Solar envelope (Run only once)

```
for ( int i = 0; i < npoints(2); i++)                                   // # of faces
{
    for ( int j = 0; j < nprimitives(3); j++)                           // # of sunrays
    {
        int Intersect_voxel[], sun = 0, num_of_rays = 0;
        float u = 0.0, v = 0.0;
        vector Sun_ray, p = {0,0,0}, q[], uvw[];

        Sun_ray = prim_normal(3, j, 0.5, 0.5) * 300;                    // create sunray
        sun = intersect(1, point(2, 'P', i), Sun_ray, p, u, v);         // check whether the face would receive sunlight

        if(sun != -1)
       // if it would receive sunlight, check whether it intersects with the voxelcloud
        {
            num_of_rays = intersect_all(0, point(2, 'P', i), Sun_ray , q, Intersect_voxel, uvw);
            for (int k = 0; k < num_of_rays; k++)                       // increase the Hit_counter of the voxel that were hit by 1
            {
                setprimattrib(0, 'Voxel_score', Intersect_voxel[k], 1, "add");
            }
        }
    }
}
```

Merge scores on point cloud (Run only once)

```
for ( int i = 0; i < nprimitives(1); i++)       // # of faces on voxelcloud
{
    int ID = prim(1, 'Point_ID', i);
    int score = prim(1, 'Voxel_score', i);

    setpointattrib(0, 'Hit_counter', ID, score, "add");
}
```

Delete floating voxels (Run only once)

```
for( int i = 0; i < npoints(0); i++)                                                    // # of points in pointcloud
{
    vector Pos_temp = point(0, 'P', i);
    Pos_temp[1] -= (chf("../Voxel_cloud/Voxel/sizey") * chi("../Voxel_cloud/Voxel/scale")); // look at the voxel underneath

    if(Pos_temp[1] > 0)                                                                 // don't delete the bottom row
    {
        int lower_voxel = nearpoint(0, Pos_temp, 0.1);                                  // check for a voxel below
        Pos_temp = point(0, 'P', i);                                                    // reset location

        if(lower_voxel == -1)                                                           // if there is no voxel underneath
        {
            removepoint(0, i);
            int top_voxel = 0;
            while(top_voxel != -1)                                          // keep looking up until there is no more voxel above
            {
                Pos_temp[1] += (chf("../Voxel_cloud/Voxel/sizey") * chi("../Voxel_cloud/Voxel/scale"));
                top_voxel = nearpoint(0, Pos_temp, 0.1);
                if(top_voxel != -1)
                {
                    removepoint(0, top_voxel);
                }
            }
        }
    }
}
```

Tune_voxel_cloud (run over points)

```
int Near_voxels[];
int scaler = chi("../Voxel_cloud/Voxel/scale");
scaler = chi("../Also_voxel/scale");

Near_voxels = nearpoints(0, @P, 1.1 * scaler, 9);   // search the surroundings for upo to 9 voxels.

if(len(Near_voxels) < 5)                            // if the point has less than 5 neighbours, delete it.
{
    removepoint(0, @ptnum);
}
```

Sun_rating (run over primitives of voxelcloud)

```
for ( int j = 0; j < nprimitives(1); j++)   // # of sunrays
{

    int Intersect_voxel[], sun = 0, num_of_rays = 0;
    float u = 0.0, v = 0.0;
    vector Sun_ray, p = {0,0,0};

    Sun_ray = prim_normal(1, j, 0.5, 0.5) * 300;

    sun = intersect(2, @P + prim_normal(0, @primnum, 0.5, 0.5) * 0.1, Sun_ray, p, u, v);
    if(sun == -1)
    {
        i@Sun_rate += 1;
    }
}
```

Generate point characteristics (run over points)

```
// Set all 4 factors from 0 - 19

float dist_x = abs(@P.x - 75);
float dist_z = abs(@P.z - 56);
vector Max_pos = detail(0, "Max_height");

i@Factor_distance_Zomerhofstraat = int(20 - (20 * (@P.x / 150)));
i@Factor_distance_Vijverhofstraat = int(20 - (20 * (@P.z / 75)));
i@Factor_height = int(20 * (@P.y / Max_pos.y));
i@Factor_distance_courtyard = int(20 - (20 * (sqrt(dist_x*dist_x + dist_z*dist_z) / 90.0)));
```

Generate point scores for all groups (run over points)

```
// for each point, go over all groups.
for(int i = 0; i < npoints(1);i++)             // # of groups
{
    int Courtyard = point(1, "Courtyard", i);
    int Vijverhof = point(1, "Vijverhof", i);
    int Zomerhof  = point(1, "Zomerhof" , i);
    int Height    = point(1, "Height"   , i);

    int Score = i@Factor_distance_courtyard * Courtyard;
    Score += i@Factor_distance_Vijverhofstraat * Vijverhof;
    Score += i@Factor_distance_Zomerhofstraat * Zomerhof;
    Score += i@Factor_height * Height;                  //Combine point characteristics with Group factors

                                                        // Set the right score to the right rating.
    if( i == 0) i@Rate_Students_free_time = Score;
    if( i == 1) i@Rate_Work = Score;
    if( i == 2)
    {
        if (@P.y <= 4.8) i@Rate_Day_2_day = Score;      // Day_2_day may only grow up to 2 levels.
    }
    if( i == 3)
    {
        if (@P.y >= 4.8) i@Rate_Elderly = Score;        // Elderly may not grow on the ground level.
        if (@P.y < 4.8) i@Rate_Elderly = -100;
    }
    if( i == 4) i@Rate_Starters = Score;
    if( i == 5) i@Rate_Library = Score;
    if( i == 6) i@Rate_Students = Score;
    if( i == 7)
    {
        if(@P.y < 8) i@Rate_Cinema = Score;
        if(@P.y >= 8) i@Rate_Cinema = -100;
    }
}

i@floor = int((@P.y - 1.6)/3.2);
```

Generate total voxel score per group

```
int scaler = chi("../Voxel_cloud/Voxel/scale");
float size_x = chf("../Voxel_cloud/Voxel/sizex");
float size_y = chf("../Voxel_cloud/Voxel/sizey");
float size_z = chf("../Voxel_cloud/Voxel/sizez");
float radius = sqrt(pow(scaler*size_x, 2) + pow(scaler*size_y, 2) + pow(scaler*size_z, 2)) + 0.05;
int N_points[] = nearpoints(0, @P, radius);

for(int i = 0; i < len(N_points); i++)              // add up all scores from nearpoints
{
    if(point(0, 'floor', N_points[i]) == i@floor)
    {
        i@T_score_Students_free_time  += point(0, 'Rate_Students_free_time'    , N_points[i]);
        i@T_score_Work                += point(0, 'Rate_Work'                  , N_points[i]);
        i@T_score_Day_2_day           += point(0, 'Rate_Day_2_day'             , N_points[i]);
        i@T_score_Elderly             += point(0, 'Rate_Elderly'               , N_points[i]);
        i@T_score_Starters            += point(0, 'Rate_Starters'              , N_points[i]);
        i@T_score_Library             += point(0, 'Rate_Library'               , N_points[i]);
        i@T_score_Students            += point(0, 'Rate_Students'              , N_points[i]);
        i@T_score_Cinema              += point(0, 'Rate_Cinema'                , N_points[i]);
    }
}
```

Calculate average position per group (run over points)

```
vector Pos_list_Cinema[], Pos_list_Starters[], Pos_list_Students[], Pos_list_Elderly[], Pos_list_Student_free_time[], Pos_list_Day_2_
day[], Pos_list_Work[], Pos_list_Library[];
for(int i = 0; i < npoints(0); i++)
{
    vector Pos = point(0, 'P', i);
    if(inpointgroup(0, 'Starters', i) == 1)
    {
        append(Pos_list_Starters, Pos);
    }
    if(inpointgroup(0, 'Students', i) == 1)
    {
        append(Pos_list_Students, Pos);
    }
    if(inpointgroup(0, 'Elderly', i) == 1)
    {
```

```
        append(Pos_list_Elderly, Pos);
    }
    if(inpointgroup(0, 'Student_free_time', i) == 1)
    {
        append(Pos_list_Student_free_time, Pos);
    }
    if(inpointgroup(0, 'Day_2_day', i) == 1)
    {
        append(Pos_list_Day_2_day, Pos);
    }
    if(inpointgroup(0, 'Work', i) == 1)
    {
        append(Pos_list_Work, Pos);
    }
    if(inpointgroup(0, 'Library', i) == 1)
    {
        append(Pos_list_Library, Pos);
    }
    if(inpointgroup(0, 'Cinema', i) == 1)
    {
        append(Pos_list_Cinema, Pos);
    }
}

v@Avg_Starters = avg(Pos_list_Starters);
v@Avg_Students = avg(Pos_list_Students);
v@Avg_Elderly = avg(Pos_list_Elderly);
v@Avg_Student_free_time = avg(Pos_list_Student_free_time);
v@Avg_Day_2_day = avg(Pos_list_Day_2_day);
v@Avg_Work = avg(Pos_list_Work);
v@Avg_Library = avg(Pos_list_Library);
v@Avg_Cinema = avg(Pos_list_Cinema);

i@Max_Starters = point(1, 'Size', 4);
i@Max_Students = point(1, 'Size', 6);
i@Max_Elderly = point(1, 'Size', 3);
i@Max_Student_free_time = point(1, 'Size', 0);
i@Max_Day_2_day = point(1, 'Size', 2);
i@Max_Library = point(1, 'Size', 5);
i@Max_Work = point(1, 'Size', 1);
i@Max_Cinema = point(1, 'Size', 7);
```

Select grower (run over points in group Starters) (this is the same for all groups, therefore the other groups are not included.

```
setpointgroup(0, 'Available', @ptnum, 0, 'set');
int Group_size = npointsgroup(0, 'Starters');                        // # points in group starters
int Max_group_size = detail(0, 'Max_Starters');                            // size of group starters
if(Group_size <= Max_group_size)                                     // as long as the size is not reached, continue
{
    vector Pos_nghbr[];
    int Nghbrs[], Nghbrs_score[];
    for( int i = 0; i < 6; i++)
    {
        Pos_nghbr[i] = @P;
    }
    Pos_nghbr[0][0] -
= (chf("../../../../Create_point_cloud/Voxel_cloud/Voxel/sizex") * chi("../../../../Create_point_cloud/Voxel_cloud/Voxel/scale"));
    Pos_nghbr[1][0] += (chf("../../../../Create_point_cloud/Voxel_cloud/Voxel/sizex") * chi("../../../../Create_point_cloud/Voxel_clo
ud/Voxel/scale"));
    Pos_nghbr[2][1] -
= (chf("../../../../Create_point_cloud/Voxel_cloud/Voxel/sizey") * chi("../../../../Create_point_cloud/Voxel_cloud/Voxel/scale"));
    Pos_nghbr[3][1] += (chf("../../../../Create_point_cloud/Voxel_cloud/Voxel/sizey") * chi("../../../../Create_point_cloud/Voxel_clo
ud/Voxel/scale"));
    Pos_nghbr[4][2] -
= (chf("../../../../Create_point_cloud/Voxel_cloud/Voxel/sizez") * chi("../../../../Create_point_cloud/Voxel_cloud/Voxel/scale"));
    Pos_nghbr[5][2] += (chf("../../../../Create_point_cloud/Voxel_cloud/Voxel/sizez") * chi("../../../../Create_point_cloud/Voxel_clo
ud/Voxel/scale"));

    int Edge_point[] = nearpoints(0, '!Starters', @P, 4.5);

    if (len(Edge_point) < 4) setpointgroup(0, 'Edge_Starters', @ptnum, 1, 'set');

    for( int i = 0; i < 6; i++)
    {
         int Near_point = nearpoint(0, 'Available', Pos_nghbr[i], 0.1);
         if( Near_point != -1)
         {
            append(Nghbrs, Near_point);
            int Score = point(0, 'Rate_Starters', Near_point);
            append(Nghbrs_score, Score);
            if(i == 2 || i == 3)
            {
                Nghbrs_score[i] = int(0.7 * float(Nghbrs_score[i]));
            }
```

```
        }
    }

    if(len(Nghbrs) > 1)                                                 // sort list
    {
        for( int i = len(Nghbrs) - 1; i >= 0 ; i--)
        {
            for( int j = 0; j < len(Nghbrs) - 1; j++)
            {
                if (Nghbrs_score[j] < Nghbrs_score[j+1])
                {
                    int temp = Nghbrs[j];
                    int temp_s = Nghbrs_score[j];
                    Nghbrs[j] = Nghbrs[j+1];
                    Nghbrs_score[j] = Nghbrs_score[j+1];
                    Nghbrs[j+1] = temp;
                    Nghbrs_score[j+1] = temp_s;
                }
            }
        }
    }
    if(len(Nghbrs) > 0)                                                 // if the point has available neighbours
    {
        setpointgroup(0, 'Temp_Starters', Nghbrs[0], 1, 'set');
    }
}
```

Grow grower (detail; run only once)

```
int rate = chi("../../../../controller/Grow_rate");
int Group_size = npointsgroup(0, 'Starters');                          // # points in group starters
int Max_group_size = detail(0, 'Max_Starters');

int Close_Work = 0, Close_Day_2_day = 0, Close_Self = 0, Raise = chi("../../../Attraction_strength"), Raise_self = chi("../../../Attraction_self");
float dist_Work = 1000.0, dist_Day_2_day = 1000.0, dist_Self = 1000.0;

if((Max_group_size - Group_size) > 0)
{
    int Score_list[];
    int ID_list[] = expandpointgroup(0, 'Temp_Starters');
    for( int i = 0; i < len(ID_list); i++)
    {
        int Score = point(0, 'Rate_Starters', ID_list[i]);
        append(Score_list, Score);
    }

    for(int j = 0; j < len(ID_list); j++)
    {
        vector Pos_temp = point(0, 'P', ID_list[j]);
        float Temp_dist_self = distance(Pos_temp, detail(0, 'Avg_Starters'));
        float Temp_dist_Work = distance(Pos_temp, detail(0, 'Avg_Work'));
        float Temp_dist_Day_2_day = distance(Pos_temp, detail(0, 'Avg_Day_2_day'));
        if (Temp_dist_self < dist_Self)                            // If the distance is smaller, update closest distance and ID in Nghbrs
        {
            dist_Self = Temp_dist_self;
            Close_Self = j;
        }
        if (Temp_dist_Work < dist_Work)                            // If the distance is smaller, update closest distance and ID in Nghbrs
        {
            dist_Work = Temp_dist_Work;
            Close_Work = j;
        }
        if (Temp_dist_Day_2_day < dist_Day_2_day)                          // If the distance is smaller, update closest distance and ID in Nghbrs
        {
            dist_Day_2_day = Temp_dist_Day_2_day;
```

```
            Close_Day_2_day = j;
        }

    }
    Score_list[Close_Work] += Raise;
    Score_list[Close_Day_2_day] += Raise;
    Score_list[Close_Self] += Raise_self;


    if(len(Score_list) > 1)                                                 // bubble sort list
    {
        for( int i = len(Score_list) - 1; i >= 0 ; i--)
        {
            for( int j = 0; j < len(Score_list) - 1; j++)
            {
                if (Score_list[j] < Score_list[j + 1])
                {
                    int temp = ID_list[j];
                    int temp_s = Score_list[j];
                    ID_list[j] = ID_list[j+1];
                    Score_list[j] = Score_list[j+1];
                    ID_list[j+1] = temp;
                    Score_list[j+1] = temp_s;
                }
            }
        }
    }

    if(len(ID_list) < rate)                 rate = len(ID_list);
    if(Max_group_size - Group_size < rate)  rate = Max_group_size - Group_size;

    for(int k = 0; k < rate; k++)
    {
        setpointgroup(0, 'Temp_Starters', ID_list[k], 0, 'set');
        setpointgroup(0, 'Starters', ID_list[k], 1, 'set');
        setpointgroup(0, 'Available', ID_list[k], 0, 'set');
    }
}
if((Max_group_size - Group_size) > 1)
{
    int E_Score_list[];
    int E_ID_list[] = expandpointgroup(0, 'Edge_Starters');
```

```
    for( int i = 0; i < len(E_ID_list); i++)
    {
        int Score = point(0, 'Rate_Starters', E_ID_list[i]);
        append(E_Score_list, Score);
    }

    if(len(E_Score_list) > 1)                                   // bubble sort list from high to low
    {
        for( int i = len(E_Score_list) - 1; i >= 0 ; i--)
        {
            for( int j = 0; j < len(E_Score_list) - 1; j++)
            {
                if (E_Score_list[j] > E_Score_list[j + 1])
                {
                    int temp = E_ID_list[j];
                    int temp_s = E_Score_list[j];
                    E_ID_list[j] = E_ID_list[j+1];
                    E_Score_list[j] = E_Score_list[j+1];
                    E_ID_list[j+1] = temp;
                    E_Score_list[j+1] = temp_s;
                }
                   }
               }
           }
           setpointgroup(0, 'Starters', E_ID_list[0], 0, 'set');
           setpointgroup(0, 'Available', E_ID_list[0], 1, 'set');
       }
```

Create lines between all points for path finding(run over points)

```
setpointgroup(0, 'Available', @ptnum, 0, 'set');

int Neighbours[] = nearpoints(0, 'Available', @P, 4.5);

for (int i = 0; i < len(Neighbours); i++)
{
    int nprim = addprim(0, "polyline", @ptnum, Neighbours[i]);
}
```

Find closest point to function centers.

```
i@center_Starters = nearpoint(0, v@Avg_Starters);
i@center_Students = nearpoint(0, v@Avg_Students);
i@center_Elderly = nearpoint(0, v@Avg_Elderly);
i@center_Work = nearpoint(0, v@Avg_Work);
i@center_Student_free_time = nearpoint(0, v@Avg_Student_free_time);
i@center_Library = nearpoint(0, v@Avg_Library);
i@center_Cinema = nearpoint(0, v@Avg_Cinema);
i@center_Day_2_day = nearpoint(0, v@Avg_Day_2_day);

setpointgroup(0, 'Elderly_s', i@center_Elderly, 1, 'set');
setpointgroup(0, 'Elderly_e', i@center_Library, 1, 'set');

setpointgroup(0, 'Work_s', i@center_Work, 1, 'set');
setpointgroup(0, 'Work_e', i@center_Starters, 1, 'set');

setpointgroup(0, 'Day_2_day_s', i@center_Day_2_day, 1, 'set');
setpointgroup(0, 'Day_2_day_e', i@center_Starters, 1, 'set');
setpointgroup(0, 'Day_2_day_e', i@center_Students, 1, 'set');
setpointgroup(0, 'Day_2_day_e', i@center_Elderly, 1, 'set');
```

Define stairs (run over points)

```
int ID_Highest_point;
vector Highest_P = {0,0,0};
vector Stair_2 = {82.5, -1.6, 10.0};
vector Stair_3 = {14.0, -1.6, 59.0};
vector Stair_4 = {130.0, -1.6, 29.0};
vector Stair_5 = {6.0, -1.6, 6.0};

for(int i = 0; i < npoints(0); i++)
{
    vector Pos = point(0, 'P', i);
    if (Pos.y > Highest_P.y)
    {
        Highest_P = Pos;
        ID_Highest_point = i;
    }
}
setpointgroup(0, 'Stair_1', ID_Highest_point, 1, 'set');

for(int i = 0; i < 20; i++)
{
    Highest_P.y -= 3.2;
    int ID = nearpoint(0, 'not_Connected', Highest_P, 1);
    if(ID != -1)
    {
        setpointgroup(0, 'Stair_1',     ID, 1, 'set');
    }
}

for(int i = 0; i < 20; i++)
{
    Stair_2.y += 3.2;
    int ID = nearpoint(0, 'not_Connected', Stair_2, 1);
    if(ID != -1)
    {
        setpointgroup(0, 'Stair_2', ID, 1, 'set');
    }
}

for(int i = 0; i < 20; i++)
{
```

```
    Stair_3.y += 3.2;
    int ID = nearpoint(0, 'not_Connected', Stair_3, 1);
    if(ID != -1)
    {
        setpointgroup(0, 'Stair_3', ID, 1, 'set');
    }
}

for(int i = 0; i < 20; i++)
{
    Stair_4.y += 3.2;
    int ID = nearpoint(0, 'not_Connected', Stair_4, 1);
    if(ID != -1)
    {
        setpointgroup(0, 'Stair_4', ID, 1, 'set');
    }
}

for(int i = 0; i < 20; i++)
{
    Stair_5.y += 3.2;
    int ID = nearpoint(0, 'not_Connected', Stair_5, 1);
    if(ID != -1)
    {
        setpointgroup(0, 'Stair_5', ID, 1, 'set');
    }
}

int acces = chi("../../../controller/Accessibility");

int ID_list[] = expandpointgroup(0, 'not_Connected');
for(int i = 0; i < len(ID_list); i++)
{
    if(ID_list[i] % acces == 0)
    {
        setpointgroup(0, 'Acces', ID_list[i], 1, 'set');
    }
}
```