Documentation
Oasis Generative Design Project

Jonas Althuis
Adrian Wong
Hannah van der Sluijs
Benjamin Laagewaard

This document contains the pseudo-code to all of our systems in Houdini, it is structured in the same way our Houdini file is structured, which also represents the way data flows linearly through our process. The actual code, the majority of which is vex code in attribute wranglers in Houdini, can be found in the appendices of this document, which follow the same structure as the pseudo-code. Please note, the pseudo-code we've written is largely representative of our processes, but not every last detail is included.

Table of Contents

1. Graph Relaxation

> Spatial Configuration

Location	File > Graph Relaxation > Spatial Configuration
Purpose	Simulating relaxed graph state of building connectivity graph
Inputs	spatial_configuration.csv (function information and connection relationships), function_relation_strength.csv (function connection strength matrix)
Outputs	Relaxed state graph of functions based on function area and connection weights.

- Place functions as points on a circle.
- **FOR** every point:
    - Create a polyline primitive from current point to points of functions it should be connected with.
    - Set 'pscale' attribute based on function area
    - Place spheres on points using *copy to points,* these are are automatically scaled by pscale value
- **FOR** each primitive (connections between points):
    - Create connection strength attribute based on connection strength matrix.
    - Calculate rest length value by adding the radius of the spheres at each end of primitive, write rest length as attribute on primitive.
- Simulate connection graph using a houdini rigid body solver.

See code snippet: 1.1.1, 1.1.2, 1.1.3, 1.1.4, 1.1.5

2. Starting Geometry

> Building Voxelization

Location	File > Starting Geometry > building_voxelization
Purpose	Voxelization of the building volume.
Inputs	polyline of plot, point grid with voxel size as spacing
Outputs	Voxel points of building volume
● Copy flat **point grid** up to desired max height for building, with voxel_size as vertical translation ● Extrude **polyline of plot** up to desired max height for building volume ● Group grid points by bounding box of polyline extrusion, excluding points not in bounding box ● Blast points not in grid group	
No code snippet for this process as it’s almost exclusively done with houdini nodes.	

3. Envelope Shaping

> Entrance squares

Location	File > Envelope Shaping > entrance_squares
Purpose	Place squares on the plot as space for vegetation
Inputs	Voxel points, points of interest, solar path and context
Outputs	New shape of building with entrance squares

- Set the points of interest in the context.
- Find the nearest points of the point grid (on ground level) of the building to the points of interest.
- Polyextrude the round squares placed on the nearest points to the top of the building.
- Remove the points of the building within that group, only keeping the bottom layer (as input for the shadow analysis)
- Do the sun analysis and convert the outcome into percentages.
- Multiply the percentages with the divided area of the 30% vegetation (divided by the amount of squares(4)), to give the squares with the best/worst sun value the possibility to grow relatively bigger/smaller.
- Redo the first few steps: polyextrude the squares and delete the points of the building in the group.

See code snippet: 3.1.1 and 3.1.2

> Shadow casting

Location	File > Envelope Shaping > shadow_casting
Purpose	Calculate intersections between voxel points and geometry of surroundings on the vector of every sun point to every voxel points, representing the amount of shadow every point casts on the surrounding geometry.
Inputs	Grid of voxel points, sun points, geometry of surrounding buildings, threshold value for point removal
Outputs	Ratio value as attribute on every voxel point, solar envelope with points removed based on threshold

```
● FOR every voxel point:
    ○ FOR every sun point:
        ■ Get the vector direction from every voxel point to the sun point
        ■ Check for intersect between the voxel point and the sun point
        ■ IF the voxel point can see the sun point (i.e. there is no buildings
          blocking the sun from reaching the voxel point):
            ● Reverse the vector and check for intersections between the
              voxel point and the surrounding geometry (i.e. where the voxel
              would cast a shadow over geometry)
            ● Add 1 to the count if an intersection is found
    ○ Ratio = count / total number of sun points
    ○ Fit ratio into range from 0 to 1
    ○ IF ratio is greater than threshold:
        ■ Remove voxel point
```

See code snippet: 3.2.1

4. Data Analysis

> Acoustic Analysis

Location	File > Data Analysis > acoustic_analysis
Purpose	Envelope building volume based on acoustic pollution & calculate acoustic data for all voxel points
Inputs	Grid of voxel points, acoustic data .psd files, threshold value for point removal
Outputs	Ratio value as data attribute on every voxel point

```
● FOR each .psd file (split up by color, each representing acoustic level):
    ○ Trace images to polylines
    ○ Remove small areas, clean polylines
    ○ Populate polyline area with points
    ○ Set color & acoustic value as attributes on points in area
● FOR each voxel point:
    ○ Find all the acoustic points in a maximum search radius (50 meters)
    ○ FOR each acoustic point:
        ■ Read the acoustic value from each acoustic point
        ■ Calculate distance between voxel point and acoustic point
        ■ Recalculate acoustic value based on initial acoustic value and distance
          (using acoustic formula)
    ○ Sum acoustic values (using acoustic formula)
    ○ Write acoustic value as data in attribute
● Fit ratio into inverse range from 0 to 1, such that the maximum acoustic value is 0 and
  the minimum acoustic value is 1.
● IF ratio is greater than threshold:
    ○ Remove voxel point.
```

See code snippet: 4.1.1

> Sunlight Analysis

Location	File > Data Analysis > sunlight_analysis
Purpose	Calculate sunlight data for all voxel points
Inputs	Grid of voxel points, sun points, geometry of surrounding buildings, threshold value for point removal
Outputs	Ratio value as data attribute on every voxel point

```
● FOR every voxel point:
    ○ FOR every sun point:
        ■ Get the vector direction from every voxel point to the sun point
        ■ Check for intersect between the voxel point and the sun point
        ■ IF the voxel point can see the sun point (i.e. there is no buildings
          blocking the sun from reaching the voxel point):
            ● Add 1 to the count if an intersection is found
    ○ Ratio = count / total number of sun points
    ○ Fit ratio into range from 0 to 1
```

See code snippet: 4.2.1

> Daylight Analysis

Location	File > Data Analysis > daylight_analysis
Purpose	Calculate daylight data for all voxel points
Inputs	Grid of voxel points, sky points, geometry of surrounding buildings, threshold value for point removal
Outputs	Ratio value as data attribute on every voxel point

```
● FOR every voxel point:
    ○ FOR every sky point:
        ■ Get the vector direction from every voxel point to the sky point
        ■ Check for intersect between the voxel point and the sky point
        ■ IF the voxel point can see the sky point (i.e. there is no buildings
          blocking the sun from reaching the voxel point):
            ● Add 1 to the count if an intersection is found
    ○ Ratio = count / total number of sky points
    ○ Fit ratio into range from 0 to 1
```

See code snippet: 4.3.1

> Isovist Analysis

Location	File > Data Analysis > acoustic_analysis
Purpose	Calculate view quality data for all voxel points
Inputs	Grid of voxel points, geometry of surrounding buildings, threshold value for point removal, grid of isovist points with bigger spacing than voxel points
Outputs	Ratio value as data attribute on every voxel point

- **FOR** every isovist grid point:
    - Create a circle of 360 points with given radius around grid point
    - Get vector from grid point to points on circle
    - Check for intersects with surrounding geometry using vector
    - Replace circle points in point list with intersection points if an intersection with geometry was found
    - Draw a polyline passing through every point
    - Calculate area of polyline using *measure* node
    - Calculate ratio between area of polyline and area of circle with start radius
- **FOR** every voxel point:
    - Find closest point in isovist point grid using *nearpoints*
    - Get “view-quality” value from closest point
    - Fit values into range from 0 to 1

See code snippet: 4.4.1, 4.4.2, 4.4.3, 4.4.4

> Closeness Ground

Location	File > Data Analysis > closeness_groundlevel
Purpose	Calculate closeness to ground data for all voxel points
Inputs	Grid of voxel points
Outputs	Closeness value as data attribute on every voxel point

```
● FOR every voxel point:
    ○ Get Y value of point position
    ○ Set Y value as height attribute
    ○ Fit height into inverse range from 0 to 1, such that the maximum height value is
      0 and the minimum height value is 1
```

See code snippet: 4.5.1

> Closeness Vegetation

Location	File > Data Analysis > closeness_vegetation
Purpose	Calculate closeness to vegetation data for all voxel points
Inputs	Grid of voxel points, vegetation points
Outputs	Closeness value as data attribute on every voxel point

- **FOR** every voxel point:
    - Find closest point in vegetation points using *nearpoints*
    - Calculate distance from voxel point to near point
    - Fit distance into inverse range from 0 to 1, such that the maximum distance value is 0 and the minimum distance value is 1

See code snippet: 4.6.1

5. Weighing Points & Growing Functions

> Weighing points

Location	File > Weighing Points and Growing Functions > weighing_points
Purpose	The weighted product aggregates each data value on every point together based on a value weight per function, in turn creating a weighted product value on every point for every function. This means that every point ends up with the same number of W.P. values as there are functions in our building.
Inputs	Voxel points with all the analysis data on them, analysis_weighing.csv
Outputs	Weighted product value for every function on every point, function seeds placed in voxel points

```
● FOR every voxel point:
    ○ Raise data values to the power of data weights for each function, multiply
      these values together to get the weighted product.
    ○ Create “wp” attribute on every point for every function. (e.g. wp1, wp2, wp3)
    ○ Create a “temp” attribute with value -1

● FOR each function:
    ○ Sort point list by W.P. value for current function from best to worst
    ○ Keep the same number of points as functions, discard other points
    ○ IF temp value is equal to -1 (meaning point is empty):
        ■ Place function seed on point
    ○ ELSE
        ■ Check next points until an empty point is found
        ■ Place function on next empty point
```

See code snippet: 5.1.1, 5.1.2

> Growing Functions

Location	File > Weighing Points and Growing Functions > growing_functions
Purpose	Growing each function up to its desired area based on the optimal locations for that function.
Inputs	Voxel points with function seeds placed, spatial_configuration.csv
Outputs	Fully grown function spaces

```
● FOR every point:
    ○ Create occupied group
    ○ Place function seeds in occupied group

● FOR each function:
    ○ IF current area < desired area:
        ■ Get list of points in function (i.e. children)
        ■ FOR each point in function:
            ● Find neighboring points:
                ○ IF point is not occupied:
                    ■ Add to list of unoccupied neighbors
        ■ Sort list of neighbors by W.P. value for current function
        ■ Add the best W.P. value point (index 0 of list of neighbors) to the
          function, setting "parent" attribute to current function and adding it to
          occupied group.
```

See code snippet: 5.2.1

6. Circulation and Final Building Generation

> Clustering

Location	File > Circulation and Final Building Generation > clustering
Purpose	Clustering functions together to create an efficient circulation network
Inputs	Voxel points with functions placed, spatial_configuration.csv
Outputs	Center of functions clustered by closeness

- **FOR** each function:
    - Find center point of function
    - Move center point on to voxel grid by finding nearest voxel point and taking it's coordinates
- Generate clusters from function center points based on closeness, number of clusters can be specified as desired (currently 4)
- **FOR** each cluster:
    - Find center point of the cluster
    - Move center point on to voxel grid by finding nearest voxel point and taking it's coordinates

See code snippet: 6.1.1 and 6.1.2

> Shortest Paths

Location	File > Circulation and Final Building Generation > shortest_paths
Purpose	Connecting cluster to cluster via shortest paths and connecting from cluster centers to its function centers.
Inputs	Voxel points with functions placed, function center positions, cluster center positions
Outputs	Shortest path between clusters & from clusters to functions

- **FOR** every point in voxel grid:
    - Connect every point along the grid using *connect adjacent pieces* houdini node
- **FOR** cluster centers:
    - Find shortest paths from cluster center to every other cluster center along network of connected points using find *shortest path houdini* node
    - Remove duplicate geometry using *clean* node
    - Place voxel boxes along points of shortest path to create primary corridors, using *copy to points* node
- **FOR** function centers:
    - Find shortest paths from cluster centers to every function center in its cluster along network of connected points using find *shortest path houdini* node
    - Remove duplicate geometry using *clean* node
    - Place voxel boxes along points of shortest path to create secondary corridors, using *copy to points* node
- Merge primary and secondary corridors

No code snippet for this process as it’s almost exclusively done with houdini nodes.

> Placing Shafts

Location	File > Circulation and Final Building Generation > placing_shafts
Purpose	Placing shafts to connect the cluster points to the ground level .
Inputs	Cluster points, ground plane, voxel points
Outputs	Vertical means to get from the corridors to ground level

- **FOR** every cluster center point:
  - Copy and translate down below plot level
- Polyextrude the **plot** down.
- Group points by bounding box of extrusion such that all unnecessary points under plot level are in group
- Remove points in group such that only shaft points above the plot are kept
- Place voxel box on points to create shafts

No code snippet for this process as it’s almost exclusively done with houdini nodes.

> Placing Parking

Location	File > Circulation and Final Building Generation > placing_parking
Purpose	Place the function parking in the building
Inputs	Voxel points with function seeds placed
Outputs	Points with the function parking
• **IF** point is not occupied & y-value is higher that two voxel sizes (such that the first two layers of voxels are free, as these are dedicated to public space under the building): ○ Place parking on points	
See code snippet: 6.2.1	

> Final Processing

Location	File > Circulation and Final Building Generation > final_processing
Purpose	To implement all the elements of the building and make it into one building. This includes placing columns under the building and carving skylights through the building.
Inputs	Voxel points with function seeds placed, shafts, corridors and parking voxels
Outputs	The final building

- **FOR** parking voxels (for making columns):
    - Get bottom points of voxels
    - Give each bottom point a random value between 0 and 1 as attribute
    - Keep values under a certain threshold such that a percentage of points are removed (currently 20% kept, 80% removed)
    - Place columns on remaining points

- **FOR** parking voxels (for making light shafts):
    - Move all points to the same level
    - Delete duplicate points
    - Cluster points by closeness, with number of clusters as input
    - Get cluster center points and place a vertically extruded box on them
    - Group points belonging to parking and functions in extrusion boxes and delete them, such that light shafts are created, cutting through the building

- Merge all geometry: corridors, shafts, parking, functions, columns

See code snippet: 6.3.1

7. Changes after Final Presentation

> Method for vegetation placement

Location	File > Changes after Final Presentation > placement_of_vegetation
Purpose	Placing vegetation on the plot based on most suitable spots with best sunlight quality throughout the year.
Inputs	Grid of voxel points, sun points, final building geometry, threshold for sunlight quality based on area needed for vegetation, plot polyline.
Outputs	Most suitable voxel spots for vegetation

```
● FOR grid of voxel points:
    ○ IF y-value of point is greater than 0:
        ■ Remove points, keeping only the points on the ground level

● Calculate the area needed for vegetation, 30% of plot as described in building brief,
  divide by area of voxel to get amount of voxels necessary for vegetation

● FOR remaining ground-level points:
    ○ Apply sunlight calculation in the same way as for sunlight analysis, except the
      final building geometry is also included in intersection geometry, so that
    ○ Sort point list by sunlight quality, keep the same amount of points as necessary
      for vegetation
    ○ Place footprint of voxels on points, representing the vegetation area.
```

> Facade Implementation

Location	File > Changes after Final Presentation > facade_implementation
Purpose	
Inputs	
Outputs	

8. Appendices

These appendices contain vex code snippets from the different systems in our Houdini file. We've decided to only include the most important parts of these systems, smaller code snippets running in point or detail wranglers are not included. These smaller snippets can be found in our Houdini file.

The ID for each snippet (e.g. 1.1.1) represent: Section, Geometry group, Snippet Number.

1. Graph Relaxation

ID	**1.1.1**
Location	File > Graph Relaxation > spatial_configuration > convert_strings_to_int_list
Type	Vex code in attribute wrangler running over detail
Purpose	Re-writes data read from a CSV document for a list of strings into a list of integers and writes it into a point attribute.
Author	Jonas Althuis
Notes	

```
int num_points = npoints(0);

for(int all_points = 0; all_points < num_points; all_points++)
{

    string indexConnections = point(0, "indexConnections", all_points);

    string connectionsList[] = split(indexConnections,",");

    int num_connections = len(connectionsList);

    int intList[];

    for(int Connection = 0; Connection < num_connections; Connection++)
    {
        append(intList, atoi(connectionsList[Connection]));
    }

    setpointattrib(0, "newConnectionList", all_points, intList, "set");
}
```

ID	**1.1.2**
Location	File > Graph Relaxation > spatial_configuration > draw_lines
Type	Vex code in attribute wrangler running over detail
Purpose	Creates a polyline primitive between all points that should be connected to each other.
Author	Jonas Althuis
Notes	

```
int num_points = npoints(0);

for(int all_points = 0; all_points < num_points; all_points++)
{
    int connection_list[] = point(0, "newConnectionList", all_points);

    foreach(int connection; connection_list)
    {

        int startpoint = all_points;
        int endpoint = connection;

        addprim(0, "polyline", startpoint, endpoint);

    }

}
```

ID	**1.1.3**
Location	File > Graph Relaxation > spatial_configuration > write_weights_into_array_attr
Type	Vex code in attribute wrangler running over detail
Purpose	Re-writes data read from a CSV document into a more easy to manage array for every point.
Author	Jonas Althuis
Notes	

```
int num_points = npoints(0);
int attrib_success = 0;

for (int all_points = 0; all_points < num_points; all_points++)
{
    int array_size = pointattribsize(0, "connectionStrengths");

    int weight_list[] = point(0, "connectionStrengths", all_points);
    int array_column = weight_list[all_points];

    float combined_weights[]; // an array to put every weight in for each
point: [4,3,0,0,0...}

    //loop through the columns of the weight_list and add values into an
array
    for(int weights = 0; weights < array_size; weights++)
    {
        float normalised_weights = fit(weight_list[weights], 0, 5, 0, 1);
        append(combined_weights, normalised_weights);
    }

    setpointattrib(0, "combinedWeights", all_points, combined_weights,
"set");

    //setpointattrib(0, "attrTest", all_points, array_column, "set");
}
```

ID	**1.1.4**
Location	File > Graph Relaxation > spatial_configuration > write_connection_weights_to_prims
Type	Vex code in attribute wrangler running over detail
Purpose	Writes the weight of connections as a primitive attribute on the primitive that represents that connection.
Author	Jonas Althuis
Notes	

```
int num_prims = nprimitives(0);
int num_points = npoints(0);

for (int all_prims = 0; all_prims < num_prims; all_prims++)
{
    int prim_points[] = prim(0, "pointsInPrim", all_prims);
    int prim_start = prim_points[0];
    int prim_end = prim_points[1];

    float get_weight[] = point(0, "combinedWeights", prim_start);

    float connection_weight = get_weight[prim_end];
    setprimattrib(0, "weight", all_prims, connection_weight, "set");
}
```

ID	**1.1.5**
Location	File > Graph Relaxation > spatial_configuration > set_pscale_based_on_area
Type	Vex code in attribute wrangler running over detail
Purpose	Sets the pscale of each connection graph point based on the area of the function on that point, these areas are reduced significantly by taking the square root and dividing by a multiplier to make the solver work properly.
Author	Jonas Althuis
Notes	

```
int num_points = npoints(0);
float multiplier = chf("multiplier");

for (int all_points = 0; all_points < num_points; all_points++)
{
    float function_area = point(0, "Area", all_points);
    float graph_weight = point(0,"graphAreaWeight", all_points);
    float sphere_scale = sqrt(function_area) / multiplier;
    setpointattrib(0, "pscale", all_points, sphere_scale, "set");

}
```

3. Entrance Squares

ID	**3.1.1**
Location	File > Envelope Shaping > entrance_squares > add_sun_count_per_group
Type	Vex code in attribute wrangler running over detail
Purpose	Adds up sunlight values for different groups of points, fitting these values to a range from 0 - 1 and writing them to a detail attribute.
Author	Jonas Althuis
Notes	

```
int num_points = npoints(0);
float count0 = 0;
float count1 = 0;
float count2 = 0;
float count3 = 0;

for (int points = 0; points < num_points; points++)
{
    float ratio = point(0, "ratio", points);
    int group0 = inpointgroup(0, "group0", points);
    int group1 = inpointgroup(0, "group1", points);
    int group2 = inpointgroup(0, "group2", points);
    int group3 = inpointgroup(0, "group3", points);

    if (group0 == 1 )
    {
        count0 = count0 + ratio;
    }
    if (group1 == 1 )
    {
        count1 = count1 + ratio;
    }
    if (group2 == 1 )
    {
        count2 = count2 + ratio;
    }
    if (group3 == 1 ){
        count3 = count3 + ratio;
    }
}

setdetailattrib(0, "count0", fit(count0, 0, 100, 0, 1), "set");
setdetailattrib(0, "count1", fit(count1, 0, 100, 0, 1), "set");
setdetailattrib(0, "count2", fit(count2, 0, 100, 0, 1), "set");
setdetailattrib(0, "count3", fit(count3, 0, 100, 0, 1), "set");
```

ID	**3.1.2**
Location	File > Envelope Shaping > entrance_squares > multiply_area_and_percentage
Type	Vex code in attribute wrangler running over detail
Purpose	Sets point scale based on area and sunlight quality at vegetation so that circle placed on points later are scaled properly.
Author	Jonas Althuis
Notes	

```
float area30percent = detail(1, "area30percent", 0);
float areaBaseCircles = detail(1, "areaBaseCircles", 0);
float margin = chf("margin");

float scaleMultiplier = areaBaseCircles / @pscale ;

float areaForpoints = @percentage * area30percent;

float newPscale = @percentage * scaleMultiplier * margin;

setpointattrib(0, "pscale1", @ptnum, newPscale, "set");
```

3. Shadow Casting

ID	**3.2.1**
Location	File > Envelope Shaping > shadow_casting >
Type	Vex code in attribute wrangler running over points
Purpose	Calculate intersections between voxel points and geometry of surroundings on the vector of every sun point to every voxel points, representing the amount of shadow every point casts on the surrounding geometry.
Author	Shervin Azadi
Notes	

```
//initialization
int num_sun = npoints(1);
vector hitp;
float u;
float v;
int count = 0;

//iterate over sun points
for (int i=0; i<num_sun; i++)
{
    //intersect the ray to the context
    vector dir = point(1, "P", i);
    int ind_1 = intersect(2, v@P, dir, hitp, u, v);
    if (ind_1==-1)
    {
        int ind_2 = intersect(2, v@P, -dir, hitp, u, v);
        if (ind_2>-1)
        {
            count = count + 1;
        }
    }
}
//output
int total = detail(1, "total");
f@ratio = count / float(total);
i@c = count;
```

ID	**4.1.1**
Location	File > Data Analysis > acoustic_analysis > calc_acoustic_value
Type	Vex code in attribute wrangler running over points
Purpose	
Author	Adrian Wong, Jonas Althuis
Notes	

```
float max_distance = 50.0;
float values[];
float multiplier = chf("multiplier");
int closest_points[] = nearpoints(1, "soundpoints", v@P, max_distance);

foreach(int closest_point; closest_points)
{
    float sound_value = point(1, "soundLevel", closest_point);
    vector sound_point_position = point(1, "P", closest_point);
    float pointDistance = distance(v@P, sound_point_position)/multiplier;
    float Lp = sound_value-20*log(pointDistance)-8;
    float value = pow(10,Lp/10.0);
    append(values, value);
}

float weighted_sound_level = 10.0*log10(sum(values));
if (weighted_sound_level < 0) weighted_sound_level = 0;
setpointattrib(0, "weightedSoundLevel", @ptnum, weighted_sound_level,
"set");
```

ID	**4.1.2**
Location	File > Data Analysis > acoustic_analysis > fit_range
Type	Vex code in attribute wrangler running over points
Purpose	Works in combination with two sort nodes, used to find the minimum and maximum values, fits the given data into a range from 0 to 1.
Author	Jonas Althuis
Notes	This same wrangler is used for every data analysis and several other locations in our Houdini file, but for efficiency's sake is not included as code here in every analysis section.

```
float sound = point(0, "analysis1", @ptnum);
float max = detail(1, "max", 0);
float min = detail(2, "min", 0);
float data = fit(sound, min, max, 0, 1);
setpointattrib(0, "analysis1", @ptnum, data, "set");
```

ID	**4.1.3**
Location	File > Data Analysis > acoustic_analysis > acoustic_threshold
Type	Vex code in attribute wrangler running over points
Purpose	Removes points with values above acoustic threshold
Author	Jonas Althuis
Notes	

```
float ratio = point(0, "analysis1", @ptnum);
float threshold = chf("acoustic_threshold");

if (threshold > ratio)
{
    removepoint(0, @ptnum);
}
```

ID	**4.2.1**
Location	File > Data Analysis > sunlight_analysis > calc_sunlight
Type	Vex code in attribute wrangler running over points
Purpose	Calculate sunlight ratio for every voxel point, based on intersections with the surrounding geometry.
Author	Shervin Azadi, Hannah van der Sluijs, Jonas Althuis
Notes	

```
//initialization
int num_sun = npoints(1);
vector hitp;
float u;
float v;
int count = 0;

//iterate over sun points
for (int i=0; i<num_sun; i++)
{
    //intersect the ray to the context
    vector dir = point(1, "P", i);
    int ind = intersect(2, v@P, dir, hitp, u, v);
    if (ind==-1)
    {
        count = count +1;
    }
}

//output
int total = detail(1, "total");
f@ratio = count / float(total);
i@c = count;
```

4. Daylight Analysis

ID	**4.3.1**
Location	File > Data Analysis > daylight_analysis > calc_daylight
Type	Vex code in attribute wrangler running over points
Purpose	Calculate daylight (sky view) ratio for every voxel point based on intersections with the surrounding geometry.
Author	Shervin Azadi, Jonas Althuis
Notes	

```
//initialization
int num_sky = npoints(1);
vector hitp;
float u;
float v;
int count = 0;

//iterate over sky points
for (int i=0; i<num_sky; i++)
{
    //intersect the ray to the context

    vector dir = point(1, "P", i);
    int ind = intersect(2, v@P, dir, hitp, u, v);
    if (ind==-1)
    {
        count = count +1;
    }
}

//output
int total = detail(1, "total");
f@ratio = count / float(total);
i@c = count;
```

ID	**4.4.1**
Location	File > Data Analysis > isovist_analysis > draw_circle_points_with_radius
Type	Vex code in attribute wrangler running over points
Purpose	Draws a circle of points with a specified radius and a specified number of points on it.
Author	Jonas Althuis
Notes	

```
int circle_divisions = chi("Circle_Divisions");
float circle_radius = chf("Circle_Radius");

vector start_point = point(0, "P", @ptnum);
vector origin = {0,0,0};
vector origin_to_point = origin - start_point;

for (int degree_counter = 0; degree_counter < circle_divisions;
degree_counter++)
{
    float angle = radians(degree_counter);

    vector circle_points = set((circle_radius * sin(angle)), 0,
(circle_radius * cos(angle)));

    vector surrounding_points = circle_points - origin_to_point;

    addpoint(0, surrounding_points);
}
```

ID	**4.4.2**
Location	File > Data Analysis > isovist_analysis > vector_from_center_to_circle_points
Type	Vex code in attribute wrangler running over detail
Purpose	Intersects vectors of center point to point on circle with given radius into geometry of surroundings, writing intersection location as attribute into each circle point, to be used later.
Author	Jonas Althuis
Notes	

```
int num_points = npoints(0);
vector start_point = point(0, "P", 0);

for (int all_points = 0; all_points < num_points; all_points++)
{
    vector intersection_point;
    vector dump;

    vector circle_points = point(0, "P", all_points);
    vector direction = circle_points - start_point;

    int rays = intersect(1, "surroundings", start_point, direction,
intersection_point, dump);

    setpointattrib(0, "foundIntersect", all_points, rays, "set");
    setpointattrib(0, "intersectPoint", all_points, intersection_point,
"set");
}
```

ID	**4.4.3**
Location	File > Data Analysis > isovist_analysis > replace_circle_point_with_intersect
Type	Vex code in attribute wrangler running over points
Purpose	Replaces points on the circle with points that have intersected geometry, creating a new point list containing the isovist.
Author	Jonas Althuis
Notes	

```
int group_check = point(0, "parentpoints", @ptnum);
int intersect_check = point(0, "foundIntersect", @ptnum);

if (group_check != 1)
{
    if (intersect_check != -1 && intersect_check != 0)
    {
        vector intersect_point = point(0, "intersectPoint", @ptnum);
        setpointattrib(0, "P", @ptnum, intersect_point, "set");
    }
}
```

ID	**4.4.4**
Location	File > Data Analysis > isovist_analysis > find_nearpoints_get_ratio
Type	Vex code in attribute wrangler running over points
Purpose	Aggregates isovist data into voxel points by finding closest isovist point and taking its value.
Author	Jonas Althuis
Notes	

```
vector base_point = point(0, "P", @ptnum);

int near_point = nearpoint(1, "findnear", base_point);

float view_ratio = point(1, "ratio", near_point);

setpointattrib(0, "ratio", @ptnum, view_ratio, "set");
```

4. Closeness Ground

ID	**4.5.1**
Location	File > Data Analysis > closeness_groundlevel > set_height
Type	Vex code in attribute wrangler running over points
Purpose	Get the y value of every point and set it as an attribute called “height”
Author	Jonas Althuis
Notes	

```
vector point_xyz = point(0, "P", @ptnum);

float point_y = point_xyz[1];

f@height = point_y;
```

4. Closeness Vegetation

ID	**4.6.1**
Location	File > Data Analysis > closeness_vegetation > set_closeness
Type	Vex code in attribute wrangler running over detail.
Purpose	Find closest point in voxel points to vegetation points, calculate distance between each point to its closest point.
Author	Jonas Althuis
Notes	

```
int num_points = @numpt;

for (int all_points = 0; all_points < num_points; all_points++)
{
    vector current_point = point(0, "P", all_points);

    int closest_point_id = nearpoint(1, "vegetation", current_point);
    vector sound_point = point(1, "P", closest_point_id);

    float distance = distance(current_point, sound_point);
    setpointattrib(0, "distance", all_points, distance, "set");
}
```

ID	5.1.1
Location	File > Weighing Points and Growing Functions > weighing_points > product
Type	Python code running in a python node
Purpose	Calculates the weighted product of a set of incoming data and incoming weights using a numpy array.
Author	Shervin Azadi, Pirouz Nourian
Notes	

```
__version__ = '0.1'
__date__ = '28-11-2019'
__author__ = 'Shervin Azadi & Pirouz Nourian'

import numpy as np
node = hou.pwd()


#function to put the attributes of the houdini geometry into a numpy array
def attrib_to_nparray(inputIndex, attributeList):

    #loading the geometry of the corresponding input
    geo = node.inputGeometry(inputIndex)

    #counting the number of the elements of the attributes (vectors have
more than one for eg)
    numElementAttrib = 0
    for attrib in attributeList:
        #retrieve the attribute value
        val = geo.point(0).attribValue(attrib)
        if isinstance(val, tuple):
            numElementAttrib += len(val)
        else:
            numElementAttrib += 1
    #getting the number of the points
    numpt = len(geo.points())
    # initialize the numpy array
    DataSet = np.zeros((numpt, numElementAttrib), dtype=float)

    # iterate over points
    for i in range(numpt):

        #iterate over attribs
        j = 0
        for attrib in attributeList:

            #retrieve the attribute value
            val = geo.point(i).attribValue(attrib)

            #check if it is a vector
            if isinstance(val, tuple):
                #iterate over the vector elements and store all of them
                for k in range(len(val)):
                    DataSet[i][j] = val[k]
                    j += 1
            else:
                DataSet[i][j] = val
                j += 1
    return DataSet


#function to write the data of a numpy array onto the houdini geometry
def nparray_to_attrib(DataSet, name):
    #load the main geometry
    geo = node.geometry()
    #iterate over columns of DataSet (attributes)
    for j in range(DataSet.shape[1]):
        #create the name of the attribute
        attribName = name + str(j)
        #initialize the attribute in the geometry
        geo.addAttrib(hou.attribType.Point, attribName, 0.0)
        #iterate over the rows of DataSet (points)
        for i in range(DataSet.shape[0]):
            # read the value that corresponds to each point from DataSet
and write it to the attribute
            geo.iterPoints()[i].setAttribValue(attribName, DataSet[i][j])


#read the attribute list for voxels
```

```
attribs0 = hou.evalParm("attributes").split()

#put the VoxelData in a numpy array
VoxelData = attrib_to_nparray(0, attribs0)

#read the attribute list for functions
attribs1 = hou.evalParm("weight").split()

#put the FunctionData in a numpy array
FunctionData = attrib_to_nparray(1, attribs1)


#initialize the WP matrix with the number of points as the numberof rows
AND number of functions as number of attributes
WeightedProduct = np.zeros((VoxelData.shape[0],FunctionData.shape[0]),
dtype=float)

#iterate of the functions
for i in range(FunctionData.shape[0]):

    #raising each voxel value to the power of the function criteria
    powers = np.float_power(VoxelData, FunctionData[i])

    #multiplying all the value powers together for each voxel
    product = np.prod(powers, axis=1)

    #placing the result in the corresponding column of the weighted product
matrix
    WeightedProduct[:,i] = product

#place the calculated WP as attribute 'func' on the voxels
nparray_to_attrib(WeightedProduct, "wp")
```

ID	**5.1.2**
Location	File > Weighing Points and Growing Functions > weighing_points > check_empty
Type	Vex code in attribute wrangler running over detail
Purpose	When placing building functions in most suitable locations based on weighted product, this node checks if the next best point is already occupied or not.
Author	Jonas Althuis, Hans Hoogenboom
Notes	

```
i@counter = 0;

for(int i = 0; i < @numpt; i++) {

   int temp = point(0, "temp", i);

   @counter = i;

   if (temp == -1) {
       break;
   }
}
```

ID	5.2.1
Location	File > Circulation and Final Building Generation > growing_functions > GrowthModel > Agents_Growth
Type	Vex code in attribute wrangler running over detail
Purpose	Grows the building functions iteratively using weighted product, looking at neighboring points and growing into the most suitable point.
Author	Shervin Azadi, Jonas Althuis
Notes	Flatness is being imported but not used.

```
/*
__version__ = '0.2'
__date__ = '10-01-2020'
__author__ = 'Shervin Azadi and Jonas Althuis'
*/

// 0: initialize variables
int nfunc = npoints(1);
float squarness = chf("squareness");
float voxel_size = chf("voxel_size");


// 1: iterate over each function (agent)
for (int j=0; j<nfunc; j++)
{
    // 1.1: check if the function(agent) have enough voxels

    //current area
    float flatness = 1 + point(0, "Flatness", j);
    float blockiness = 1 + point(0, "Blockiness", j);
    int cur_area = (findattribvalcount(0, "point", "parent", j) *
(pow(voxel_size,2)));
    string attr_name = "area" + itoa(j);
    setdetailattrib(0, attr_name, cur_area, "set");

    //desired area
    int des_area = point(1, "area", j);
```

```
//compare current with desired
if (des_area > cur_area )
{
    // 1.2: prepare children to find the boundary of function
    //get the number of children
    int num_child = findattribvalcount(0, "point", "parent", j);

    //initialize the boundary lists
    int func_bounds_id[];
    float func_bounds_wp[];

    // 1.3: iterate over children
    for (int i=0; i<num_child; i++)
    {
        // 1.3.1: find the neighbours of the child
        //retrieve the child id
        int child_id = findattribval(0, "point", "parent", j, i);
        //retrieve the child neighbours
        int child_neighs[] = neighbours(0, child_id);

        // 1.3.2: iterate over the neighbours of the child
        foreach (int neigh; child_neighs)
        {
            //1.3.2.1: check if they are occupied
            if (1 - inpointgroup(0, "occupied", neigh))
            {
                //find it in the currently existing boundary list
                int found = find(func_bounds_id, neigh);

                //if you did not find it add it to the list
                if (found < 0)
                {
                    // add the id
                    append(func_bounds_id, neigh);

                    // add the wp
                    string attr_name = "wp" + itoa(j);
                    float wp = point(0, attr_name, neigh);
                    append(func_bounds_wp, wp);

                }
```

```
                    else
                    {
                        //multiply by squareness factor
                        func_bounds_wp[found] *= blockiness;
                    }
                }
            }
        }
        // 1.3: sort the list of boundary voxels in decreasing order
        int sorted_indicies[] = reverse(argsort(func_bounds_wp));
        func_bounds_id = reorder(func_bounds_id, sorted_indicies);

        // 1.4: set the first voxel in the (sorted) boundary list as the
new child (occupy it by the function)
        setpointgroup(0, "occupied", func_bounds_id[0], 1, "set");
        setpointattrib(0, "parent", func_bounds_id[0], j, "set");
    }
}
```

6. Clustering

ID	**6.1.1**
Location	File > Circulation and Final Building Generation > clustering > move_to_voxel_grid
Type	Vex code in attribute wrangler running over points
Purpose	Moves the current point onto the voxel grid by finding the nearest voxel point and copying its position attribute.
Author	Jonas Althuis
Notes	This code snippet is used several times in different places, but we will only include it once here.

```
@cluster = @cluster + 1;
int near_point = nearpoint(1, @P);
vector replace = point(1, "P", near_point);
@P = replace;
```

ID	**6.1.2**
Location	File > Circulation and Final Building Generation > shortest_paths > create_groups
Type	Vex code in attribute wrangler running over points
Purpose	Creates groups of points based on cluster. This is uses an if condition because the “clusterHub” attribute is a list filled with many empty points that have a value of -1, which we don’t want included in the groups but are necessary to create a network in which the shortest path can solve.
Author	Jonas Althuis
Notes	

```
string groupset = "hub" + itoa(@clusterHub);

if (@clusterHub > 0)
{
    setpointgroup(0, groupset, @ptnum, 1,"set");
}
```

ID	**6.2.1**
Location	File > Circulation and Final Building Generation > placing_parking > select_parking_points
Type	Vex code in attribute wrangler running over points
Purpose	Placing parking points under a certain height-cap.
Author	Jonas Althuis
Notes	

```
float voxel_size =
chf("../../building_voxelization/size_switch/voxel_size");
int multiplier = chi("multiplier");
float v_start = voxel_size / 2 ;

float height_cap = v_start + multiplier * voxel_size;

if (@P[1] > height_cap)
{
setpointgroup(0, "bound", @ptnum, 1, "set");
}
```

6. Final Processing

ID	**6.3.1**
Location	File > Circulation and Final Building Generation > final_processing > get_taken_points
Type	Vex code in attribute wrangler running over points
Purpose	Removes all points that are empty after functions have been placed.
Author	Jonas Althuis
Notes	

```
int check = point(0, "parent", @ptnum);

if (check == -1)
{
    removepoint(0, @ptnum);
}
```

ID	**6.3.2**
Location	File > Circulation and Final Building Generation > final_processing > threshold_columns
Type	Vex code in attribute wrangler running over points
Purpose	Removes points to reduce the number of columns randomly.
Author	Jonas Althuis
Notes	

```
float rand = point(0, "rand", @ptnum);
float threshold = ch("threshold");
if (rand > threshold )
{
removepoint(0, @ptnum);
}
```