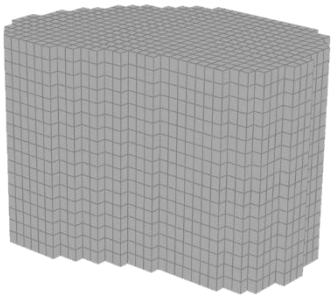
Points of Interest

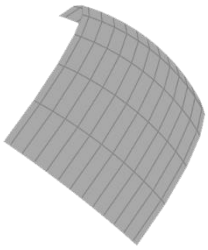
Given Mesh Model of Surroundings
Output Evenly Spread Amount of Points of Interests (on the Facades facing the Site)
❖ **Do** - Import site geometry (.obj file) - Reposition geometry to global origin - Fuse geometry/meshes - Clip any geometry not needed (existing buildings on site) - Create mesh of the building site perimeter - **For each** (Mesh Face) + Check **if** normal vector is pointing towards site perimeter (range of dot-product) +**if** it does, keep the face +**else** delete the face - "Remesh" remaining faces for similar face sizes - **For each** (Remeshed Face) + Create point in its centre + Delete the face

Point/Box Cloud



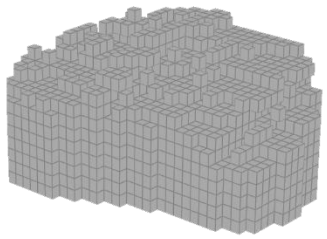
Given Dimension/Outline of the Site, Voxel Dimension
Output Point Cloud and Box Model of the Volume
❖ **Do** - ""Null Note" Defining the box size - Grid with rows/columns dependent on box dimensions - Copy up x-number of times according to building height / voxel height - Align with site boundaries (Transform) - **For each** face in the grid + Create point in its centre + Delete the Face ✓ Will leave you with the **point cloud** - Either use a box volume or extruded building outline as shape volume - If necessary align in the site position - Group points that are within the bounding volume & delete the unused - Create a box with the voxel dimensions and aligned rotation - Copy box onto points ✓ Will leave you with the **box cloud**

Simplified Sunray Model



Given Longitude of the Building Site
Output Simplified Model of the Sunrays
❖ **Do** - Create sphere & reverse mesh faces - **Clip** the sphere (vertical planes 37.5 deg, horizontal xz-plane moved up/down by radius * sin(23.5) - Rotate around X-Axis by 52 degrees (~altitude of Rotterdam) - Reverse mesh faces back (to save troubles during the solar envelope calculation)

Solar Envelope



Given Points of Interests, Sunrays Model, Voxel Point Cloud and Voxel Boxes

Output Voxel Points not blocking the daylight of Neighbouring Buildings

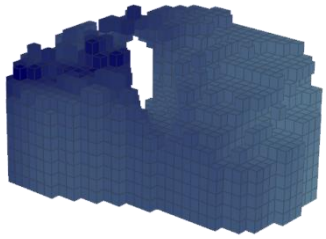
❖ **Do**

- Initialize “Hit_Count” Attribute on the all box faces
- **For each** Point of Interest (Inside of Attribute Wrangle)
  + **For each** sun ray
    # Create sun ray vector
    # I**f** the sun ray is not intersecting with the site
    - Intersect all box faces with the sun ray
    - **for each** intersected face
    +Add 1 to the “”Hit_Count” attribute

✓ Boxes with “Hit_Count” attribute on its faces

- Transfer the attribute from the 6 faces to corresponding box centroid (Point_ID)
- Delete every point/centroid with “Hit_Count” > 0
- Copy a box onto remaining points and fuse them together
- **For each** connected piece (same class)
  +**If** number of connected pieces < 300 delete the boxes
- Import new point cloud and only keep points within the remaining volume

Voxel Attributes

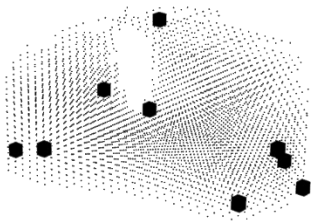


Given Usable Points

Output Points with various site specific attributes

- ❖ **Do**
  - **For each** point in the point cloud
    + Floor Level Attribute = floor(@P.y / 3) (3 signifying intended ceiling height)
- ✓ Floor Attribute
  - Fuse Boxes & delete the interior faces (left with exterior shell)
  - Delete ground faces based on normal vector
  - **For each** face (Sunrate Calculation)
    +**For each** sun ray
    # Create sun ray vector
    # I**f** it does not intersect the site geometry or its own faces
    - Add 1 to the sunrate attribute
  - **For each** point (transfer sunrate onto point cloud)
    +**For each face**
    #Store centroid & sunrate from faces
    #**If** the current point is on the same height as face centroid
    -Calculate distance between centroid and current point
    -Add face_sunrate * $(1/1.1^{distance})$ to the point sunrate
- ✓ Daylight Attribute
  - **For each** face (Daylight Envelope)
    + Set View attribute to 1 for associated box centroids (Point ID)
    +Based on direction of face normal vector store view direction
- ✓ View and Orientation Attribute
  - Define Points of Interest around the perimeter of the site
  - **For each** point in the point cloud
    +Calculate the shortest distance to each of those points
    +Set the Connectivity attribute to the smallest one of those distances
- ✓ Connectivity Attribute
  - Import the centre lines of the roads around the site
  - **For each** point in the point cloud
    +Calculate the shortest distance to the road
    +Set the Noise Attribute to **80 - (20 * log(dist))** based on the loss of noise intensity over distance
- ✓ Noise Attribute
  - Use the minimum and maximum value of each attribute to map the values from 0-1
  - (This will achieve a better result when calculating the weighted product)

Function Starting Points



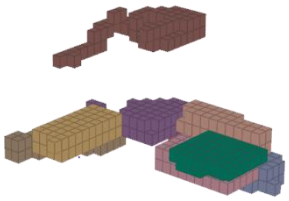
Given Point Cloud with various Attributes

Output Most suitable point for each function

- ❖ **Do**
  - **For each** point in point cloud
    - + Read and store the relevant point attributes on it as variables
    - + **For each** function
      - #**If** the point is within specified floor range -
        - -Calculate the performance for the function (weighted product)
        - - **Store** the performance as point attribute
- ✓ Function Performance Attributes
  - **For each** point in point cloud
    - + Read and store the agent attributes from its node
    - + Read and store the relevant point attributes on it as variables
    - + **For each** Function Agent
      - #**If** the point is within specified floor range; is free and is within the function agent boundaries for the attributes
        - - Look at the neighbouring points and calculate the average performance of its neighbourhood (adding up the performance attribute/neighbours)
        - - **If** the average performance is better than stored performance **AND** no other function is close to it
          - +overwrite previous point_id and performance
    - + **If** id <5000000 (Check for if it does not find a point)
      - #Set func_id and group of that voxel to corresponding function

Weighted Product = $P(A_K/A_L) = \prod_{j=1}^{n} (a_{Kj}/a_{Lj})^{w_j}, \text{ for } K, L = 1, 2, 3, \ldots, m.$

Agent based growing



Given Starting Points (Optimal Position)

Output Functions in adequate positions within building mapped onto the points

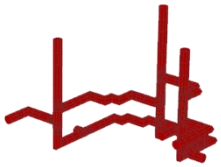
- **Do for each agent/function**
  - Every fifty iterations try growing again
  - Read and store the agent attributes from its node and the growth variable
  - **If** the function still needs to grow; **For each** point in point cloud
    - + Calculate the centroid (loc) of function ($\sum_{k=0}^{n} P.xyz$)/n
    - + **If** the points x,y or z-distance is to far from the function centre
      #remove the point from the function
    - + **Else; For each** of its neighbours :
      # Read and store the relevant point attributes on it as variables
      # **If** the point is within specified floor range; is free; has enough connectivity and day light as well as acceptable noise level
      - Calculate the performance (weighted product)
      - **If** performance > stored performance **AND** it has not been taken by this agent before **AND** the function still needs more points
      +overwrite previous point_id & performance

    - + **If** point amount is reached **AND** still wants to grow
      # Check the point performance furthest away from the centroid compared to the ones it could grow to
      # **If** there is a better performing point
      - Overwrite the previous point with the new one
      # **Else**
      - Stop the growth for this agent

    - + **If** id <5000000 (Check for if it does not find a point)
      #Set func_id and group of that point to corresponding function
      #Add 1 to its occupation attribute

Weighted Product = $P(A_K/A_L) = \prod_{j=1}^{n} (a_{Kj}/a_{Lj})^{w_j}, \text{ for } K, L = 1, 2, 3, \ldots, m.$

Path Finding



Given Point Cloud after the Agent Based Growing

Output A Circulation Pathway connecting the Entrance to all Functions

❖ **Do**

- **For each** function needing direct access to the ground floor
  + Set func_id and group of every point straight above or below the centre to "Circulation"

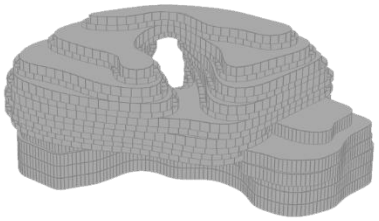
✓ First Circulation Shafts

- **For each** Function
  + **If** the Circulation Shafts goes through them
    # Store the point as Function Access Point (Group)
    # Set Function Stair Attribute to 1 (no need for separate staircase)
- **For each** Function
  + **If** the Circulation Shafts is within close proximity to it on the same floor
    # Store the point as Function Access Point (Group)
    # Set Function Stair Attribute to 1 (no need for separate staircase)
- **For each** Function
  + **If** the Function Stair Attribute is 0 (no access to staircases)
    # Set func_id and group of every point straight above or below the centroid to "Circulation"

✓ Function Access Points / Circulation Shafts (All Functions are connected to the Ground)

- Connect all points through polylines
- **Run** "Find Shortest Path" SOP between Function Centres and their Access point
- **Run** "Find Shortest Path" SOP between the Entrance and all Circulation points on 0F
- Set the func_id and group of every point on the circulation pathway to "Circulation"

Envelope Smoothening



Given Box Cloud after Growing / Pathfinding

Output A smoothened envelope with balconies for Living Unit

❖ **Do**

- **Split** Living Units Boxes from the other Functions

- **Living Unit:**
  + Convert boxes into a Isosurface and afterwards into a **Mesh**
  + **Remesh** it to further smoothen the shape
  + **For each** Floor Level
    # **Clip** and **Divide** the Mesh obtaining the cut floor shape
    # Delete Faces with less than 100m^2
    # **Sort** the line segments of the outline
    # **Resampl**e and **Smooth** the outline getting more regular line distance
    # **Polyextrude** the face by the ceiling height

✓ Individual smoothened floor envelops
    # **Duplicate** the envelope
    # **Clip** it at half its height and blast its bottom face
    # **Polyextrude** these faces twice separately (one keeping front faces or the side faces)
    # For the created side faces only keep the bottom face (Balcony Slap)
    # **Merge** both extrusions with the envelope

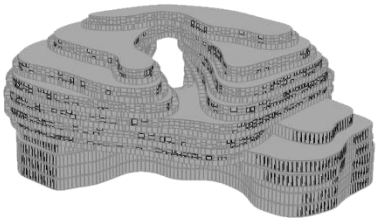
✓ Individual smoothened floor envelops and balconies

- **Other Functions:**
  + Convert Voxels into a Isosurface and afterwards into a **Mesh**
  + **Remesh** it to further smoothen the shape
  + **For each** Floor Level
    # **Clip** and **Divide** the Mesh obtaining the cut shape
    # Delete Faces with less than 100m^2
    # **Sort** the line segments of the outline
    # **Resampl**e and **Smooth** the outline getting more regular line distance
    # **Polyextrude** the face by the ceiling height

✓ Individual smoothened floor envelops
  + Split the bottom three floors from the rest
    # Only keep the face with the highest area **(measure, sort, group, blast)**
    # Move it to the ground (y=0)
    # **Polyextrude** the face by the ceiling height
    # Copy the floor twice upwards

✓ Stable and more uniform base

Façade Tiling



Given Building Envelope Mesh after Smoothening (still separated)

Output Façade Meshes split into Tiles and Windows

❖ **Do**

- **Split** top and bottom meshes from the sides
- **Top and Bottom:**
  + **PolyExtrude** them by a regular floor thickness

✓ Floor Slaps

- **Sides:**
  + **For each (Iteration)**
    # **For each** face
    - randomly set a cutting plan (clipped to 1/4, 1/2 ,3/4 of the edges)
    - randomly select either a horizontal or vertical cut
    - **If** one edge is too short select the other one **OR if** both are too small; Move cutting plane outside of the face

    - Based on the UV-Coordinates at the middle of the edges calculate two vectors
    - Calculate the cross product between the face normal and those vectors (Used as the normal of the cutting planes)
    - **Clip** the face along the chosen calculated plane

✓ Tiled Facades

- **Sides:**
  + **For each** Face
    # Calculate its performance based on how much sunlight it is getting, preferred cardinal direction and its area
    # **Sort** the faces by their performance
    # Add to each floors window group until specified window area has been reached
    # **Split** façade panels from window panels

    # Façade:
    - **PolyExtrude** each façade face

    # Window:
    - **Split** into two (Frame and Pane)
    - **PolyExtrude** both twice
    - Merge both parts together with the façade faces

✓ Tiled Facades with Windows (and window frames)