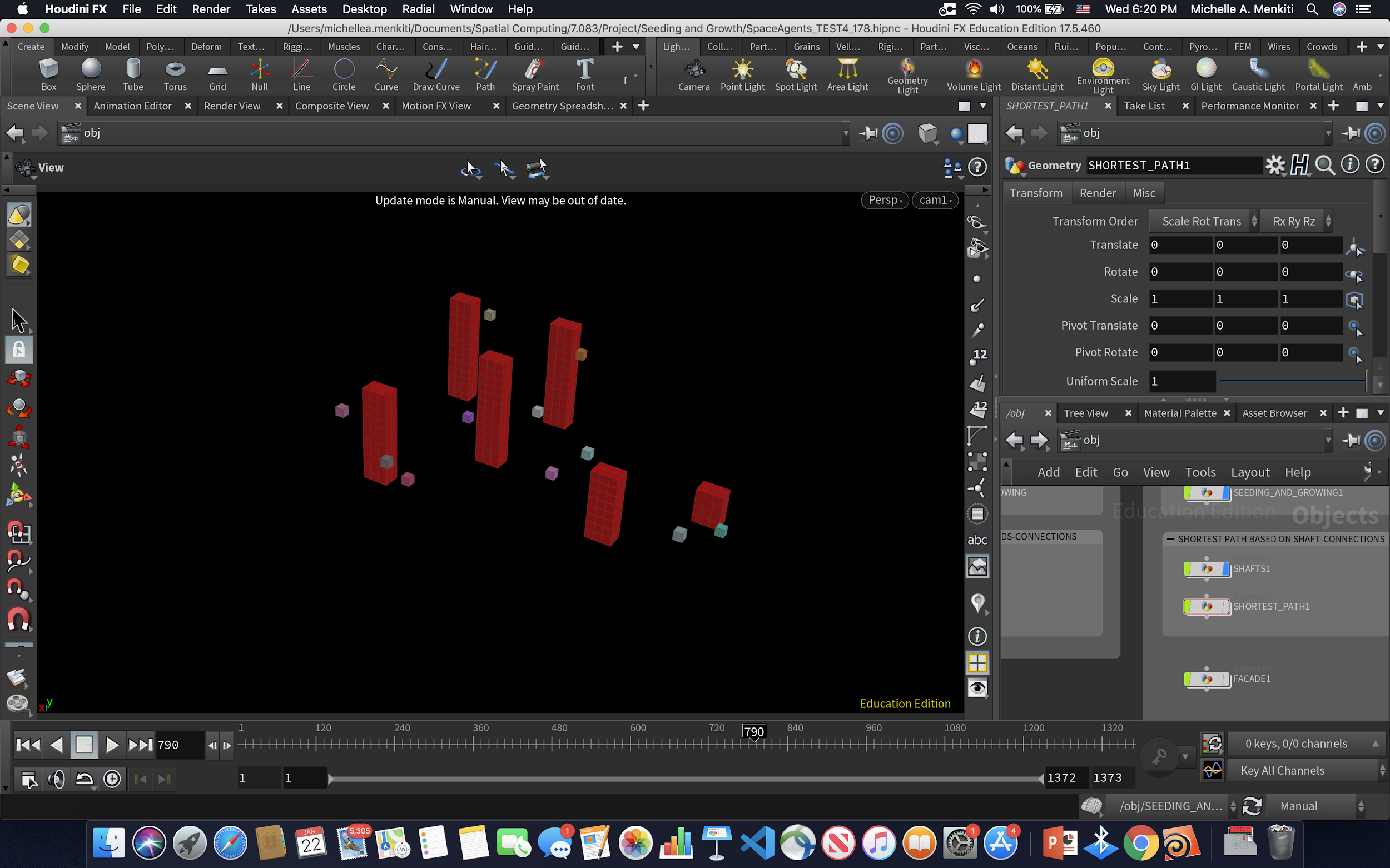
SHAFT PLACEMENT

//Date: 24-01-2019

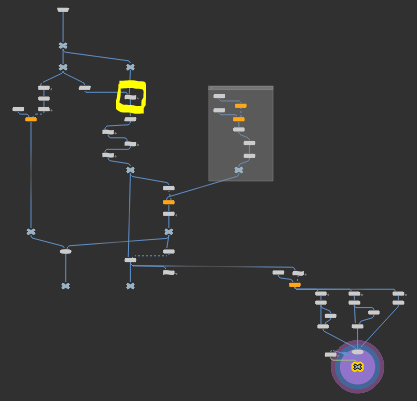
//Authors: Jolt Wiersma, Michelle A. Menkiti, Arthur Masure

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

//Purpose: To find points within the envelope that, based on sliders and position, are ideal for the placement of lift/stairwell shafts



SHAFTS SHOWN IN RELATION TO ALL SEEDED FUNCTIONS



//SLIDER 'MAX. DIST. SHAFT-FUNCTION' -> MAXIMUM ALLOWABLE DISTANCE OF SHAFT FROM FUNCTIONS

float max\_dist\_s\_f = chf("max\_dist\_s\_f");

//SLIDER 'MIN. DIST. SHAFT-SHAFT' -> MINIMUM ALLOWABLE DISTANCE OF SHAFT FROM OTHER SHAFT

float min\_dist\_s\_s = chf("min\_dist\_s\_s");

//SLIDER 'SHAFT COUNT' -> MAXIMUM AMOUNT OF SHAFTS TO CREATE

int shaft\_count\_slider = chi("shaft\_count");

//GET X-VALUE OF ALL INITIAL SEEDS (STORE IN LIST -> SEED\_POINTS\_X)

//GET Z-VALUE OF ALL INITIAL SEEDS (STORE IN LIST -> SEED\_POINTS\_Z)

float seed\_points\_x[];

float seed\_points\_z[];

//PREPARE LIST OF SEED POSITIONS

vector seed\_pos[];

//CREATE ITERATOR LIMITERS

int num\_points\_0 = npoints(0);

int num\_points\_1 = npoints(1);

//iterating through all points/voxels

//getting position, x, and z values of all seeds

//adding them to respective list

for (int point\_id = 0; point\_id < num\_points\_1; point\_id++)

{

int seed = point(1,"parent",point\_id); //seeds are presented in parent attribute

if (seed != -1) // all seeds have value other than -1

{

vector pos = point(1,"P",point\_id);

float x\_value = pos[0];

float z\_value = pos[2];

append(seed\_points\_x,x\_value);

append(seed\_points\_z,z\_value);

append(seed\_pos, pos);

}

}

//SET SHAFT\_COUNT = 0 (A VARIABLE TO BE CHECKED WHEN IT GETS 'FULL' -> ACCORDING TO THE SLIDER 'SHAFT COUNT')

int shaft\_count = 0;

//SET SHAFT\_ORIGIN\_LIST (AN EMPTY LIST OFF ALL SHAFTS CONTAINING THE POSITION VALUE OF ITS ORIGIN-> TO BE FILLED IF THE BELOW CODE SUCCEEDS IN PLACING A SHAFT)

int shaft\_origins[];

//GET LOWEST POINTS OF THE ENVELOPE (SHAFT WILL ORIGINATE ON GROUND FLOOR)

int ground\_floor[];

for (int point\_id = 0; point\_id < num\_points\_0; point\_id++)

{

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

float y\_value = pos[1];

if (abs(y\_value) < 0.001) // the lowest y-value equals 0.0

{

for (int seed\_id = 0; seed\_id < len(seed\_pos); seed\_id++)

{

vector this\_seed\_pos = seed\_pos[seed\_id];

this\_seed\_pos \*= set(1,0,1);

float dist = distance(this\_seed\_pos, pos);

if (dist < max\_dist\_s\_f && dist > 5)

{

append(ground\_floor,point\_id);

}

}

}

}

//FOR EVERY LOWEST POINT OF THE ENVELOPE (SHAFT WILL ORIGINATE ON GROUND FLOOR)

for (int ground\_point = 0; ground\_point < len(ground\_floor); ground\_point++) // for ground point on ground floor

{

//IF THE DESIRED SHAFT COUNT HAS BEEN EXCEEDED, BREAK THE LOOP (FINISH SHAFT PLACEMENT)

if (shaft\_count >= shaft\_count\_slider) break;

//set a toggle for getting out of coming for loops if other if statements are false

int toggle = 0;

//get the x and z positions of the possible shaft origin point

vector pos = point(0,"P",ground\_floor[ground\_point]);

float x\_value = pos[0];

float z\_value = pos[2];

float shaft\_points\_x = x\_value;

float shaft\_points\_z = z\_value;

//IF 'SHAFT\_COUNT' IS EQUAL TO 0 (FOR PLACING FIRST SHAFT)

if (shaft\_count == 0)

{

//APPEND ITS ORIGIN POINT INTO THE LIST 'SHAFT\_ORIGIN\_LIST'

append(shaft\_origins,ground\_floor[ground\_point]);

//PLACE THE SHAFT AT THAT ORIGIN POINT BY SETTING AN ATTRIBUTE

setpointattrib(0,"shaft\_origins",ground\_floor[ground\_point],1, "set"); //1 means center of shaft

setpointgroup(0, "shaft\_base", ground\_floor[ground\_point], 1, "set");

//INCREASE 'SHAFT\_COUNT' BY 1

shaft\_count++;

//GO TO NEXT LOWEST POINT OF THE ENVELOPE (PLACING A SHAFT HERE WILL PUT IT TOO FAR FROM ANY SEED, ACCORDING TO THE SLIDER

toggle = 1; // set toggle to 1

}

//check toggle

if (toggle) // toggle (1) will equal true

{

continue; // continue the loop: go to next iteration of the loop this is nested in

}

//IF 'SHAFT\_COUNT' IS GREATER THAN 0 (CHECK FOR SURROUNDING SHAFTS)

if (shaft\_count > 0)

{

//FOR SHAFTS IN 'SHAFT\_ORIGIN\_LIST'

for (int i = 0; i < len(shaft\_origins); i++)

{

//getting x and z value of point

vector shaft\_origin\_pos = point(0,"P",shaft\_origins[i]);

float shaft\_origin\_x = shaft\_origin\_pos[0];

float shaft\_origin\_z = shaft\_origin\_pos[2];

//calculate distance between this point and other shaft points

float dist = distance(pos, shaft\_origin\_pos);

//IF THE DISTANCE BETWEEN THAT ORIGIN POINT AND THE POINT OF ANY EXISTING SHAFT IS LESS THAN 'MIN. DIST. SHAFT-SHAFT'

if ( dist <= min\_dist\_s\_s )

{

//GO TO NEXT LOWEST POINT OF THE ENVELOPE (PLACING A SHAFT HERE WILL PUT IT TOO CLOSE TO ANOTHER SHAFT, ACCORDING TO THE SLIDER)

toggle = 1; // set toggle to 1

break; // break the loop: skip all next iterations (of the loop this is nested in)

}

}

//check toggle

if (toggle) // toggle (1) will equal true

{

continue; // continue the loop: go to next iterations (of the loop this is nested in)

}

//APPEND ITS ORIGIN POINT INTO THE LIST 'SHAFT\_ORIGIN\_LIST'

append(shaft\_origins, ground\_floor[ground\_point]);

//PLACE THE SHAFT AT THAT ORIGIN POINT BY SETTING AN ATTRIBUTE

setpointattrib(0,"shaft\_origins",ground\_floor[ground\_point],1,"set"); //1 means center of shaft

setpointgroup(0, "shaft\_base", ground\_floor[ground\_point], 1, "set");

//INCREASE 'SHAFT\_COUNT' BY 1

shaft\_count++;

}

}

//FOR EACH POINT ON THE GROUND ADD TO A GROUP ATTRIBUTE

foreach (int ground\_point; ground\_floor) setpointgroup(0, "ground", ground\_point, 1, "set");

//FOR EACH POINT IN THE ENVELOPE

for (int i = 0; i < num\_points\_0; i++)

{

//get a position

vector pos\_2 = point(0,"P",i);

//project to ground plane

pos\_2 \*= set(1,0,1);

//for each shaft

foreach (int shaft; shaft\_origins)

{

//get a position

vector shaft\_pos = point(0, "P", shaft);

//if the shaft position is at the same spot as the projected envelope position

if ( distance(pos\_2,shaft\_pos) < 0.001 )

{

//set those envelope positions to occupied

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

break;

}

}

}