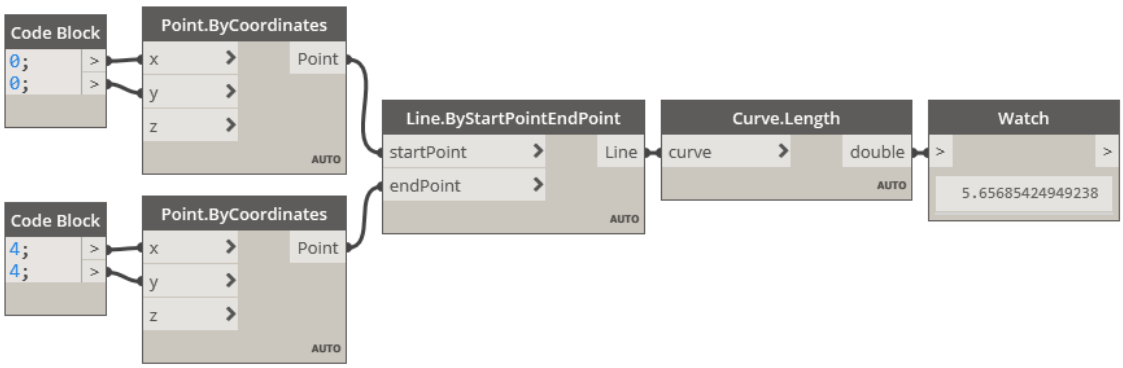
Računalniško podprto modeliranje predstavlja širšo področje, v katerem je zajeto parametrično modeliranje. Modele izdelujemo z opisom procesa preko zbirke ukazov oziroma algoritmov. Namesto da neposredno izdelujemo modele, računalniku podamo navodila kako model zgraditi. Preidemo iz osredotočenosti na končni model na osredotočenost opisa modela. To nam omogoča prilagodljivo okolje, ki je potrebno za generativni pristop, saj lahko ukaze prilagajamo, dodajamo vmesne korake in izvlečemo lastnosti oziroma podatke o modelu. [4] [5]

Kot primer si lahko pogledamo izris preproste geometrije:

1. *Nariši točko v koordinati (0,0)*
2. *Nariši točko v koordinati (4,4)*
3. *Poveži točko (0,0) in točko (4,4) z ravno črto*
4. *Izračunaj dolžino krivulje*

Algoritem nam bo izrisal ravno črto med podanima točkama pri tem pa izpisal lastnost krivulje. Z eksplicitnimi ukazi lahko po tem principu opišemo različne vrste problemov. Pri tem uporabljamo programe, ki prepoznavajo ukaze in jih izvedejo. Dva od teh programom sta Dynamo in Grasshopper, v katerima programiramo vizualno.

Vizualno programiranje je gradnja algoritmov oziroma zaporedja ukazov preko grafičnega povezovanja ukazov. Pri tem jih moramo med seboj logično povezati. [4]



Slika 3.3: Primer vizualnega programiranja.

Figure 3.3: Example of visual programming.

4 OPTIMIZACIJA

Optimizacija predstavlja pomembno vlogo na različnih področjih, kot so strojno učenje, nevronske mreže, umetna inteligenca, idr. Področje raziskovanja algoritmov za reševanje je obsežno in pri tem zajema veliko število možnih algoritmov za iskanje rešitev. Naloga projektantov, ki uporabljajo omenjena orodja, je prepoznati zahteve algoritmov in prevesti obravnavan problem v matematični zapis, ki ga matematični programi razumejo in znajo rešiti. Z dobljenimi rezultati projektant lažje razume problem in kako različni parametri vplivajo nanj ter tako lažje poišče optimalne rešitve. Pri kompleksnejših problemih ni točne optimalne rešitve, ampak mora projektant pretehtati katerim kriterijem želi bolj zadostiti in katera rešitev mu na koncu najbolj ustreza. [6]

Problem optimizacije je sestavljen iz štirih ključnih delov:

- Spremenljivke: $\bar{x} = \{x_1, x_2, \dots, x_n\}$
- Namenska funkcija: $F_{(x_i)} \rightarrow min/max$ $\quad i = 1,2,\dots,n$
- Meje: $a_i \leq x_i \leq b_i$ $\quad i = 1,2,\dots,n$
- Pogoji: $g_j = 0\ ali\ g_j \leq 0\ ali\ g_j \geq 0$ $\quad j = 1,2,\dots,m$

Spremenljivke predstavljajo vhodne parametre, ki jih v analizi spreminjamo, da dobimo različne rezultate, med kateri poiščemo optimalne.

Namenska funkcija dodeli numerično oceno vsaki rešitvi, odvisni od spreminjanja vhodnih parametrov. Glede na ocene, jih razvrsti in poišče minimume ali maksimume.

Meje predstavljajo omejitev dovoljenih vrednosti za spremenljivke. Te tvorijo prostor rešitev.

Pogoji povedo, katere rešitve so sprejemljive.

Za reševanja problema optimizacije z algoritmi, je potrebno podati pravila kako naj ocenijo in razvrstijo različne rezultate. Običajno to predstavlja zahteven in zelo pomemben del optimizacije. Kako vhodne neodvisne spremenljivke z neko funkcijo spremenimo v izhodne odvisne spremenljivke, ki jih lahko razvrstimo od najboljše do najslabše. Želimo poiskati objektivno numerično oceno, s katero algoritmi znajo operirati.

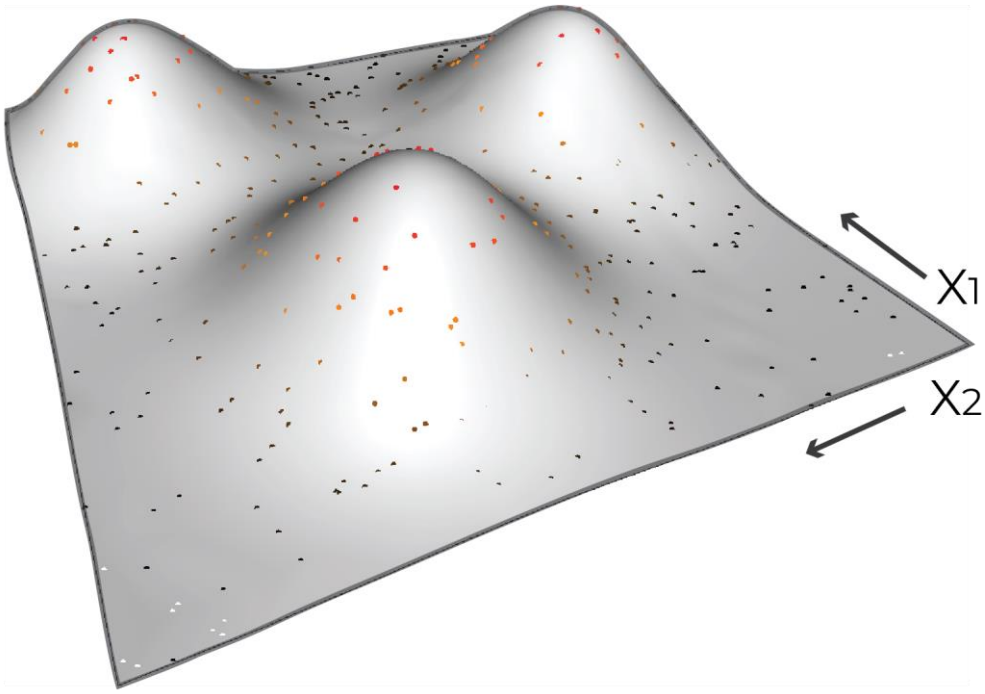
Za reševanje optimizacijskih problemov obstaja mnogo algoritmov, v nadaljevanju si bomo pogledali evolucijske algoritme, izmed katerih so genetski algoritmi pogosto uporabljeni za reševanje optimizacije.

4.1 Genetski algoritmi

Genetski algoritem deluje po naslednjih korakih:

1. Naključno generiraj začetno populacijo
2. Vsakemu rezultatu podaj oceno
3. Ponavljaj naslednje korake do konca analize:
    a. Selekcija: izberi del populacije z najboljšimi ocenami z namenom reprodukcije za naslednjo generacijo
    b. Generiraj novi del populacije preko reprodukcije s križanjem in mutacijo
    c. Ponovno vsakemu rezultatu podaj oceno
    d. Nadomesti najslabše rezultate z novimi rezultati

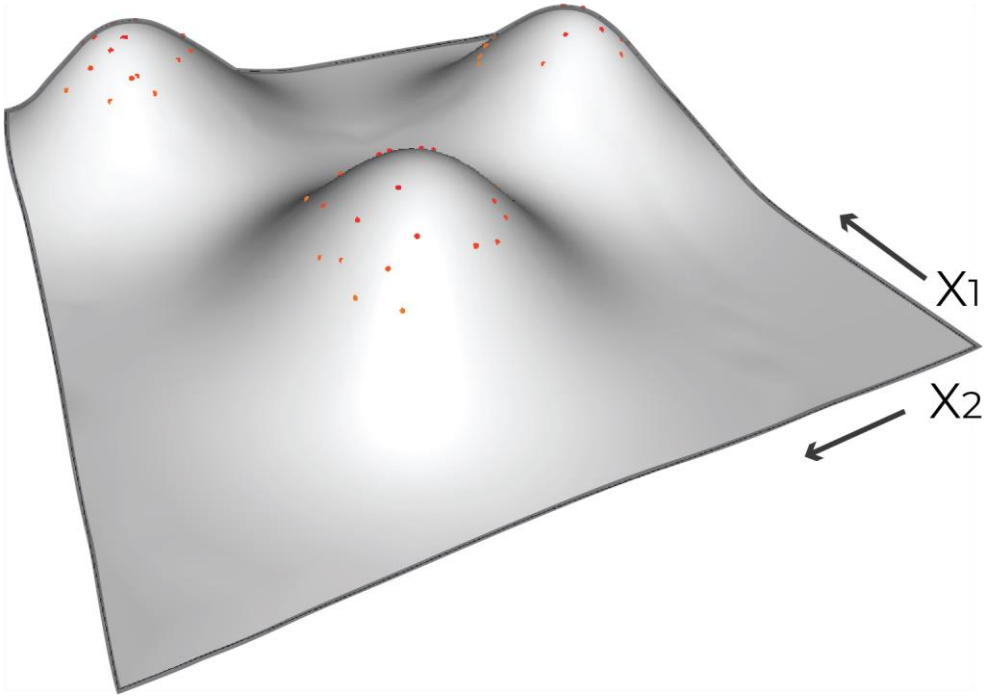
Posebnost genetski algoritmov je, da se vsaka generacija izboljšuje, to nam zagotavlja algoritem, ko iz množice rezultatov vedno najslabše nadomesti z boljšimi rezultati. Preko tega postopka iščemo optimalne vrednosti problemov. [7]



Slika 4.1: Naključno generiranje začetne populacije.

Figure 4.1: Random generation of the initial population

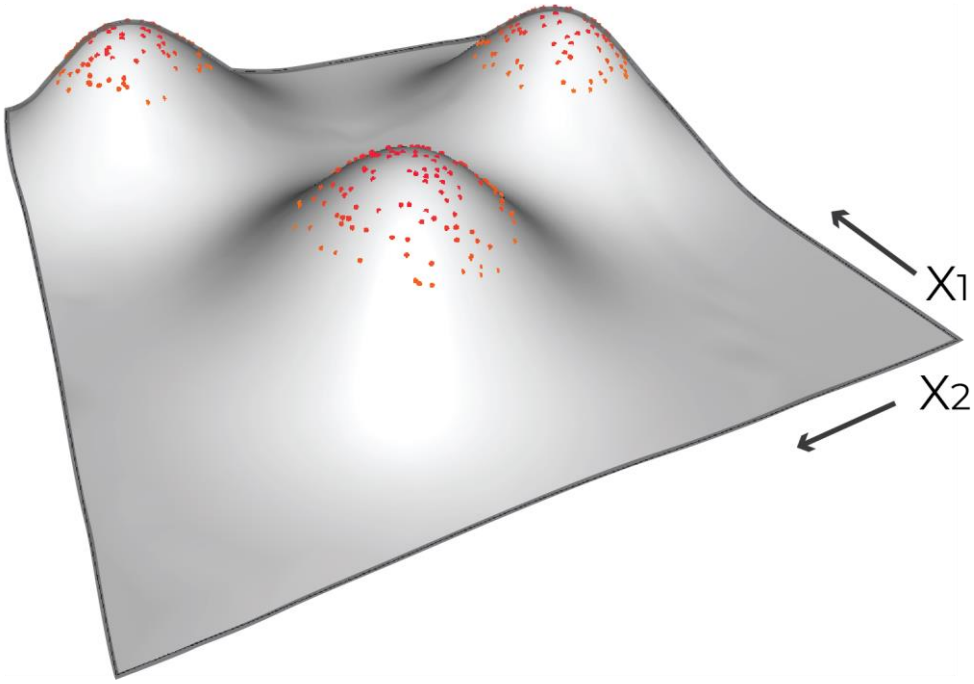
Selekcija izbere del populacije, ki ima najboljšo oceno, za tvorbo naslednje generacije. Obstaja več vrst selekcije, ena izmed njih je elitizem, ki takoj najboljši delež prenese v naslednjo generacijo. [7]



Slika 4.2: Izbor deleža populacije z najboljšimi ocenami.
Figure 4.2: Selection of the best-fit part of the population

Križanje generira nove rezultate iz križanja genov populacije izbrane iz selekcije. Križanje naključno izbere gene, ki se križajo med dvema primerkoma. Na enostavnem primeru prikazanemu na slikah, to pomeni, da naključno izmenjuje gene, ki predstavljajo vrednosti spremenljivk X1 in X2. [7]

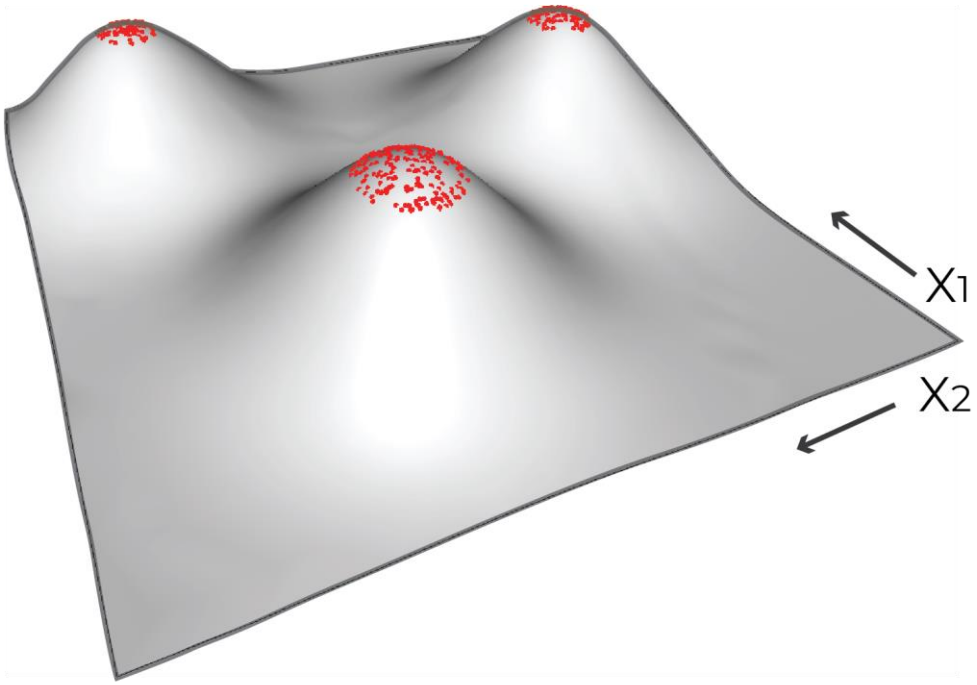
Mutacija je naključni dogodek s katerim naključno spremenimo vrednost gena oziroma spremenljivke. Z mutacijo preprečujemo monotonost reševanja in iščemo globalne optimalne vrednosti v primeru, da se algoritem reproducira le v okolici lokalne optimalne vrednosti. [7]



Slika 4.3: Naslednja iteracija.

Figure 4.3: Next iteration.

Z vsako naslednjo generacijo se nam rešitve izboljšujejo in prehajajo proti optimalnim vrednostim. Da se nam rešitve ne slabšajo, nam to zagotavlja selekcija, saj vedno izbere delež najboljših rešitev, kar pomeni, da izbrane vrednosti oziroma ocene vedno rastejo.



Slika 4.4: Z vsako novo generacijo se zelo približamo optimalnim rezultatom

Figure 4.4: With each new generation we are closer to optimal results

5 PROGRAMSKO ORODJE

5.1 Autodesk Revit

Autodesk Revit je programsko orodje v kateremu je omogočeno informacijsko modeliranje zgradb (BIM). Podpira sodelovanje različnih disciplin v skupnem okolju v katerem lahko projektanti sodelujejo od zasnove do končne rešitve. V programskem okolju izdelujemo parametrične modele stavb z vključenimi informacijami. [8]

5.2 Dynamo

Dynamo je programsko okolje, ki omogoča vizualno programiranje in služi kot dodatno orodje projektantom, ki ga lahko prilagajajo njihovim potrebam. Uporabnikom omogoča sestavljanje logičnih ukazov, preko katerih lahko upravljajo z informacijami ali izdelujejo modele z računalniško podprtim modeliranjem. Dynamo deluje kot samostojni program ali kot dodatek programu Revit. [9]

V procesu generativnega pristopa nam omogoča računalniško podprto modeliranje, preko katerega lahko parametrično opredelimo probleme za reševanje z algoritmi, ki nam generirajo rešitve. [9]

5.3 Project Refinery

Project Refinery je programski dodatek za program Dynamo. Orodje nam omogoča generativni pristop, saj privzame spremenljivke in namenske funkcije iz okolja Dynamo ter v povezavi z njim generira rešitve preko izbranih algoritmov. Dodatno vsebuje tudi uporabniški vmesnik, ki omogoča pregledno prikazovanje prostora rešitev in sortiranja med rešitvami. [10]

Opozoriti je potrebno, da program Project Refinery deluje kot dodatek samostojnemu programu Dynamo Sandbox in ne programskemu orodju Dynamo, ki je dodatek ostalim programom. Razlika je v tem, da Dynamo Sandbox deluje kot samostojen program in ni v povezavi z ostalimi programi kot so Revit, Civil3D. To pomeni, da vsi ukazi, ki se sklicujejo na te programe, tukaj ne delujejo. V našem primeru so to ukazi, ki privzamejo geometrijo iz okolja Revit, kot so *Select Face* in *Select Model Elements*. Vso geometrijo je potrebno tvoriti v programu Dynamo ali pa uporabimo posebni ukaz, ki se imenuje *Data.Remember*. S tem ukazom shranimo podatke izbranega ukaza in nadaljujemo funkcijo s shranjenimi podatki. [10]

6 ŠTUDIJA UPORABE GENERATIVNEGA PRISTOPA NA PRAKTIČNEM PRIMERU

Generativni pristop je uporabljen pri načrtovanju kompleksne fasade oziroma elementov za senčenje. Obravnavana bo stavba, ki je bila zasnovana v okviru mednarodnega projekta Project Based Learning na Univerzi Stanford. Stavbo je v okviru študija Project Based Learning, projektirala skupina Team Pacific 2019, v kateri je kot gradbeni inženir in BIM koordinator sodeloval tudi avtor te magistrske naloge. V okviru projekta so nastajali digitalni modeli stavbe in je kot taka primerna za študijski primer.

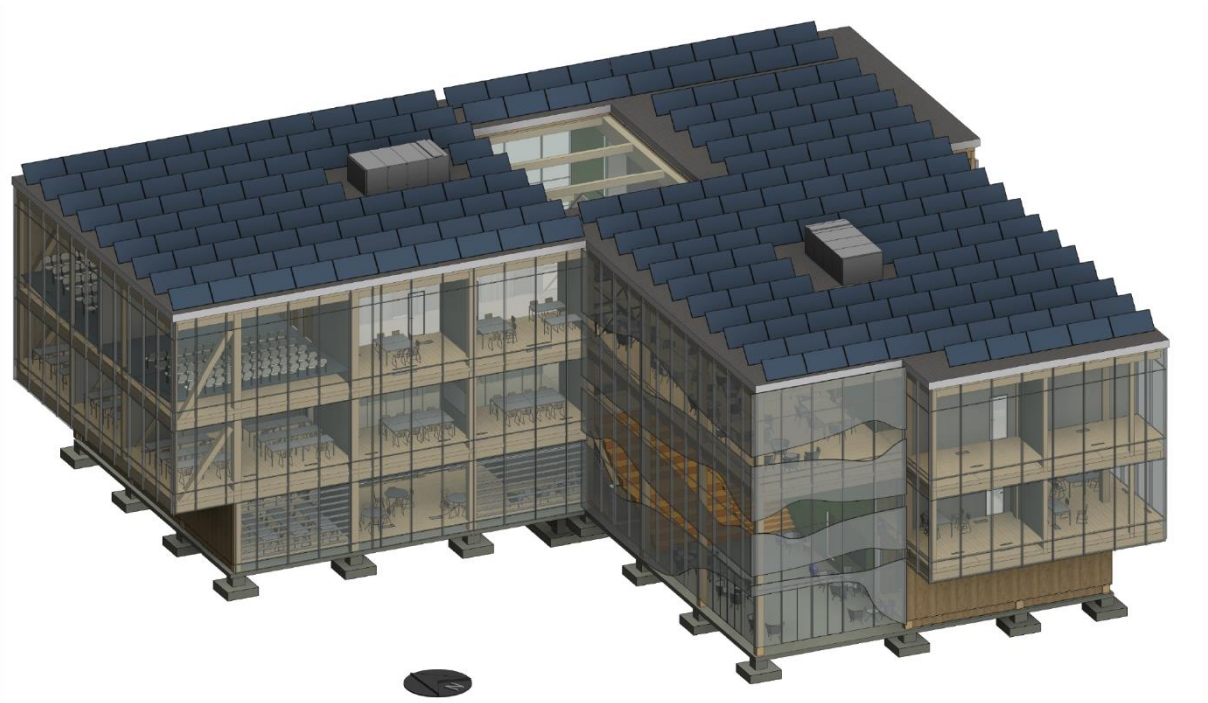
Slika 6.1: Rešitev uporabljena na mednarodnem projektu na univerzi Stanford. [11]
Figure 6.1: Render of the solution used on the international project on Stanford University. [11]

6.1 Predstavitev problema

V zadnjem desetletju daje družba čedalje večji poudarek trajnostni gradnji in zmanjšanju porabe energije. Zaradi velike površine zastekljenih površin na obravnavanem objektu, je bilo potrebno poiskati rešitev za zmanjšanje energetskih izgub. Za senčenje so bili izbrani nepomični vodoravni elementi, ki bi poleg svoje prvotne vloge ponazarjali tudi naravne oblike. Navdih za obliko so bila lesena vlakna. V preliminarnih analizah je bila uporabljena virtualna resničnost za določitev smeri elementov. Ugotovljeno je bilo, da najbolj ustrezajo vodoravni nepremični elementi. V podrobnejši analizi želimo modelirati elemente in pri tem poiskati optimalno obliko, ki bo zagotavljala učinkovito senčenje.

Zahteva je, da se zmanjša energijske izgube zaradi hlajenja poleti in zaradi gretja pozimi. Poleti želimo čim manj direktnega pretoka svetlobe v notranjost in s tem segrevanja prostorov, pozimi pa želimo čim več pretoka svetlobe v notranjost, ker bo segrevala prostore. Zanima nas tudi cena

(oziroma poraba materiala), ki jo želimo zmanjšati. Ena od možnih rešitev je na sliki 6.1. To seveda ni edina rešitev. Vprašanje je, kakšno bi bilo optimalno število in širina elementov za senčenje.



Slika 6.2: Obravnavani objekt na katerem želimo steklene površine pokriti z elementi za senčenje.
Figure 6.2: Example building on which we want to cover glass surfaces with shading elements.

6.2 Metode reševanja

Izhodišče za izdelavo računskega in parametričnega modela kompleksne fasade je informacijski model stavbe v programu Revit. Izbrane geometrije elementov zaradi kompleksnejše oblike ni možno modelirati ročno, temveč jo je potrebno modelirati preko računalniško podprtega modeliranja. V ta namen uporabimo programsko orodje Dynamo, ki je orodje vgrajeno v programu Revit.

Pri načrtovanju uporabimo generativni pristop, preko katerega želimo raziskati prostor rešitev in poiskati optimalno rešitev. Generativni pristop je omogočen s Project Refinery, ki je programski dodatek za Dynamo. Pri iskanju optimalne rešitve se uporabijo genetski algoritmi, ki z generiranjem rezultatov in iteracij iščejo prostor optimalnih rešitev. Rešujemo torej matematični problem optimizacije, poleg tega pa želimo tudi izdelati geometrijski model elementov za senčenje.

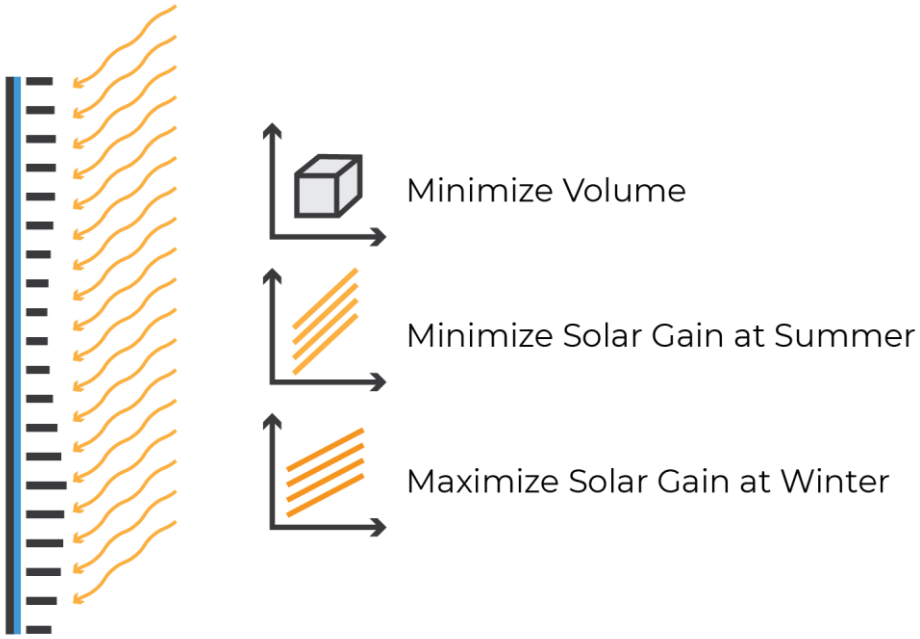
Cilj analize je poiskati vrednosti vhodnih spremenljivk, pri katerih lahko ocenimo, da imamo optimalno rešitev. Odvisno od problema imamo lahko optimalnih rešitev več, poleg tega pa je rezultat odvisen tudi od projektanta, saj le-ta analizira prostor rešitev in izbere njemu najbolj ustrezno. Lahko se tudi zgodi, da matematični programi ne najdejo optimalne rešitve, projektant pa mora kljub temu končno rešitev podati.

6.3 Opredelitve in poenostavitve modela

Da lahko uporabimo optimizacijske algoritme moramo problem opredeliti tako, da bo vseboval vhodne spremenljivke, namensko funkcijo, izhodne spremenljivke in pogoje. Ko je problem zapisan v rešljivi obliki, ga lahko prepustimo v reševanje izbranemu algoritmu. Ta zahteva definirano namensko funkcijo, ki različnim vhodnim spremenljivkam dodeli prepoznavno oceno (običajno numerično vrednost), ki predstavlja izhodne spremenljivke. Te želimo maksimizirati ali/in minimizirati. V začetku je zelo pomembno, da se problem razčleni in kritično presodi, katere parametre v analizi uporabimo kot vhodne spremenljivke in prek katerih kriterijev ocenjujemo rešitve. Pri tem želimo zmanjšati število vhodnih in izhodnih spremenljivk.

Vhodne spremenljivke so:

- Število elementov.
- Širina elementov za senčenje.



Slika 6.3: Iščemo optimalne vrednosti števila in širine elementov za senčenje, glede na porabo materiala ter glede na toplotne pribitke zaradi segrevanja v zimskih in poletnih mesecih.

Figure 6.3: We want to find optimal values for the number and the width of the shading elements, based on the material needed and the solar gains during the winter and summer months.

Izhodne spremenljivke so:

- Količina porabljenega materiala
- Število sončnih žarkov v poletnih mesecih, ki jim je preprečen prehod v notranjost
- Število sončnih žarkov v zimskih mesecih, ki jim je preprečen prehod v notranjost

Namenska funkcija je:

- Zmanjšati porabo materiala.
- Povečati število sončnih žarkov v poletnih mesecih, ki jim je preprečen prehod v notranjost.
- Zmanjšati število sončnih žarkov v poletnih mesecih, ki jim je preprečen prehod v notranjost.

Meje so:

- Omejitev največjega in najmanjšega možnega števila elementov.
- Omejitev največje in najmanjše širine elementov.

Če razmislimo, bi lahko količino porabljenega materiala nadomestili s stroški, ki bi jih zajeli kot pogoj v analizi, kjer stroški ne smejo prekoračiti proračuna za izdelavo fasade. Lahko bi analizirali tudi stroške skozi življenjsko dobo, kjer bi zajeli prihranke zaradi segrevanja in hlajenja v daljšem časovnem obdobju glede na začetne stroške izdelave. Pri tem bi povečali kompleksnost modela oziroma izračuna.

Sama razčlenitev realnega problema oziroma zapisa problema v matematično obliko predstavlja najbolj obsežen del. Reševanje problema lahko opravi matematični program, ki gleda le na začetne neodvisne in končne odvisne spremenljivke. Pri tem se število vhodnih in izhodnih spremenljivk ter zahtevnost matematičnega zapisa odraža v času reševanja potrebnem za rešitev problema.

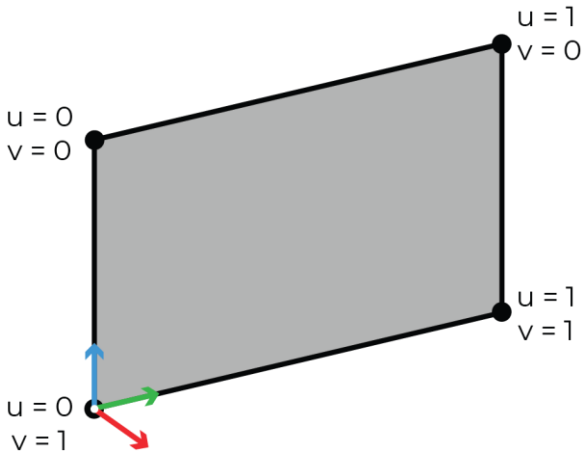
6.4 Izdelava modela

Model izdelujemo v programskem okolju Dynamo, v katerem gradimo model z vizualnim programiranjem. Pri tem model formiramo s sestavljanjem ukazov, ki jih lahko prilagajamo in pri tem spreminjamo obliko in lastnosti. V tem se izraža prednost računalniško podprtega modeliranja, saj lahko kadar koli popravimo modele in pri tem ponovno poženemo analizo. Tako lahko hitro in enostavno uporabimo že zastavljene modele za podobne ali nove probleme, nasprotno, če modeliramo ročno enkratne modele, te težje prilagodimo novim zahtevam.

V nadaljevanju bomo obravnavali del izbrane fasade. Želimo zgraditi model, ki bo vseboval vhodne in izhodne spremenljivke, pri tem pa želimo na koncu ustvariti geometrijo, ki jo lahko prenesemo v ostale programe.

6.4.1 Določitev površine fasade

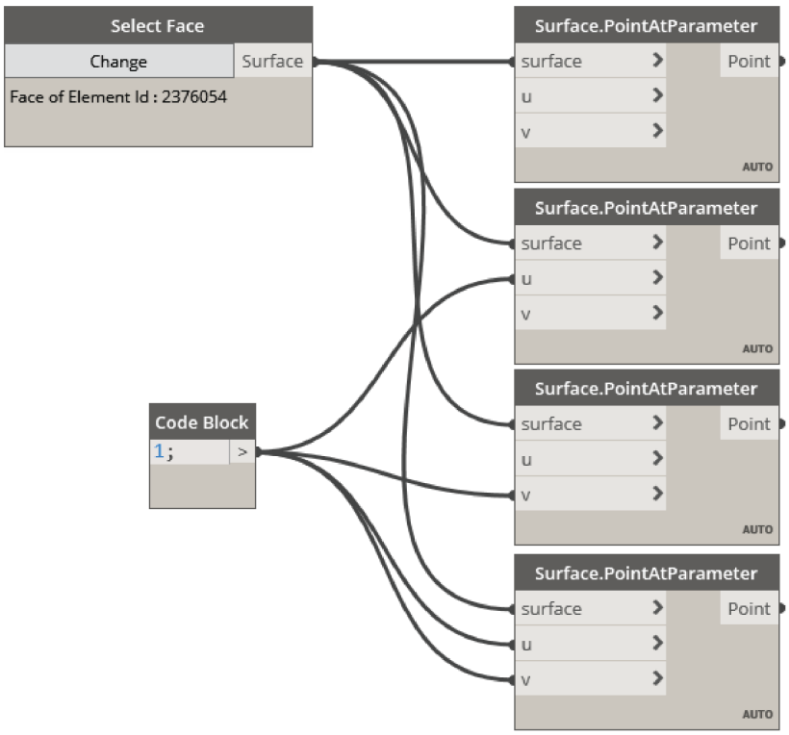
Izgradnjo modela pričnemo z izbiro površine fasade objekta, ki jo želimo analizirati. Na izbrani površini se bo modeliralo elemente za senčenje.



Slika 6.4: Obravnava površina in njene lastnosti. Barve vektorjev rgb ustrezajo smerem xyz.

Figure 6.4: Example surface and its parameters. Vector colors rgb correspond with the xyz directions.

Z ukazom *Select Face* prenesemo izbrano geometrijo iz okolja Revit v programsko okolje Dynamo. Za nadaljnjo obravnavamo moramo pridobiti lastnosti izbrane površine, natančneje robne točke in normalo ploskve. Robne točke pridobimo z ukazom *Surface.PointAtParameter* s katerim generiramo točko na ploskvi v odvisnosti od parametrov *u [0,1]* in *v [0,1]*. Robne točke dobimo z vsemi možnimi kombinacijami vrednostih 0 in 1.

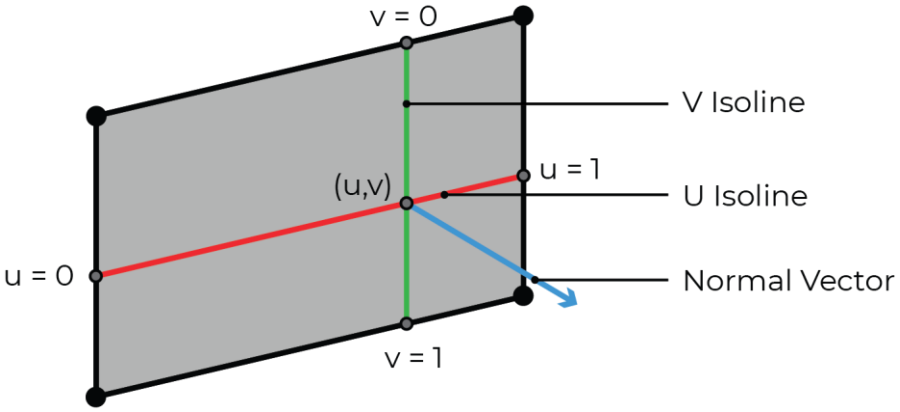


Slika 6.5: Izbira površine in določitev robnih točk.

Figure 6.5: Surface selection and defining border points.

6.4.2 Koordinatni sistem

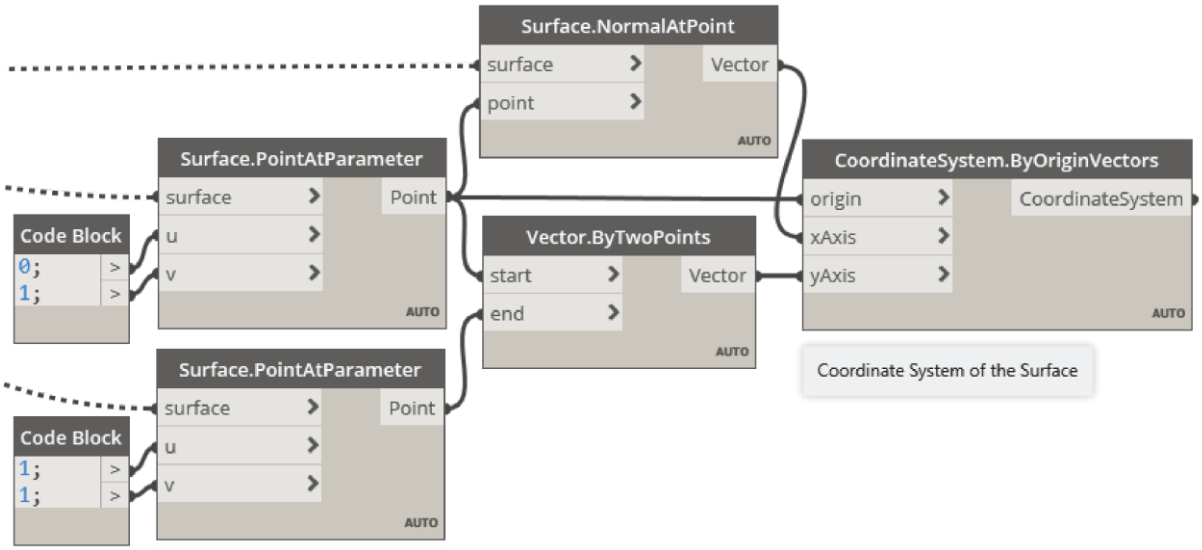
Koordinatni sistem je potreben za formiranje elementov, za katere želimo, da se tvorijo v obravnavi ravnini ali njenih vzporednih ravninah. Koordinatni sistem določimo iz lastnosti obravnavane ploskve.



Slika 6.6: Parametrične lastnost površine v programu Dynamo.

Figure 6.6: Surface parameter properties in Dynamo.

Nov kartezični koordinatni sistem tvorimo z ukazom *CoordinateSystem.ByOriginVectors*, v katerega podamo izhodiščno točko in dva smerna vektorja, ki predstavljata novi osi x in y. Kot smerni vektor za abscisno os je bila izbrana normala na ploskev, ki jo dobimo z ukazom *Surface.NormalAtPoint*. Ker je obravnavana površina ravna ploskev, točka, v kateri pridobimo normalni vektor, ni pomembna in lahko izberemo poljubno točko. Za Ordinatno os vzamemo vektor, generiran z ukazom *Vector.ByTwoPoints*, v katerega vnesemo dve robni točki v vodoravni ravnini.

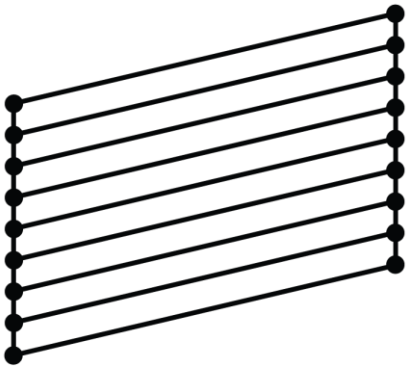


Slika 6.7: Določitev kartezičnega koordinatnega sistema za izbrano ploskev.

Figure 6.7: Definition of the Cartesian coordinate system for the selected surface.

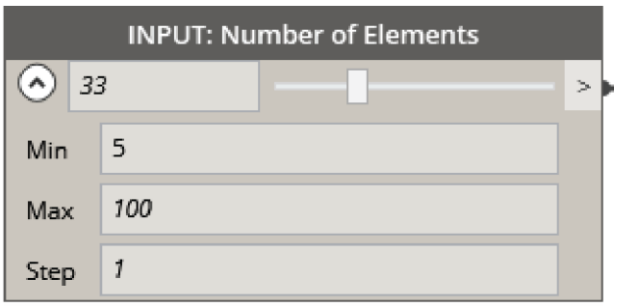
6.4.3 Zaporedje vodoravnih linij

Po izbrani ploskvi želimo razporediti elemente za senčenje. Te pričnemo sestavljati iz vodoravnih linij razporejenih v enakomernem zaporedju po ploskvi. Pred tem je potrebno razdeliti površino na enakomerno število delov, število katerih bo predstavljalo prvo vhodno spremenljivko.



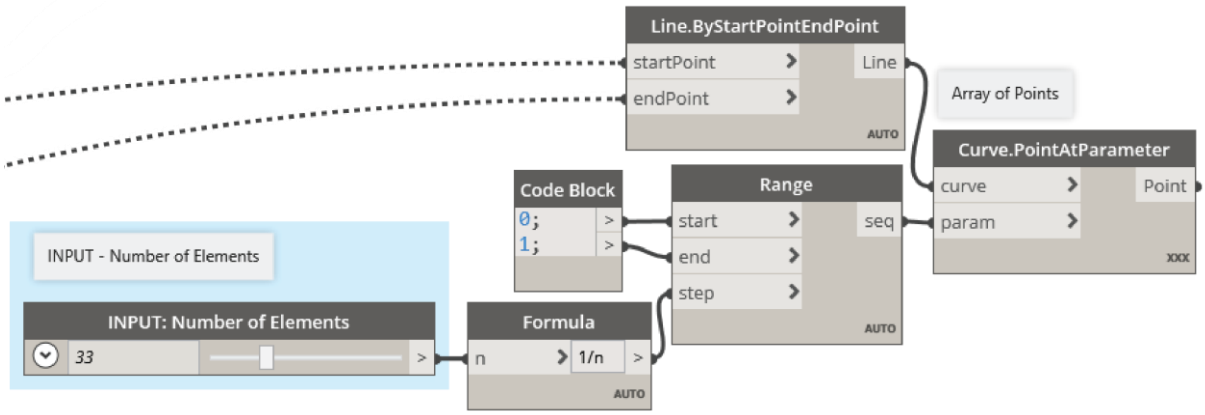
Slika 6.8: Prikaz zaporedja vodoravnih linij, ki predstavljajo število elementov.
Figure 6.8: Illustration of the array of lines, which represent the number of the elements.

Spremenljivko definiramo z ukazom *Integer Slider*, ki omogoča parametrično spreminjanje celega števila znotraj določenega območja. Izbrano območje predstavlja meje



Slika 6.9: Podajanje mejnega območja vhodne spremenljivke.
Figure 6.9: Defining the bounds of the input variales.

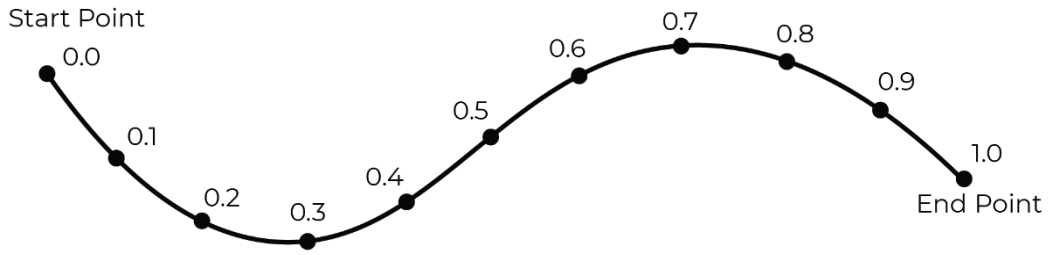
Iz podanega števila elementov se nato tvori enakomerno množico števil v obsegu 0…1 s korakom 1/n, kjer n predstavlja število elementov. Množico števil se ustvari z ukazom *Range*. Ta tvori zaporedje števil v določenem obsegu z določenem koraku.



Slika 6.10: Določitev parametra za število elementov za senčenje. Število predstavlja na koliko delov se razdeli navpični rob površine iz katerih bomo generirali elemente.

Figure 6.10: Input parameter for the number of elements for the shading. Input number is the number of divisions of a surface edge from which the elements will be modeled.

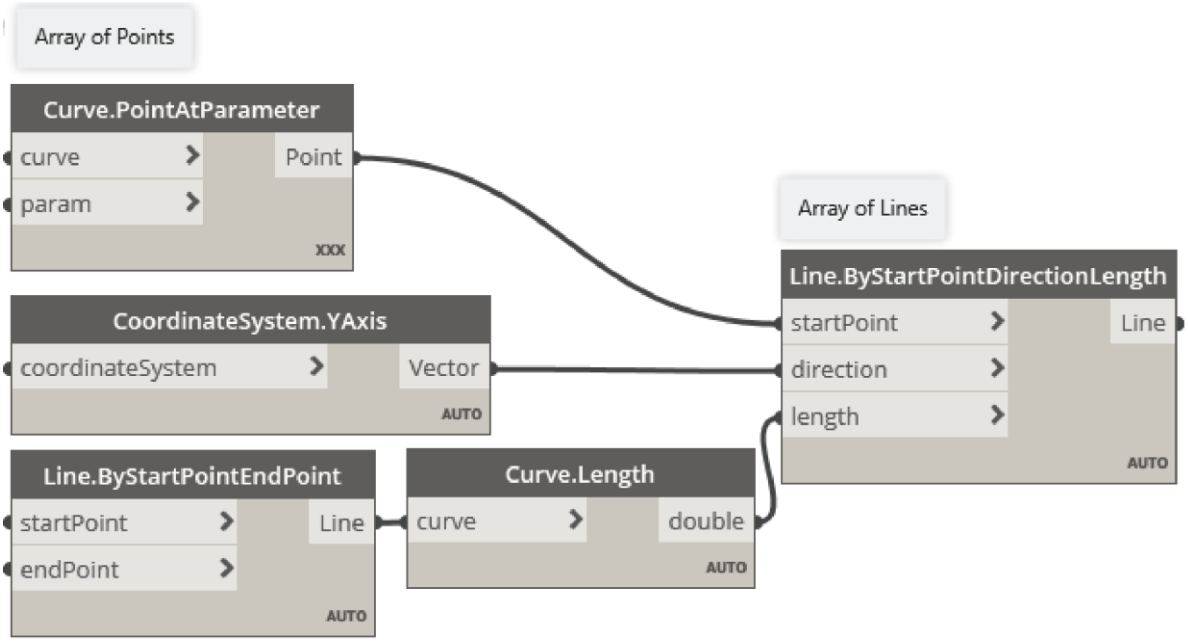
Iz enakomernega zaporedja števil tvorimo točke vzdolž zunanjega navpičnega robu ploskve z ukazom *Curve.PointAtParameter*. Ta nam tvori točke vzdolž izbrane krivulje glede na podane dolžinske parametre. Eden izmed parametrov krivulje v programu Dynamo je dolžinski parameter. Vrednost v območju [0,1] nam pove kje se nahajamo na krivulji. Vrednost 0 pomeni začetek krivulje in vrednost 1 konec krivulje.



Slika 6.11: Dolžinski parameter krivulje v programu Dynamo

Figure 6.11: Distance parameter of a curve in Dynamo

Enakomerno razporejene vodoravne linije vzdolž obravnavane ploskve tvorimo z ukazom *Line.ByStartPointDirectionLenght*. Ukaz zahteva vnos izhodiščnih točk, smeri in dolžine. Za izhodiščne točke uporabimo enakomerno razporejene točke na zunanjem navpičnem robu. Smer predstavlja smerni vektor pridobljen iz koordinatnega sistema kot ordinatna os. Iz zaporedja točk se tako formirajo linije v vzdolžni smeri ploske z dolžino vodoravnega robu.

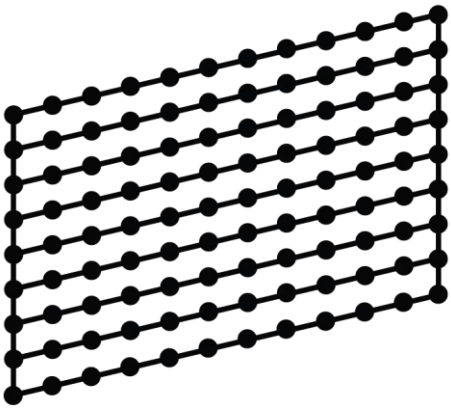


Slika 6.12: Generiranje zaporedje vodoravnih linij vzdolž ploskve glede na zaporedje točk.

Figure 6.12: Generating array of horizontal lines along the surface based on the array of points.

6.4.4 Zaporedje točk vzdolž linij

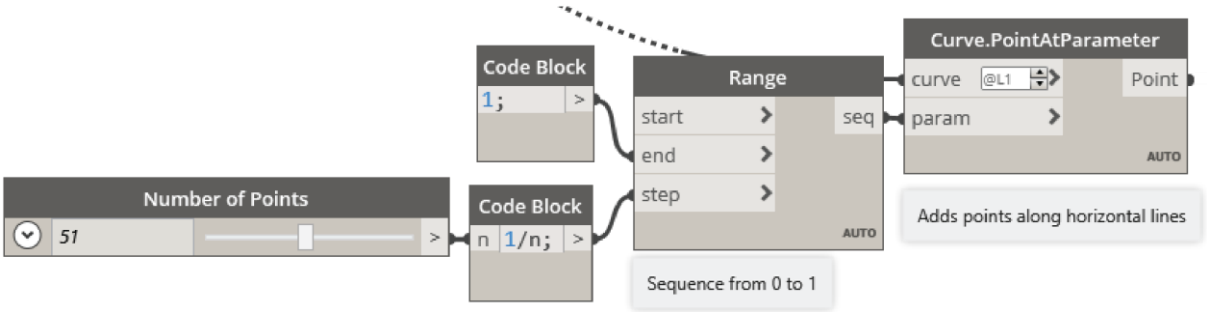
Za ponazoritev odprtin je potrebno vodoravne linije zakriviti. To storimo, tako da tvorimo zaporedje točk, izmed katerih zamaknemo le točke v okolici odprtin. Nato preko vseh točk povlečemo krivulje, ki bodo zakrivljene v območju odprtin.



Slika 6.13: Prikaz generacije točk vzdolž vodoravnih linij.

Figure 6.13: Illustration of the point generation along the horizontal lines.

Točke tvorimo vzdolž zaporedja vodoravnih linij, tako da razdelimo krivuljo na enakomerni razpored točk, glede na izbrano število. To storimo z ukazom *Curve.PointAtParameter*, ki tvori točke vzdolž krivulje glede na podano zaporedje števil od 0 do 1.

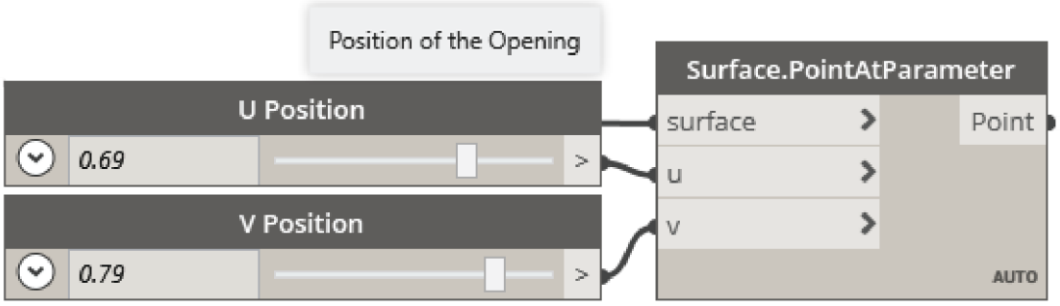


Slika 6.14: Tvorba zaporedja točk vzdolž vodoravnih linij.

Figure 6.14: Generating array of points along the horizontal lines.

6.4.5 Lega odprtin

Lego odprtin definiramo s točko, ki leži na začetni ploskvi, to določimo preko površinskih parametrov, ki jih podamo ukazu *Surface.PointAtParameter*.

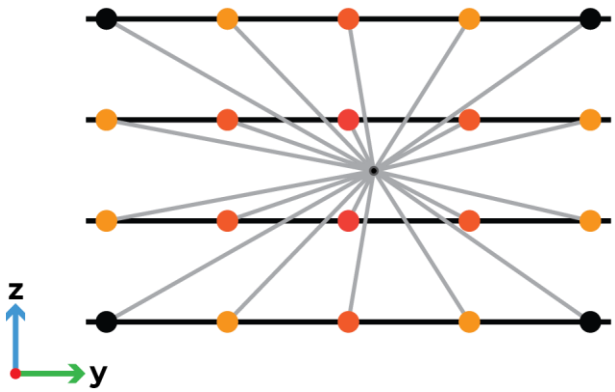


Slika 6.15: Lego odprtine definiramo z U, V koordinatami na površini.

Figure 6.15: Position of the opening is defined with U,V coordinates on the surface.

6.4.6 Razdalja med zaporedjem točk in središčem odprtine

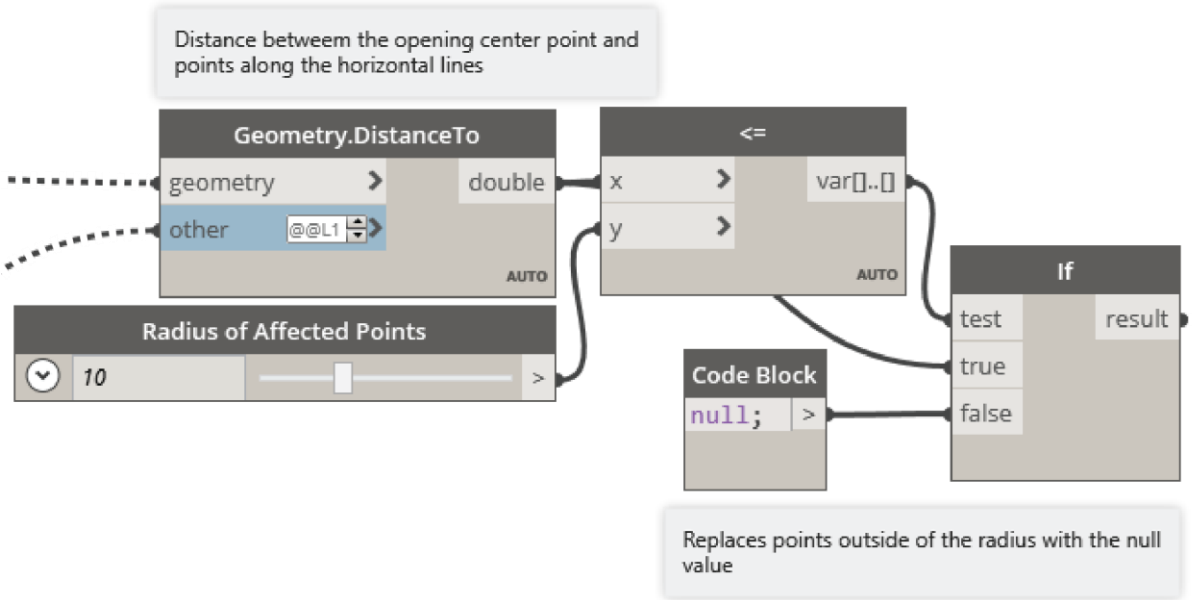
Točke znotraj območja lege odprtin določimo, tako da poiščemo razdalje med vsemi točkami vzdolž vodoravnih linij in med središčem odprtine. Tako lahko ugotovimo, katere točke so v bližini odprtine in katere so daleč stran. Točke nato ločimo na tiste, ki ležijo znotraj izbranega območja in na točke izven območja odprtine.



Slika 6.16: Prikaz obravnavanih točk v odprtini, ki so pridobljene glede na oddaljenost od središča.

Figure 6.16: Illustration of the affected points, which are filtered based on the distance from the center point.

Razdaljo med točkami poiščemo preko ukaza *Geometry.DistanceTo*, v katerega podamo množico točk in lego odprtine. Ukaz nam tako poračuna oddaljenosti vseh točk od središča odprtine. Razdalje nato sortiramo s pomočjo *If* stavka, ki obdrži točke znotraj podanega radija, ostalim točkam pa dodeli vrednosti *null*. Točkam, ki so znotraj območja želimo spreminjati lastnosti, ostalim točkam pa želimo ohraniti prvotne lastnosti. Zaradi tega je bila točkam izven območja dodeljena ničta vrednost *null*, saj nadaljnji ukazi ne vplivajo na ničte vrednosti, pri tem pa se še vedno ohranja struktura seznama. Strukturo želimo ohraniti, zato da se točke povežejo pravilno in da tvorimo vodoravne krivulje, ki so zakrivljene v območju odprtin.

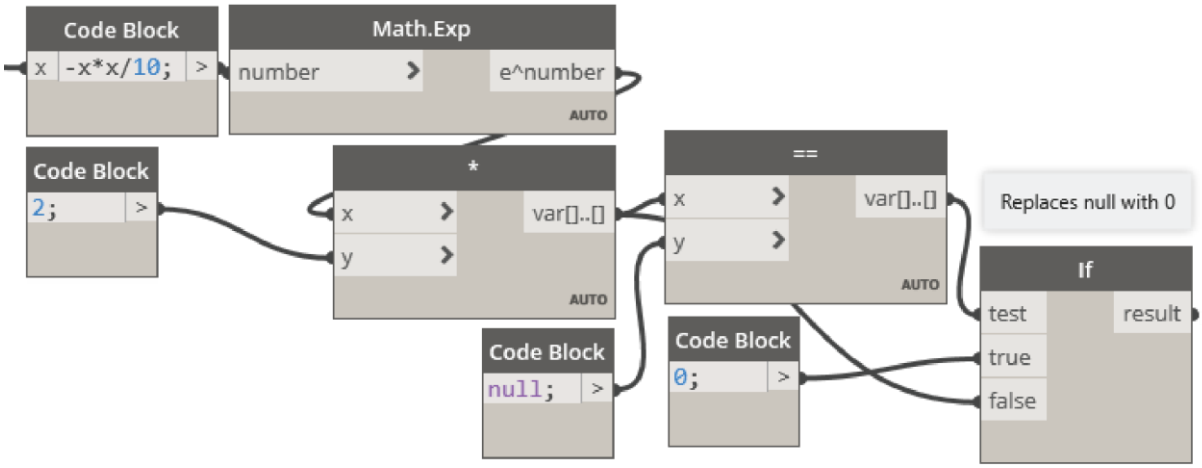


Slika 6.17: Sortiranje dolžin med središčem in zaporedju točk. Dolžine, katerih velikost je manjša od radija ohranimo, ostalne nadomestimo z vrednostjo null, da ohranimo strukturo seznama.

Figure 6.17: Filtering the distances between the centerpoint and the array of points. Distances which are larger than the radius are kept, others are replaced with value null to preserve list structure.

6.4.7 Enačba za odprtino

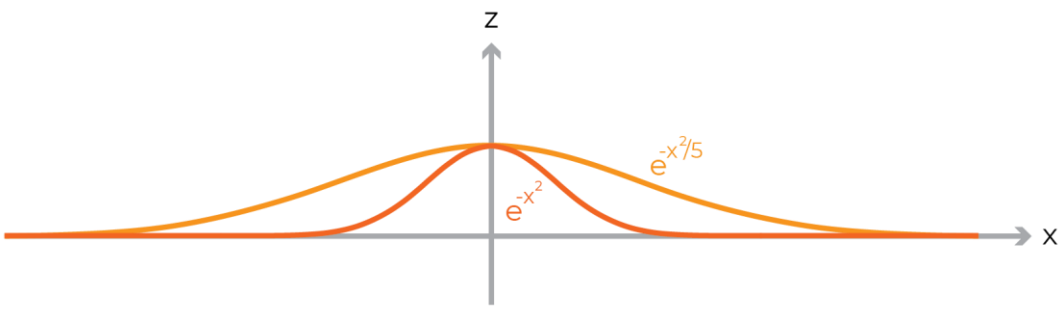
Izbranim točkam znotraj radija odprtine, priredimo vrednost zamika glede na njihovo oddaljenost od središča. V prejšnjem koraku smo ostalim točkam izven radija podali vrednost *null*, zato da algoritem ne zgublja časa z računanjem točk izven območja obravnave. Nato še nadomestimo ničte vrednosti z vrednostjo 0, kar pomeni, da se bodo točke zamaknile le tiste točke, ki ne vsebujejo vrednosti 0, ostale bodo pa ostale na istem mestu. S tem postopkom smo ohranili strukturo seznama točk, da lahko v naslednjih korakih povežemo točke v krivulje. Če strukture ne bi ohranjali, bi imeli samo točke, ki smo jih zamaknili brez povezave s prvotnimi točkami in jih ne bi bilo možno povezati v krivulje.



Slika 6.18: Podajanje enačbe, ki podeli vrednost za koliko zamaknemo točke glede na njihovo oddaljenost od središča. Točkam, ki so izven radija dodelimo vrednost 0. To pomeni, da se te točke ne bodo zamaknile.

Figure 6.18: Equation which assigns the values for the point translation based on the distance from the center point. Points which are beyond radius are attributed with value 0, which means they will not be translated.

V primeru, da bi namesto ničte vrednosti *null* podali vrednost 0 pred vnosom v enačbo, bi nam izračun vrnil vrednost 1, saj je e^0 enako 1.

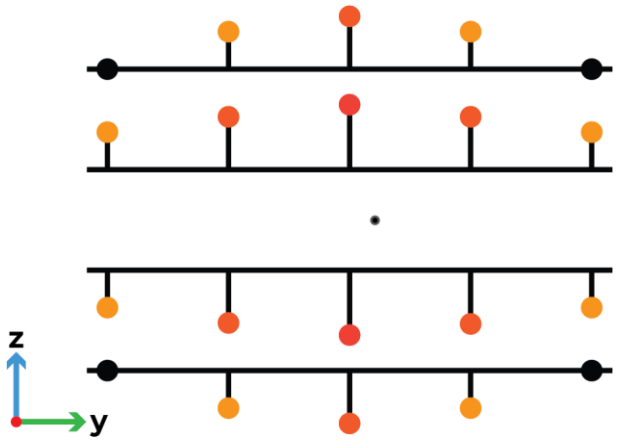


Slika 6.19: Grafični prikaz enačbe za premik točk za odprtine.

Figure 6.19: Illustration of the offset equation for the openings.

6.4.8 Smer premika točk

Poiščemo razdaljo med lego odprtin in vsemi točkami vzdolž linij. Zdaj lahko filtriramo glede na dolžino, torej da izberemo vse elemente, ki so znotraj izbrane dolžine. Te točke se bodo potem zamaknile. Te smo sortirali preko *If* stavka in pogoja manjše ali enako. Točke zamaknemo glede na posebno enačbo, ki je odvisna od odmaknjenosti vsake točke od središča. Zamaknemo jih le v z-smeri



Slika 6.20: Prikaz premika točk v vertikalni smeri odvisno od odmaknjenosti od središča.

Figure 6.20: Illustration of point translation based on the distance from the center point.

Slika 6.21: Poiščemo smerne vektorje med središčem in vsako točko v zaporedju. Smeri premikov točk predstavljajo komponente vektorjev v navpični smeri.

Figure 6.21: Direction vectors are found between the center point and each point in the array. Directions for the translation of each point are defined by the z-component of the vectors.

Pri združevanju seznamov imamo različne načine združevanja elementov med seznami. Pomembno je število elementov v vsakem seznamu, saj to vpliva na njihovo združevanje. Pri iskanju vektorjev med dvema točkama, združujemo seznam, ki vsebuje eno samo točko in seznam, ki vsebuje zaporedje točk. Ker želimo poiskati vse možne razdalje med središčem in ostalimi točkami, moramo temu pravilno

izbrati način združevanja. V tem primeru moramo uporabit *CrossProduct*. Na zgornji sliki je to vidno z oznako *xxx* na ukazu *Vector.ByTwoPoints*, kjer imajo ostali oznako *Auto*.

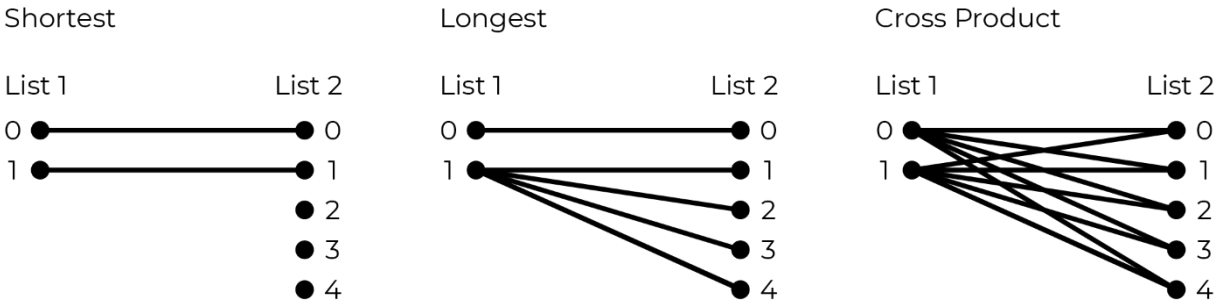
CrossProduct združi elemente iz različnih množic po principu vsak z vsakim. Predstavljamo si lahko z analogijo množenja dveh množic števil. Končno število elementov predstavlja produkt količine elementov v množicah. Za spodnji primer to pomeni $4 \cdot 2 = 8$ elementov.

$$(1 + 2 + 3 + 4) \cdot (1 + 2) = (1 \cdot 1 + 2 \cdot 1 + 3 \cdot 1 + 4 \cdot 1) + (1 \cdot 2 + 2 \cdot 2 + 3 \cdot 2 + 4 \cdot 2)$$

$$= (1 + 2 + 3 + 4) + (2 + 4 + 6 + 8)$$

V programu obstajajo 4 načini združevanja množic elementov (ang.: list, array):

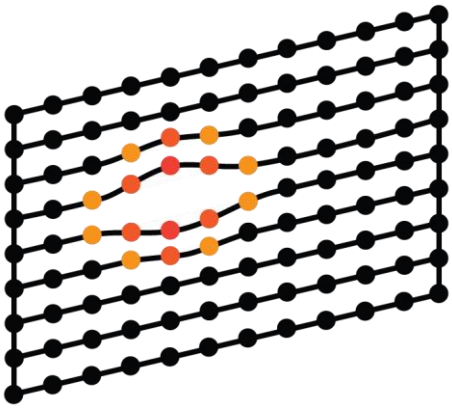
- *Auto*: samodejno izbere najbolj ustrezni način od naslednjih treh.
- *Shortest*: združuje elemente množic po principu: združi vsak element med množicami, dokler ne zmanjka elementov v eni izmed množic elementov. Ta predstavlja privzeto nastavitev.
- *Longest*: združuje elemente množic podobno kot *Shortest*, le da se zadnji element najkrajše množice združi z vsemi preostalimi elementi.
- *Cross Product*: združuje elemente množic vsak z vsakim. To nam naredi vse možne kombinacije povezav med elementi.



Slika 6.22: Prikaz različnih možnih kombinacij združevanja seznamov elementov v programu Dynamo.
Figure 6.22: Illustration of different lacing methods for list combinations in Dynamo.

6.4.9 Premik točk za odprtino

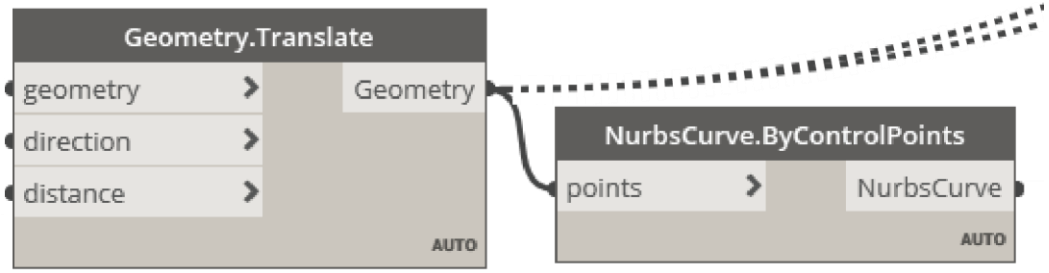
Točke v bližini odprtine zamaknemo glede na njihovo oddaljenost od središča. Tako smo tvorili odprtino med elementi za senčenje. Točke še povežemo s krivuljami, ki bodo nato tvorile zakrivljene elemente.



Slika 6.23: Prikaz rezultata odmika točk, ki ponazarjajo odprtino.

Figure 6.23: Illustration of the point translation result, which illustrates the opening.

Točke zamaknemo s funkcijo *Geometry.Translate*, tako da podamo seznam točk, smer ter velikost zamika. Ker smo ohranili strukturo seznama, lahko zdaj ponovno generiramo krivulje skozi prirejeno množico točk. Te krivulje so zdaj zakrivljene v okolici odprtine, povsod drugje pa ravne.

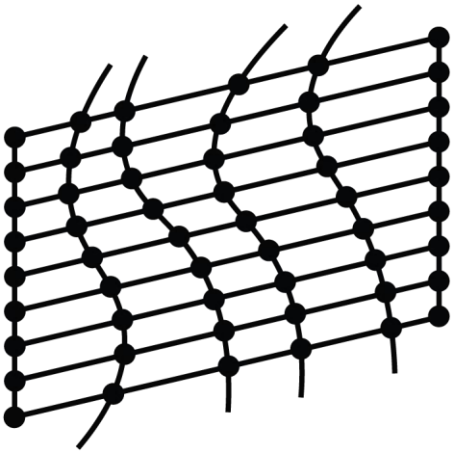


Slika 6.24: Premik točk in tvorba krivulj skozi te točke. Točke zamaknemo za vrednost

Figure 6.24: Point translation and the generation of the curves connecting the points.

6.4.10 Oblikovne krivulje

Za estetsko podobo fasade želimo valovito obliko. Za to so potrebne krivulje ki narekujejo obliko. Te so narisane v programu Revit, lahko pa so tudi narejene v okolju Dynamo. Razlika je, da je v okolju Revit mogoče lažje za preveriti videz in prilagoditi krivulje.

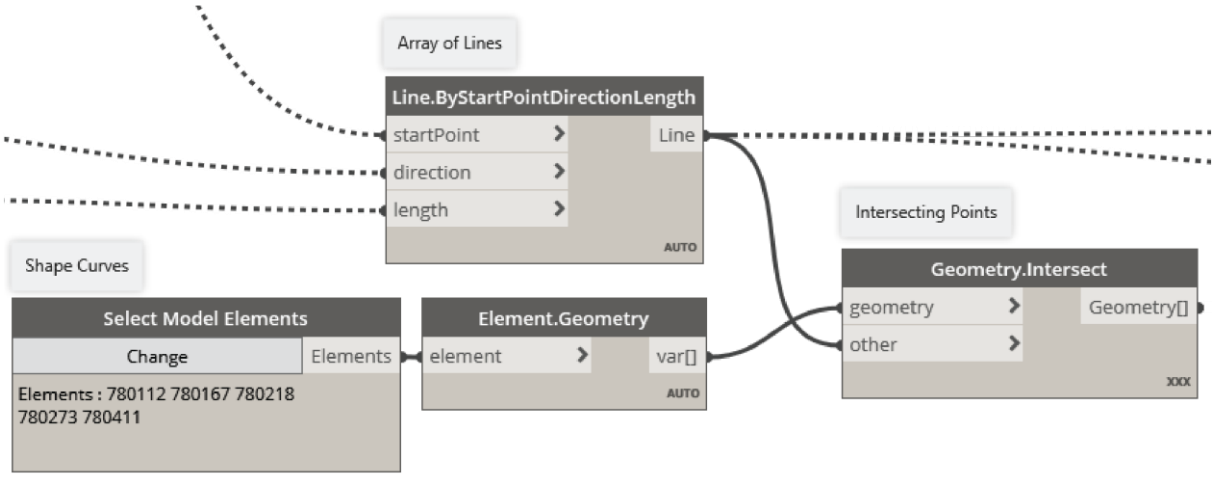


Slika 6.25: Prikaz sečišč med zaporedjem vodoravnih linij in oblikovnimi krivuljami.

Figure 6.25: Illustration of the intersection between the array of lines and the shape curves.

Pomožne krivulje, ki podajajo valovito obliko geometriji elementov so izrisane v okolju programa Revit in nato uvožene v okolje Dynamo preko *Select Model Elements*. S funkcijo *Geometry.Intersect* generiramo točke na stičiščih med pomožnimi črtami in vodoravnimi linijami. Pomembno je tudi to, katera geometrija se seka s katero. To lahko obračamo tudi z *List.Transpose*.

Geometry.Intersect naredi geometrijo, ki je za stopnjo manjša od najmanjše stopnje. Na primer, če sekamo krivuljo s površino, sečišče predstavlja točko. Če sekamo dve površini je sečišče krivulja.

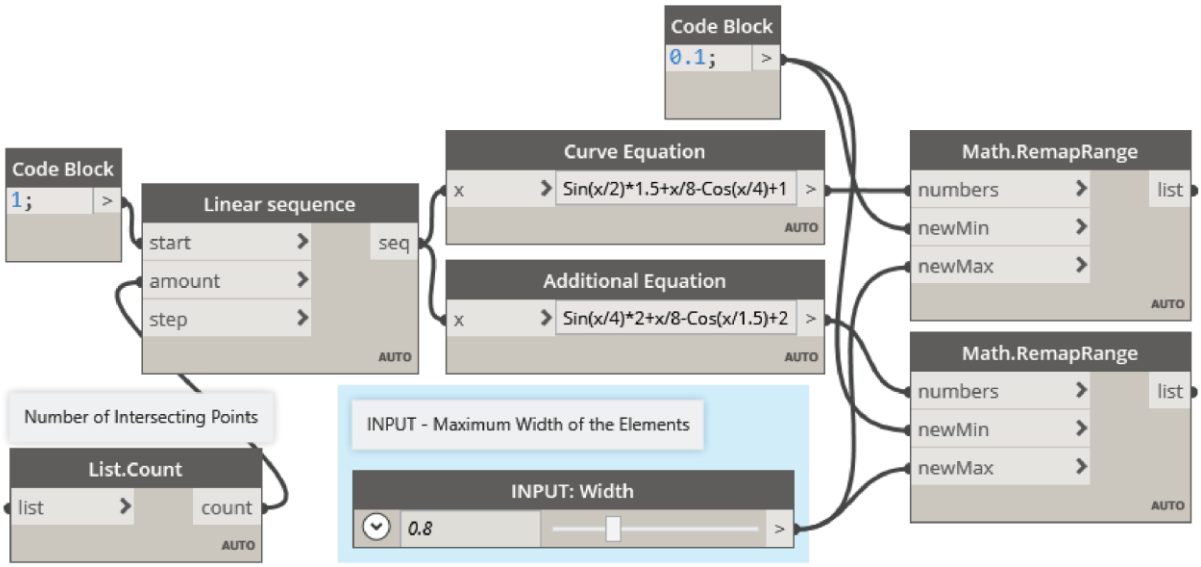


Slika 6.26: Tvorjenje točk na sečiščih oblikovnih krivulj in zaporedjem vodoravnih linij.

Figure 6.26: Generating points on the intersections between shape curves and array of horizontal lines.

6.4.11 Enačbe širine elementov

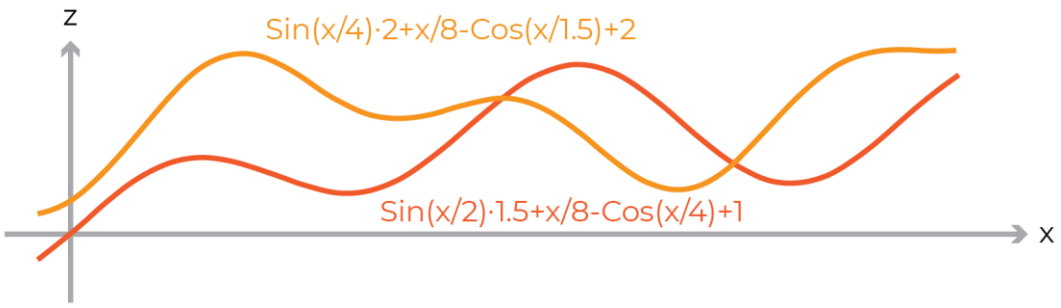
Ko imamo točke vzdolž krivulj, ki smo jih prilagodili z odprtinami, iz njih povlečemo ravne črte v smeri normale površine in ponovno uporabimo *Geometry.Intersect* med linijami in ukrivljeno površino. To nam tvori točke, kjer se ti dve geometriji sekati. Te točke zato ležijo na ukrivljeni površini in so vzporedne tistim iz linije. To smo storili zaradi teh očes, ki so nam zamaknile krivulje



Slika 6.27: Tvorjenje seznama vrednosti, katere bodo tvorile spremenljivo širino elementov za senčenje. Vrednosti zapišemo parametrično, kar predstavlja tudi drugo vhodno spremenljivko, ki je definirana kot največja možna širina elementov.

Figure 6.27: Creating a list of values for the variable width of the shading elements. Values are parametric, which is also a second input variable, defined as a maximum width of the elements.

Najprej tvorimo seznam linearnih vrednosti od 1 do števila elementov za senčenje, kar predstavlja prvo vhodno spremenljivko. Dobimo seznam oblike *[1, 2, 3, ..., n]*, kjer vrednost *n* predstavlja število elementov za senčenje. Tega smo zapisali parametrično s prvo vhodno spremenljivko. Seznam vrednosti uporabimo



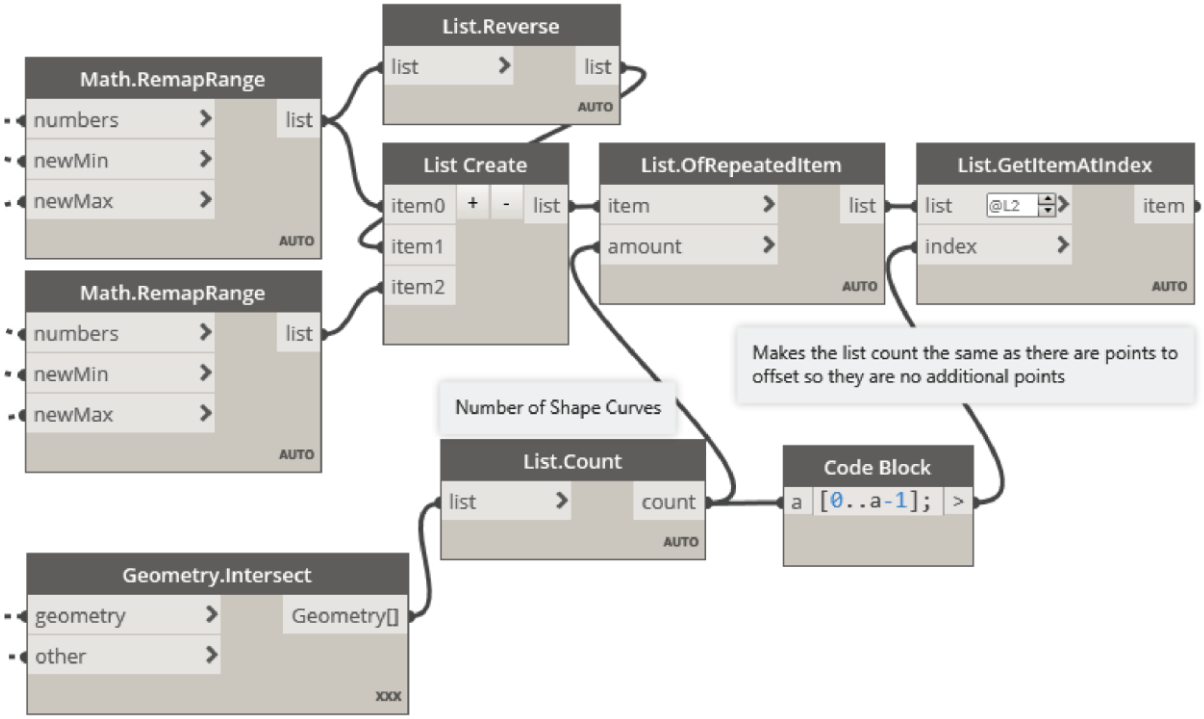
Slika 6.28: Prikaz oblike, ki nam jo tvori enačba za določitev spreminjajoče širine.

Figure 6.28: Illustration of the shape, generated by the equation for the variable width.

Za drugi parameter privzamemo širino elementov za senčenje oziroma ga podamo kot največja možna vrednost pri funkciji, ki podaja spremenljivo širino elementom. Spremenljivo širino podamo v obliki sinusne in kosinusne funkcije, ki ju nato prilagodimo z novimi mejami območja oziroma zaloge vrednosti.

6.4.12 Seznam vrednosti širin v posameznih točkah

Nato naredimo ponavljajoče zaporedje vrednosti iz kombinacije funkcij $f_1(x)$, $f_2(x)$ in $1\text{-}f_1(x)$. Na koncu želimo imeti isto število spreminjajočih širin kot za število krivulj, zato filtriramo glede na število krivulj.

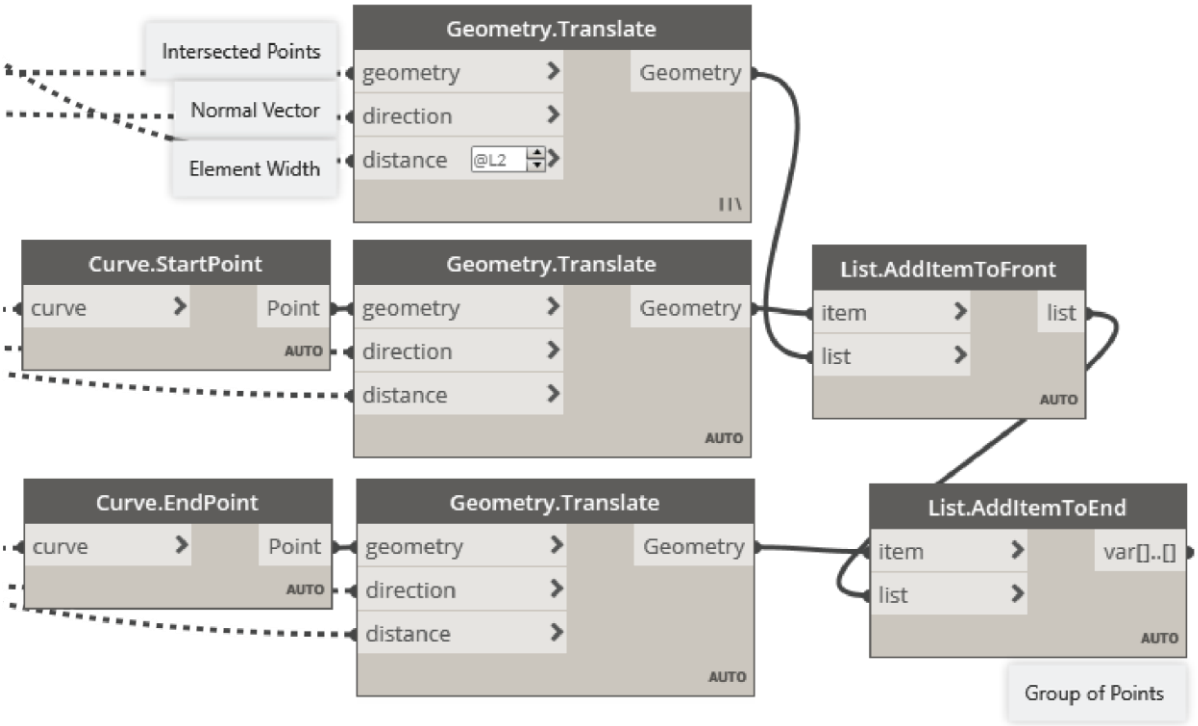


Slika 6.29: Vrednosti spremenljivih širin dobljenih iz enačb združimo v novo zaporedje naraščujočih in padajočih vrednosti.

Figure 6.29: Values of the variable widths are combined into a joint list of rising and falling values.

6.4.13 Zamik točk za odprtino

V seznam zamaknjenih točk dodamo še začetne in končne zamaknjene točke, da jih združimo v en sam seznam. Zdaj, ko imamo to seznam točk, lahko tvorimo krivulje in zaporedja točk.



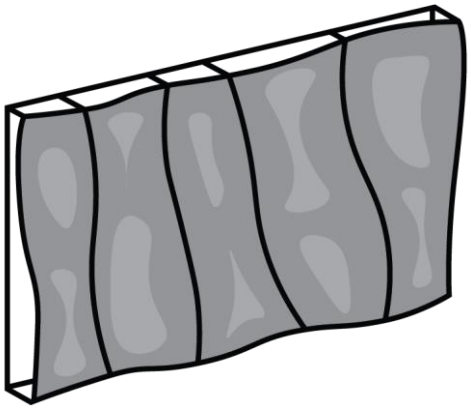
Slika 6.30: Zamikanje točk v smeri normale glede na zaporedje naraščujočih in padajočih vrednosti. Pri tem vključimo v seznam robne točke, ki jih zamaknjemo z enakimi vrednostmi.

Figure 6.30: Point translation in the normal direction based on the array of the raising and falling values. Translated edge points are also added to the list of points.

Poleg tega moramo še dodati širino na koncu in na začetku, tako da je bila tukaj izbrana ena izmed enačb in podana širina z že izbranimi parametri. S temi podatki potem zamaknemo začetne in končne točke s funkcijo *Geometry.Translate*. Začetne in končne točke pridobimo iz vodoravnih liniji s funkcijo *Curve.StartPoint*.

6.4.14 Oblikovna površina

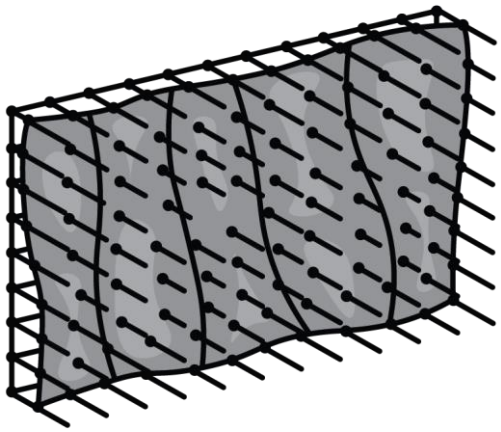
Preko zaporedja krivulj generiramo površino. Ta površina predstavlja zunanji rob elementov. Preko površine vidimo estetsko podobo, iz katere bo izhajala končna oblika.



Slika 6.31: Površina, ki je generirana preko interpolacije med oblikovnimi krivuljami.

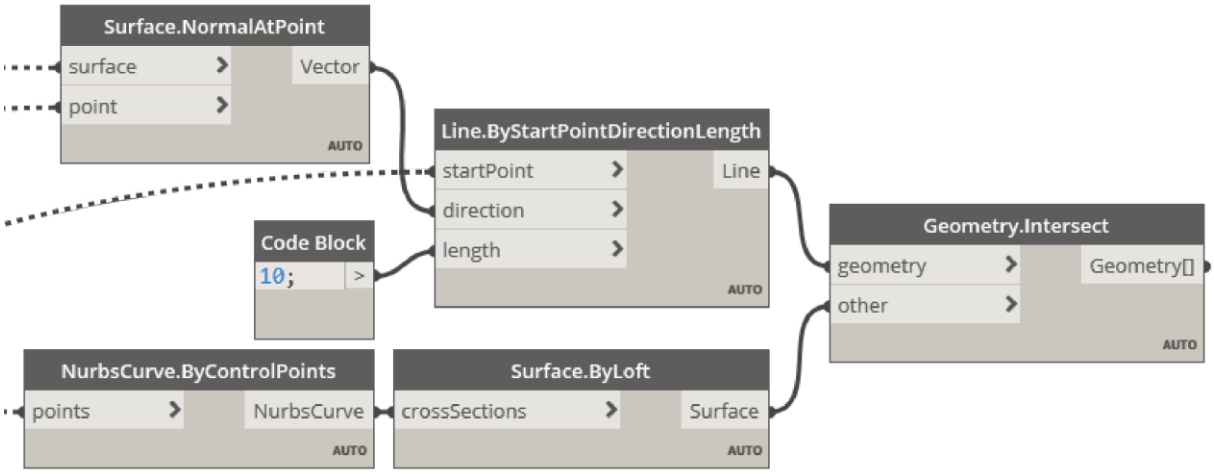
Figure 6.31: Surface, which is generated by interpolating shape curves.

Da lahko zdaj dobimo krivulje na površini, generiramo linije v smeri normale. Potem poiščemo sečišča z *Geometry.Intersect* in tvorimo točke na sečiščih. Odvisno je tudi zaporedje geometrije v ukazu, to lahko spremenimo tudi s funkcijo *List.Transpose*, ki zamenja vrstice in stolpce. Te točke potem povežemo v krivuljo, preko t ega dobimo zdaj zunanjo krivuljo, ki določa elemente. Ta postopek je bil potreben, zato ker smo dodali odprtine in moramo ohraniti obliko elementov.



Slika 6.32: Prikaz zaznavanja sečišča med oblikovno provršino in pravokotnimi linjami glede na začetno površino. Preko tega dobimo točke, katere se bo povezavlo s krivuljami.

Figure 6.32: Illustration of the intersection detection between the shape surface and the lines perpendicular on the starting surface. Intersection returns points, which will be connected with curves.

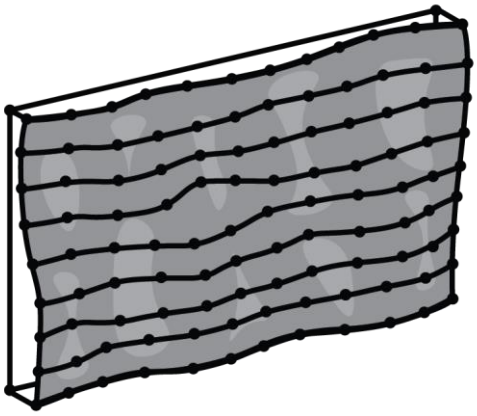


Slika 6.33: Generiranje točk na sečiščih med pravokotnimi linijami glede na začetno ploskev in med odmaknjeno ploskev. Točkam na začetni površini poiščemo sorodne točke na odmaknjeni površini.

Figure 6.33: Point generation on the intersection between the lines, perpendicular on the base surface and the offset surface. To points on the base surface we find associated points on the offset surface.

6.4.15 Nove krivulje, kot zunanji rob elementov za senčenje

Sečišče linij s površino nam da točke, ki jih povežemo v krivulje. Krivulje predstavljajo zunanji rob elementov za senčenje.



Slika 6.34: Prikaz povezave točk v krivulje na odmaknjeni površini. Krivulje predstavljajo zunanji rob elementov za senčenje.

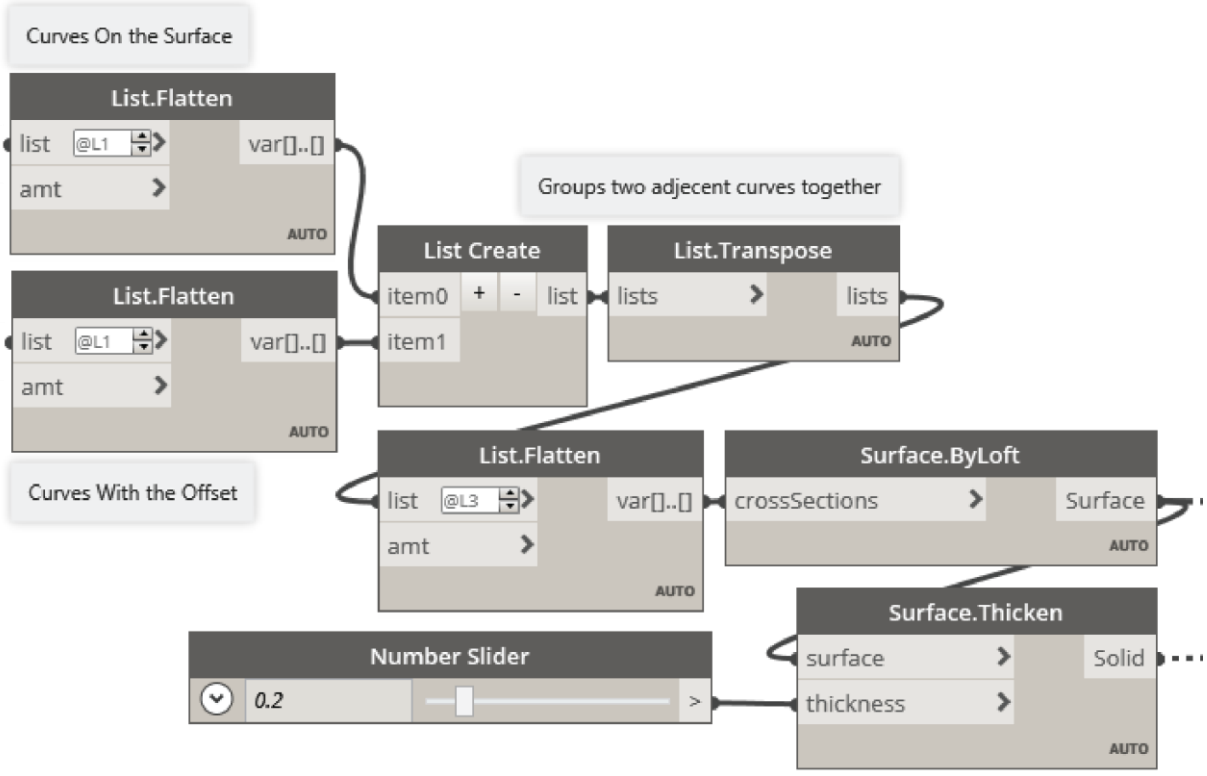
Figure 6.34: Illustration of the curves connecting the points on the offset surface. The curves are outer edge of the shading elements.

Slika 6.35: Tvorjenje krivulj na odmaknjeni ploskvi preko povezave točk, dobljenih iz sečišč.

Figure 6.35: Creating curves on the offset surface, by connecting the points from the intersection.

6.4.16 Geometrija

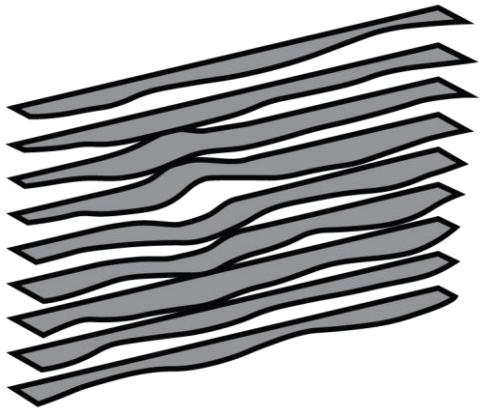
Kot smo omenili, nas kot končni rezultat nas zanima poleg razporeditve in velikosti elementov tudi geometrija, ki jo želimo uporabiti za model. Da dobimo obliko elementov, najprej generiramo površine teh elementov. Uporabimo funkcijo Surface.ByLoft, ki tvori površino preko dveh ali več krivulj. Do sedaj smo ustvarili krivulje na začetni ploski in krivulje na odmaknjeni ploski. Te moramo zdaj združiti v skupen seznam. Pomembno je, da imamo pravilno strukturo seznama, torej da sta dve sosednji krivulji v isti podmnožici. To dobimo tako, da najprej krivulje pretvorimo v seznam, ki vsebuje eno podmnožico z vsemi krivuljami, kar smo storili z ukazom *List.Flatten*, ki združi vse elemente v isto množico oziroma podmnožico. Te potem združimo v skupni seznam z *List.Create*. Zdaj je struktura sestavljena iz dveh podmnožic v katerih so krivulje na začetni površini in krivulje na odmaknjeni površini. Z ukazom *List.Transpose* zdaj zamenjamo vrstice in stolpce, kar pomeni da imamo zdaj toliko podmnožic kot krivulj, ki vsebujejo obe sosednji krivulji.



Slika 6.36: Povezava krivulj v površine, katere odebelimo, da dobimo končno volumetrično geometerijo elementov za senčenje.

Figure 6.36: Connecting the curves into the surfaces to which we add thickness to create a solid geometry for the shading elements.

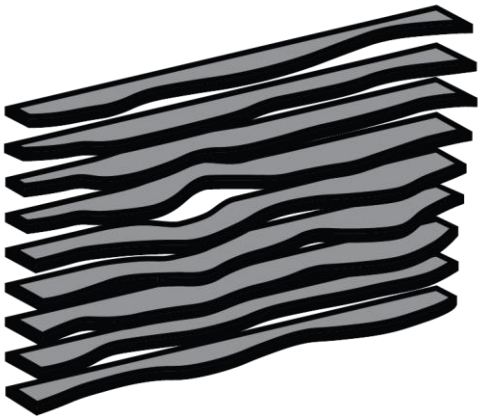
Potem ko ustvarimo začetne vodoravne linije in njim vzporedne krivulje, lahko preko teh tvorimo površino z *Surface.ByLoft*. Tako smo dobili površino elementov za senčenje, ki jih nato še odebelimo.



Slika 6.37: Generiranje površine, tako da povežemo krivulje na začetni površini in krivulje na odmaknjenji oblikovni površini.

Figure 6.37: Generating surface by connecting the curves on the starting surface and the curves on the shape surface.

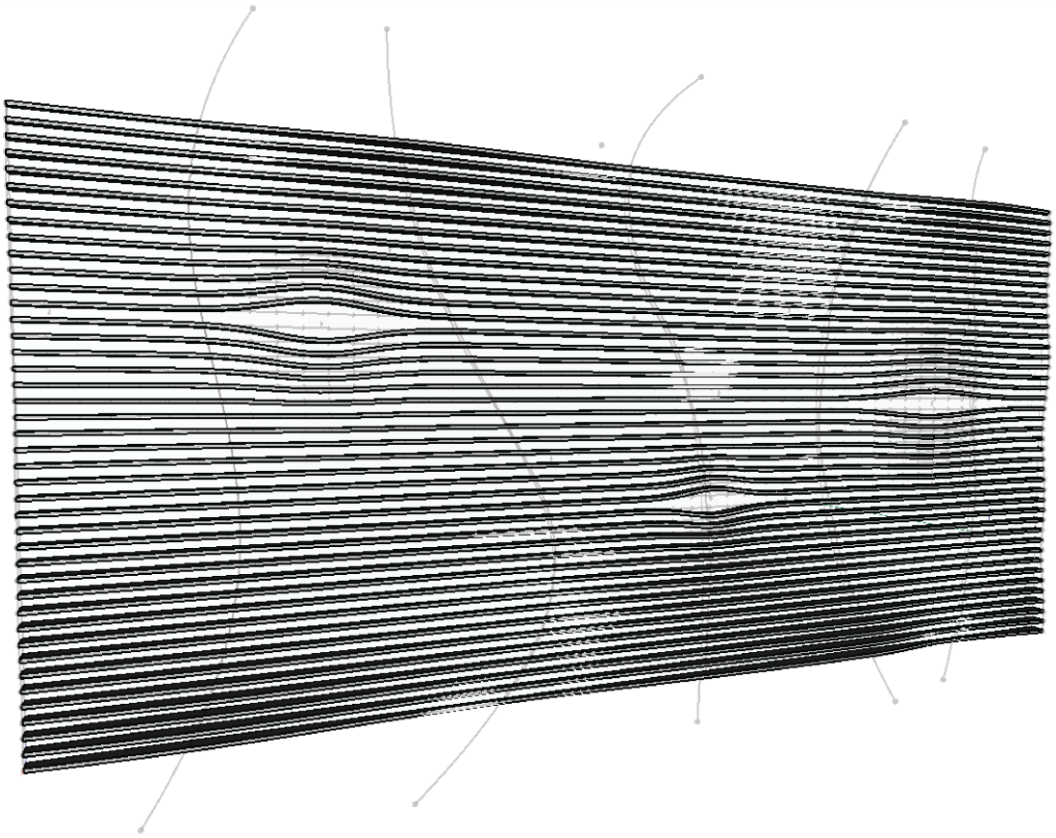
Za izris volumetrične geometrije uporabimo ukaz *Surface.Thicken*, ki odebeli površino za predpisano vrednost. Pri tem se ustvarijo končni elementi, ki jih lahko uvozimo v ostale programe.



Slika 6.38: Dodelitev debeline površinam elementov, katere pretvorimo v končno volumentrično geometrijo elementov za senčenje.

Figure 6.38: Applying thickness to the surfaces, which are transformed into the solid geometry of the shading elements.

Tako smo tvorili geometrijo elementov za senčenje v programskem okolju Dynamo. Geometrijo nato izvozimo v ostale programe, kot so npr. Revit.



Slika 6.39: Prikaz geometrije elementov za senčenje v programskem okolju Dynamo.

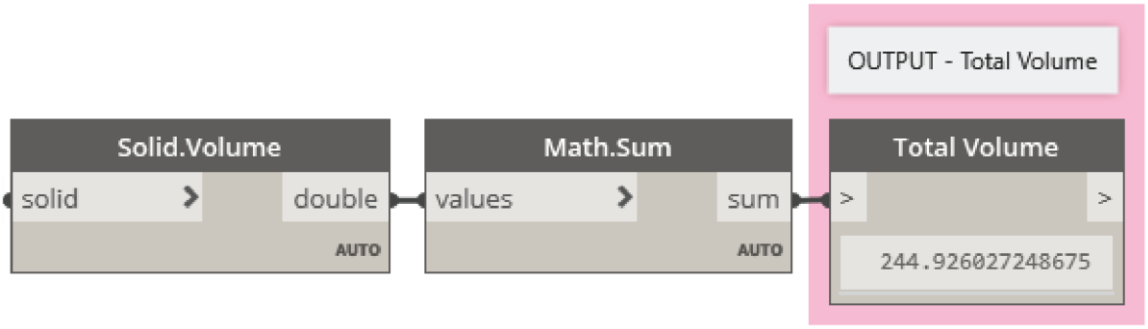
Figure 6.39: Generated geoemetry of the shading elements in Dynamo.

6.5 Kriteriji

Če ponovimo, je optimizacija običajno sestavljena iz štirih ključnih elementov: spremenljivk, namenske oziroma kriterialne funkcije, mej in pogojev. [6] V prejšnjih poglavjih smo obravnavali vhodne spremenljivke in formulacijo modela oziroma geometrije elementov za senčenje. Sedaj si moramo pogledati namensko funkcijo oziroma kriterije, ki dodelijo kazalce uspešnosti rezultatom pridobljenih s spreminjanjem vhodnih spremenljivk.

6.5.1 Količina materiala

Količino potrebnega materiala za elemente za senčenje lahko enostavno pridobimo, tako da uporabimo ukaz *Solid.Volume*, ki izračuna volumen geometrije. Ta je seveda odvisen od obeh spremenljivk, torej od števila elementov in njihove širine.



Slika 6.40: Funkcija, ki nam izračuna volumen generirane geometrije elementov za senčenje, ki predstavlja izhodno spremenljivko oziroma prvi kriterij za optimizacijo.

Figure 6.40: Objective function which calculates the volume of the generated geometry. Volume is output variable or first objective for optimization process.

Pomembno je, da se zavedamo da program Dynamo deluje brez enot oziroma ima enotske enote. Torej ko vnašamo geometrijo iz ostalih programov, privzamemo njihove enote. Če je npr. uvoženi element dolžine 100 cm je potem v programu Dynamo dolžine 100 enot in nadaljujemo modeliranje v enotah istega reda. V našem primeru smo privzeli geometrijo iz programa Revit v katerem smo modelirali v čevljih (angl. feet). Torej prikazana vrednost predstavlja volumen v kubičnih čevljih (angl. cubic feet).

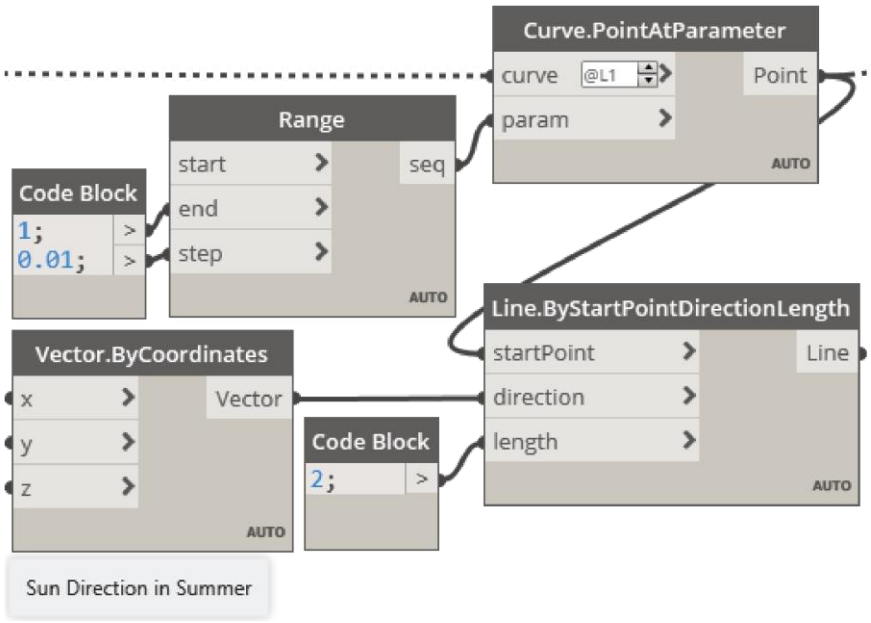
6.5.2 Učinkovitost senčenja

Vprašanje je, kako lahko ocenimo učinkovitosti senčenja v različnih polletnih časih. Poznamo programske rešitve, ki računajo toplotno učinkovitost stavb in analizirajo osvetljenost prostorov, kot so Autodesk Insight ali dodatek Ladybug Tools. Vendar uporaba dodatnih programov pri analizi

pomeni daljši čas za optimiziranje. Želimo poiskati, ali so možni enostavnejši načini kako lahko numerično ocenimo učinkovitost geometrije pri senčenju izbrane fasade oziroma površine.

Pri iskanju enostavnejše rešitve se začnemo spraševati, kako delujejo ostali programi za svetlobne analize. Če razmislimo o svetlobi v računalniški grafiki, to simuliramo s svetlobnimi žarki. Torej če znamo generirati žarke lahko potem preverimo, ali se ti sekajo z geometrijo elementov za senčenje. Preko omenjenega razmisleka je nastala predstavljena rešitev, ki je v splošnem enostavna in nam poda numerično oceno.

Pričnemo s formiranjem zaporedja točk vzdolž krivulj, ki predstavljajo notranji rob elementov za senčenje in ležijo na začetni ploskvi. Zaporedje točk nato uporabimo kot izhodišče za generacijo linij, ki potekajo v smeri sončnega kota. Smer sonca zapišemo kot vektor, ki ga pridobimo iz projekta v programu Revit. V projektu imamo namreč definirano geografsko lokacijo in orientacijo objekta na terenu.



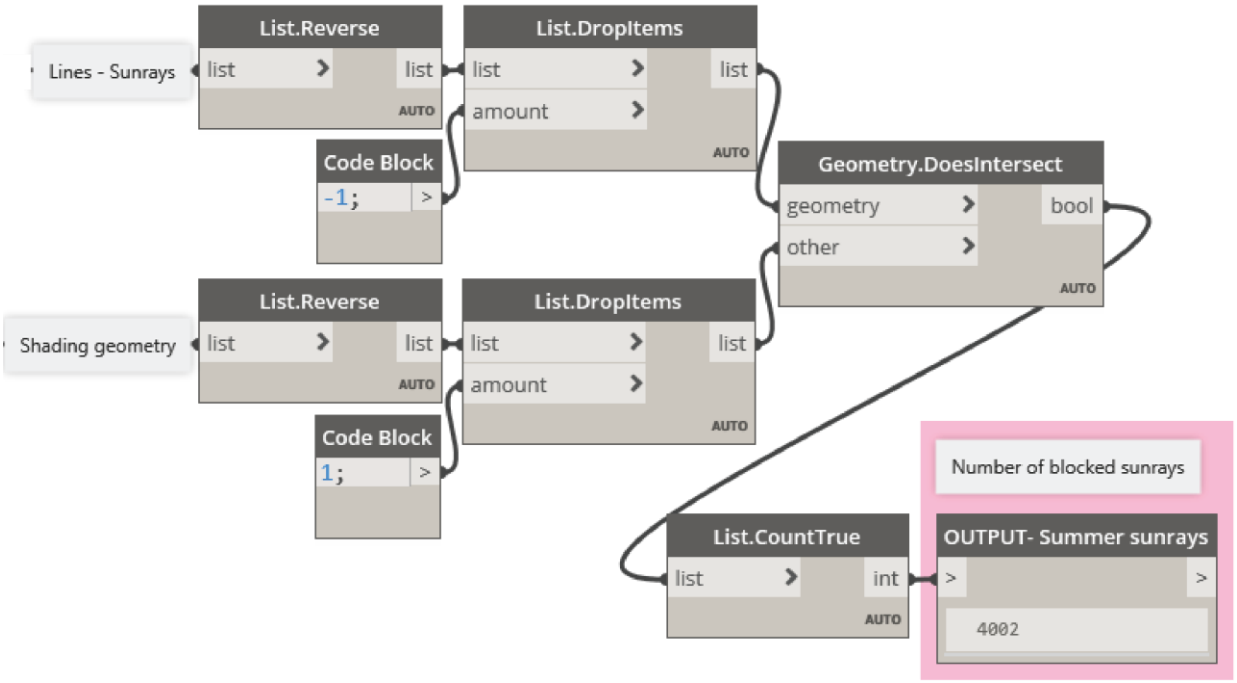
Slika 6.41: Simuliranje sončnih žarkov z zaporedjem linij v smeri sončnega kota.

Figure 6.41: Simulating sunrays with an array of lines in the direction of the sun.

Linije se tvori z ukazom *Line.ByStartPointDirectionLenght*, kot pove ime, se linije izrišejo v določeni smeri z določeno dolžino. Dolžino izberemo tako, da bo prišlo do sečišča z elementi za senčenje. Preko tega postopka smo simulirali žarke z izbrano gostoto, ki narekuje zaporedje točk in jo določi projektant. Večja kot bo gostota, natančnejša in bolj časovno potratna bo analiza.

Do sedaj imamo simulacijo sončnih žarkov, potrebno je preveriti še, v kolikšnem številu preidejo skozi senčenje oziroma v kolikšnem številu jim elementi za senčenje preprečijo pot v notranjost. Preverimo tako, da zaznamo, ali se linije, v nadaljevanju žarki, sekajo z geometrijo za senčenje, ki

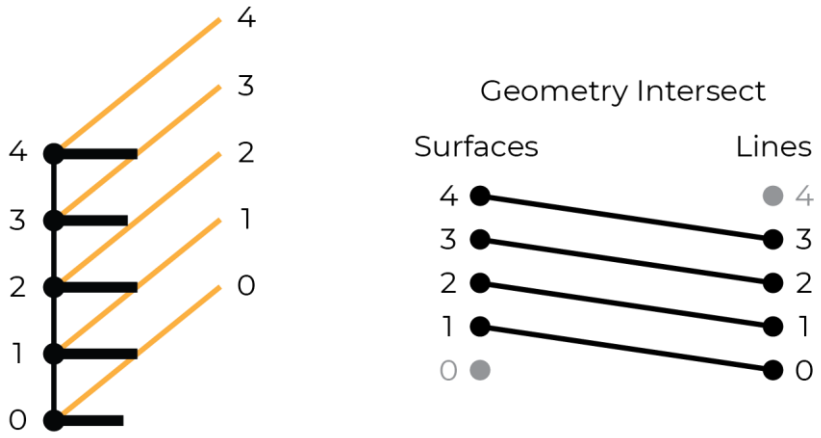
smo jo modelirali v prejšnjih poglavjih. To nam omogoča ukaz *Geometry.DoesIntersect*, ki nam vrne rezultat *da* ali *ne (true, false),* torej ali se seka ali se ne. Za pridobitev števila žarkov, ki jim je preprečen prehod v notranjost, seštejemo vrnjene vrednosti *da (true)* z ukazom *List.CountTrue*. Preko tega smo zdaj numerično opisali oceno efektivnosti senčenja. Vemo, da večja kot bo številka, večja je učinkovitost senčenja.



Slika 6.42: Funkcija, ki nam izračuna število žarkov, ki jih elementi za senčenje zaustavijo. Število predstavlja izhodno spremenljivko oziroma kriterij za optimizacijo.

Figure 6.42: Objective function which calculates the number of sunrays blocked by the shading elements. Number is output variable or objective for optimization process.

Da pravilno zaznamo, ali se žarki sekajo z geometrijo, moramo seznam žarkov in seznam geometrije povezati, tako da se žarki v vsaki posamezni vrstici povežejo z geometrijo v vrstici višje. To storimo tako, da iz seznama geometrije odstranimo prvi element, saj se ta ne bo sekal z nobenim žarkom. Zdaj je začetni element v seznamu geometrije drugi element, kar pomeni, da se bo žarek v začetni vrstici sekal z geometrijo v drugi vrstici in tako dalje za ostale vrstice. Podobno lahko storimo za seznam žarkov in odstranimo zadnji element, saj se ta ne bo sekal z nobeno geometrijo, je pa ta korak poljuben, saj na strukturo seznama ne vpliva. Prvi element v seznamu se odstrani z ukazom *List.DropItems*, ki mu podamo pozitivno vrednost 1, kar pomeni, da odstrani en element v začetku seznama, če podamo negativno vrednost, pa odstrani elemente na koncu seznama. V seznamu se elementi začnejo z zgornjimi in končajo s spodnjimi. Da funkcija deluje pravilno, želimo obrniti vrstni red, kar storimo z ukazom *List.Reverse*.



Slika 6.43: Prikaz delovanja zaznavanja sečišča med sončnimi žarki in elementi za senčenje. Seznam elementov se mora zamakniti, tako da se vsaka linija povezuje z geometrijo nad njo.

Figure 6.43: Illustration how the intersection between the sunrays and shading elements works. List of elements need to be shifted so that the each individual line interacts with the geometry above it.

Za analizo senčenja preverimo lego sonca glede na različne letne čase in glede na čas dneva. Ugotovimo, da ima največji vpliv na učinkovitost senčenja najvišja in najnižja lega sonca. Ker želimo omejiti tudi število oziroma želimo čim manjše število kriterijev pri optimizaciji, izberemo analizo senčenja pri legi sonca v letnem času in legi sonca v zimskem času ob 12. uri, kar ustreza tudi najvišji in najnižji poziciji sonca na nebu ob najmočnejši uri sonca.

Izbrani kriteriji za optimizacijo so torej:

- Količina materiala oziroma volumen elementov za senčenje. Tukaj iščemo minimum.
- Število žarkov, ki jim elementi za senčenje preprečijo prehod v notranjost v poletnih mesecih. Tukaj iščemo maksimum.
- Število žarkov, ki jim elementi za senčenje preprečijo prehod v notranjost v zimskih mesecih. Tukaj iščemo minimum.

6.6 Reševanje

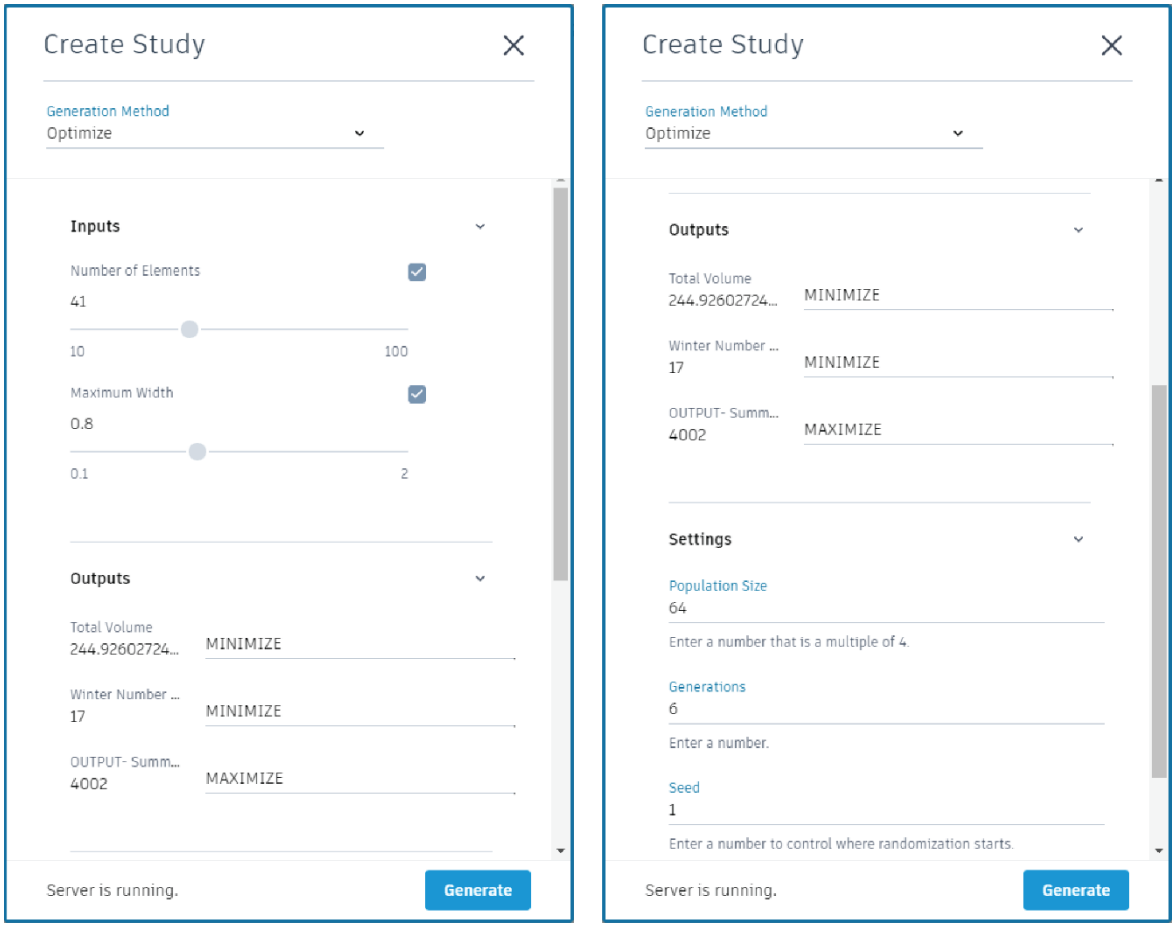
Reševanje problema poteka z orodjem Project Refinery, ki ponuja več metod reševanja. Možne metode so:

- *Randomize*: naključno generira izbrano število izidov.
- *Optimize*: išče optimalne izide glede na evolucijski razvoj in genetske algoritme.
- *Cross Product*: vse izbrane vrednosti vhodnih spremenljivk kombinira med seboj in generira izide.
- *Like This*: Generira izide glede na vrednosti iz bližnje okolice izbranih vrednosti vhodnih spremenljivk. [12]

V naši analizi nas zanima optimalna rešitev, zato izberemo optimizacijski algoritem, ki išče rešitve z genetskimi algoritmi. Ta zahteva, da izberemo katere vhodne spremenljivke želimo uporabiti in na kaj želimo optimizirati torej ali iščemo minimum ali maksimum namenskih funkcij. Podati moramo tudi nastavitve za optimizacijo torej vzorec populacije in število generacij. Velikost populacije vpliva na to, koliko primerov obdelujemo, število generacij pa vpliva na natančnost optimizacije. Poleg tega lahko v primeru, da želimo postopek ponoviti za primerjave, tudi izberemo seme (angl. seed) začetne naključne iteracije v primeru.

Pri iskanju rešitve se praviloma ne smemo zanašati le na uporabo algoritma in privzeti, da nam algoritem samodejno reši problem. Pomembna je tudi kritična presoja rezultatov in nadaljnja analiza rezultatov. Reševanje preko računalniških algoritmov nam omogoča hitro raziskovanje prostora rešitev. Torej analiza večjega števila rezultatov nam znatno pomaga pri ugotavljanju, kako različni parametri vplivajo na rezultate, kar pripomore k lažjemu in boljšemu odločanju pri izbiri končne rešitve. Z delom, ki ga vnesemo v nastavljanje problema za reševanje z algoritmi, nadomestimo delo, ki ga bi potrebovali pri iskanju različnih rešitev. To je tudi osnovna ideje generativnega pristopa pri projektiranju. Problem prevedemo na reševanje z algoritmi ali pametnimi računalniki, preko katerih z matematičnimi orodji generiramo večje število rešitev, ki bi jih lahko potem še obedujemo, da pridemo do končne rešitve.

Pri optimizaciji kompleksnejših problemov običajno nimamo samo ene optimalne rešitve, ki bi jo lahko privzeli za končno rešitev. Zato je potrebno analizirati prostor rešitev in ugotoviti, kako različne spremenljivke vplivajo na rešitve odvisno tudi od različnih kriterijev. V našem primeru rešujemo več-kriterialni problem, kjer enkrat iščemo maksimum, enkrat pa minimum.

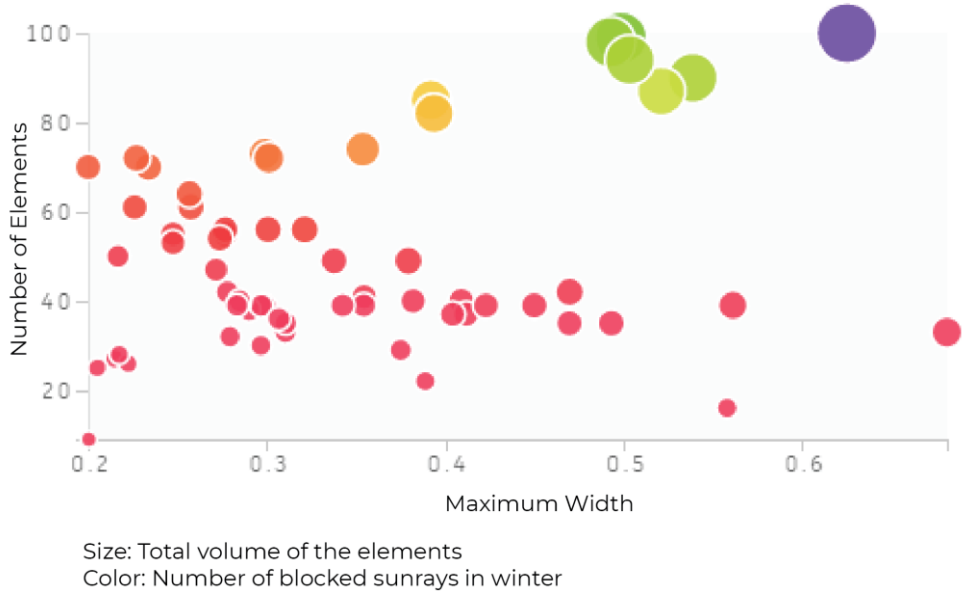


Slika 6.44: Nastavitve v programu Project Refinery. Podati je potrebno vhodne in izhodne spremenljivke, ter nastavitve optimizacije.

Figure 6.44: Settings in Project Refinery. Input and output variables need to be selected together with the optimization settings.

Če si pogledamo rezultate in primerjamo kriterije med seboj, vidimo, da večji kot bo volumen, večja učinkovitost bo poleti, vendar manjša pozimi. Iz grafa na sliki 6.45 vidimo, da se kriterija zmanjšanja volumna in efektivnosti pozimi dopolnjujeta[1]. Ko se zmanjšuje volumen, se zmanjšuje tudi učinkovitost pozimi. Če bi reševali problem le na omenjena dva kriterija, bi dobili precej manjši prostor rešitev oziroma bi lahko v tem primeru dobili tudi samo eno rešitev, saj oba kriterija želita zmanjšati vrednosti vhodnih spremenljivk. Optimalna rešitev bi bila torej najmanjša vrednost števila in širine elementov za senčenje. To seveda ne drži vedno, odvisno je kako spreminjanje parametrov vpliva na vsak kriterij posebej.

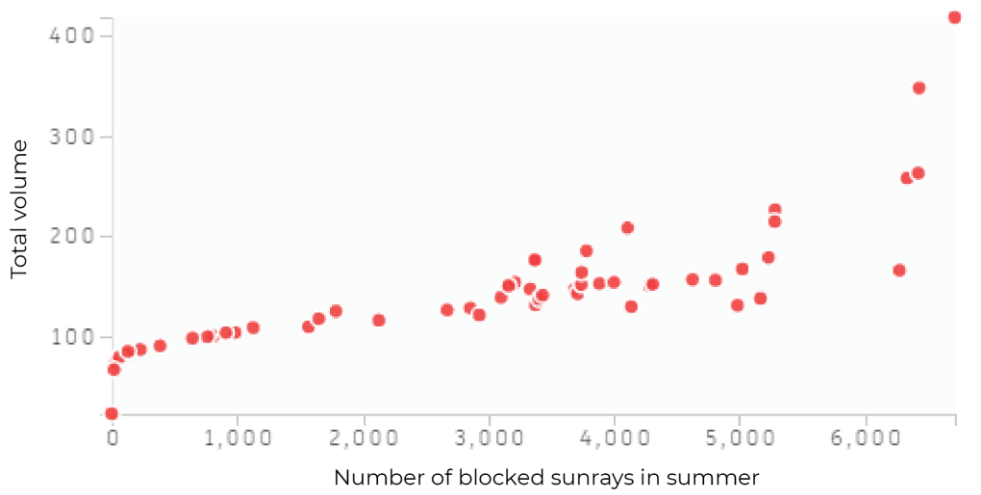
[1] Čeprav v analizi iščemo pri obeh kriterijih minimum, ni to pogoj, da se kriterija dopolnjujeta. Pomembno je kako je kriteriji definiran. Če bi zastavili vprašanje: kolikšnemu številu žarkov želimo prepustiti prehod v notranjost, potem nas bi zanimal maksimum števila. Vendar bi se kriterija še vedno dopolnjevala, saj pri manjšem volumnu tudi več žarkov preide v notranjost.



Slika 6.45: 4D graf raztrosa rešitev, ki prikazuje volumen in število sončnih žarkov pozimi, ki jim je preprečen prehod v notranjost, v odvisnosti od števila in širine elementov (Večje točke predstavljajo večje vrednosti, manjše točke pa manjše vrednosti. Podobno z barvami, rdeča barva predstavlja manjše vrednosti, vijolična pa večje vrednosti).

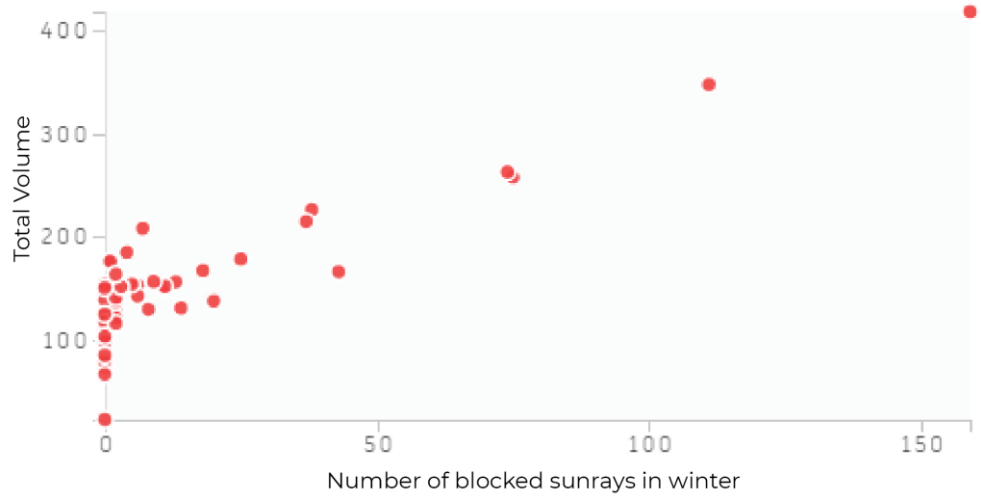
Figure 6.45: 4D scatterplot of total volume and number of blocked sunrays in summer in realtion to number and width of the elements (larger values are displayed as larger points and smaller values are displayed as smaller points. Similarly with colors, larger values are displayed as purple colors and smaller values are displayed as red colors).

Poleg prej omenjenih kriterij imamo še kriterij povečanja števila sončnih žarkov poleti, za katerega vidimo, da nasprotuje prejšnjima kriterijema, saj večja učinkovitost poleti zahteva gostejši razpored širših elementov. To pomeni, da bo volumen večji in tudi efektivnost pozimi večja, kar pa nasprotuje kriterijema. V tem koraku pride pa na vrsto projektant, da analizira rezultate in poišče ravnovesje ter izbere končno rešitev.



Slika 6.46: Graf, ki primerja rešitve med kriterijema volumna in števila sončnih žarkov poleti, ki jim je preprečen prehod v notranjost.

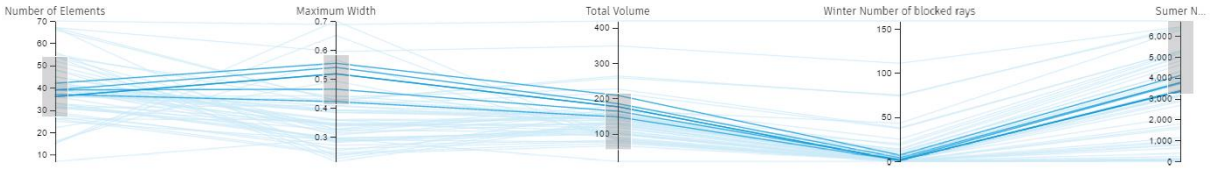
Figure 6.46: Scatterplot of solutions in comparison between the total volume and number of blocked sunrays in summer.



Slika 6.47: Graf, ki primerja rešitve med kriterijema volumna in števila sončnih žarkov pozimi, ki jim je preprečen prehod v notranjost.

Figure 6.47: Scatterplot of solutions in comparison between the total volume and number of blocked sunrays in winter.

Pri izbiri končne rešitve si pomagamo z grafičnim vmesnikom, ki omogoča omejevanje spremenljivk, preko česar zmanjšujemo prostor rešitev. Manjše število rešitev lahko lažje med seboj primerjamo in poiščemo ravnovesje. Pri tem moramo zdaj poiskati končna vhodna parametra, ki jih bomo uporabili za končni rezultat. Ker smo problem zastavili s samo dvema parametroma, zdaj lažje presodimo, ali želimo malo gostejšo razporeditev elementov z manjšimi širinami ali želimo manj širših elementov.



Slika 6.48: Sortiranje med optimalnimi rešitvami z omejevanjem vrednosti spremenljivk.

Figure 6.48: Filtering optimal solutions by adding acceptable range of variable values.

Ker nas zanima videz fasade, subjektivno opredelimo izbrane rešitve manjšega števila in izberemo končne vrednosti števila elementov in največje širine elementov. Ni nujno, da izberemo točno eno izmed predlaganih rešitev, lahko si le pomagamo s predlaganimi rešitvami. V našem primeru smo privzeli eno izmed optimalnih rešitev in nato prilagodili vhodne spremenljivke glede na videz. Pri tem izberemo tudi raje zaokroženo vrednost spremenljivk. Tako si izberemo kot končno rešitev število elementov 41 in največjo možno širino elementov 0.6 čevljev (angl. feet).

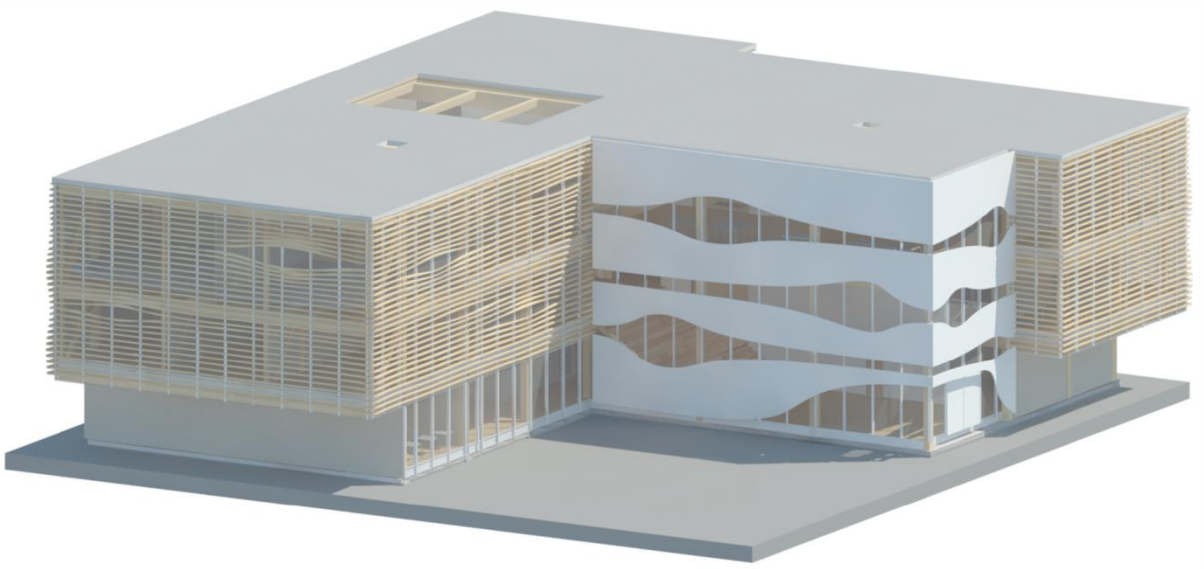
Total Volume	Winter Number of blocked rays	Sumer Number of blocked rays	Number of Elements	Maximum Width
147.459	1.0	3334.0	37	0.422
163.712	2.0	3744.0	39	0.465
176.220	1.0	3372.0	36	0.518
176.359	1.0	3375.0	36	0.519
185.139	4.0	3783.0	39	0.541
208.419	7.0	4110.0	42	0.555

Slika 6.49: Izpis vrednosti spremenljivk od rezultatov dobljenih s sortiranjem.

Figure 6.49: Printout of the variable values from the filtered selection.

6.7 Končna rešitev

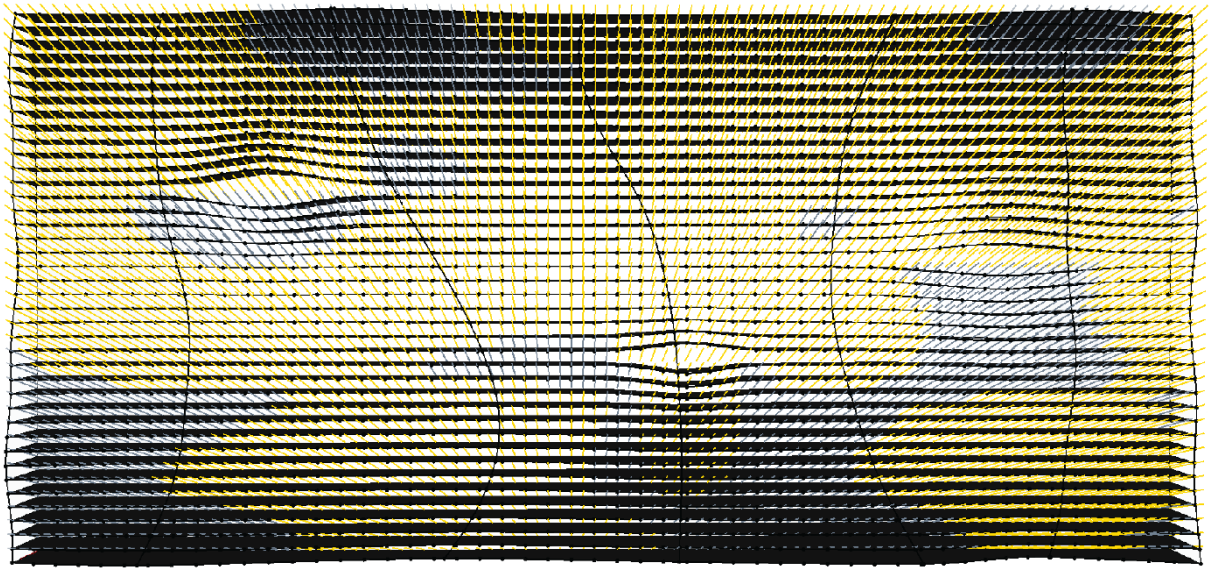
Končna rešitev je poleg objektivnih kriterijev iz postopka optimizacije sestavljena še iz subjektivnih kriterijev projektanta. Poleg objektivnih rešitev iz optimizacije je potrebno preveriti še izvedljivost, videz in ostale kriterije, ki bi jih težje matematično opredelili.



Slika 6.50: Prikaz podobe elementov za senčenje na objektu.
Figure 6.50: Visualization of how the shading elements look like when applied to the building.

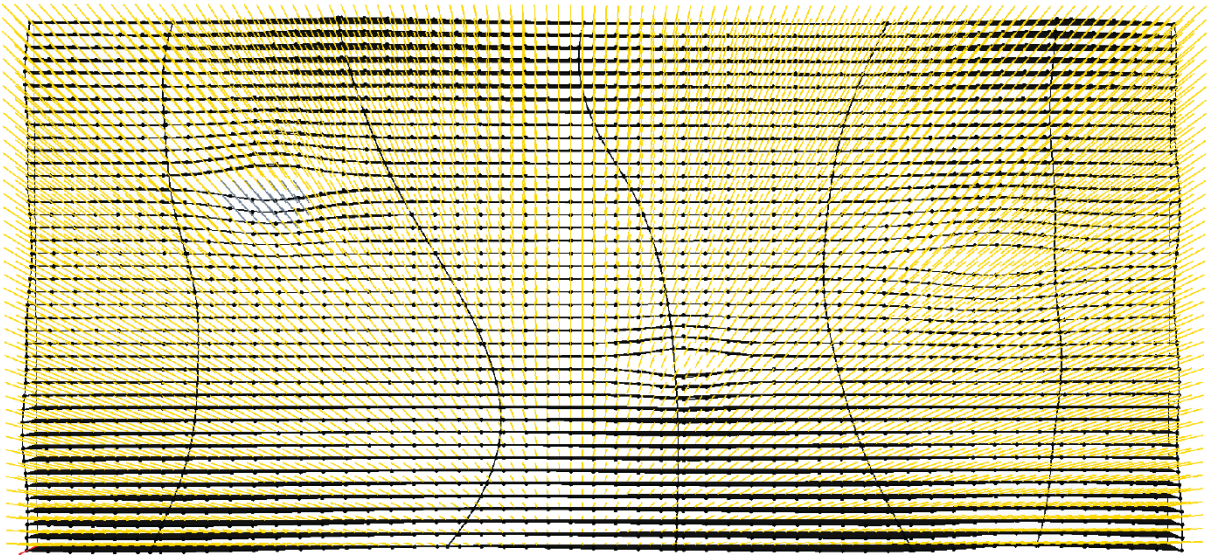
Ko se je problem reševalo brez uporabe generativnega pristopa, se je izkazalo, da težje raziščemo prostor rešitev, ki nam bi drugače pomagal pri razumevanju, kako različni parametri vplivajo na rezultate. V primerjavi z ročnim spreminjanjem parametrov zelo počasi iščemo rešitve manjšega števila, medtem ko preko algoritmov hitreje in v mnogo večjem številu raziščemo prostor rešitev. Prek samostojnega reševanja je bila dobljena rešitev, ki je bila na prosto oko zadovoljiva, vendar se je izkazalo, da obstajajo mnogo bolj optimalne rešitve. Če primerjamo rešitve pred in po uporabi algoritmov, je bil volumen pred optimizacijo 664 ft^3, po optimizaciji pa 216 ft^3, kar pomeni več kot 307% redukcijo v porabi materiala.

V tem poglavju smo spoznali, kako si s pomočjo računalniške zmogljivosti in naprednih algoritmov hitreje in lažje pomagamo pri iskanju rešitve. Obdelan postopek je primer generativnega pristopa pri projektiranju.



Slika 6.51: Grafični prikaz prehoda sončnih žarkov skozi senčenje v zimskih mesecih pred optimizacijo. Rumena barva predstavlja žarke, ki preidejo skozi senčenje, siva barva predstavlja žarke, ki jih senčenje zaustavi. Za zimske mesece želimo povečati število žarkov, ki preide v notranjost, torej želimo videti več rumenih žarkov.

Figure 6.51: Visualization of sunrays passing through the shading in winter months before the optimization process. Yellow color indicates the sunrays passing through the shading, grey color indicates sunrays blocked by the shading. For winter months we want to maximize the number of sunrays passing through, meaning we want to see more of yellow sunrays.



Slika 6.52: Grafični prikaz prehoda sončnih žarkov skozi senčenje v zimskih mesecih po optimizaciji. Rumena barva predstavlja žarke, ki preidejo skozi senčenje, siva barva predstavlja žarke, ki jih senčenje zaustavi.

Figure 6.52: Visualization of sunrays passing through the shading in winter months after the optimization process. Yellow color indicates the sunrays passing through the shading, grey color indicates sunrays blocked by the shading.

7 ZAKLJUČEK

V magistrski nalogi smo spoznali princip generativnega pristopa, ki je bil apliciran na primeru projektiranja elementov za senčenje. Primer iz realnega sveta je bilo potrebno poenostaviti in razčleniti za potrebe algoritmičnega zapisa, da ga nato lahko rešimo z uporabo optimizacijskih algoritmov. Pri tem so bile izbrane vhodne spremenljivke in kriteriji, glede na katere se alternativne rešitve oceni in izbere najboljše rešitve v trenutni iteraciji. Genetski algoritmi nato v procesu iteracij iščejo optimalne rešitve. Ker je bilo v našem primeru zastavljenih več kriterijev, po katerih algoritem ocenjuje rezultate, je bilo dobljenih več optimalnih rešitev. S pomočjo grafičnega vmesnika se je omejil prostor možnih rešitev, iz katerega lahko lažje primerjamo rezultate in izberemo končno rešitev. Ugotovljeno je bilo, da prek ročnega spreminjanjem parametrov, lahko raziščemo le manjše število rešitev, kot pa če uporabimo algoritme za reševanje, ki so sposobnejši analizirati večje število podatkov. Tako smo preko generativnega pristopa dobili rešitev, ki ima trikratno manjšo porabo materiala in še vedno učinkovito senčenje, kot pa rešitev dobljena preko ročnega reševanja, kar tudi potrjuje začetno hipotezo.

Za reševanje se je uporabil generativni pristop v katerem se je iskalo optimalne rešitve z genetskimi algoritmi. S tem pristopom se osredotočimo na analizo generiranih rešitev, da lahko razumemo problem, medtem, ko pri klasičnem projektnem pristopu analiziramo namene problema in vhodne pogoje, predno pridemo do kakršnekoli rešitve. To pomeni, da s takim pristopom pridemo do rešitve, vendar ne vemo ali je ta optimalna ali ne.

Največja prednost generativnega pristopa je ravno v tem, da analiziramo prostor rešitev in pri tem lahko ugotovimo, kje se nahaja optimalna rešitev. To lahko omogoča zelo velike prihranke, odvisno od problemov, ki jih analiziramo, izkušenj ter občutka projektanta. Ta pristop lahko uporabimo tudi za preliminarne analize, poiščemo optimalno različico, ki jo nato analiziramo po starih postopkih. Slabost pristopa je, da zapis problema v algoritmičnem načinu zahteva nova znanja, predvsem abstraktni in programerski način razmišljanja. Omejitev tega pristopa je tudi, da vseh problemov na katere naletimo v gradbeništvu, ne znamo opisati na tak način, ker so preveč kompleksni, vključujejo preveč parametrov, ki so med seboj povezani na nepredvidljive načine. Ta problem t.i. simbolične umetne inteligence rešujejo nevronske mreže, ki se reševanja problemov naučijo iz množice učnih primerov, ne da bi problem formalizirala v informacijske strukture in algoritme.

Glavna ideja generativnega pristopa je vzpostavljanje sinergije projektanta s pametnim računalnikom, ki projektanta dopolni, kjer ima le-ta slabosti: pregled velikega števila podatkov, analiza ponovljivih problemov, avtomatizacija dolgotrajnih postopkov, itd.

Uporaba genetskih algoritmov je trenutno najbolj razširjena na področju prostorskega načrtovanja. Pri tem se pojavlja vedno več možnosti uporabe, npr. iskanje optimalne trase tunela ali ceste, iskanje razporeditev prostorov v zgradbi, iskanje razporeditve konstrukcijskih elementov pri vertikalni obtežbi, iskanje konstrukcijskih sistemov zgradb, konceptualno načrtovanje mostov, ipd. Če problem lahko opišemo parametrično, potem ga lahko tudi analiziramo z raziskanimi algoritmi. Če ga ne znamo, bi bilo treba računalniško pomoč iskati v tehnologiji nevronskih mrež in drugih primerov ne-simbolične umetne inteligence.

Oboje bo v bližnji prihodnosti premet tako akademskih raziskav kot razvoja programske industrije. Računalnik bo postal več kot samo orodje. Tudi prihodnost dela inženirja bo vse bolj dobivala obliko timskega dela s strojnim pomočnikom.

8 VIRI

[1] Refinery Primer. 2019. What is Generative Design. https://refineryprimer.dynamobim.org/01-introduction/01-02_what-is-generative-design (Pridobljeno 18. 08. 2019).

[2] Stasiuk, D. 2018. Design Modeling Terminology. https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf (Pridobljeno 29. 07. 2019).

[3] DesignTech. 2019. Parametric Modelling. https://www.designtechsys.com/articles/parametric-modelling (Pridobljeno 28. 08. 2019).

[4] Dynamo Primer. 2019. What is Visual Programming. https://primer.dynamobim.org/01_Introduction/1-1_what_is_visual_programming.html (Pridobljeno 26. 08. 2019).

[5] Nagy, D. 2017. Introduction to computational design. https://medium.com/generative-design/introduction-to-computational-design-6c0fdfb3f1 (Pridobljeno 14. 08. 2019).

[6] Analytica Wiki. 2019. Optimization Characteristics. http://wiki.analytica.com/Optimization_Characteristics (Pridobljeno 26. 08. 2019).

[7] Guid, N., Strnad, D. 2015. Umetna inteligenca. Maribor, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko.

[8] Autodesk. 2019. Revit Features. https://www.autodesk.com/products/revit/features (Pridobljeno 22. 08. 2019).

[9] Dynamo Primer. 2019. What is Dynamo. https://primer.dynamobim.org/01_Introduction/1-2_what_is_dynamo.html (Pridobljeno 22. 08. 2019).

[10] Refinery Primer. 2019. What is Refinery. https://refineryprimer.dynamobim.org/01-introduction/01-09_what-is-refinery (Pridobljeno 21. 08. 2019).

[11] PBL Lab Stanford. 2019. Team Pacific Spring Presentation. http://pbl.stanford.edu/AEC%20projects/Year%202018-2019/Spring/Pacific-Spring2019.pdf (Pridobljeno 12. 08. 2019).

[12] Refinery Primer. 2019. Solvers. https://refineryprimer.dynamobim.org/05-algorithms/05-04_solvers (Pridobljeno 13. 08. 2019).

[13] Autodesk. 2019. Insight. https://www.autodesk.com/products/insight/overview (Pridobljeno 28. 08. 2019).

[14] Ladybug Tools. 2019. Tools. https://www.ladybug.tools/index.html (Pridobljeno 28. 08. 2019).