Codebook

1. 3D City Optimization

Divide each building primitive into multiple smaller apartment primitives

File	obj > optimize > divide_into_appartment_prims
Function	We calculate where to place each point on the current primitive and create new primitives of size 5x3 (or as close to that as possible) all over the primitive. We then delete the original.
Input	The 3D city grid without primitives facing away
Output	The 3D city grid divided into smaller primitives
Run Over	Primitives

Pseudocode:

Calculate the height vector of the building;
Calculate the width vector of the building;
Normalize the height and width vectors of the building;
Calculate the height and width of a single building appartment;
Multiply them with the normalized vectors to get the height and width vectors of an apartment;
FOR Number of apartments in height:
  FOR Number of apartments in width:
    Calculate positions of top and bottom, left and right points of the current apartment;
    Create a new primitive along those points;
Delete the original primitive;

Input wrangle nodes used for variables:
obj > optimize > attributewrangle1

See code snippet: 1.1

2. Sunlight Analysis

Calculate the number of sun hours and shadow score per primitive

File	obj > rays > calc_hits_advanced1
Function	This node is set to run over each point in the voxelgrid. For each run, we iterate over each sun hour and calculate whether that point gets sun. Then, if it does get sun, we calculate if it casts shadow. Finally, if it does cast shadow, calculate how many hours of sun the shadow cast primitive gets. Divide 1 by this number and add that to the shadow contribution score of the voxel. We then use the number of sun intersections, shadow intersections and the shadow contribution score in the analysis and shaping of the voxel.
Input	0: The voxelgrid 1: All sun hour points in a year 2: The optimized city grid
Output	The number of sun hours blocked, the number of shadow casts and the shadow contribution score of each voxel.
Run Over	Points

Pseudocode:

```
FOR each sun hour:
  Calculate vector from voxel to the sun;
  IF sun vector intersects with city grid geometry:
    Append hit primitive number to hits list of voxel;
  Invert sun vector as new shadow vector if there was no intersection;
  IF shadow vector intersects with city gid geometry:
    Append hit primitive number to blocked list of voxel;
    Get centroid of the hit primitive;
    FOR: each sun hour in the current day:
      Calculate whether the hit primitive gets sun this hour;
    Shadow score is 1 / sun hours of hit primitive today;
    Add shadow score to shadow contribution variable of the current voxel;
```

Input wrangle nodes used for variables:
obj > rays > attributewrangle3
obj > rays > add_attr_hits
obj > rays > attributewrangle2
Output wrangle nodes used for variables:
obj > rays > calc_num_hits
obj > rays > attributewrangle1

See code snippet: 2.1

3. Facade

Group lowest north facade points into ground floor group

File	obj > rays > attributewrangle2
Function	Check if the height of the voxel is between 1.4 and 1.6. If so, group it into the “ground floor” group.
Input	The voxelgrid points, which are grouped into 4 facades.
Output	The voxelgrid points, now grouped into 4 facades as well as a ground floor group if they’re at the lowest level
Run Over	Points

Pseudocode:
IF point vector y is between 1.4 and 1.6:   Add point to “ground floor” group;
See code snippet: 3.1

Group lowest east facade points into ground floor group

File	obj > rays > attributewrangle3
Function	Check if the height of the voxel is between 1.4 and 1.6. If so, group it into the “ground floor” group.
Input	The voxelgrid points, which are grouped into 4 facades.
Output	The voxelgrid points, now grouped into 4 facades as well as a ground floor group if they’re at the lowest level
Run Over	Points

Pseudocode:
IF point vector y is between 1.4 and 1.6:   Add point to “ground floor” group;
See code snippet: 3.1

Group lowest south facade points into ground floor group and place entrances

File	obj > rays > attributewrangle4
Function	Check if the height of the voxel is between 1.4 and 1.6. If so, check whether the x position of this voxel is equal to the x position of one of the three predetermined entrances. If so, group the voxel into the "sliding doors" group, else, group it into the "ground floor" group.
Input	The voxelgrid points, which are grouped into 4 facades.
Output	The voxelgrid points, now grouped into 4 facades as well as a ground floor group or sliding door group if they're at the lowest level.
Run Over	Points

Pseudocode:

Save x position for left entrance;
Save x position for middle entrance;
Save x position for right entrance;
IF point vector y is between 1.4 and 1.6:
  IF x position of voxel is equal to x position of left entrance:
    Add point to "sliding doors" group;
  ELSE IF x position of voxel is equal to x position of middle entrance:
    Add point to "sliding doors" group;
  ELSE IF x position of voxel is equal to x position of right entrance:
    Add point to "sliding doors" group;
  ELSE:
    Add point to "ground floor" group;

See code snippet: 3.2

Group lowest west facade points into ground floor group

File	obj > rays > attributewrangle5
Function	Check if the height of the voxel is between 1.4 and 1.6. If so, group it into the “ground floor” group.
Input	The voxelgrid points, which are grouped into 4 facades.
Output	The voxelgrid points, now grouped into 4 facades as well as a ground floor group if they’re at the lowest level
Run Over	Points

Pseudocode:
IF point vector y is between 1.4 and 1.6:   Add point to “ground floor” group;
See code snippet: 3.1

4. Weighing points

Location	File > uSeed_points
Purpose	Get seeded points on ground floor
Inputs	Weighted points in point cloud
Outputs	Seeded points

Create attribute ‘groundfl’ and set value to -1 for all points.
Set value to 1 if the y-coordinate is 1.5

Loop as many times as there are functions:

  Sort the wp values, starting with wp0 (weights for function 0).

  Run over all points:

    Keep count and store in a detail attribute.
    Stop the script when it finds an unoccupied point on the ground floor.

The counter is now the same as the point number that is the best available point on the

  ground floor.
  Set this point as a parent.

See code snippet: 4.1

5. Wireframe region growing

<table>
<tr><td>Location</td><td>File > Region_growing</td></tr>
<tr><td>Purpose</td><td>Create a wireframe for growing algorithm</td></tr>
<tr><td>Inputs</td><td>Seeded points</td></tr>
<tr><td>Outputs</td><td>Wireframe</td></tr>
<tr><td colspan="2">Find all points within a radius that's as big as the voxel width and store the points in a list.
Remove the first point in the list, since this is the point itself.
Create a line from the point to all neighbors in the list.

Loop 20 times: (after each iteration it may have happened that by deleting primitives, other points now have less than three neighbours)
    For all points:
        Create a list with the primitives connected to that point.
        If there are less than three primitives in the list:
            Remove the primitives.

Fuse everything
Go into the growth model</td></tr>
<tr><td colspan="2">See code snippet: 5.1</td></tr>
</table>

6. Tower cores analysis

Location	**File > distancecore**
Purpose	**Create cores for the towers and use the normalized distance to these cores as an attribute for growing.**
Inputs	**Point cloud from shadow casting analysis**
Outputs	**Reshaped point cloud and normalized distance attribute**

Initialize a list with as many indices as there are floors to store information about every floor.

Run over all points:

Get the y-coordinate.

Calculate which floor number it is on.

Add 1 to the index of that floor.

Sort the list. The first value in the sorted list is the index of the smallest floor.

Create a group ‘smallestfloor’.

For all points:

Get the y-coordinate of the point and calculate what floor it is on.

If the point is on the smallest floor, set the value to 1 in the group ‘smallestfloor’.

Perform k-means clustering on the points of the smallest floor with the cluster node. We're looking for two towers, so we set the clusters to 2.

We set the y-coordinate of those points to 1.5, the height of the points on the ground floor.

Copy the points to the same place, but at y = 121.5.

Create lines between the high points and the points on the ground floor (the cores) and calculate the distance from all points in the point cloud to the cores and store this in attribute ‘dist’.

For all points:

Get the y-coordinate and distance to the core.

If height > threshold and distance > threshold:

Delete the point

The ‘dist’ attribute is normalized and set as the attribute 'analysis3'.

See code snippet: 6.1

Appendix

Code Snippets

Snippet 1.1

Location	File > optimize > divide_into_apartment_prims
Type	Vex code in attribute wrangler running over primitives
Purpose	Subdivide single facade primitive into multiple smaller ones
Author	Lapo den Hollander

```
int pts[] = primpoints(0, @primnum);
int npt = len(pts);
int newpts[] = {};
float primHeight = 3.0;
float primWidth = 5.0;

// Check if this primitive actually is a rectangle
if (npt == 4)
{
    // Caluclate the height and width of the building
    vector startPos = point(0, "P", pts[0]);
    vector heightPos = point(0, "P", pts[3]) - startPos;
    vector widthPos = point(0, "P", pts[1]) - startPos;
    float height = length(heightPos);
    float width = length(widthPos);

    // If the building is smaller than 5x3, adjust the interval
    if (height < primHeight) {
        primHeight = height;
    }
    if (width < primWidth) {
        primWidth = width;
    }

    // Calculate the amount of appartments and the interval vectors
    int heightIntervals = floor(height / primHeight);
    int widthIntervals = floor(width / primWidth);
    vector primUpPos = heightPos / heightIntervals;
    vector primRightPos = widthPos / widthIntervals;

```

```
    // Add a point at every end of a single calculated appartment
    // Store its index and create primitives between those points
    for (int i = 0; i <= heightIntervals; i++) {
        for(int j = 0; j <= widthIntervals; j++) {
            vector newPos = startPos + i * primUpPos + j * primRightPos;
            int newPointIndex = addpoint(0, newPos);
            append(newpts, newPointIndex);
            if (i != 0 && j != 0) {
                int botRight = newpts[(i-1) * (widthIntervals+1) + j];
                int botLeft = newpts[(i-1) * (widthIntervals+1) + (j-1)];
                int topRight = newpts[i * (widthIntervals+1) + j];
                int topLeft = newpts[i * (widthIntervals+1) + (j-1)];
                addprim(0, "poly", botRight, topRight, topLeft, botLeft);
            }
        }
    }
    // Finally remove the original primitive and points
    removeprim(0, @primnum, 1);

}

```

Snippet 2.1

Location	File > rays > calc_hits_advanced1
Type	Vex code in attribute wrangler running over points
Purpose	Calculate sun hours, shadow casts and shadow contribution
Author	Lapo den Hollander

```
int num_sun_hours = npoints(1);
float _shadow_con = 0.0;

int day = point(1, "day", 0);
int day_hits = 0;


for(int hour = 0; hour < num_sun_hours; hour++) {

    //if (hour == num_sun_hours || point(1, "day", hour) != day) {
    //    // store
    //    append(i[]@hits_per_day, day_hits);
    //    int total_hours = day_counter - day_hits;
    //    append(i[]@sun_day, total_hours);
    //}

    vector shadow_dir = normalize(point(1, "P", hour)) * chf('max_dist');
    vector sun_dir = shadow_dir * -1;

    vector hit_pos;
    float u, v;
    int hit_prim;
    int hit_prim_shadow;

    // can our voxel see the sun?
    hit_prim = intersect(2, @P, shadow_dir, hit_pos, u, v);
    if (hit_prim > -1) {
        append(i[]@hitprims, hit_prim);
        day_hits += 1;
    }
```

```
    // do we block something in the neighborhood?
    hit_prim_shadow = intersect(2, @P, sun_dir, hit_pos, u, v);
    if (hit_prim_shadow > -1 && !inprimgroup(2, "roofs", hit_prim_shadow))
        append(i[]@blocked, hit_prim_shadow);

        int n = 0;
        vector centroid = primintrinsic(2, "centroid", hit_prim_shadow);

        for(int i = 0; i < num_sun_hours; i++) {
            vector shadow_dir_hour = normalize(point(1, "P", i)) * chf('ma
            int hit_prim_hour = intersect(2, centroid, shadow_dir_hour, hi
            if (hit_prim_hour < 0) {
                n += 1;
            }

            if (n > 0) {
                _shadow_con += 1 / n;
            }
        }

    }
}


float _t = _shadow_con / num_sun_hours;
@shadow_con = f@shadow_con + _t;
i@hits = len(@hitprims);
i@block = len(@blocked);
```

Snippet 3.1

Location	**File > uSeed_points > attributewrangle2** **File > uSeed_points > attributewrangle3** **File > uSeed_points > attributewrangle5**
Type	**Vex code in attribute wrangler running over points**
Purpose	**Adding lowest facade voxels to groundfloor group and entrances**
Author	**Bart Koppejan, based on given code**

```
float height = @P.y;

if (height > 1.4 && height < 1.6) {
    setpointgroup(0, "groundfloor", @ptnum, 1, "set");
}
```

Snippet 3.2

Location	**File > facade > attributewrangle4**
Type	**Vex code in attribute wrangler running over points**
Purpose	**Adding lowest facade voxels to groundfloor group and entrances**
Author	**Bart Koppejan and Lapo den Hollander**

```
float height = @P.y;
float offset = 3.6;
float x_wall = @P.x - offset;

if (height > 1.4 && height < 1.6) {
    // Set left door at this location
    if (x_wall > -342.8 && x_wall < -339.2) {
        setpointgroup(0, "door", @ptnum, 1, "set");
    }
    // Set right door at this location
    else if (x_wall > -274.4 && x_wall < -270.8) {
        setpointgroup(0, "door", @ptnum, 1, "set");
    }
    // Set middle door at this location
    else if (x_wall > -306.8 && x_wall < -303.2) {
        setpointgroup(0, "door", @ptnum, 1, "set");
    }
    // Set window walls everywhere else
    else {
        setpointgroup(0, "groundfloor", @ptnum, 1, "set");
    }
}
```

Snippet 4.1

Location	File > uSeed_points > create_id_and_bla File > uSeed_points > check_empty File > uSeed_points > write_attributes
Type	Vex code in attribute wrangler running over points Vex code in attribute wrangler running over detail Vex code in attribute wrangler running over points
Purpose	Select seed points
Author	Bart Koppejan, based on given code

VEXpression

```
i@id = @ptnum;
i@temp = -1;
i@func_id = -1;
s@func = "empty";

i@groundfl = -1;
vector point_xyz = point(0, "P", @ptnum);
float point_y = point_xyz[1];

if (point_y == 1.5) {
    i@groundfl = 1;
}
```

VEXpression

```
i@counter = 0;

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

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

    @counter = i;

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

VEXpression

```
i@temp = @id;
i@func_id = @loop;
s@func = point(1, "Function", @loop);
```

Snippet 5.1

Location	**File > Region_growing > find_neighbours** **File > Region_growing > attribwrangle8**
Type	**Vex code in attribute wrangler running over points** **Vex code in attribute wrangler running over points**
Purpose	**Create wireframe**
Author	**Bart Koppejan**

Radius `ch("../../voxelgrid/control_grid/width") + 0.1`

VEXpression

```
int np[] = nearpoints(0, @P, chf('radius'));
removeindex(np, 0);
i[]@pts = np;

for (int i = 0; i < len(i[]@pts); i++) {
    if (@ptnum < @pts[i]) {
        int result = addprim(geoself(), "polyline", @ptnum, @pts[i]);
    }
}
```

VEXpression

```
int plist[] = pointprims(0, @ptnum);

if (len(plist) < 3) {
    for (int j = 0; j < len(plist); j++) {
        removeprim(geoself(), plist[j], 0);
    }
}
```

Snippet 6.1

Location	**File > distancecore > calculate_floorsizes** **File > distancecore > add_to_group** **File > distancecore > shape_tower**
Type	**Vex code in attribute wrangler running over detail** **Vex code in attribute wrangler running over points** **Vex code in attribute wrangler running over points**
Purpose	**Select the smallest floor and shape based on distance to a core**
Author	**Bart Koppejan, based on given code**

VEXpression

```
int sf[];
int floors = chi("floors");
for (int i = 0; i < floors; i++) {
    append(sf, 0);
}

for (int i = 0; i < npoints(0); i++) {
    vector temp_point = point(0, "P", i);
    float point_y = temp_point.y;
    int floornr = floor(rint((point_y - 1.5) / 3));
    sf[floornr] = sf[floornr] + 1;
}

i[]@sort = argsort(sf);
i[]@sizefloor = sf;
```

VEXpression

```
int floorsize[] = detail(0, "sort");
int floornr = floorsize[0];

float point_y = @P.y;
int floorpoint = floor(rint((point_y - 1.5) / 3));

if (floorpoint == floornr) {
    setpointgroup(0, "smallestfloor", @ptnum, 1, "set");
}
```

VEXpression

```
if (@dist > 12 && @P.y > 30.6) {
    removepoint(geoself(), @ptnum);
}
if (@dist > 15 && @P.y > 24.6) {
    removepoint(geoself(), @ptnum);
}
if (@dist > 22 && @P.y > 16.6) {
    removepoint(geoself(), @ptnum);
}
if (@dist > 30 && @P.y > 10.6) {
    removepoint(geoself(), @ptnum);
}
if (@dist > 40 && @P.y > 7.6) {
    removepoint(geoself(), @ptnum);
}
```