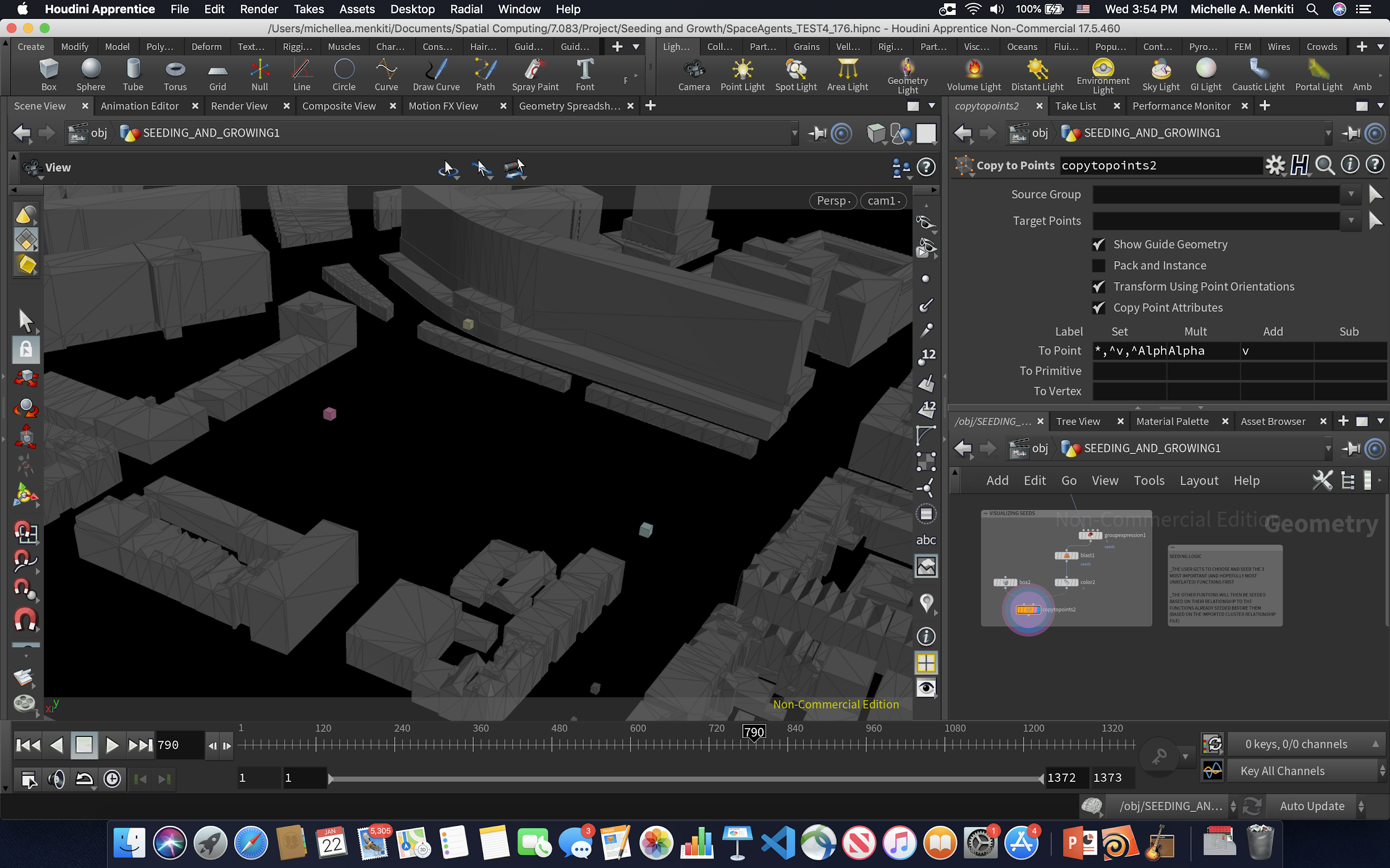
SEED PLACEMENT

//Date: 24-01-2019

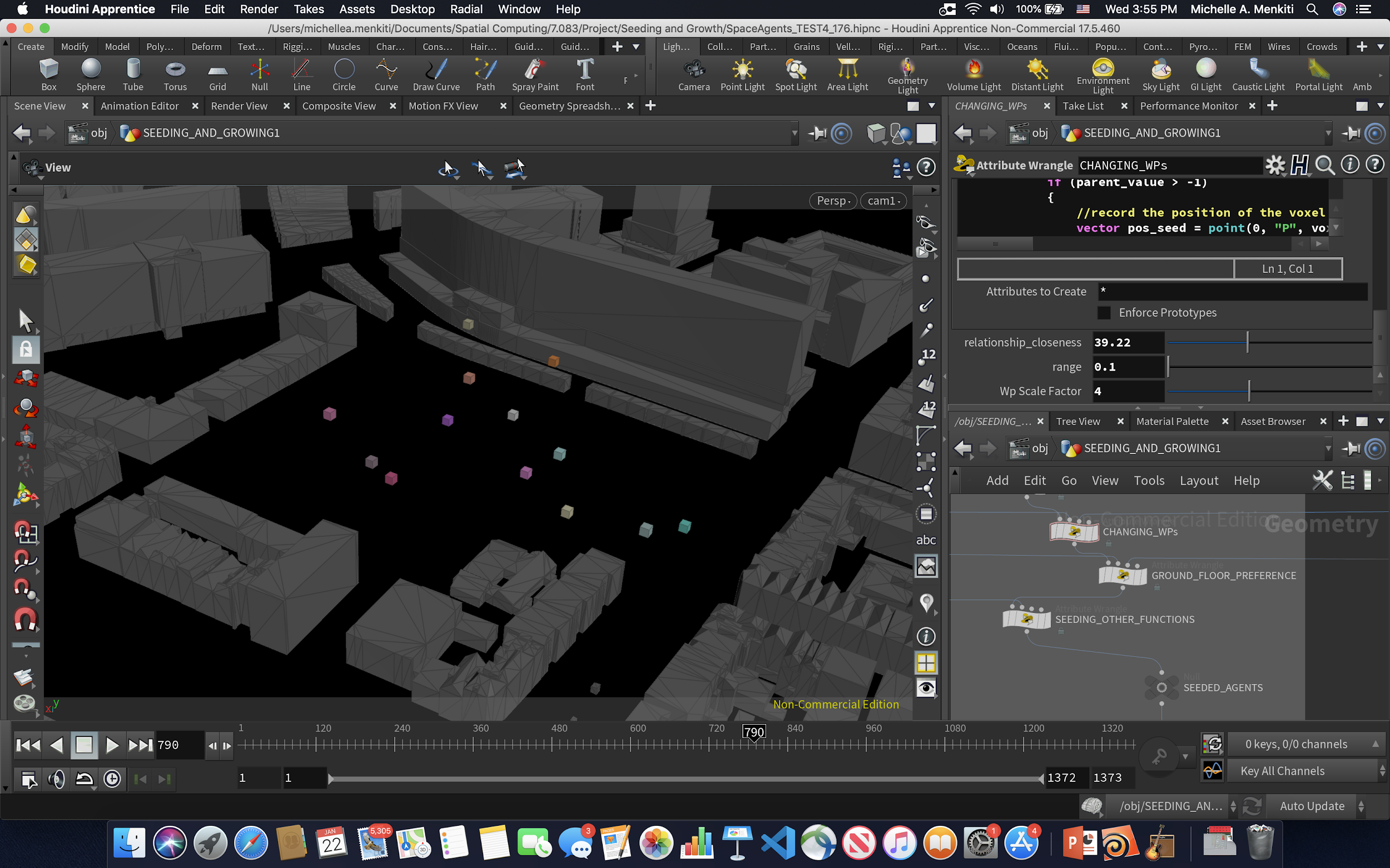
//Authors: Jolt Wiersma, Michelle A. Menkiti, Arthur Masure

//Adapted from work by: Shervin Azadi, Dr.ir. Pirouz Nourian, Hans Hoogenboom

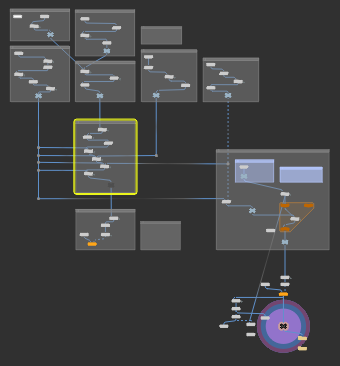
//Purpose: Determines the best voxel location for each function starting with three most important functions determined by the designer



STEP A: PLACEMENT OF 3 MOST IMPORTANT SEEDS



STEP D: PLACEMENT OF REMAINING SEEDS AFTER MANIPULATION OF WP’S DUE TO THEIR DESIRED RELATIONSHIP RADIUS TO INITIAL SEEDS AND GROWTH LEVEL PREFERENCE (STEP B,C)



//A: SEED PLACEMENT OF 3 MOST IMPORTANT FUNCTIONS (CLUSTER START)

//number of functions

//number of points

int num\_func = npoints(1);

int num\_points = npoints(0);

//initialize lists

int func\_seed\_pos[];

int occupied\_voxels[];

//loop through first 3 to be placed (most important) functions

for (int curr\_func = 0; curr\_func < 3; curr\_func++)

{

//name of weighted function

string wpName = "wp" + itoa(curr\_func);

float max\_weights[];

int max\_id[];

//going through each voxel

for (int curr\_point = 0; curr\_point < num\_points; curr\_point++)

{

//weight of voxel, add to list

float curr\_weight = point(0,wpName,curr\_point);

append(max\_weights,curr\_weight);

//add point id to list

append(max\_id,curr\_point);

}

//sorting list of weights

int sorted\_weights\_indicies[] = reverse(argsort(max\_weights));

int sorted\_seed\_points[] = reorder(max\_id,sorted\_weights\_indicies);

//initializing index and truth value

int index = 0;

int no\_result = 0;

//creating while loop

while ( (no\_result == 0) && (index < len(sorted\_seed\_points)) )

{

//getting the point id

int point\_id = sorted\_seed\_points[index];

//if point is occupied

if ( (find(occupied\_voxels,point\_id)) > -1)

{

//increasing index, going to next largest point

index += 1;

}

//if point is not occupied

else

{

//adding to list of seed positions

append(func\_seed\_pos,point\_id);

//add to list of occupied voxels

append(occupied\_voxels,point\_id);

//endng while loop, setting no result to 1

no\_result = 1;

}

}

}

//loop through first 3 to be placed (most important) functions

for (int curr\_func = 0; curr\_func < 3; curr\_func++)

{

int parent = func\_seed\_pos[curr\_func];

setpointattrib(0, "parent",parent,curr\_func, "set");

setpointgroup(0,"occupied",parent,1,"set");

}

//B: PER REMAINING FUNCTION, DETERMINE DESIRED RELATIONSHIP RADIUS TO INITIALLY SEEDED FUNCTION AND INCREASE WP OF RADIUS POINTS FOR FUNCTION ACCORDINGLY

//number of functions

//number of points

int num\_points = npoints(0);

int num\_functions = npoints(1);

//user inputs

float rel\_distance = chf("relationship\_closeness");

float range = chf("range");

int wp\_scale = chi("wp\_scale\_factor");

//creating a list of the attributes to go over

string parameterList[] = array("LivingUnits","StartUpOffices","Arcade","Library","CoworkingSpaces","FabLabsWorkshops","CommunityCentre","EntranceHall","IndoorMarket","Gym","Restaurant","CafePub","Cinema");

//go through each function you want to seed

for(int func=0; func < num\_functions; func++)

{

//making sure its not the most important function

if (func > 2)

{

//going through each voxel

for (int voxel = 0; voxel < num\_points; voxel++)

{

//if the voxel is a seed

int parent\_value = point(0, "parent", voxel);

if (parent\_value > -1)

{

//record the position of the voxel

vector pos\_seed = point(0, "P", voxel);

//getting function name of the seed from function ID

string func\_Name = parameterList[parent\_value];

//getting relationship between seed and new function

int relation = point(2, func\_Name, func);

//if the relationship exists

if (relation > 0)

{

//iterate through all points again to change their WP value

for (int point = 0; point < num\_points; point++)

{

//record the position of the point

vector pos = point(0, "P", point);

//determine the distance between that point and the seed

float distance = distance(pos\_seed, pos);

//if the distance is within a certain range according to relationship

if (((rel\_distance\*relation - range) < distance) && (distance < (rel\_distance\*relation + range)))

{

//determine the wp name

string wp\_name = "wp" + itoa(func);

//change the wp of the chosen voxels

float old\_wp = point(0, wp\_name, point);

float wp = old\_wp\*wp\_scale;//pow(10,0.8);

//setting new wp

setpointattrib(0,wp\_name,point,wp,"set");

}

}

}

}

}

}

}

//C: PER REMAINING FUNCTION, DETERMINE DESIRED GROUND LEVEL PLACEMENT AND INCREASE WP OF FLOOR LEVEL POINTS FOR FUNCTION ACCORDINGLY

//number of functions

//number of points

int num\_points = npoints(0);

int num\_functions = npoints(1);

//FOR EACH FUNCTION

for (int func = 0; func < num\_functions; func++)

{

//READ THE GROUND PREFERENCE

int ground\_pref = point(2,"CloseGround",func);

//ITERATE OVER ALL VOXELS

for(int point = 0; point < num\_points; point++)

{

//CHECK Y VALUE

vector pos = point(0,"P",point);

float y\_value = pos[1];

//GET WP VALUE

//get the correct name

string wp\_name = "wp" + itoa(func);

float wp = point(0,wp\_name,point);

//if the floor placement matters

if (ground\_pref < 3.0)

{

//ground floor

if (ground\_pref == 1)

{

//if on the ground floor

if (y\_value < 6.0)

{

float new\_wp = wp\*chf("floor\_preference\_wp");

//CHANGE WP VALUES TO BE EXTRA HIGH

setpointattrib(0,wp\_name,point,new\_wp,"set");

}

}

//middle floor

if (ground\_pref == 2)

{

//if close to the ground

if (6.0 < y\_value && y\_value < 18.0)

{

float new\_wp = wp\*chf("floor\_preference\_wp");

//CHANGE WP VALUES TO BE EXTRA HIGH

setpointattrib(0,wp\_name,point,new\_wp,"set");

}

}

}

}

}

//D: SEED PLACEMENT OF REMAININGFUNCTIONS (BASED ON NEW WP’S)

//number of functions

//number of points

int num\_func = npoints(1);

int num\_points = npoints(0);

//initialize lists

int func\_seed\_pos[];

int occupied\_voxels[];

//loop through all functions

for (int curr\_func = 0; curr\_func < num\_func; curr\_func++)

{

//name of weighted function

string wpName = "wp" + itoa(curr\_func);

float max\_weights[];

int max\_id[];

//going through each voxel

for (int curr\_point = 0; curr\_point < num\_points; curr\_point++)

{

//weight of voxel, add to list

float curr\_weight = point(0,wpName,curr\_point);

append(max\_weights,curr\_weight);

//add point id to list

append(max\_id,curr\_point);

}

//sorting list of weights

int sorted\_weights\_indicies[] = reverse(argsort(max\_weights));

int sorted\_seed\_points[] = reorder(max\_id,sorted\_weights\_indicies);

//initializing index and truth value

int index = 0;

int no\_result = 0;

//creating while loop

while ( (no\_result == 0) && (index < len(sorted\_seed\_points)) )

{

//getting the point id

int point\_id = sorted\_seed\_points[index];

//if point is occupied

if ( (find(occupied\_voxels,point\_id)) > -1)

{

//increasing index, going to next largest point

index += 1;

}

//if point is not occupied

else

{

//adding to list of seed positions

append(func\_seed\_pos,point\_id);

//add to list of occupied voxels

append(occupied\_voxels,point\_id);

//endng while loop, setting no result to 1

no\_result = 1;

}

}

}

//loop through all functions

for (int curr\_func = 0; curr\_func < num\_func; curr\_func++)

{

if (curr\_func > 2)

{

int parent = func\_seed\_pos[curr\_func];

setpointattrib(0, "parent",parent,curr\_func,"set");

setpointgroup(0,"occupied",parent,1,"set");

}

}