Capstone Project

ART 803 - Algorithmic Design Thinking

Automate generation of building that fit with rules of set backs and building permit checks

Darshan Chavan
C21107575

Content

01 | Algorithmoc design problem

Problem statement

The proposed design problem focuses on generating a building on a site with setbacks rules and building permits having controls on all parameters. The proposal attempt is to create a set of components in Rhino Grasshopper that is user friendly for designers.

Introduction

The problem is based on the space planning where the user can define site boundary and design the whole site layout with certain parameters. Further, the site can be divided into small blocks according to the user on which building layout can be generated by given parameters. Later, the building layout will be defined in detail and extruded upwards randomly with some parametric control, or the building layout can be divided into small blocks and extruded randomly. Any building permit rules can be incorporated with the algorithm for the practical use.

Users

The proposed algorithm is designed for an architect, city planner who can quickly generate site planning and massing with few inputs and parameters. As it says generative, it can produce multiple options for the same site. The same model can be analysed further by other components like TopologyEnergy. It will enhance the process of architects, the algorithm will take care of precision, which will assist architects to invest time in other complex details.

Design problem significance

The nature of the problems is very realistic and practical. The generation of site layout and building with respect to the bye-laws has a good potential to explore and many things to look at. The challenge is to make it as flexible as possible by giving parameters that can be controlled later for a better result. The automate generation could be useful for complex model, where building can be generative by a few inputs and parameters.

02 | Existing methods

There are methods available on various platforms which attempts to solve this design problem. Some methods are solely developed software for this particular problem and offer a wide range of options in it. Other methods are the combination of components(script) developed for software like Rhino-Grasshopper, Re-vit-Dynamo. There is not much explicitly explanation of the process, how the algorithm works at the back end.

Softwares :

There are softwares that focuses on generative building design problems. Softwares like TestFit, Kreo, Digital Blue Foam, Archistar, Hypar provides a solution to generative design and most of them are based on Artificial Intelligence, mainly on Machine Learning models for accurate and quick response. The software provides many different facilities based on the user(Architect, developer etc.).

Some of the common features of the software's are :

- Planning rules
- Fexlibility in editing
- Interactive interface
- Building shapes
- Environmental analysis
- Building type
- Comparison of data
- Surrounding context
- 2D, 3D export
- Site planning
- Generative design

Digital Blue Foam:

For instance, we will further briefly explore the Digital Blue Foam software to understand the workflow of it and other similar sofwares.

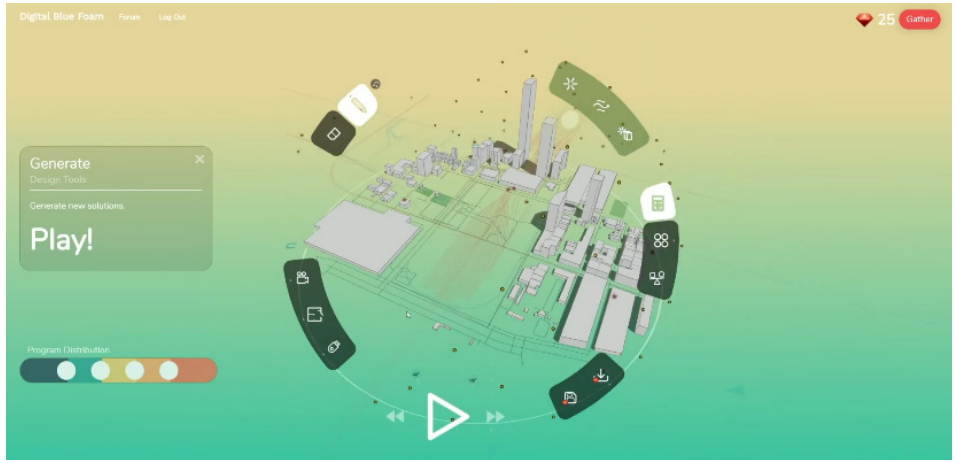


Fig 01. Interface of Digital Blue Foam Software

Context:

- Neighbouring building, roads, building types, terrain, sun path and wind path analysis.

Generating design:

- Defining of site and generating designs are by parameters like no. of subplots, site coverage, max height, total park area, floor area ratio(FAR), Gross floor area(GFA).

Pencil:

- The feature creates a building design along the curve and curve is translated to road.

Modification:

- Any building block can be selected and edit independently .
- It can be edited in the sense of its footprint, height, volume, scale, rotation, building type etc
- It is possible to carry out complex operations like adding podium to two different building, merging plot, editing.

Analysing and saving:

- After saving, five generative design solution can be viewed and compared.
- If not satisfied again we can go back to the generating design process.

Exporting:

- The final outcome can be exported in image via Portable Graphics Format(PNG), in 2D through Drawing Exchange Format (DXF) file and 3D in Standard Tessellation Language (STL) file

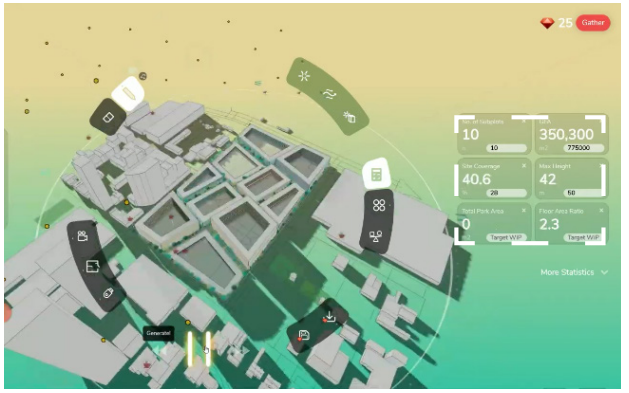


Fig 02. Generative design initial parameters.

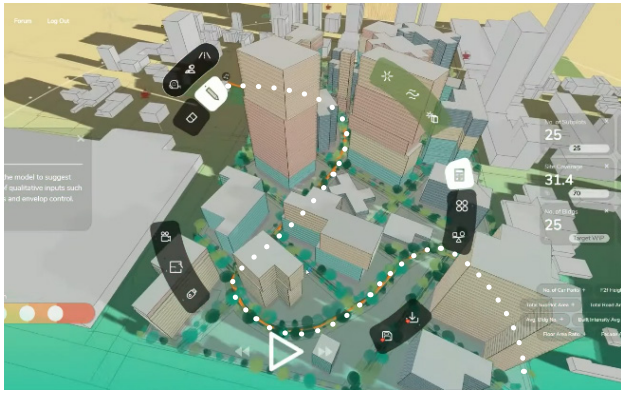


Fig 03. Pencil feature, line acting as road

Fig 04. Modification of building block by selection.

Revit/ Dynamo:

Generative Design for Architectural Space Planning

The case study was conducted by Autodesk to explore generative design in terms of Architectural Space Planning. It is very much related to the first part of the problem, the design site. The goal of the case study is to create a parametric site with a generative aspect and analyse it.

The project is based on a dynamo script and heavily dependent on python coding for computing the inputs. The script is divided into:
Parameters, Layout, Inputs, Python scripts, Visualisation, Python scripts to export into scalable vector graphics(SVG).
Further inhouse optimisation library o2 is used to analyse the model. Multiple SVG's are created and analysed at the backend of the dynamo which can be viewed by the web link located in the o2 package.

Script logic:

(2a) Primary main avenuesare created on site.

(2b) Later main avenues are divided into two small region resulting in secondary avenues.

(3) In secondary avenues, regular gridwill be developed with varying orientation.

(4) Seeds are spreaded on the primary avenue path with different zones some in between the line and at the end points.

(5) The seeds grows in direction of the neighbouring blocks untill the program meet its requirement.

(6)Finally. remaining cells are populated with booths and exhibitos are assigned

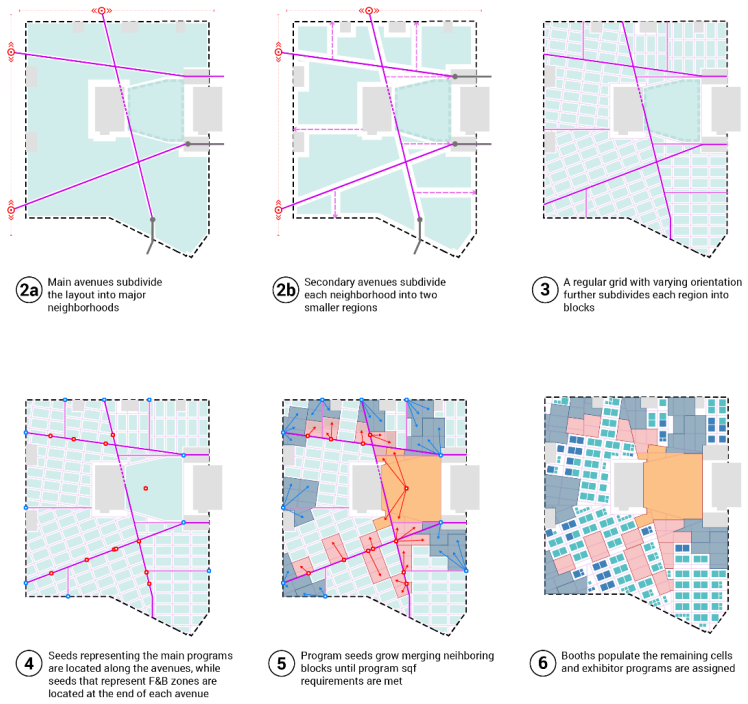


Fig 05. Explanation of Script logic.

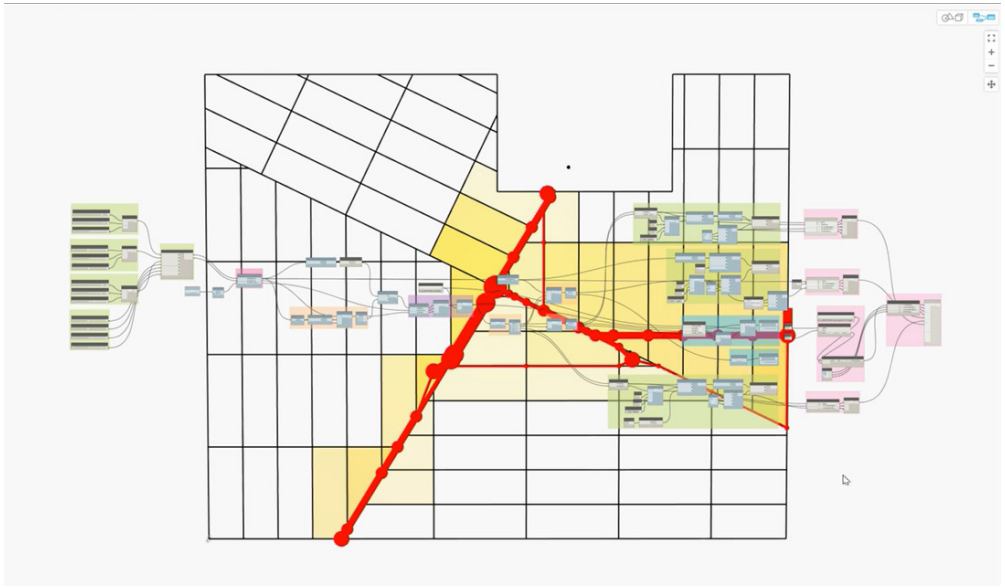


Fig 06. Script of the Generative architectural space planning.

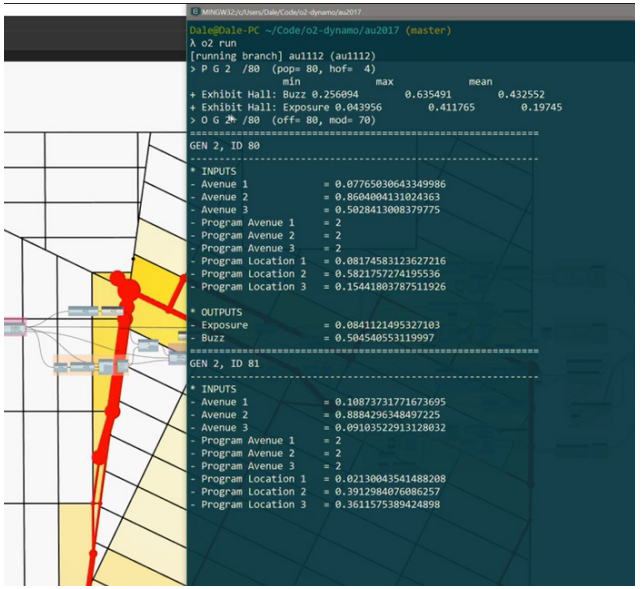


Fig 07. O2 python library to generate, bake(into imaage) and analyse design options.

Fig 09. Analysing and comparing the generated design option.

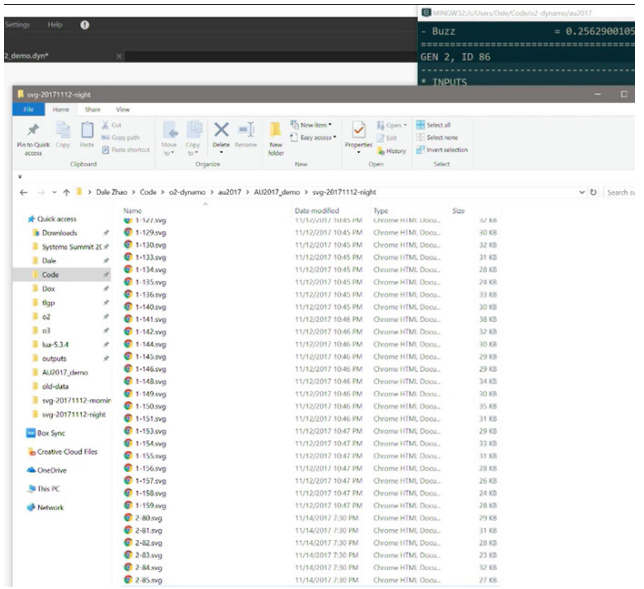


Fig 08. All design options are saved in SVG

Generative House

A generative house is a small exercise of generating an internal block floor plan within the site boundary. The script is built with the dynamo component and some customised code blocks. The focus of the example is to use the space planning principle to achieve a floor layout with having few options. Any area/block can be targeted to study. Later the model creates multiple design and analyse it.

The script is designed, having some parameters like site dimension, building block length-width, min-max area, corridor width, some optional functions. The script is processed and gives a single solution based on the set parameters. To generate multiple design option script is linked with Generative design Addon and exported to Revit as a study. Generative design addon gives few options to generate designs and later analyse them.

Dynamo script is exported to Revit with the generative design Addon to create multiple design options and to analyse it further.In 2d there are few options to generate design options, later in (Fig.) all parameters are available which were in ealier used in dynamo script. It can be change according to the user.

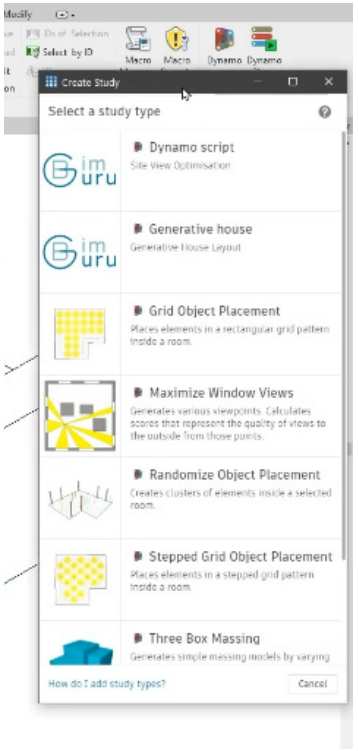


Fig 11.Generative design addon.

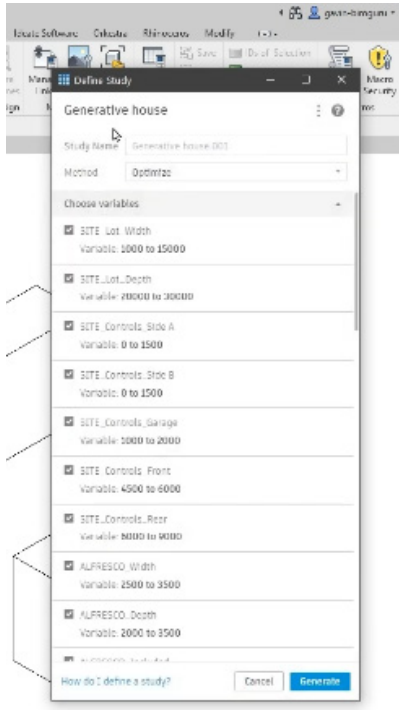


Fig 12.Parameters to control design

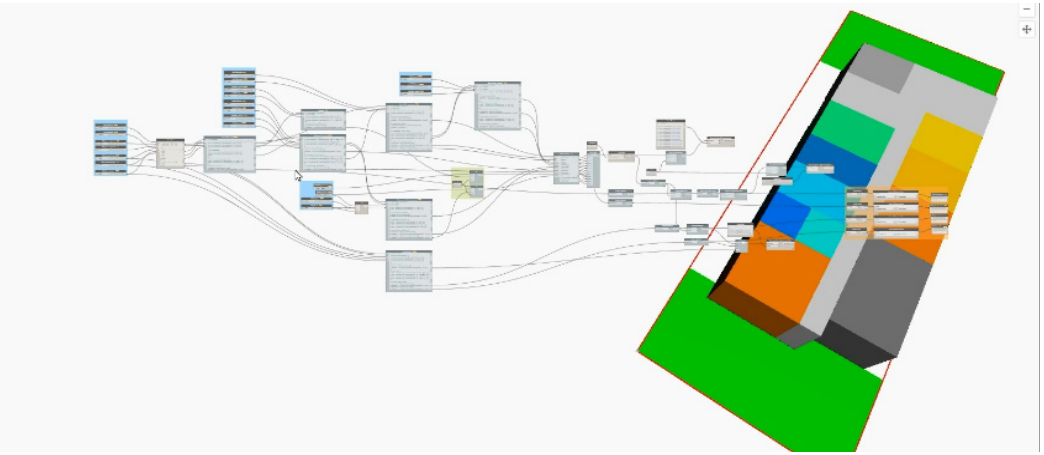


Fig 10.Script of generative house.

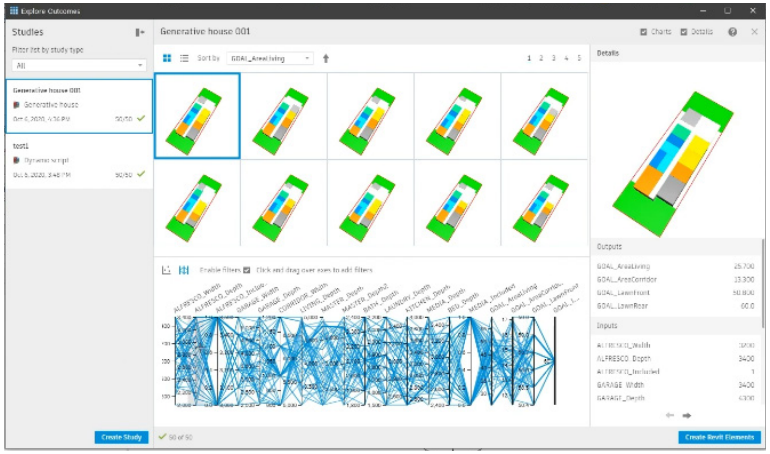


Fig 13.There are sorting and refine mode to make it presise and accurate as per user need.

Rhino-Grasshopper :

Probably, there are very less and under-developed grasshopper plugin which directly deals with the generative design building with site conditions. Few plugins are available which are based on the generative space planning design.

Spheniscidae

Spheniscidae is a grasshopper plugin for automatic generation of buildings with respect to site boundary and indexes such as FAR, height, length and width. To generate buildings, site boundary, building unit information snd indexes in Regulation in required. It also import and exports comma seperated values(CSV) files. The plugin seems to have many lmitation and there are no tutorial available of the plugin.

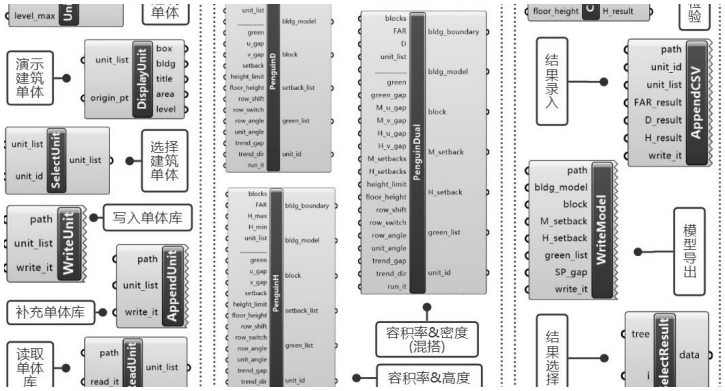


Fig 14. Components of Spheniscidae plugins

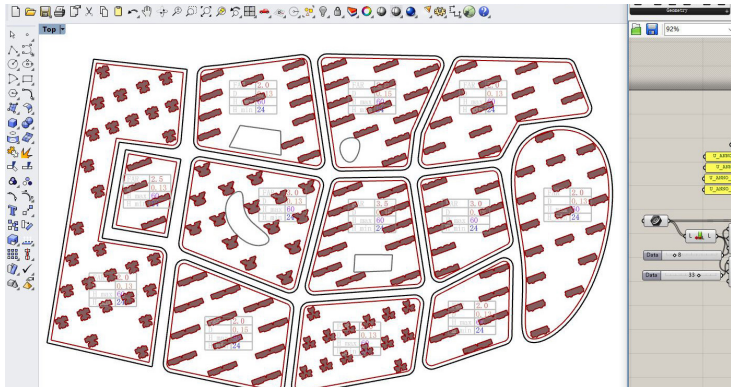


Fig 15. Demo of the plugin

Termite nest

The plugin is developed to provide utilities for space planning. It uses graph analysis and syntactic tools which helps to generate spatial graph within any given boundary.The user defined generator translate the generated spatial graph into diffierent morphological shapes. Further it can geometry can be synced to diifer-ent energy plugin to optimise it.

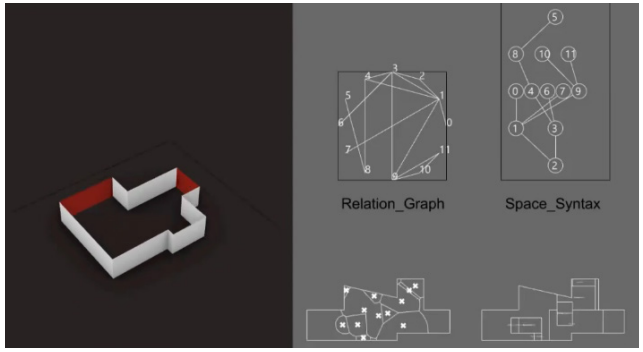


Fig 16. Implementation of the plugin

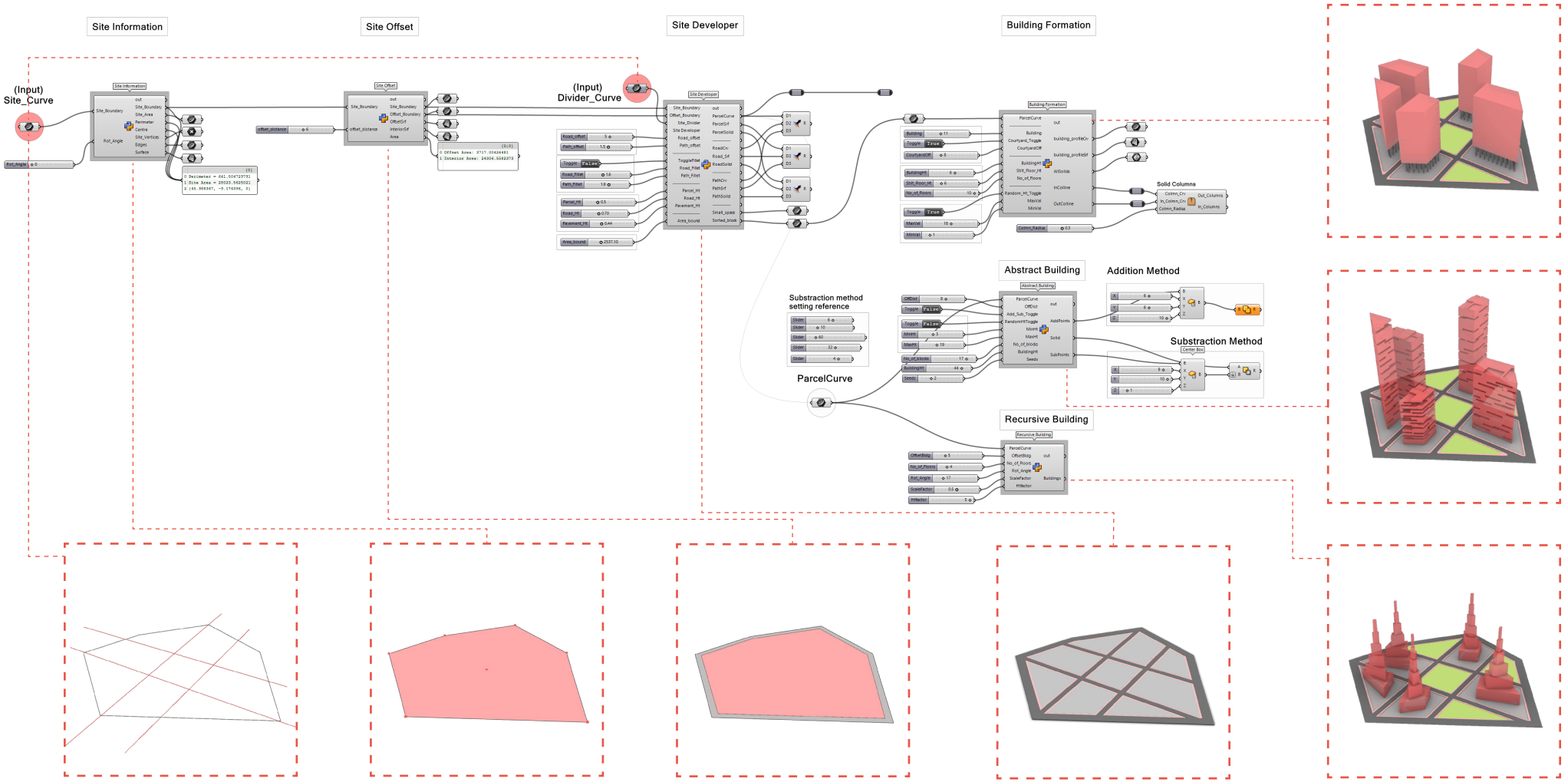
Python:

There is no direct python learning, the learning and logic development was developed by ART-803 Classes and other learning is done by bits and pieces, mostly looking into the forums and some tutorials. To code and explore, the rhino Api's was prefered which has detailed information about the functions.

03| Overview of Components

The aim of the project is to code a sets of components which can be used for site development and further for buiding massing. All together there as 6 components scripted in Rhino - Grashopper Python and each perform different task. The components are targeted to all the users and components are design with some inbuilt parameter which can be incorporated with site planning rules.

Set of components

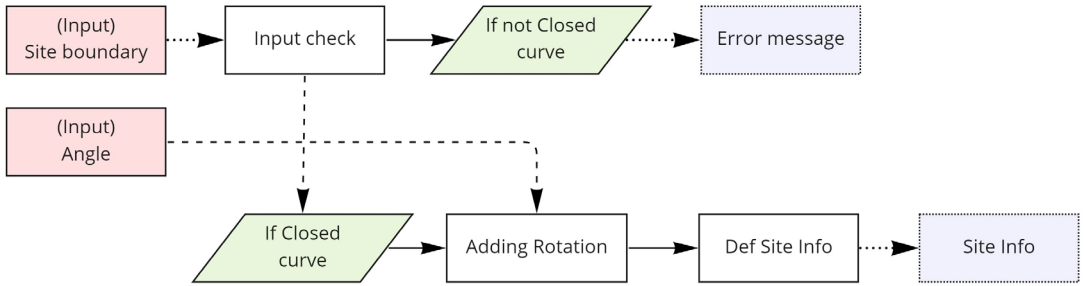


04.1 | Component 01

Site Information

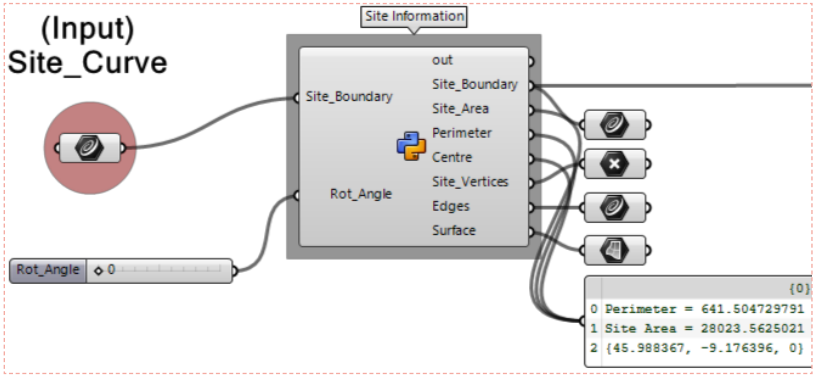
The components gives the basic information about the site(closed curve) and allows to rotate the site incase to adjust north direction at any stage of designing.

Script logic:

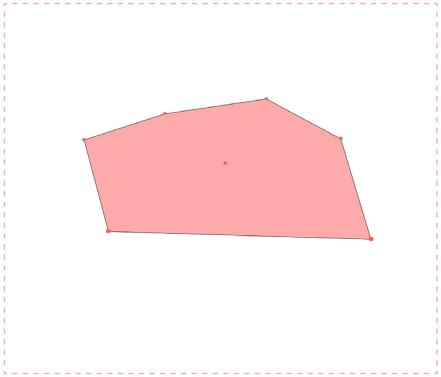



```
'''      Importing library       '''

import rhinoscriptsyntax as rs

'''      check IF its a closed curve      '''

def checkcurve(Site_Boundary):

    #If input is not curve
    if not rs.IsCurve(Site_Boundary):
        print("Error !!! Input needs to be closed curve")
    #If curve is not Closed curve
    if not rs.IsCurveClosed(Site_Boundary):
        print("ERROR !!! Input curve needs to be closed")

    else:
        #If it is a closecurve
        pass

#Calling function "checkcurve"
checkcurve(Site_Boundary)
```

Code to check if input is closed curve



Site information component



Expected visual output

```
'''      Function General information       '''

#Rotation
Rot_site_Crv = rs.RotateObject(Site_Boundary, rs.CurveAreaCentroid(

def General_Info(Rot_site_Crv):
    
    #centre
    centre = rs.CurveAreaCentroid(Rot_site_Crv)
    
    #site Area
    Site_area = rs.CurveArea(Rot_site_Crv)
    
    #Perimeter
    Perimeter = rs.CurveLength(Rot_site_Crv)
    
    #vertex count
    vertex_count = rs.CurvePoints(Rot_site_Crv)
    
    #Edges
    Edges = rs.ExplodeCurves(Rot_site_Crv)
    
    #Site surface
    Surface = rs.AddPlanarSrf(Edges)
    
    return Rot_site_Crv, Site_area[0], Perimeter, centre[0], verte
```

Code for site infomation

Input
1) Site boundary(close curve)
2) Rotation angle

Output
1) Rotated Site
2) Site area, perimeter
3) Site centre point, vertices
4) Site curves, surface

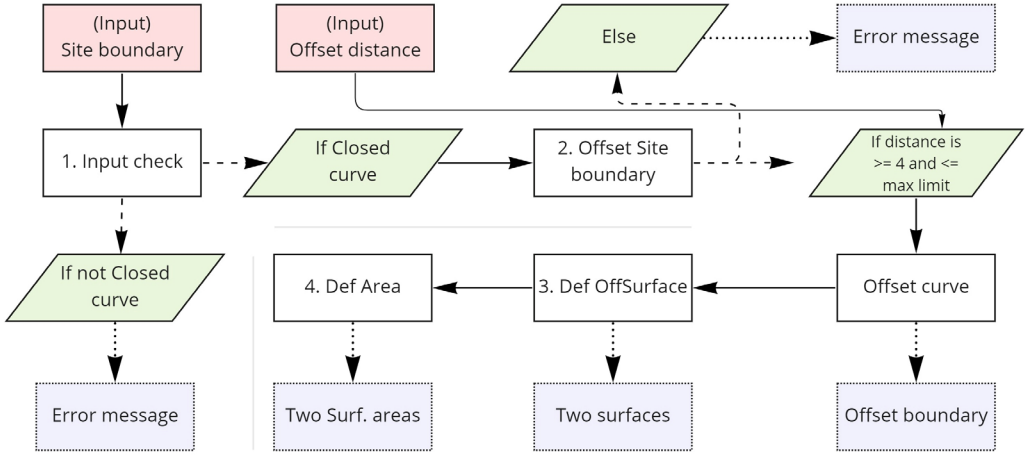
This same component further can be utilise to get information about the divided land parcels.

04.2 | Component 02

Site Offset

The component can offset the site curve. The offset have some inbuilt parameter for building setback rules to avoid any design failiure.

Script logic:



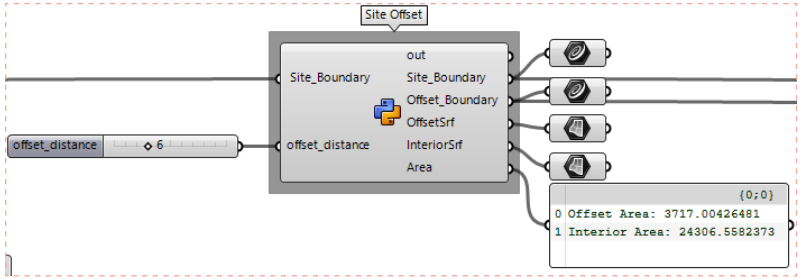


Fig 23. Site offset component

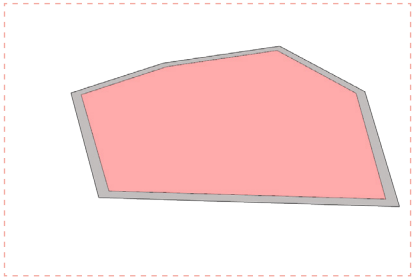


Fig 23. Output of the component

Input
1) Site boundary(close curve)
2) Offset distance

Output
1) Site boundary, offset curve
2) Surface and area of usable and offset area.

The offset site curve will be divided to get sets of land parcels for the development.

```python
'''      Function Offset      '''

def Offset(Site_Boundary, offset_distance):
    #Centre of the curve
    centreCC = rs.CurveAreaCentroid(Site_Boundary)

    #Getting vertex of the site
    vertex = rs.CurvePoints(Site_Boundary)

    #Converting centre 3dpoint to GUID
    Guidpoint = sc.doc.Objects.AddPoint(centreCC[0])

    #Creating line to from centre to site vertex
    line = rs.AddLine(vertex[0], Guidpoint)

    #Length og the line
    length = rs.CurveLength(line)

    #Setting max limit of offset
    maxlimit = length / 5

    #Setting the minimum and maximum offset parameter
    if offset_distance >= 4 and offset_distance <= maxlimit:
        offsetcrv = rs.OffsetCurve(Site_Boundary, centreCC[0], float(offset_distance))
    else:
        # Error message for user
        offsetcrv = "Offset limit cannot be less than 4 M"
    #Return
    return offsetcrv

#Calling function offset
offsetcrv = Offset(Site_Boundary, offset_distance)
```

If statement used to set parameter for offset.
offset <=4 and
<= half distance from centre

Fig 23. Code to offset site boundary

```python
'''      Function for Surface      '''

def offsetSrf(Site_Boundary, offsetcrv):
    #Apending curves to empty list
    curves = []
    curves.append(Site_Boundary)
    curves.append(offsetcrv)
    # Surface from curves
    offsetsurface = rs.AddPlanarSrf(curves)

    #Return
    return offsetsurface

#Calling def offsetSrf
offsetSrf = offsetSrf(Site_Boundary, offsetcrv)

#Interior surface
InteriorSrf = rs.AddPlanarSrf(offsetcrv)

'''      Function to append Area of the two seperate Closedcurve      '''

def Area(offsetSrf, InteriorSrf):
    #Area
    OffsetArea = rs.SurfaceArea(offsetSrf)
    IntArea = rs.SurfaceArea(InteriorSrf)

    #List to append both curve areas
    Area = []
    Area.append("Offset Area: " + str(OffsetArea[0]))
    Area.append("Interior Area: " + str(IntArea[0]))

    #Return
    return Area
```

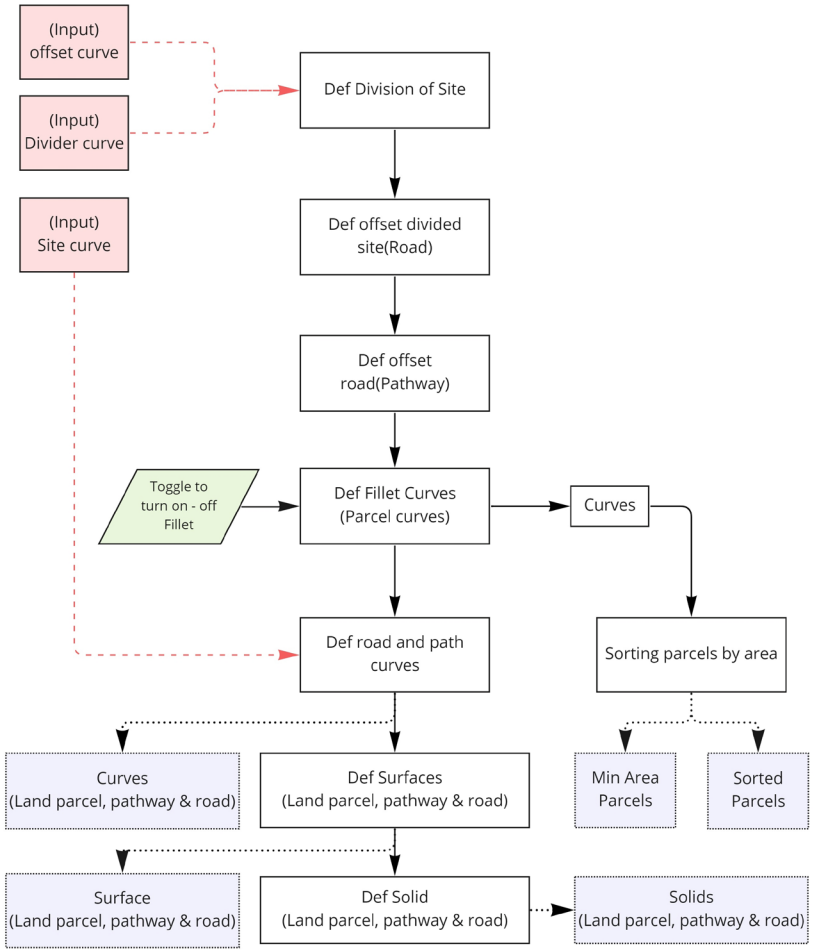
Fig 23. Surface and Area of two surfaces

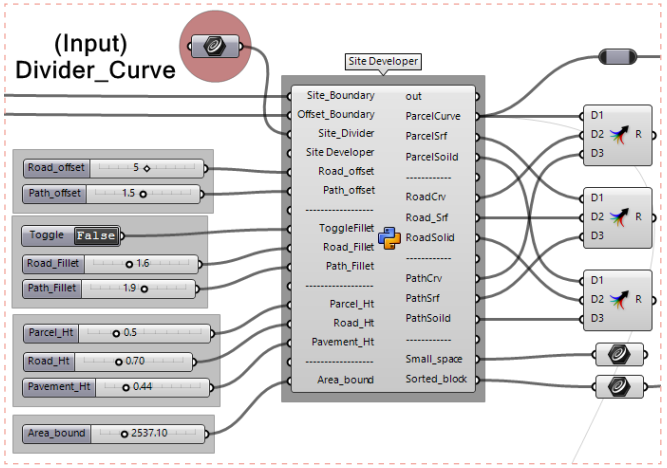
04.3 | Component 03

Site Development

The component can be use to develop the site. It can be divided by the divider(curves) to make set of land parcels and further road and pathways can be created. The land parcel can be sorted according to the site area by the given site area bound. Inside the components there are some set parameter to control the design developed by user.

Script Logic

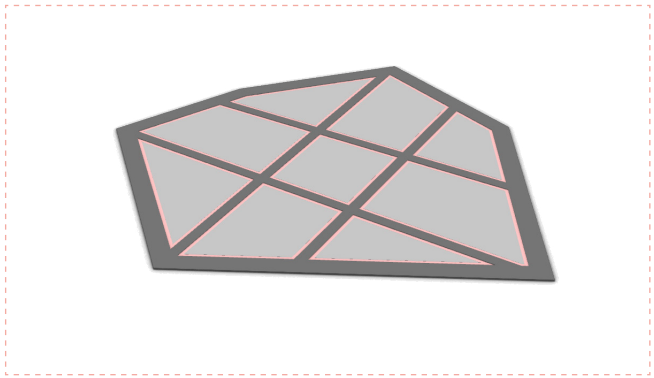




Site development component

Input
1. Site boundary(close curve)
2. Offset curve
3. Divider

Output
1. Curves of parcels, road, path.
2. Surface of parcel, road, path.
3. Solid of parcel, roadm path.
4. Green space(smaller area)
5. Sorted parcels



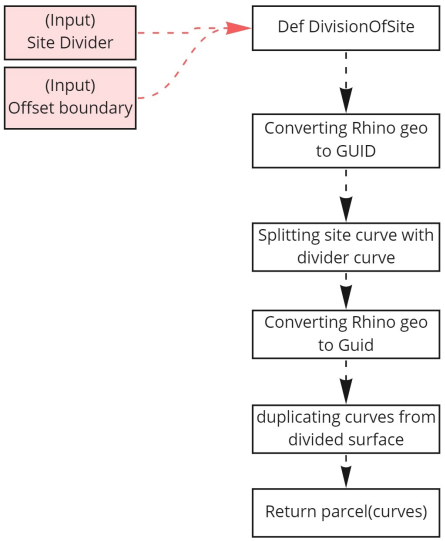
Output of component

The divided land parcels can be further divided in to different development area like buildings plots, park, parking etc.

Division of Site

```
'''       Importing libraries     '''

import rhinoscriptsyntax as rs
import ghpythonlib.treehelpers as th
import Rhino.Geometry as rg
import scriptcontext as sc
import ghpythonlib.components as ghc #used only thrice in this script

'''      Division of Site         '''

def DivisionOfSite(Site_Divider, Offset_Boundary):
    # Converting Guid to Rhino Geometry( Dividing Curves)
    RG_Site_Divider = []
    for i in Site_Divider:
        RG_Site_Divider.append(rs.coercecurve(i))

    # Converting Guid to Rhino Geometry( Boundary Site Curves)
    RG_Offset_Boundary = rs.coercecurve(Offset_Boundary)

    # Spliting site boundary using Grasshopper component in Python
    RGParcels = ghc.SurfaceSplit(RG_Offset_Boundary, RG_Site_Divider)

    # Converting Rhino geo Brep to GUID
    Parcel = []
    for i in RGParcels:
        Parcel.append(sc.doc.Objects.AddBrep(i))

    # Extracting edges from Surfaces
    Parcels = []
    for i in Parcel:
        Parcels.append(rs.DuplicateSurfaceBorder(i))

    #Return
    return Parcels

Parcels = DivisionOfSite(Site_Divider, Offset_Boundary)
```

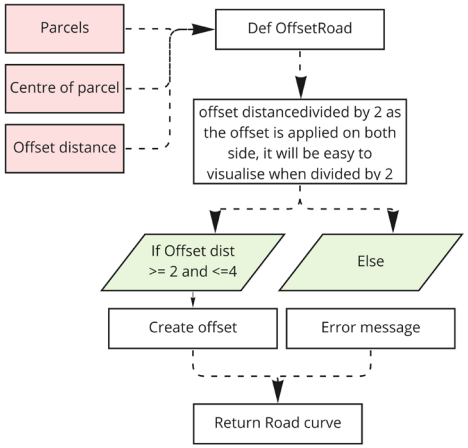
Script logic:

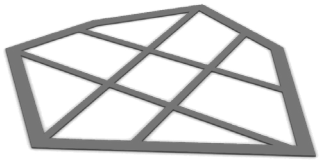


Offset road

```
'''     Centre of the Parcels       '''

#Centre of every curve
centre = []
for i in Parcels:
    centre.append(rs.CurveAreaCentroid(i))

'''     Offset for Road     '''
def OffsetRoad(Parcels, centre, Road_offset):
    '''
    #Dividing by 2 to make easier for user
    #Explaination : If Input from user is 4, 8m wide road is created(fillet on b
    #To avoid it it is divided by 2.

    Offparam = Road_offset/2

    #Setting restriction for offset value
    if Offparam >= 2 and Offparam <= 4:
        #Iterating and appending the offset curve to empty list
        OffsetRd = []
        for i, j in zip (Parcels, centre):
            OffsetRd.append(rs.OffsetCurve(i, j[0], Offparam))

    else:
        #Error message to user guidance
        OffsetRd = ("ERROR!!! Road width cannot exceed 8M")

    #Return curves
    return OffsetRd

#Calling function Offset
OffsetRd = OffsetRoad(Parcels, centre, Road_offset)
```

Script logic:

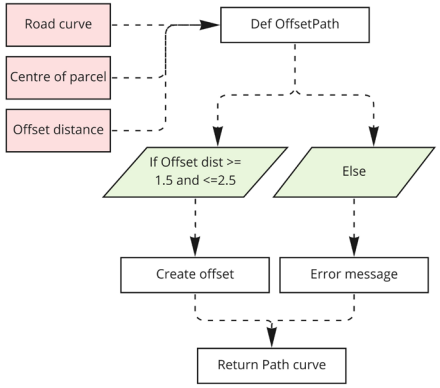


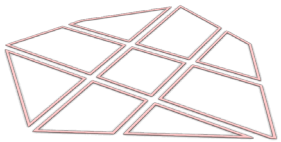


Offset path

```
'''     Offset for Path     '''
def OffsetPath(Path_offset,OffsetRd):
    #Setting restriction for offset value
    if Path_offset >= 1.5 and Path_offset <= 2.5:
        #Centre of every curve and appending to empty list
        OffsetRdCentre = []
        for i in OffsetRd:
            OffsetRdCentre.append(rs.CurveAreaCentroid(i))

        #Iterating and appending the offset curve to empty list
        PathCrv = []
        for i, j in zip(OffsetRd, OffsetRdCentre):
            PathCrv.append(rs.OffsetCurve(i, j[0], Path_offset))

    else:
        #Error message to user guidance
        PathCrv = ("ERROR!!! Pathway width cannot be less than 1M and more tha

    #Return curves
    return PathCrv

# Calling function OffsetPath
PathCrv = OffsetPath(Path_offset,OffsetRd)
```

Script logic:

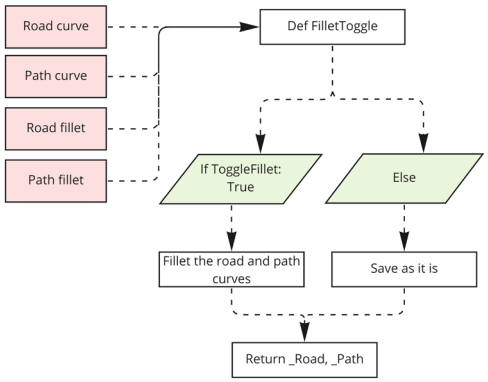


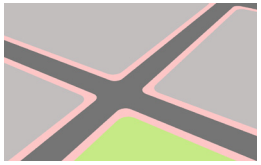


Fillet curve

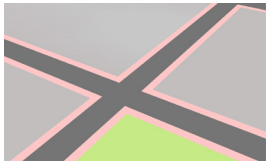
```
'''        Fillet Function     '''

def Fillet(OffsetRd, PathCrv, Road_Fillet, Path_Fillet):
    # Toggle for fillet option(Toggle: On)
    if ToggleFillet:
        # Iterating road and path curves to fillet it and storing in empty list
        _Road = []
        _Path = []
        for i,j in zip(OffsetRd, PathCrv):
            _Road.append(rg.Curve.CreateFilletCornersCurve(
                rs.coercecurve(i), float(Road_Fillet), 0.001, 0.001))
            _Path.append(rg.Curve.CreateFilletCornersCurve(
                rs.coercecurve(j), float(Path_Fillet), 0.001, 0.001))

    #(Toggle: Off)
    else:
        # Storing road and path curves without filleting
        _Road = OffsetRd
        _Path = PathCrv

    # Return of road and path curves
    return _Road, _Path

# Calling function Fillet
_Road = Fillet(OffsetRd, PathCrv, Road_Fillet, Path_Fillet)[0]
_Path = Fillet(OffsetRd, PathCrv, Road_Fillet, Path_Fillet)[1]
```

Script logic:





Fillet Toggle On

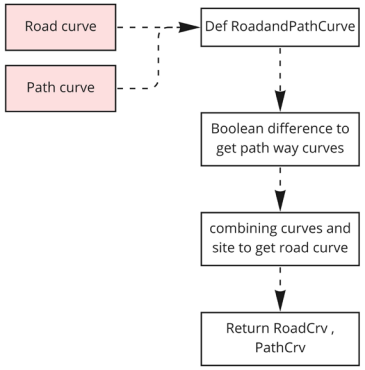


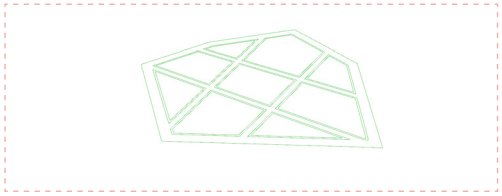
Fillet Toggle Off

Road and path curves

```
'''     Road and Path curves      '''

def RoadandPathCurves(_Road, _Path):

    #Creating only Path curves to further make surface
    PathCrv = []
    for r,p in zip(_Road, _Path):
        PathCrv.append(rs.CurveBooleanDifference(r, p, 0.001))
    
    # Creating only Road curves to further make surface
    RoadCrv = []
    # Appending Site curve to an empty list
    RoadCrv.append(Site_Boundary)
    #Iterating list if curves(path innercrv = parcels boundary) and appending to
    for i in _Road:
        RoadCrv.append(i)

    # Return
    return RoadCrv, PathCrv

# Calling function RoadandPathCurves
RoadCrv = RoadandPathCurves(_Road, _Path)[0]
PathCrv = RoadandPathCurves(_Road, _Path)[1]
```

Script logic:

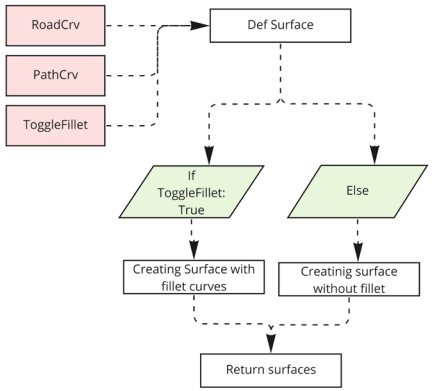




Surface

```
'''     Surface     '''

#Roaf Surface
def Surfaces(RoadCrv, PathCrv, _Path, ToggleFillet):

    #Toggle Fillet On
    if ToggleFillet:
        # Road surface using grasshopper component in python(with Fillet)
        Road_Srf = ghc.BoundarySurfaces(RoadCrv)

    #Toggle Fillet Off
    else:
        # Convertnig Guid to Rhino Geometry
        curvePath = []
        for i in _Road:
            curvePath.append(rs.coercecurve(i))

        # Appending site boundary and set of path curves in an empty list
        NEWlist = []
        NEWlist.append(Site_Boundary)
        for i in curvePath:
            NEWlist.append(i)
        # Road surface using grasshopper component in python(without fillet)
        Road_Srf = ghc.BoundarySurfaces(NEWlist)

    # Path surface from curves  #
    PathSrf = []
    for i in PathCrv:
        PathSrf.append(rs.AddPlanarSrf(i))

    #Parcel Surface

    #Toggle Fillet On
    if ToggleFillet:
        #Re-adding curve to convert into GUID
        p = []
        for i in _Path:
            p.append(sc.doc.Objects.AddCurve(i))
        #Parcel surface from curves(with Fillet)
        ParcelSrf = []
        for i in p:
            ParcelSrf.append(rs.AddPlanarSrf(i))

    #Toggle Fillet Off
    else:
        # Parcel surface from curves(without Fillet)
        ParcelSrf = []
        for i in _Path:
            ParcelSrf.append(rs.AddPlanarSrf(i))

    #Return
    return Road_Srf, PathSrf, ParcelSrf

# Calling individual items using index no. based on return value of the function
Road_Srf = Surfaces(RoadCrv, PathCrv, _Path, ToggleFillet)[0]
PathSrf = Surfaces(RoadCrv, PathCrv, _Path, ToggleFillet)[1]
ParcelSrf = Surfaces(RoadCrv, PathCrv, _Path, ToggleFillet)[2]
```

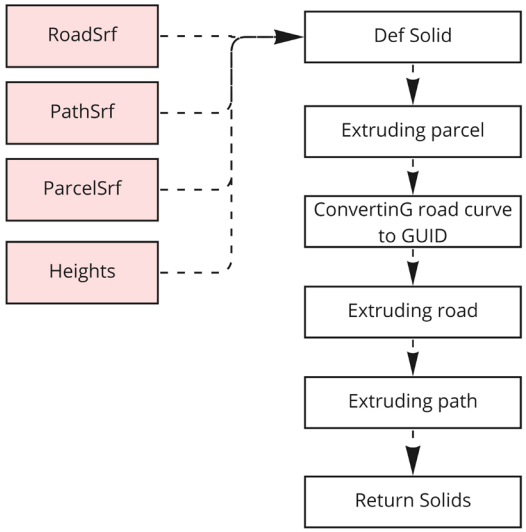
Script logic:



Solids

```
'''     Solids     '''

# Height(Ht) is a parameter for the user.
# Function Solid
def Solid(ParcelSrf, Road_Srf, PathSrf, Parcel_Ht, Road_Ht, Pavement_Ht):

    # Parcel Soild
    ParcelSoild = []
    for i in ParcelSrf:
        ParcelSoild.append(rs.ExtrudeSurface(i, (rs.AddLine((0,0,0), (0,0,Parcel_Ht))), True))

    # Re-adding curve to convert into GUID
    GuidRoad = sc.doc.Objects.AddBrep(Road_Srf)

    # Road Solid
    RoadSolid = rs.ExtrudeSurface(GuidRoad, (rs.AddLine((0,0,0), (0,0,Road_Ht))), True)

    # Path Solid
    PathSoild = []
    for i in PathSrf:
        PathSoild.append(rs.ExtrudeSurface(i, (rs.AddLine((0,0,0), (0,0,Pavement_Ht))), True))

    #Return
    return ParcelSoild, RoadSolid, PathSoild

# Calling individual items using index no. based on return value of the function
ParcelSoild = Solid(ParcelSrf, Road_Srf, PathSrf, Parcel_Ht, Road_Ht, Pavement_Ht)[0]
RoadSolid = Solid(ParcelSrf, Road_Srf, PathSrf, Parcel_Ht, Road_Ht, Pavement_Ht)[1]
PathSolid = Solid(ParcelSrf, Road_Srf, PathSrf, Parcel_Ht, Road_Ht, Pavement_Ht)[2]
```

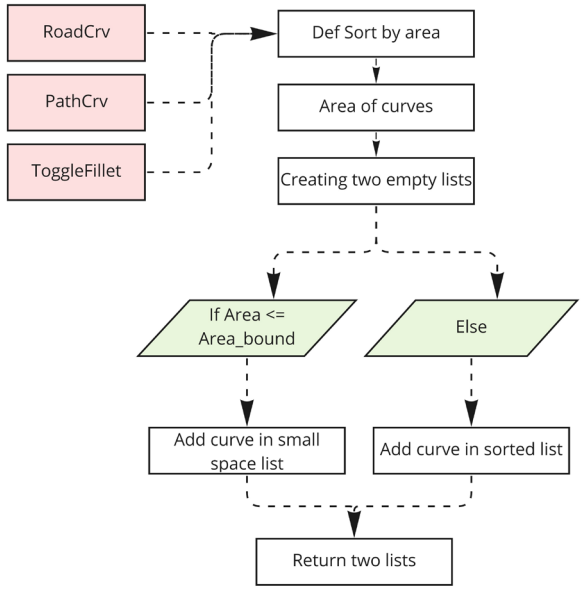
Script logic:

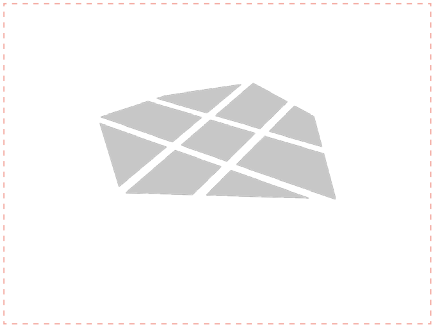


Sorting by area

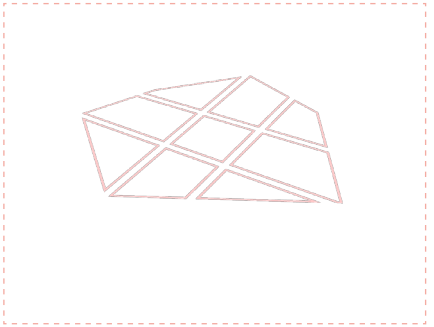
```
'''     Sorting of parcels with area bound     '''

def Sort_by_Area(Parcels, Area_bound):
    #Area of the parcels
    Area = []
    for i in Parcels:
        Area.append(rs.Area(i))
    
    #Empty list to append sorted curves
    Small_space = []
    Sorted_block = []
    #Accessing each curve using for loop
    for i in Parcels:
        # if condition to seperate curves by area
        if rs.Area(i)<= Area_bound:
            Small_space.append(i)
        else:
            Sorted_block.append(i)

    #Return
    return Small_space, Sorted_block

# Calling individual items using index no. based on return value of the function
Small_space = Sort_by_Area(Parcels, Area_bound)[0]
Sorted_block = Sort_by_Area(Parcels, Area_bound)[1]
```

Script logic:

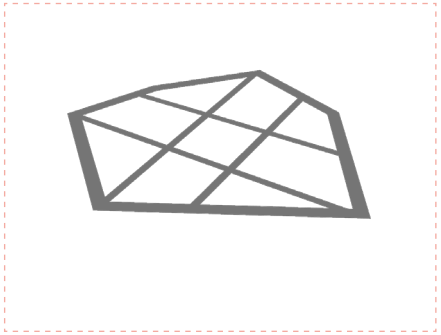




Land parcels



Pathways



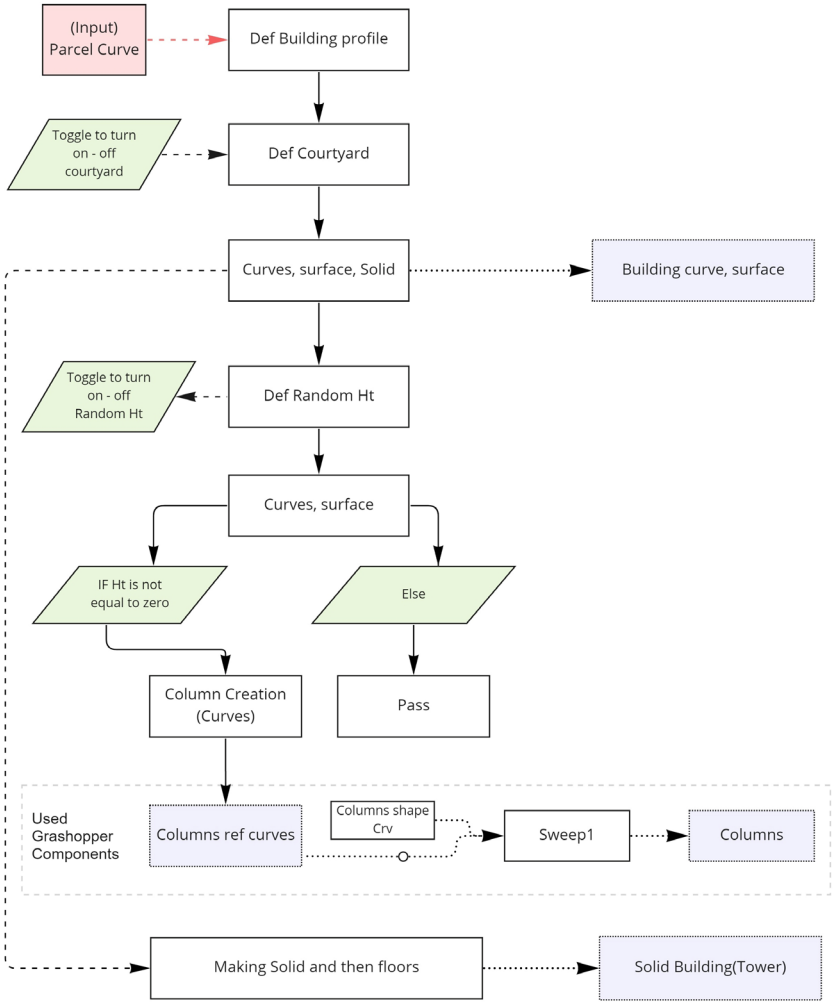
Roads

04.4 | Component 04

Building Formation

This component creates a building with respect to the shape of parcel. The components has ability to make courtyard if area of the building allows randomize the stilt floor height. Addition if building is elevated it can have columns as a structural element.

Script Logic:



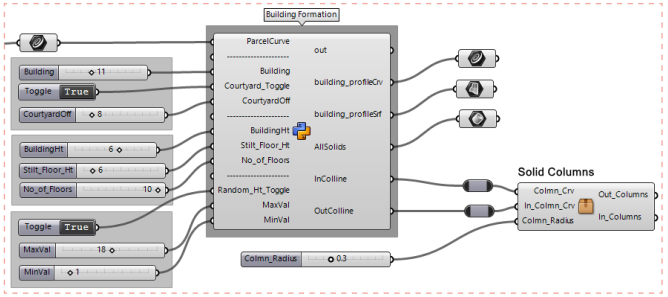


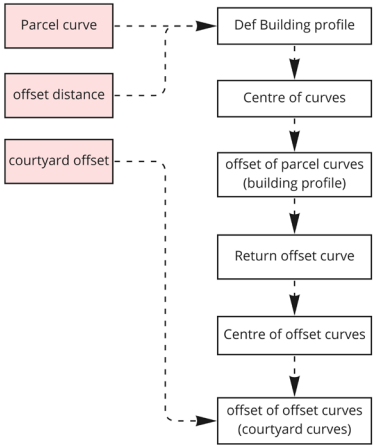
Fig 24. Building formation component

```
'''      Importing Libraries       '''

import rhinoscriptsyntax as rs
import scriptcontext as sc
import ghpythonlib.treehelpers as th
import random as rd

#Converting rhinogeo to GUID
ParcelCrv = []
for i in ParcelCurve:
    ParcelCrv.append(sc.doc.Objects.AddCurve(i))

'''      Building_profile       '''

def Building_profile(ParcelCrv):
    #Centre of the Parcels
    CentreParcel = []
    for i in ParcelCrv:
        CentreParcel.append(rs.CurveAreaCentroid(i))

    #Offset of Parcels to make building outline
    OffsetCrv = []
    for i, j in zip(ParcelCurve,CentreParcel):
        OffsetCrv.append(rs.OffsetCurve(i, j[0], float(Building)))
    return OffsetCrv

# Calling function Courtyard
OffsetCrv = Building_profile(ParcelCrv)

# Centre of the curves
CentreOffsetCrv = []
for i in OffsetCrv:
    CentreOffsetCrv.append(rs.CurveAreaCentroid(i))

# Courtyard inner curve
InnerCrv = []
for i, j in zip(OffsetCrv,CentreOffsetCrv):
    InnerCrv.append(rs.OffsetCurve(i, j[0], float(CourtyardOff)))
```

Input
1. Parcel curve

Output
1. Building curve, surface
2. Solid building
3. column ref. curve

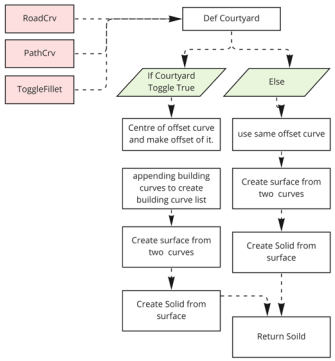
Script logic:



Courtyard

```
'''     Courtyard       '''

def Courtyard(OffsetCrv, BuildingHt, CourtyardOff) :
    if Courtyard_Toggle:
        
        # Centre of the curves
        CentreOffsetCrv = []
        for i in OffsetCrv:
            CentreOffsetCrv.append(rs.CurveAreaCentroid(i))

        # Courtyard inner curve
        InnerCrv = []
        for i, j in zip(OffsetCrv,CentreOffsetCrv):
            InnerCrv.append(rs.OffsetCurve(i, j[0], float(CourtyardOff)))

        # Building profile Curve(1Bldg = 2 curve)
        building_profileCrv = []
        for i, j in zip(OffsetCrv, InnerCrv):
            building_profileCrv.append([i,j])

        # Buildng profile Surface
        building_profileSrf = []
        for i in building_profileCrv:
            building_profileSrf.append(rs.AddPlanarSrf(i))

        # Building profile Solid
        Building_Solid = []
        for i in building_profileSrf:

            Building_Solid.append(rs.ExtrudeSurface(
                i, rs.AddLine((0,0,0), (0,0,BuildingHt))))

    #Toggle Off
    else:
        # Building profile Curve(1Bldg = 1 curve)
        building_profileCrv = OffsetCrv

        # Building profile Surface
        building_profileSrf = []
        for i in OffsetCrv:
            building_profileSrf.append(rs.AddPlanarSrf(i))

        # Building profile Solid
        Building_Solid = []
        for i in building_profileSrf:
            Building_Solid.append(rs.ExtrudeSurface(
                i, rs.AddLine((0,0,0), (0,0,BuildingHt) )))

    return building_profileCrv, building_profileSrf, Building_Solid
```

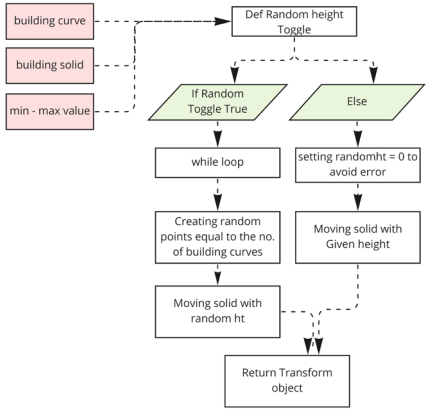
Script logic:



Random height stilt floor

```
'''     Random Stilt_Floor_Ht Toggle     '''

def randomHt(building_profileCrv, Building_Solid, MinVal, MaxVal):

    #Toggle on
    if Random_Ht_Toggle:
        #Creating random no. equal to total no. of parcels
        #and storing in an empty list
        rndmStilt_Floor_Ht = []
        i = 1
        while i <= len(building_profileCrv):
            #min and max values are users parameter for some sort of control
            rndmStilt_Floor_Ht.append(rd.randint(MinVal, MaxVal))
            i += 1

        #Moving the solid building parcels with random Stilt_Floor_Ht
        TransformObj = []
        for i,j in zip(Building_Solid, rndmStilt_Floor_Ht):
            TransformObj.append(rs.MoveObject(i, [0, 0, j]))

    #Toggle Off
    else:
        rndmStilt_Floor_Ht = 0
        # Moving solid building parcels with users input Stilt_Floor_Ht
        TransformObj = []
        for i in Building_Solid:
            TransformObj.append(rs.MoveObject(i, [0, 0, Stilt_Floor_Ht]))

    #Return Object
    return TransformObj, rndmStilt_Floor_Ht

# Calling function randomHt
TransformObj = randomHt(building_profileCrv, Building_Solid, MinVal, MaxVal)[0]
rndmStilt_Floor_Ht = randomHt(building_profileCrv, Building_Solid, MinVal, MaxVal)[1]
```

Script logic:



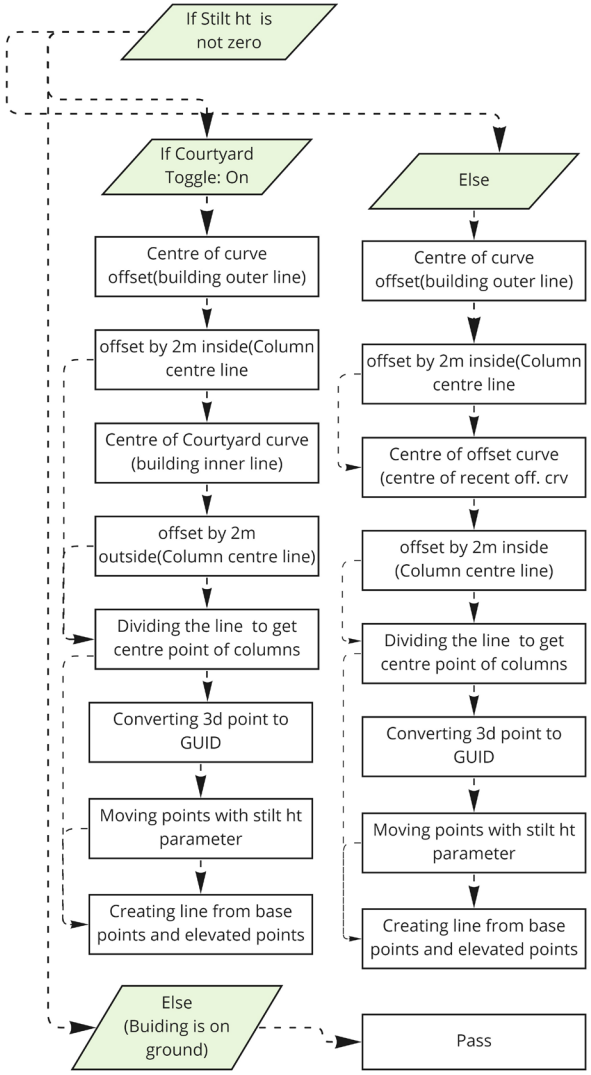
Columns If ...

```
'''     Columns of building is elevated     '''

if Stilt_Floor_Ht != 0:

    if Courtyard_Toggle:

        """     Centre line offset of Outter Column     """
        #Centre of the OffsetCrv
        CentreOffsetCrv = []
        for i in OffsetCrv:
            CentreOffsetCrv.append(rs.CurveAreaCentroid(i))

        #Offset of Parcels( Outer)
        OutColOff = []
        for i, j in zip(OffsetCrv,CentreOffsetCrv):
            OutColOff.append(rs.OffsetCurve(i, j[0], float(2)))

        """     Centre line offset of Inner Column     """
        #Centre of the OffsetCrv
        CentreInnerCrv = []
        for i in InnerCrv:
            CentreInnerCrv.append(rs.CurveAreaCentroid(i))

        #Offset of Parcels (Inner)
        InColOff = []
        for i, j in zip(InnerCrv,CentreInnerCrv):
            InColOff.append(rs.OffsetCurve(i, j[0], float(-2)))

        """     division points     """
        #division of colum centre line
        DivOutPnts = []
        DivInPnts = []
        for i, j in zip(OutColOff, InColOff):
            DivOutPnts.append(rs.DivideCurveLength(i, 5, False, True))
            DivInPnts.append(rs.DivideCurveLength(j, 5, False, True))

        #Convering 3Dpoint to Guid
        BaseOutGuid = []
        BaseInGuid = []
        for Outpnts in DivOutPnts:
            for i in Outpnts:
                BaseOutGuid.append(sc.doc.Objects.AddPoint(i))
        BaseInGuid = []
        for Inpnts in DivInPnts:
            for i in Inpnts:
                BaseInGuid.append(sc.doc.Objects.AddPoint(i))

        #Convering 3Dpoint to Guid
        GuidpointOut = []
        GuidpointIn = []
        for Outpnt in DivOutPnts:
            for i in Outpnt:
                GuidpointOut.append(sc.doc.Objects.AddPoint(i))
        for Inpnt in DivInPnts:
            for i in Inpnt:
                GuidpointIn.append(sc.doc.Objects.AddPoint(i))

        #moving Points
        TopOutPnts = []
        TopInPnts = []
        for i in GuidpointOut:
            TopOutPnts.append(rs.MoveObject(i, [0,0,Stilt_Floor_Ht*2]))
        for i in GuidpointIn:
            TopInPnts.append(rs.MoveObject(i, [0,0,Stilt_Floor_Ht*2]))

        #Line
        OutColline = []
        for i, j in zip(BaseOutGuid, TopOutPnts):
            OutColline.append(rs.AddLine(i, j))
        InColline = []
        for i, j in zip(BaseInGuid, TopInPnts):
            InColline.append(rs.AddLine(i, j))
```

Columns Else ...

```
    #When there is no courtyard and Stilt_Floor_Ht is not equal to zero
    else:
        #Centre of the OffsetCrv
        CentreOffsetCrv = []
        for i in OffsetCrv:
            CentreOffsetCrv.append(rs.CurveAreaCentroid(i))

        #Offset of Parcels( Outer)
        OutColOff = []
        for i, j in zip(OffsetCrv,CentreOffsetCrv):
            OutColOff.append(rs.OffsetCurve(i, j[0], float(2)))

        #Centre of the OutColOff
        CentreOutColOff = []
        for i in OutColOff:
            CentreOutColOff.append(rs.CurveAreaCentroid(i))

        #Offset of OutColOff( Outer)
        InnerColOff = []
        for i, j in zip(OutColOff,CentreOutColOff):
            InnerColOff.append(rs.OffsetCurve(i, j[0], float(6)))

        """     division points     """
        #division of column centre line
        DivOutPnts = []
        DivInPnts = []
        for i, j in zip(OutColOff, InnerColOff):
            DivOutPnts.append(rs.DivideCurveLength(i, 5, False, True))
            DivInPnts.append(rs.DivideCurveLength(j, 5, False, True))

        #Convering 3Dpoint to Guid
        BaseOutGuid = []
        BaseInGuid = []
        for Outpnts in DivOutPnts:
            for i in Outpnts:
                BaseOutGuid.append(sc.doc.Objects.AddPoint(i))
        BaseInGuid = []
        for Inpnts in DivInPnts:
            for i in Inpnts:
                BaseInGuid.append(sc.doc.Objects.AddPoint(i))

        #Convering 3Dpoint to Guid
        GuidpointOut = []
        GuidpointIn = []
        for Outpnt in DivOutPnts:
            for i in Outpnt:
                GuidpointOut.append(sc.doc.Objects.AddPoint(i))
        for Inpnt in DivInPnts:
            for i in Inpnt:
                GuidpointIn.append(sc.doc.Objects.AddPoint(i))

        #moving Points
        TopOutPnts = []
        TopInPnts = []
        for i in GuidpointOut:
            TopOutPnts.append(rs.MoveObject(i, [0,0,Stilt_Floor_Ht*2]))
        for i in GuidpointIn:
            TopInPnts.append(rs.MoveObject(i, [0,0,Stilt_Floor_Ht*2]))

        #Line
        OutColline = []
        for i, j in zip(BaseOutGuid, TopOutPnts):
            OutColline.append(rs.AddLine(i, j))
        InColline = []
        for i, j in zip(BaseInGuid, TopInPnts):
            InColline.append(rs.AddLine(i, j))

        #Flatten to visualise in Rhino-Grashopper
        InColline = th.list_to_tree(InColline)
        InColline.Flatten()

        OutColline = th.list_to_tree(OutColline)
        OutColline.Flatten()

#When Stilt_Floor_Ht is equal to zero
else:
    pass
```

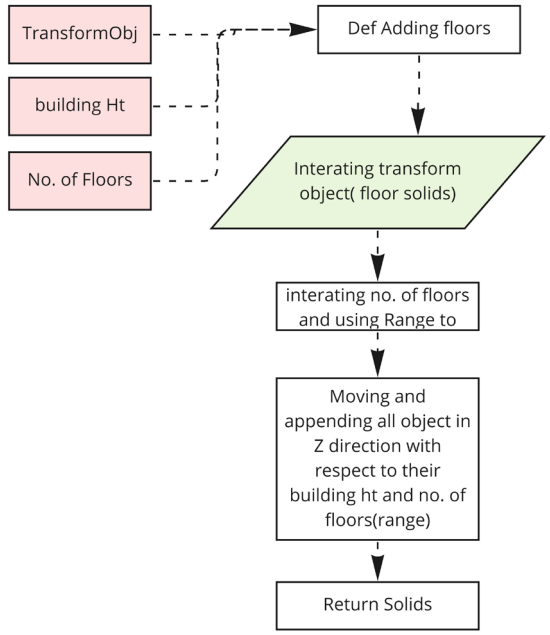
Script logic:

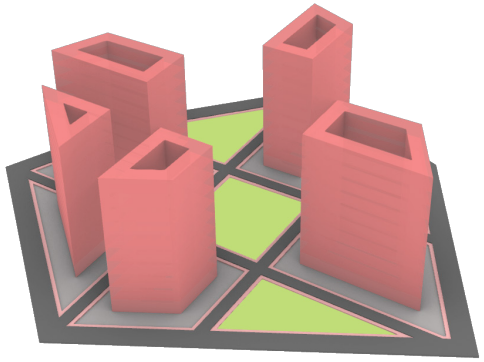


Floors

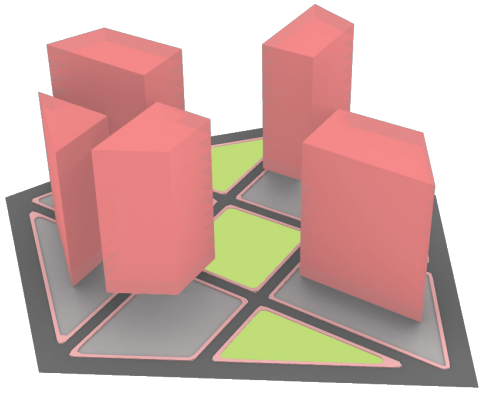
```
    '''     Adding Floors     '''

def AddingFloors(TransformObj, BuildingHt, No_of_Floors):
    #Storing floor in empty list
    AllSolids = []
    # Iterating through building solids
    for i in TransformObj:
        #Adding range to define no. of copies
        for j in range(No_of_Floors):
            # Floor copy
            AllSolids.append(rs.CopyObject(i, (0,0, BuildingHt*j)))
    #Return
    return AllSolids

#Calling function AddingFloors
AllSolids = AddingFloors(TransformObj, BuildingHt, No_of_Floors)


'''     Outputs     '''

#Flatten building_profile
building_profileCrv = th.list_to_tree(building_profileCrv)
building_profileCrv.Flatten()

#Flatten building_Surface
building_profileSrf = th.list_to_tree(building_profileSrf)
building_profileSrf.Flatten()

#Flatten Building_Solid
AllSolids = th.list_to_tree(AllSolids)
AllSolids.Flatten()
```

Script logic:

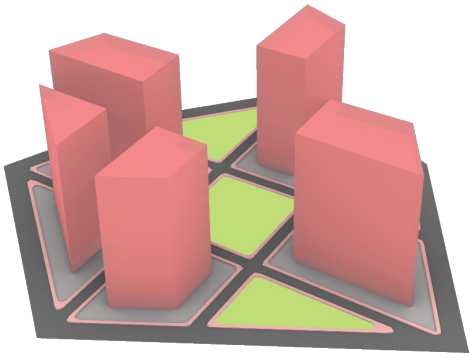




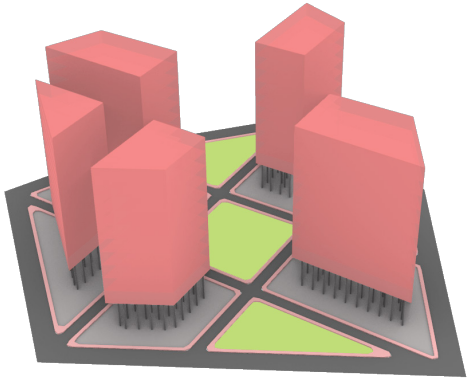
Courtyard Building



Randomly elevated solid building



Solid Building



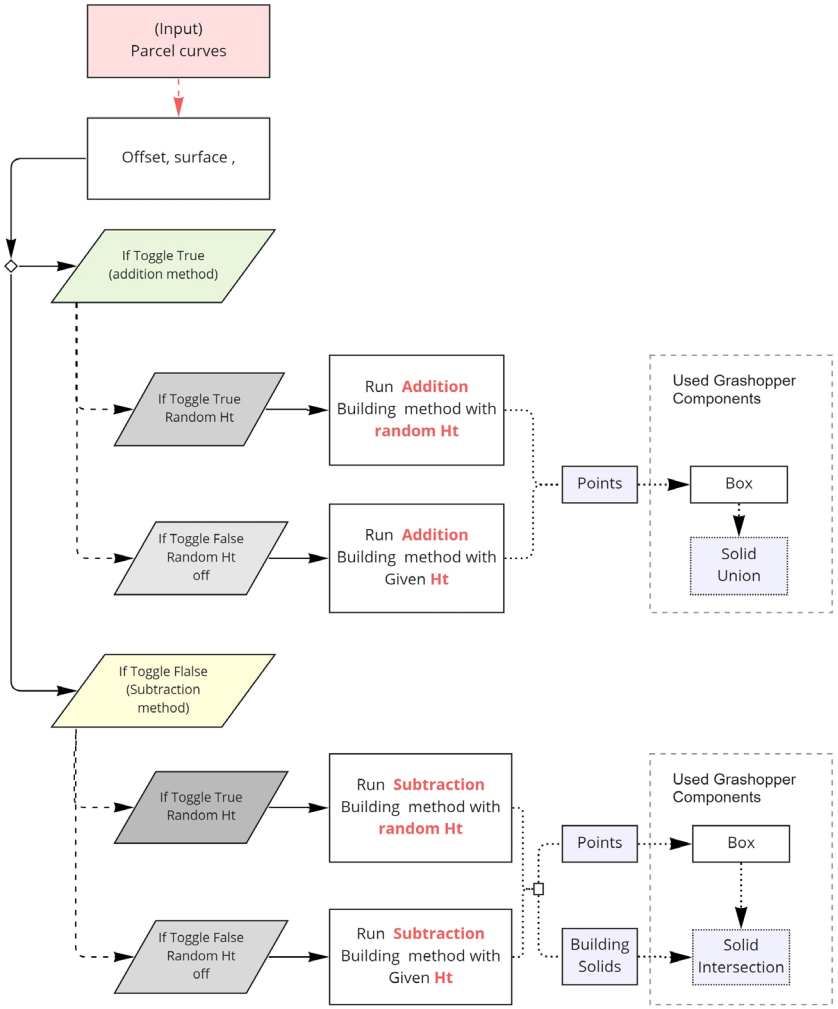
Solid building with columns

04.5 | Component 05

Abstract Building(Additive and Subtractive method)

This component creates an abstract building with two basic methods : additive and subtractive. Two different components are created to as the function are opposite to each other.

Script Logic



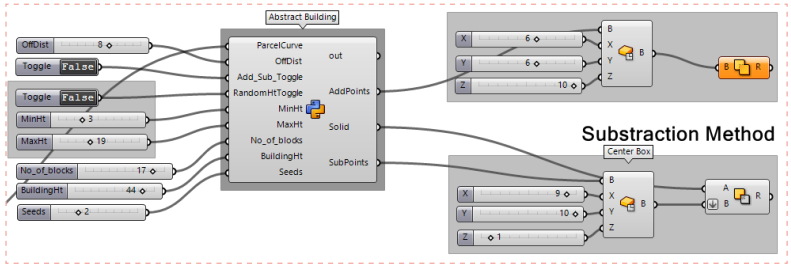
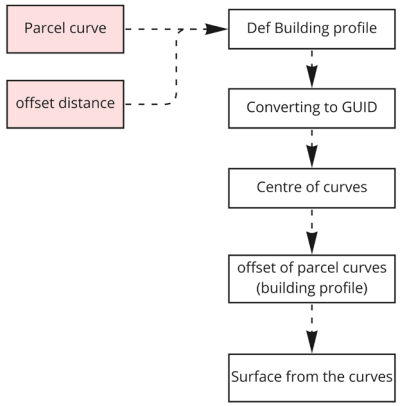


Fig 24. Abstract Building component

Input
1. Parcel curve
Output
1. Points for additive building method
2. Points for subtractive building method

```
__author__ = "DARSHAN.H.CHAVAN.H"
__version__ = "2022.05.21"

'''     Importing Libraries       '''

import rhinoscriptsyntax as rs
import scriptcontext as sc
import ghpythonlib.treehelpers as th
import random as rd
import ghpythonlib.components as ghc #used four times for same command

'''         Parcel Offset, surface and solid        '''

#Convertnig rhino geometry to GUID
GuidCrv = []
for i in ParcelCurve:
    GuidCrv.append(sc.doc.Objects.AddCurve(i))

#Centre of the curve
CentreCrv = []
for i in GuidCrv:
    CentreCrv.append(rs.CurveAreaCentroid(i))

#Converting to guid point
Guidpt = []
for i in ParcelCurve:
    Guidpt.append(sc.doc.Objects.AddCurve(i))

#Offset Curve
OffsetCrv = []
for i, j in zip(GuidCrv, CentreCrv):
    OffsetCrv.append(rs.OffsetCurve( i, j[0], OffDist))

#Surface from curve
ParcelSrf = []
for i in OffsetCrv:
    ParcelSrf.append(rs.AddPlanarSrf(i))
```

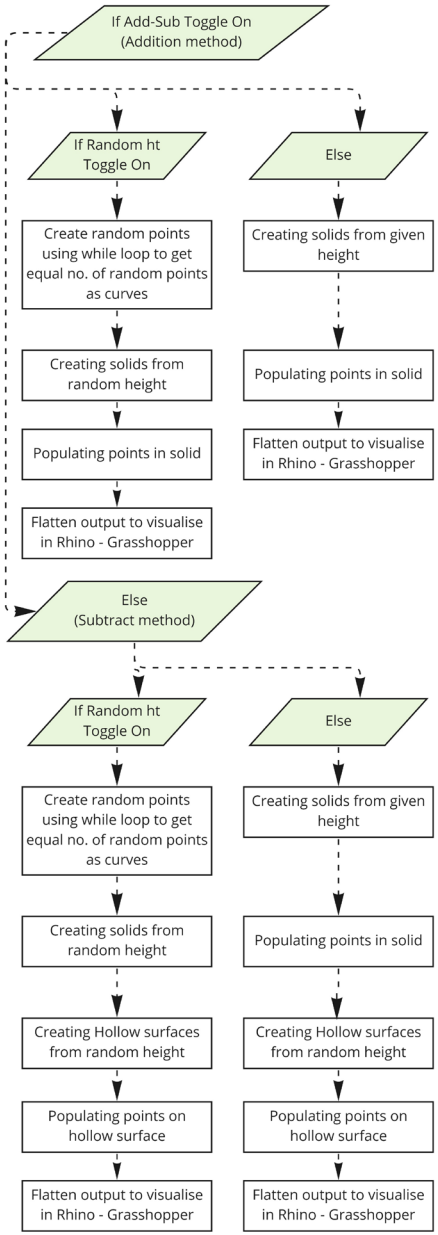
Script logic:



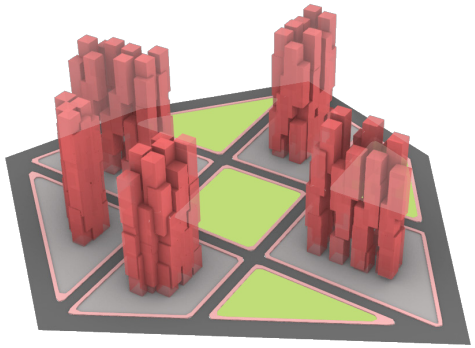
Building form method and heights

```
'''         Building forming method and height        '''
"""
There are two building forming methods: additive and substractive. First IF
condiion is used to turn On - Off Add-substract method and second If condition
is used to turn On - Off random height of the building.
"""

# Toggle True(Addition method)
if Add_Sub_Toggle:
    # Toggle True(Random ht)
    if RandomHtToggle:

        #Creating random numbers equal to total no. of parcels
        #storing in an empty list
        RandomHt = []
        i = 1
        while i <= len(ParcelCurve):
            #min and max values are users parameter for some sort of control
            RandomHt.append(rd.randint(MinHt, MaxHt))
            i += 1
        #Extruding parcels with random Stilt_Floor_Ht
        solid = []
        for i, j in zip(ParcelSrf, RandomHt):
            solid.append(rs.ExtrudeSurface(
                Si, rs.AddLine((0,0,0), (0,0,j)), True))

        #Populating points inside soild (for additive method)
        AddPopuPnts = []
        for i in solid:
            AddPopuPnts.append(ghc.PopulateGeometry(
                [rs.coercebrep(i)], [No_of_blocks], [Seeds]))

        #Flattening to visualise in Rhinoceros-Grasshopper
        AddPoints = th.list_to_tree(AddPopuPnts)
        AddPoints.Flatten()

        Solid = th.list_to_tree(solid)
        Solid.Flatten()
    # Toggle False(Random ht Off)
    else:

        #Extruding surface with given height
        solid = []
        for i in ParcelSrf:
            solid.append(rs.ExtrudeSurface(
                i, rs.AddLine((0,0,0), (0,0,BuildingHt)), True))

        #Populating points inside soild (for additive method)
        AddPopuPnts = []
        for i in solid:
            AddPopuPnts.append(ghc.PopulateGeometry(
                [rs.coercebrep(i)], [No_of_blocks], [Seeds]))

        #Flattening to visualise in Rhinoceros-Grasshopper
        AddPoints = th.list_to_tree(AddPopuPnts)
        AddPoints.Flatten()

        Solid = th.list_to_tree(solid)
        Solid.Flatten()
```

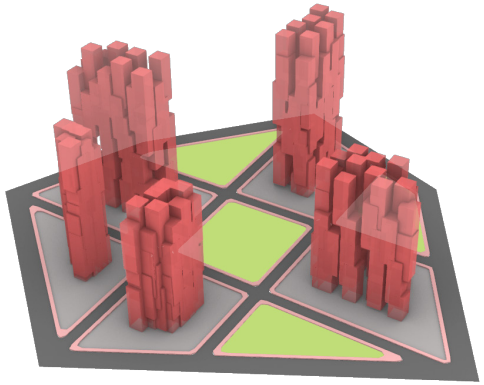
Script logic:



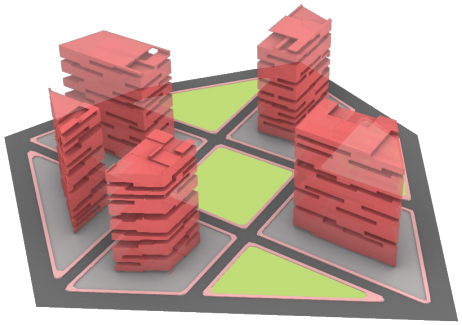

```
# Toggle Flase(Substraction method)
else:
    # Toggle True(Random ht)
    if RandomHtToggle:

        #Creating random numbers equal to total no. of parcels
        #storing in an empty list
        RandomHt = []
        i = 1
        while i <= len(ParcelCurve):
            #min and max values are users parameter for some sort of control
            RandomHt.append(rd.randint(MinHt, MaxHt))
            i += 1

        #Extruding parcels with random Stilt_Floor_Ht
        solid = []
        for i, j in zip(ParcelSrf, RandomHt):
            solid.append(rs.ExtrudeSurface(
                i, rs.AddLine((0,0,0), (0,0,j)), True))

        hollowSrf = []
        for i, j in zip(ParcelSrf, RandomHt):
            hollowSrf.append(rs.ExtrudeSurface(
                i, rs.AddLine((0,0,0), (0,0,j)), False))

        #Populating points on surface of the soild (for subtractive method)
        SubPopuPnts = []
        for i in hollowSrf:
            SubPopuPnts.append(ghc.PopulateGeometry(
                [rs.coercebrep(i)], [No_of_blocks], [Seeds]))

        #Flattening to visualise in Rhinoceros-Grasshopper
        SubPoints = th.list_to_tree(SubPopuPnts)
        SubPoints.Flatten()

        Solid = th.list_to_tree(solid)
        Solid.Flatten()

    # Toggle False(Random ht Off)
    else:
        #Extruding surface with given height
        solid = []
        for i in ParcelSrf:
            solid.append(rs.ExtrudeSurface(
                i, rs.AddLine((0,0,0), (0,0,BuildingHt)), True))

        hollowSrf = []
        for i in ParcelSrf:
            hollowSrf.append(rs.ExtrudeSurface(
                i, rs.AddLine((0,0,0), (0,0,BuildingHt)), False))

        #Populating points on surface of the soild (for subtractive method)
        SubPopuPnts = []
        for i in hollowSrf:
            SubPopuPnts.append(ghc.PopulateGeometry(
                [rs.coercebrep(i)], [No_of_blocks], [Seeds]))

        #Flattening to visualise in Rhinoceros-Grasshopper
        SubPoints = th.list_to_tree(SubPopuPnts)
        SubPoints.Flatten()

        Solid = th.list_to_tree(solid)
        Solid.Flatten()
```



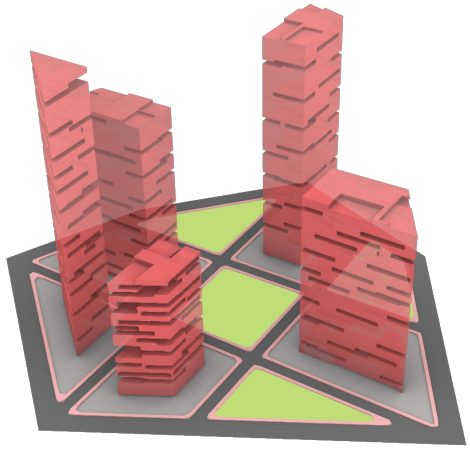
Additive Building with random height



Additive Building with given height



Subtractive Building with random height



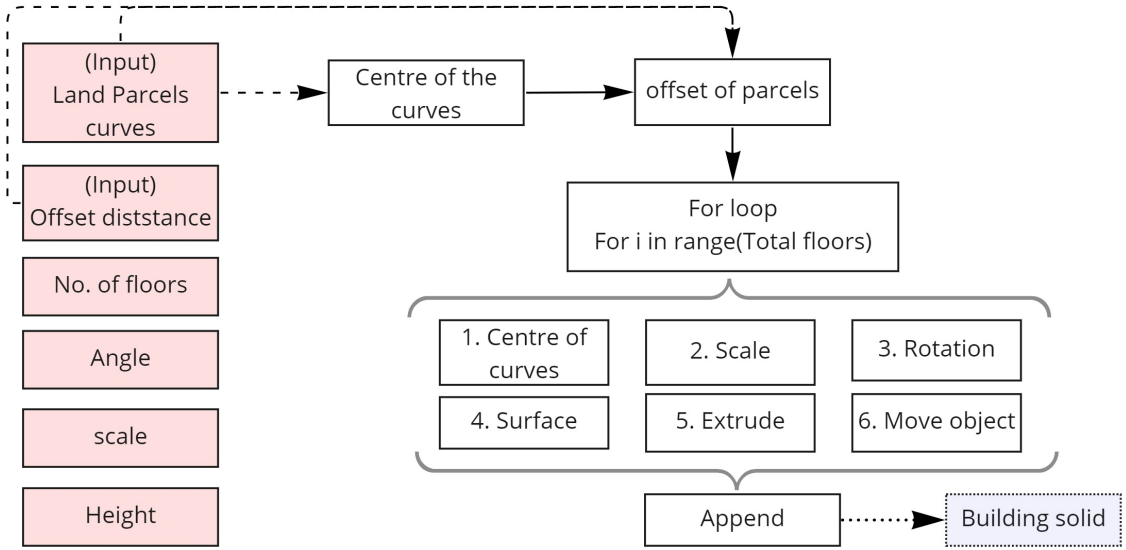
Subtractive Building with given height

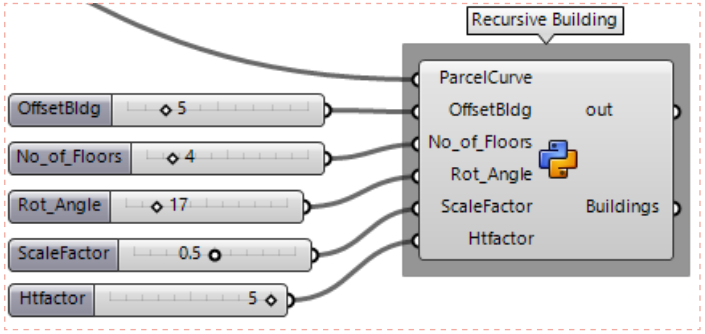
04.6 | Component 06

Recursive Building

This component creates a recursive tower by the parcel curve. The tower can bi manipulated by changing its height, size, rotation and no, of floors.

Script Logic





Input
1. Parcel curve
Output
1. Soild building

Fig 24. Recursive building component

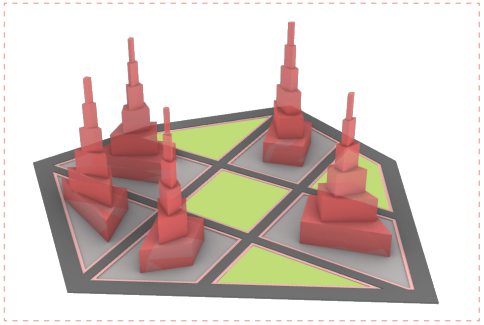


Fig 24. Result of the component

Grasshopper Python Script Editor

File Edit Tools Mode Help Test OK

```
__author__ = "DARSHAN.H.CHAVAN.H"
__version__ = "2022.05.21"

import rhinoscriptsyntax as rs

#Centre of the curves
centre = rs.CurveAreaCentroid(ParcelCurve)
#Offset curve
offsetcurve = rs.OffsetCurve(ParcelCurve, centre[0], OffsetBldg)

#Empty list to append tranform object
Buildings = []
for i in range(No_of_Floors):
    #centre
    c = rs.CurveAreaCentroid(offsetcurve)
    #Scale
    a = rs.ScaleObject(offsetcurve, c[0],[ScaleFactor, ScaleFactor, 0])
    #Rotation
    a = rs.RotateObject(a, c[0], Rot_Angle)
    #Surface
    a = rs.AddPlanarSrf(a)
    #Extrude Surface
    a = rs.ExtrudeSurface(a, rs.AddLine((0,0,0),(0,0,3*Htfactor)), True)
    #Moving object and appending to empty list
    Buildings.append(rs.MoveObject(a, (0,0,3*i*Htfactor)))
```

Fig 24. Recursive tower script

05 | Limitation and failure

Most of the goals are achieved through this components which were set but there are many ways to make components more powerful and flexible.
As a beginner in coding, it was limiting me to code what I was imagining of. Some components ideas are not being completed due to lack of time and knowlege.
The libraries/ Api's are vast and limited in some cases which affect the component growth aswell.

As there are multiple data types in grashopper python, some time it becomes hard to manipulate the data to achieve certain result. For instance, in Abstract Building component, it became hard to make box from points as the box command needs a plane to create a box and I did not get any conversion reference from point to plane in grasshopper python.
Due to lack on time and my understanding it was not managable to creates columns in randon floor height. Although the logic is clear(height of the column will be equal to random height) but lack of grashopper python knowledge makes it time consuming, resulting in not completing it.

To ingore the problem of repetition of running scripts, list access was used and to access that list every time for loop was used which makes scripts lengthy.

Machine learning can be incorporated to design a floor plan of the buildings where python has some limitation.

References:

https://www.grasshopper3d.com/forum/topics/splitting-surfaces-in-grasshopper-python

https://www.grasshopper3d.com/profiles/blogs/trim-line-python

https://discourse.mcneel.com/t/ghpython-convert-polycurve-to-guid/94581

https://www.linkedin.com/learning/grasshopper-and-rhino-python-scripting/welcome?autoplay=true&u=35392996

https://developer.rhino3d.com/api/RhinoScriptSyntax/

https://developer.rhino3d.com/api/RhinoCommon/html/R_Project_RhinoCommon.htm

and many rhino grasshopper forums sites to check my issues.