SKILL COURSE REPORT

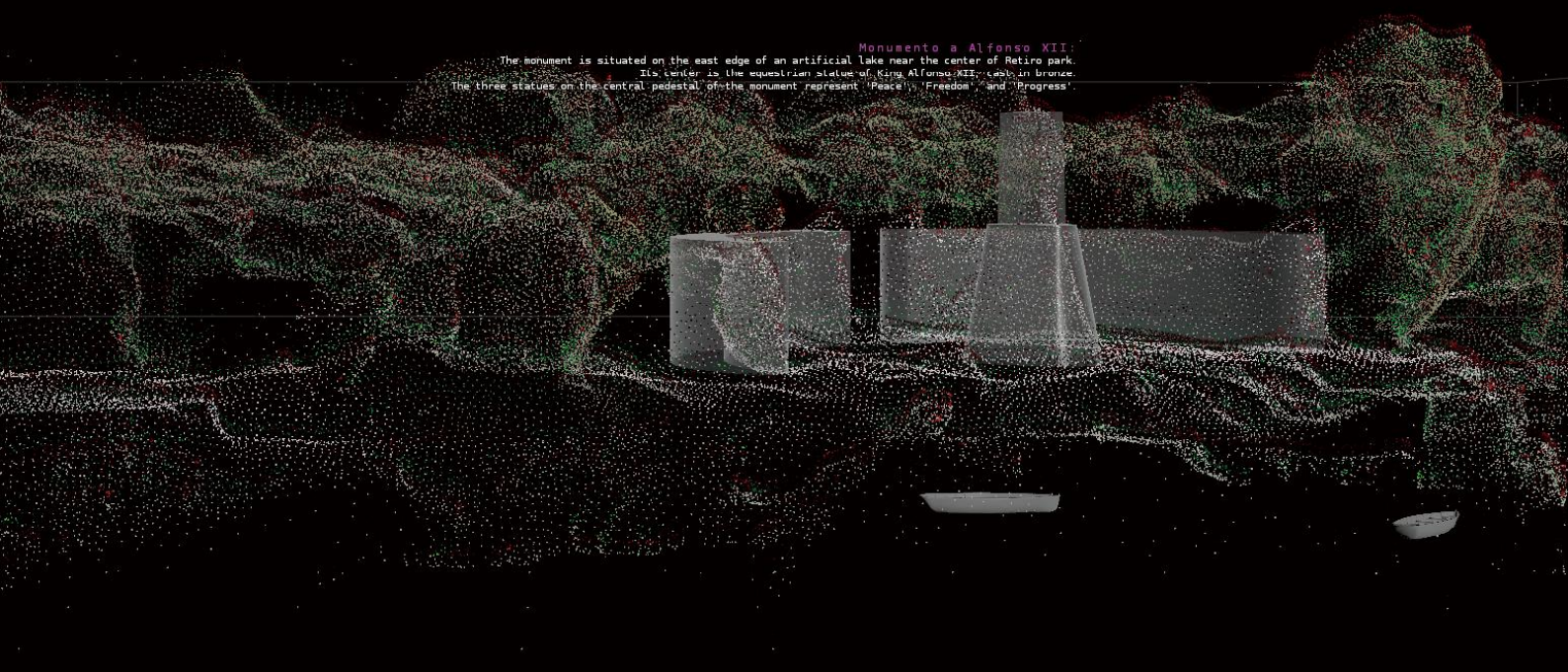
RC11_22161736

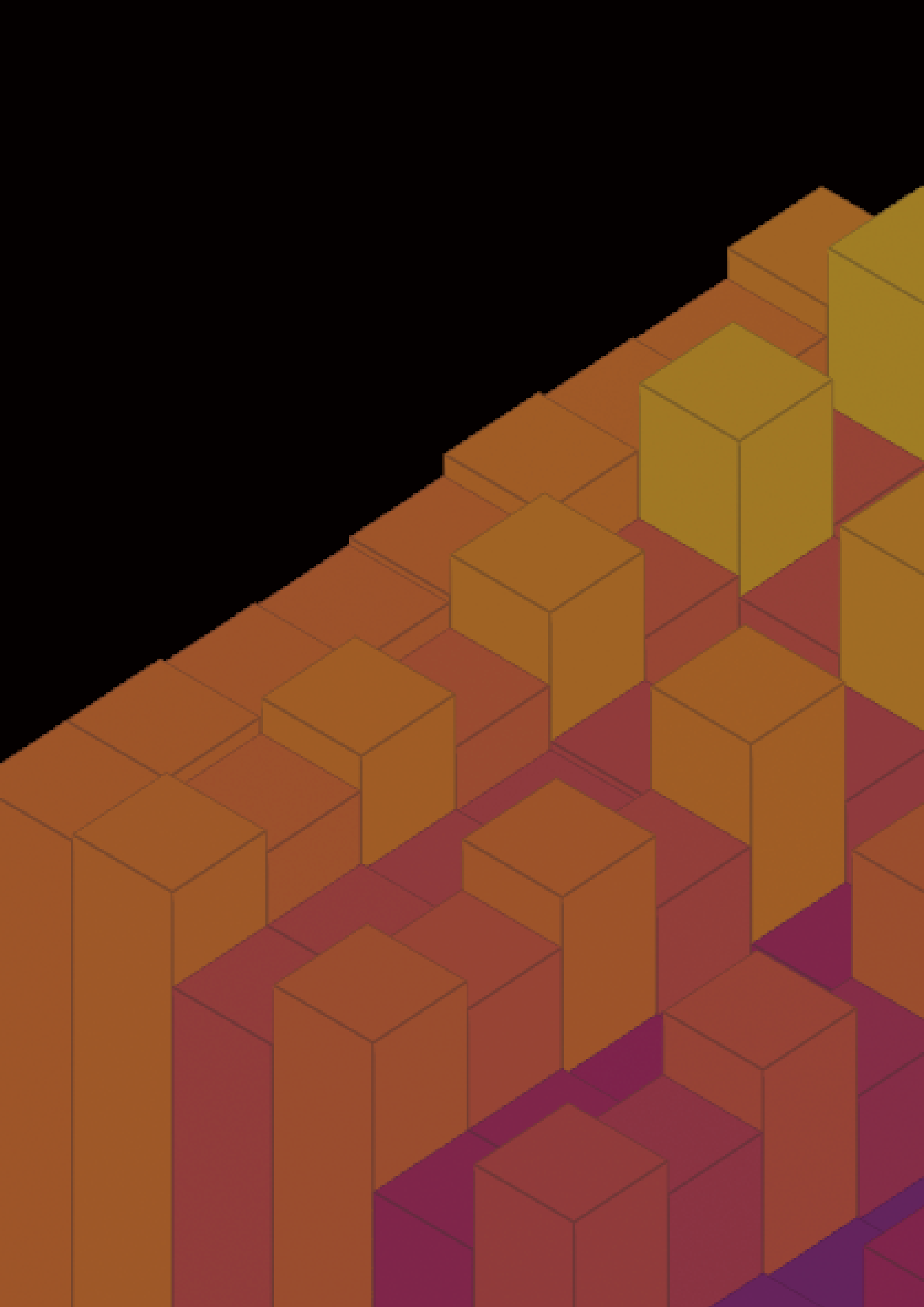
BARTLETT 23/24

Contents

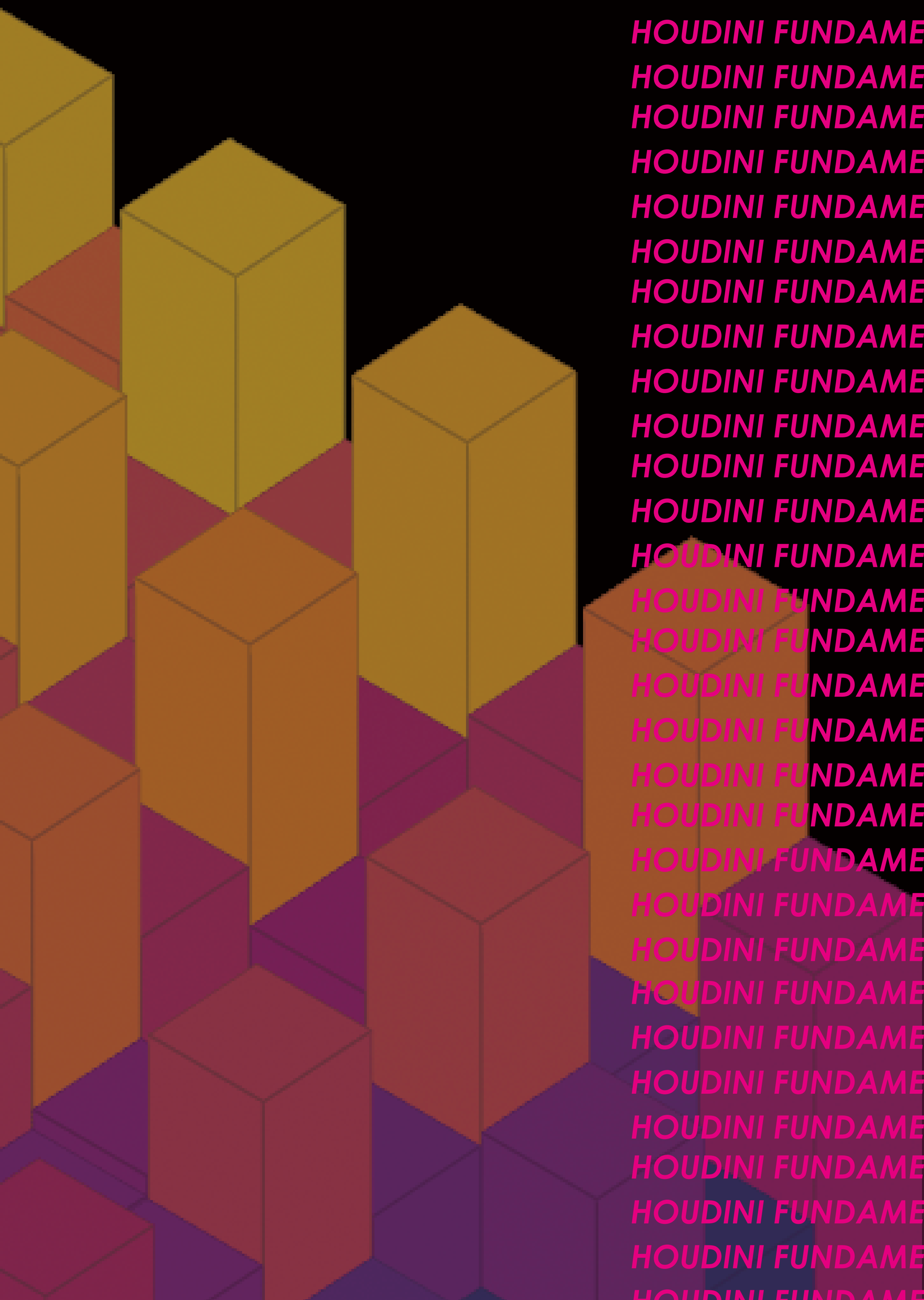
HOUDINI ASSESSMENT

RC11_22161736_HoudiniReport





HOUDINI 1

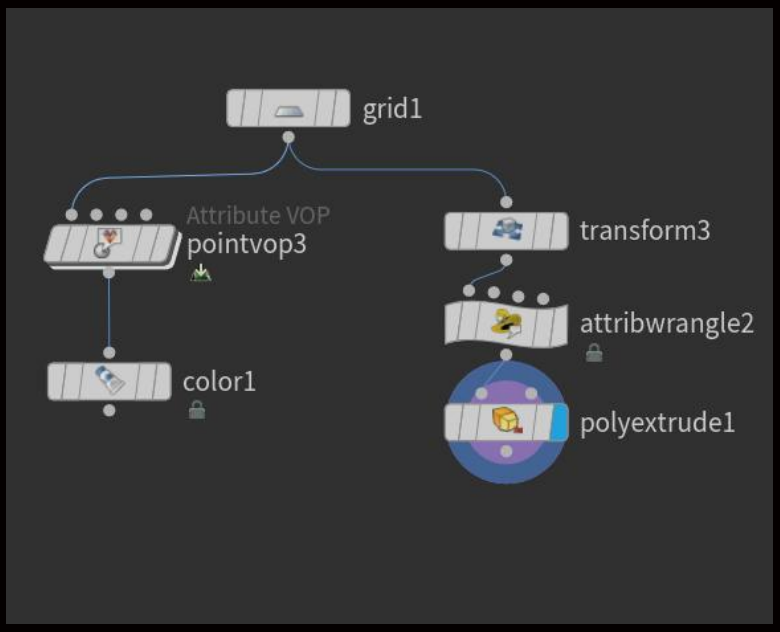


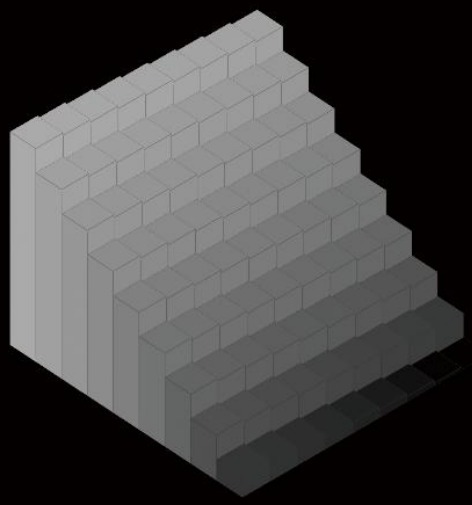
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS
HOUDINI FUNDAMENTALS

In this assessment, I completed all the exercises and understood whole functional modules in class. Afterwards, based on what was taught, I made modifications to two of exmaples.

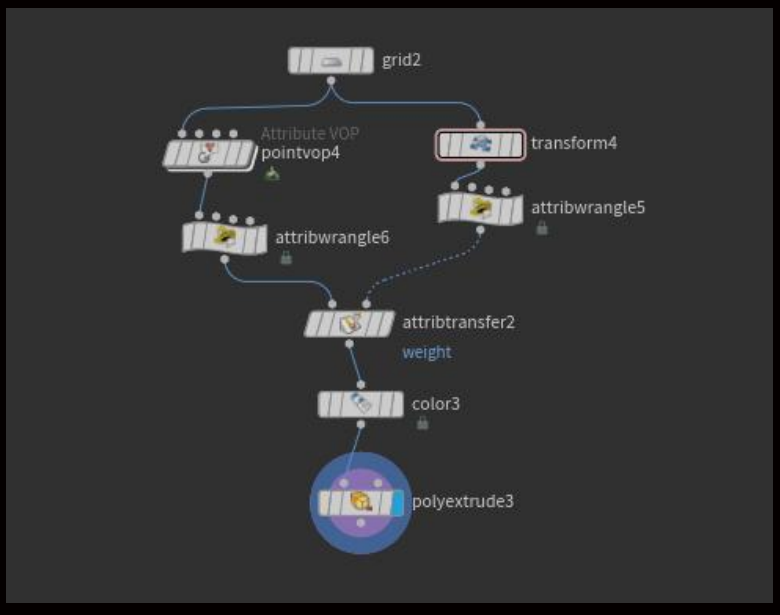
1.1 Generate a staircase, steps could be controlled

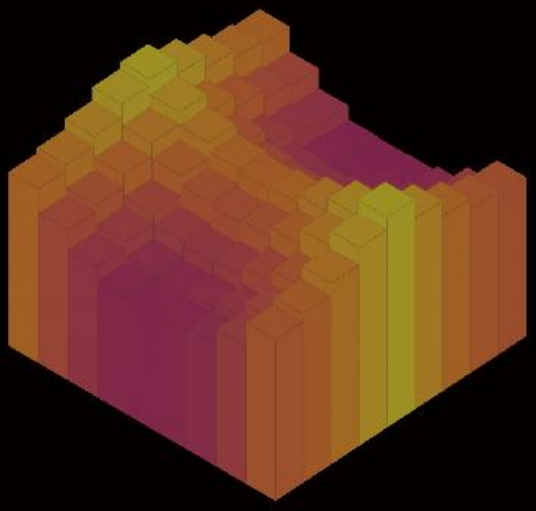
Original exmaple:





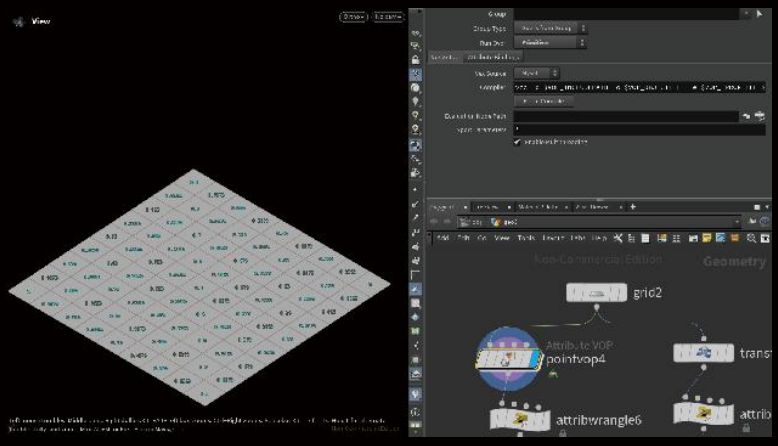
Altered result:





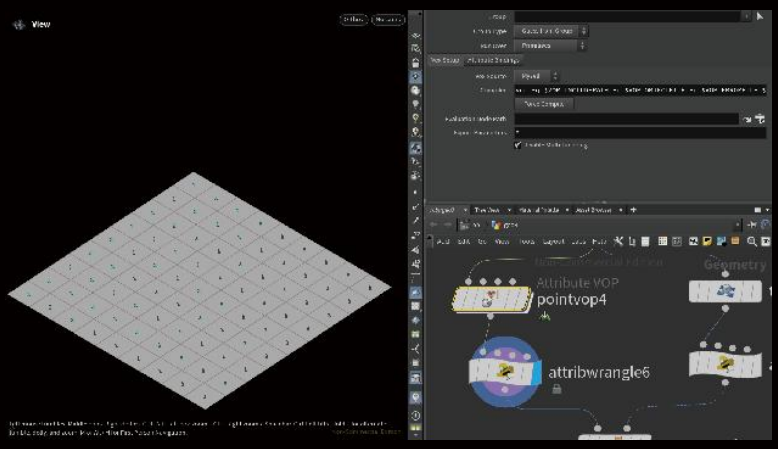
1.1.1 Grid SOP, Pointvop SOP

*The Grid is of primitive type: polygon. One of the basic geometric datatypes that contains points,primitives of type polygon. Here the grid I set is 10*10 in ZX Plane, and be divided into 10*10 (row and column). Pointvop including more detail procedure which aim to create float value for each grid from 0 to the number of grids.*



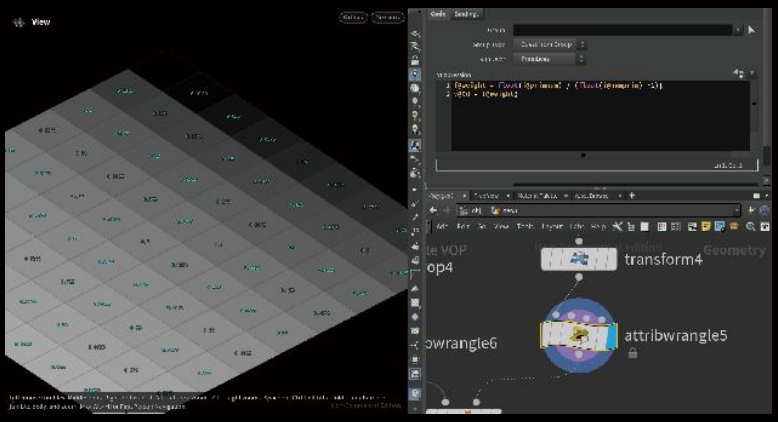
1.1.2 Attribwrangle SOP 1

Attributwrangle SOP 1(added node), where I wrote 'weight' as attribute. The attribute is of type float as described by the "f@" declaration. The attribute is set to run over primitives, because in the node setup later, I use a PolyExtrude node where we will need a primitive attribute to load the extrusion value. The value is set to 1(Limit the maximum value of extrusion).



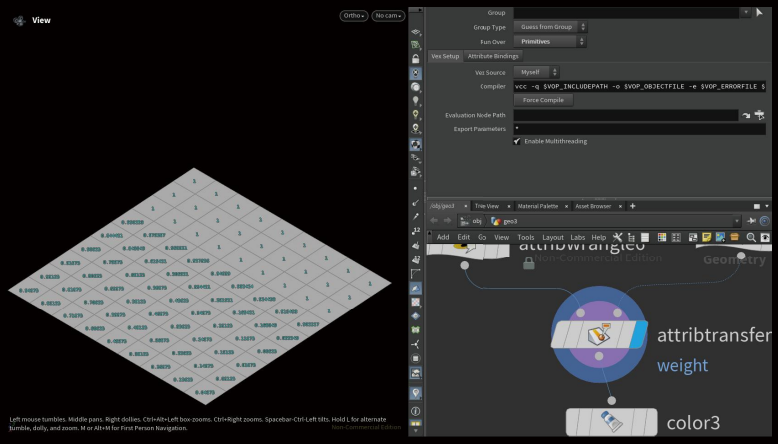
1.1.3 Transform SOP & Attribwrangle 2

Then, I laid down a Transform SOP.This will be the attractor; it is just a duplicate of the original grid. The distance between this duplicate (attractor) and the original grid will effect the final extrusion zones. Attribwrangle 2 I wrote this code, in order to make the value from a corner to another from 0 to 1.



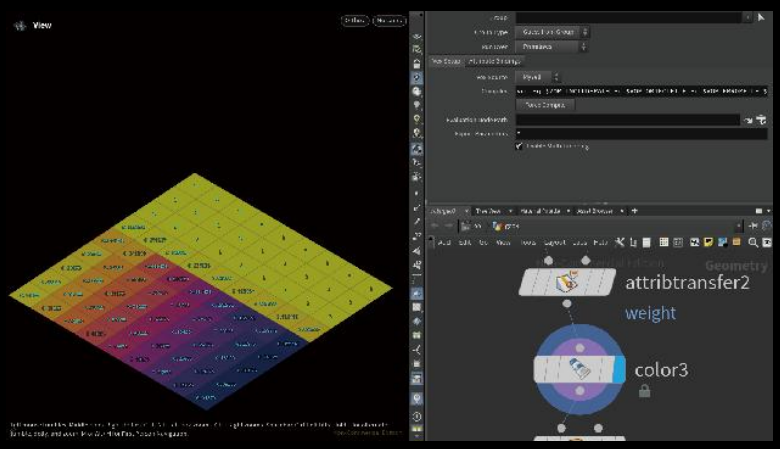
1.1.4 Attribtransfer SOP

Attribute Transfer SOP (added node) uses the "weight" primitive attribute to create the distance value between the original grid and the transformed one. By setting the original 'weight' attribute as float, we can have value interpolations between 0 and 1.



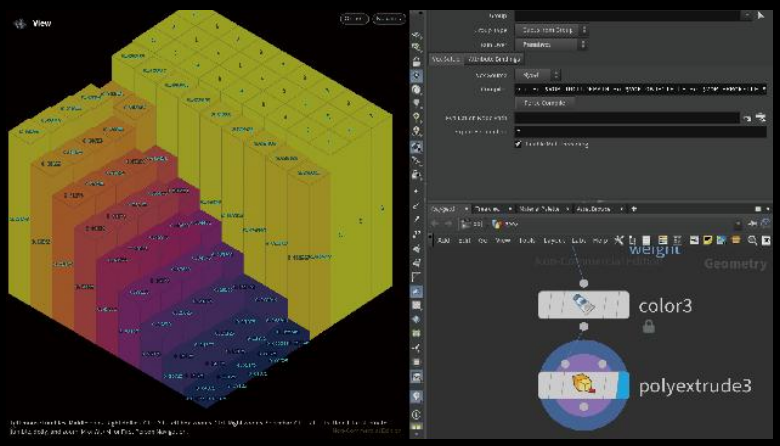
1.1.5 Color SOP

Visualize the primitive attributes of "weights" based on the "viridis" set color scheme. Because I normalized my values between 0 and 1, it is more stable if I change the distance or geometric input.



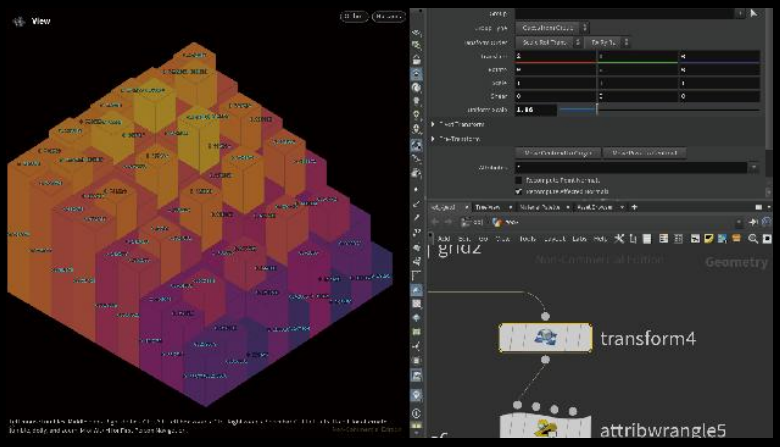
1.1.6 Polyextrude SOP

The Polyextrude SOP, the volume be extruded be divide into individual elements. 'Distance' column could be adjusted to control the distance be extruded (here it could be understand as height). 'Distance scale' is attributed by 'weight'.



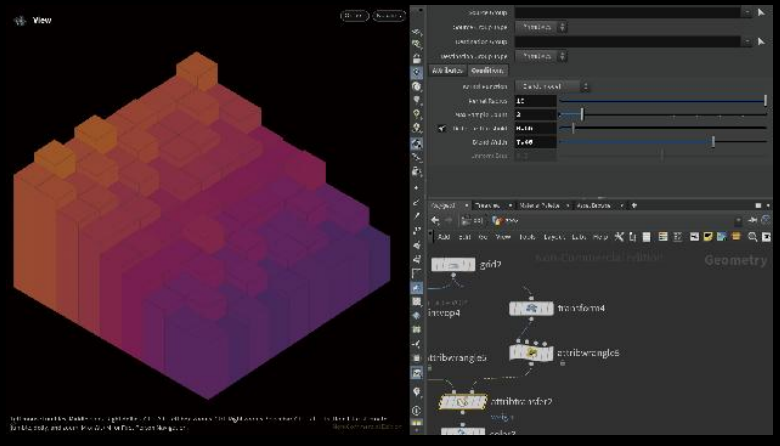
1.1.7 Adjustment_a

The effect is mainly influenced by the Transform, by adjusting settings of transform. The parameter of 'translate' , 'Rotate', 'Scale', 'Shear' and 'uniform Scale'



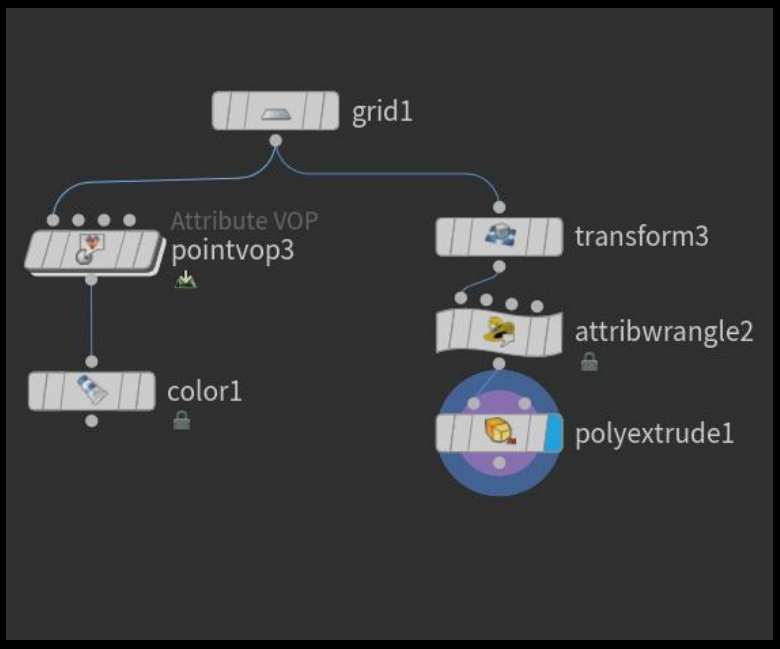
1.1.8 Adjustment_b

Attribtransfer also effect the performance, by adjusting parameter named 'Max Sample Count', 'Distance Threshold' and 'Blend Width'.

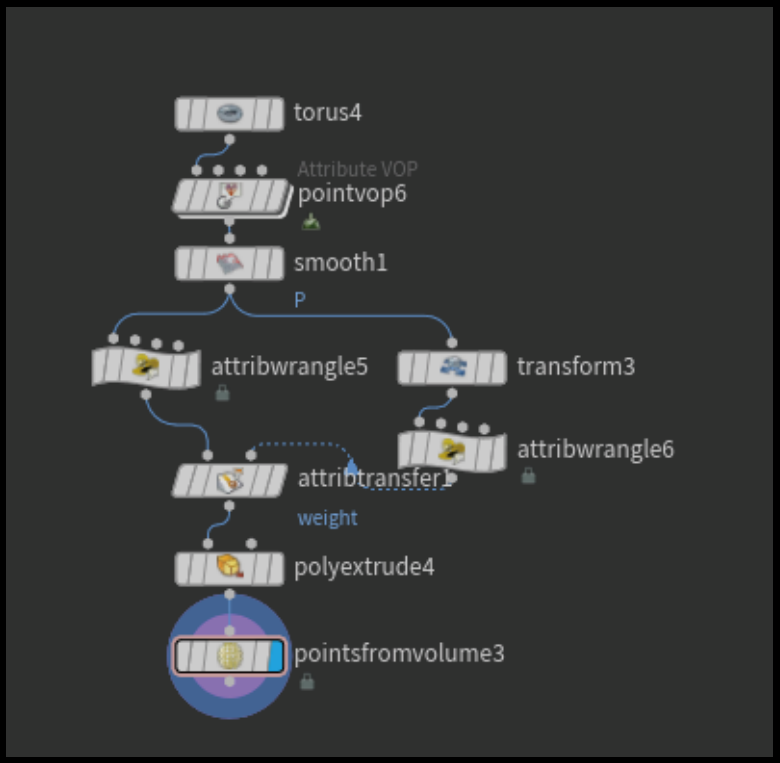


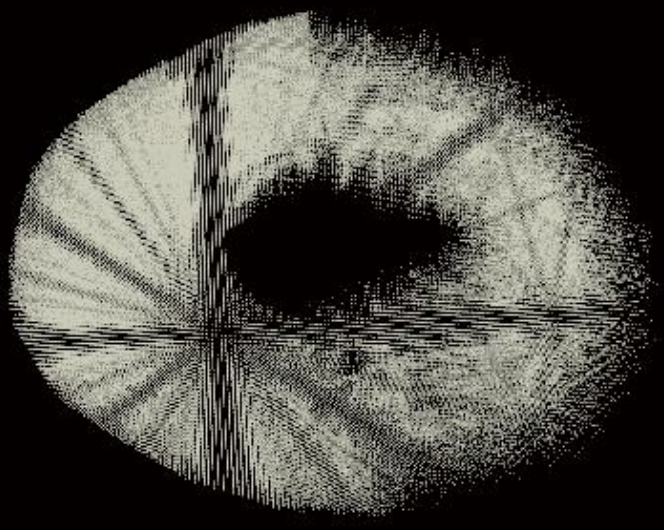
1.2 Generate a staircase, steps could be controlled

Original exmaple:



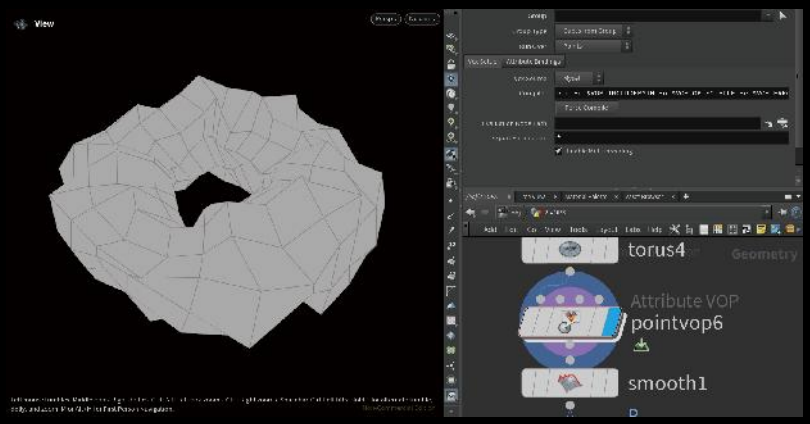
Altered result:





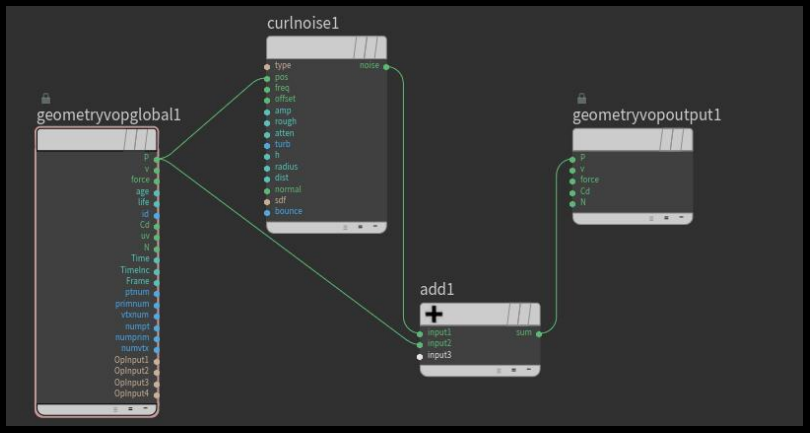
1.2.1 Torus SOP & Pointvop SOP

The Torus is of primitive type: polygon. One of the basic geometric datatypes that contains points,primitives of type polygon. Here the torus I set on Y Axis, with 0.5 radius and 0.25 ring. It be divided into 12 rows and 24 columns.



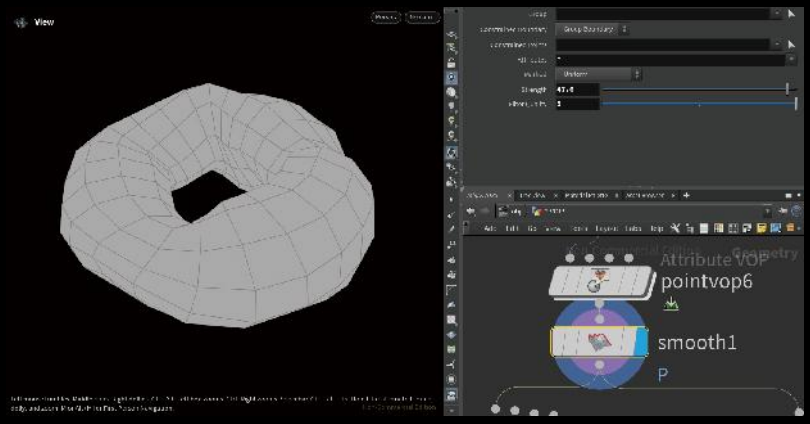
1.2.2 Pointvop SOP

The Vop SOP(added node) is similar to Attribwrangle in my understanding (VEX), Here I work in Vop, linking P with Position of curlnoise, and input to add_1, curinoise also input to add_1. Then the combined value input to geometryvop and make output. I also used parameter like 'Amplitude' and 'Roughness' to control the curvature.



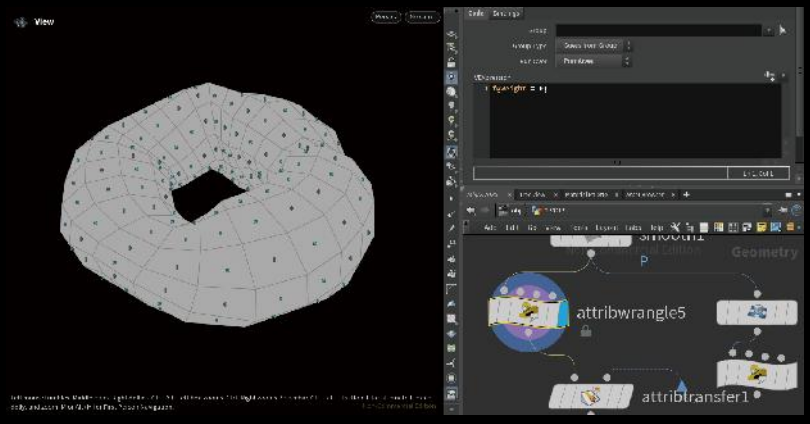
1.2.3 Smooth SOP

The Smooth (added node)rearranges the existing points in the geometry to reduce roughness (attribute in my case: Point Position). The parameters effect this SOP are 'Strength' and 'Filiter Quality', The volume would be smoother when raising the 'Strength', and be smoother when input lower 'Filiter Quality'

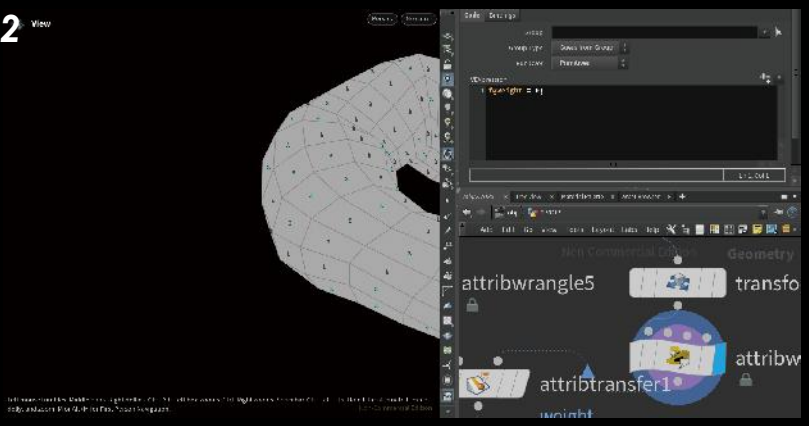


1.2.4 Attribwrangle SOP 1

Attribute Wrangle 1 I created which I made a 'weight' attribute. The attribute is of type float as described by the "f@" declaration. The attribute running over primitives. This is because later I will use a PolyExtrude which need a primitive attribute to drive the extrusion value. The value is set to 0.

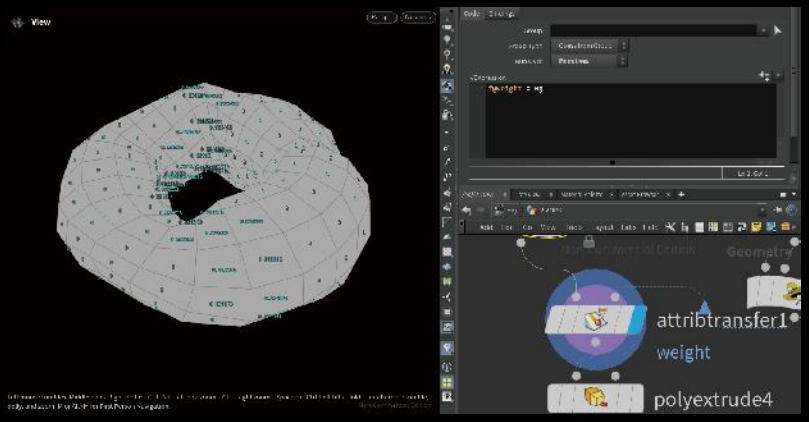


1.2.5 Transform & Attribwrangle SOP 2



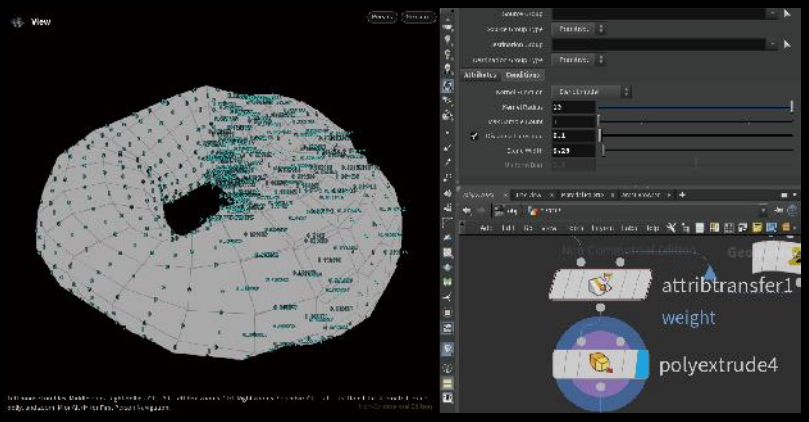
The Transform SOP could create the attractor, just a duplicate of the original sphere. The distance between this attractor and the original torus will determine the final extrusion value. Adjusting the 'Uniform scale'. The Attribwrangle 2 here set value to1. I could interpolate between 0 and 1 in the way later.

1.2.6 Attribtransfer SOP



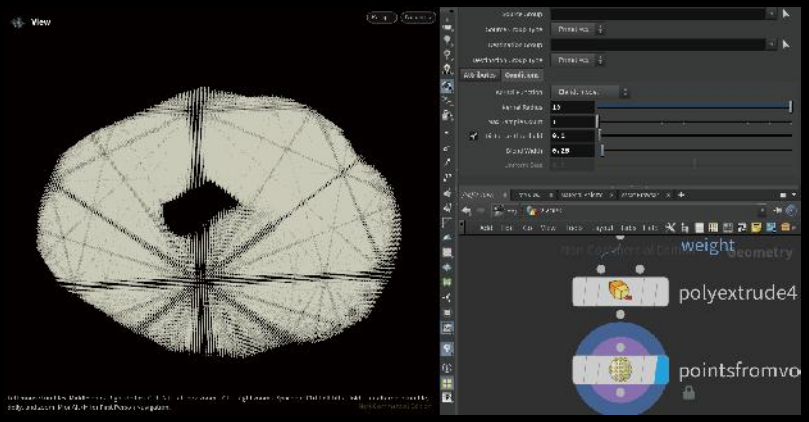
The "Attribute Transfer" SOP (added node) uses the 'weight' primitive attribute to create the distance value between the original grid and the transformed one. By setting the original 'weight' attribute as float, I can get the value interpolations between 0 and 1.

1.2.7 Polyextrude SOP



The Polyextrude SOP, the volume be extruded be divide into individual elements. 'Distance' column could be adjusted to control the distance be extruded.

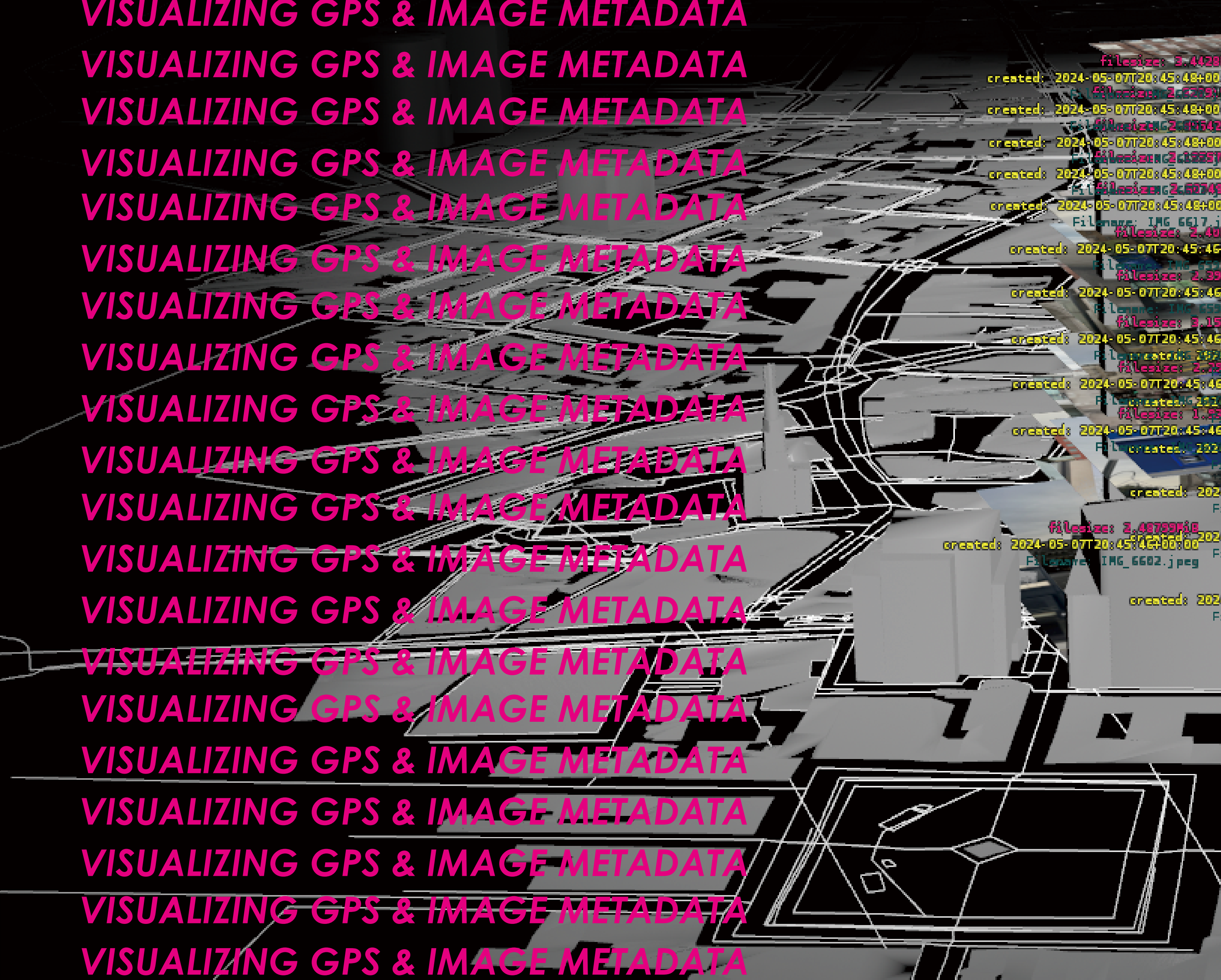
1.2.8 Pointsfromvolume SOP

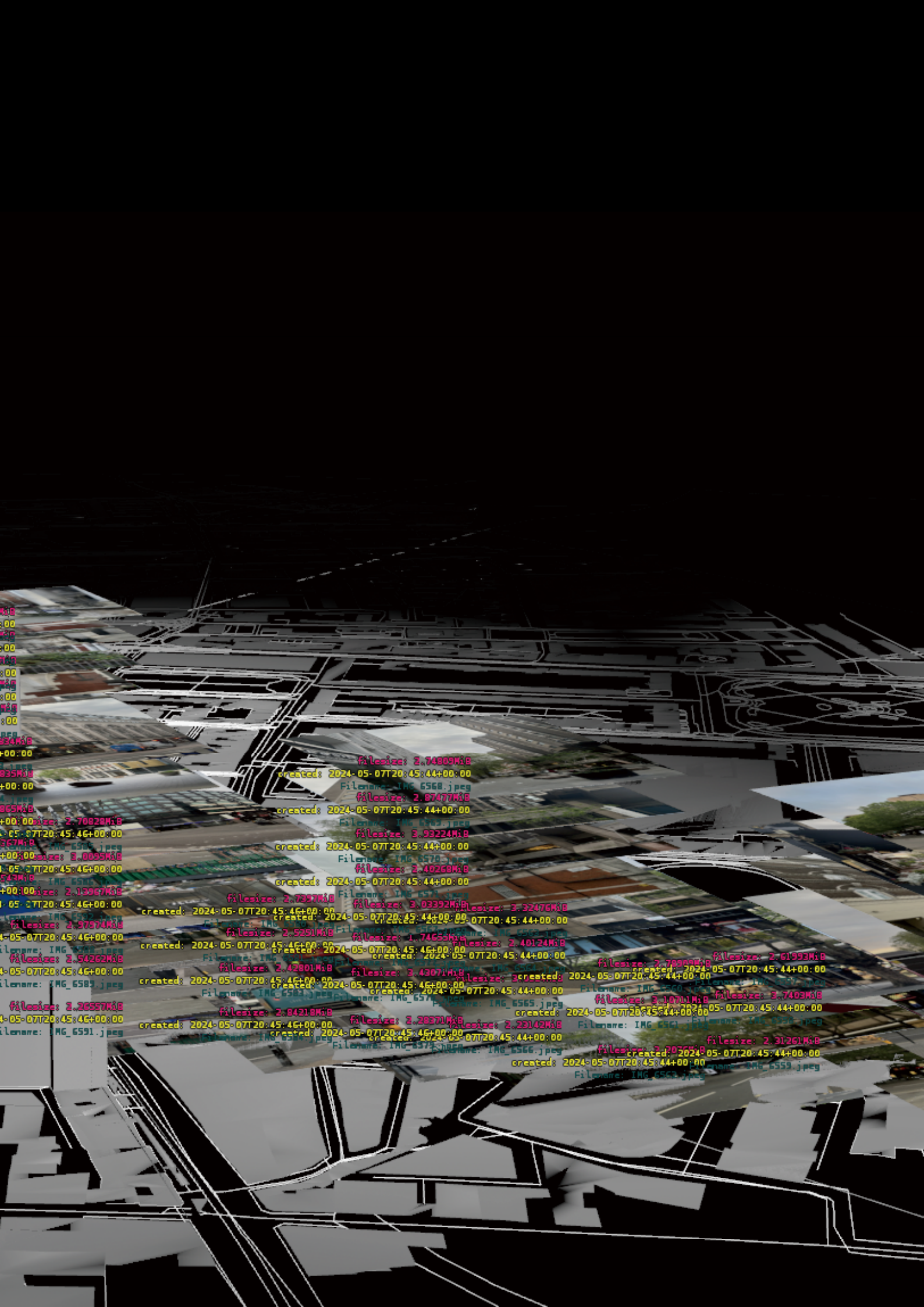


Pointsfromvolume SOP(added node) The type of incoming geometry. In auto-detect, if the input is a single volume primitive, the Fog or SDF method will be used depending whether the volume primitive has the SDF flag set. The adjustable parameter I used is 'Point Separation'. The point pixel woudl be dense (more) if that value lowller.

HOUDINI 2

VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA
VISUALIZING GPS & IMAGE METADATA

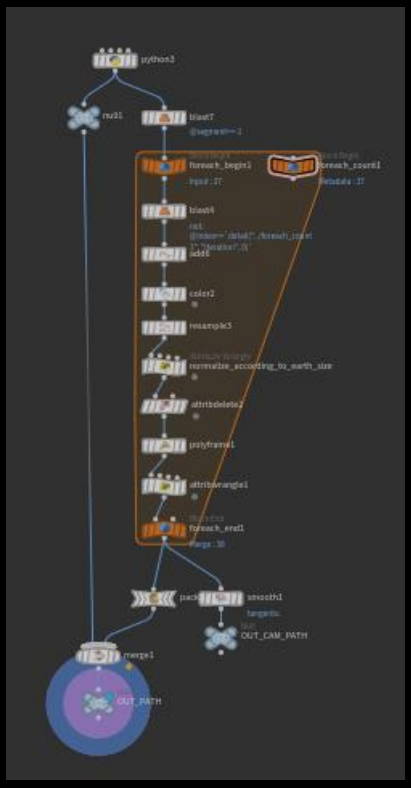



filesize: 2.74809MiB
created: 2024-05-07T20:45:44+00:00
Filename: IMG_6568.jpeg
filesize: 2.87477MiB
created: 2024-05-07T20:45:44+00:00
filesize: 3.93224MiB
created: 2024-05-07T20:45:44+00:00
filesize: 2.40269MiB
created: 2024-05-07T20:45:44+00:00
filesize: 2.73957MiB
created: 2024-05-07T20:45:46+00:00
filesize: 3.03392MiB
created: 2024-05-07T20:45:44+00:00
filesize: 2.5251MiB
created: 2024-05-07T20:45:46+00:00
filesize: 2.42801MiB
created: 2024-05-07T20:45:46+00:00
filesize: 2.84218MiB
created: 2024-05-07T20:45:46+00:00
filesize: 3.32476MiB
created: 2024-05-07T20:45:44+00:00
filesize: 2.40124MiB
created: 2024-05-07T20:45:44+00:00
filesize: 2.61933MiB
created: 2024-05-07T20:45:44+00:00
filesize: 2.7403MiB
created: 2024-05-07T20:45:44+00:00
filesize: 2.31261MiB
created: 2024-05-07T20:45:44+00:00

2.1 Extract GPX data (Path)

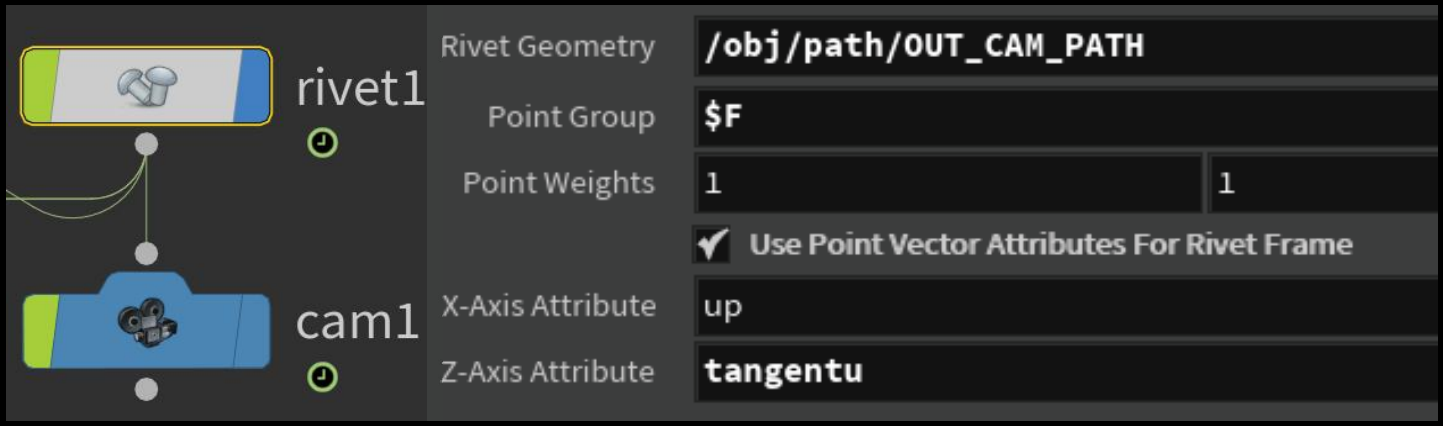
Before starting to run the program in Houdini. Based on the Timeline in my Google Map cannot be displayed, I found a Chinese mobile app called Six Feet, that can record travel routes, generate data, and generate GPX files.

Then I upload the GPX file in python module of Houdini. Add modules/ functions to ForeachPoint to handle the path, and combine it with Smooth to generate OUT_CAM_PATH, which will be used as the Operator input Rivet1 to match the CAM and path line together.



2.2 Camera setting

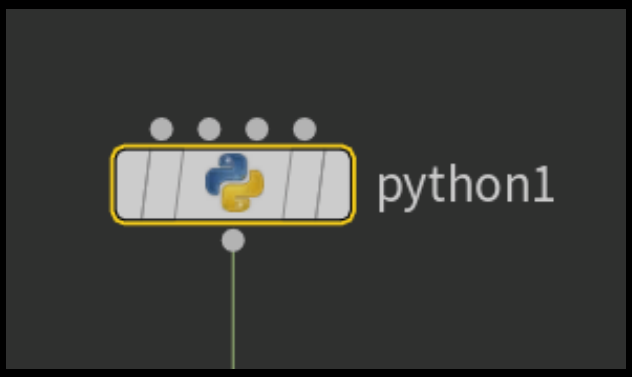
In order to let the CAM follow the path, OUT_CAM_PATH would be used as Operator uploaded in Rivet_1.

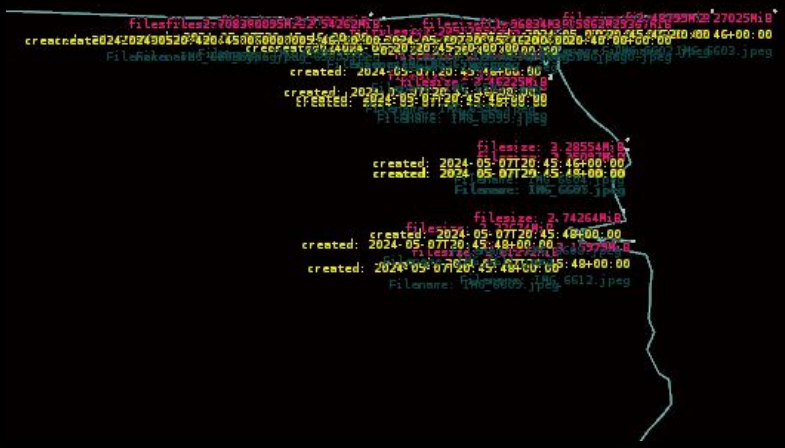


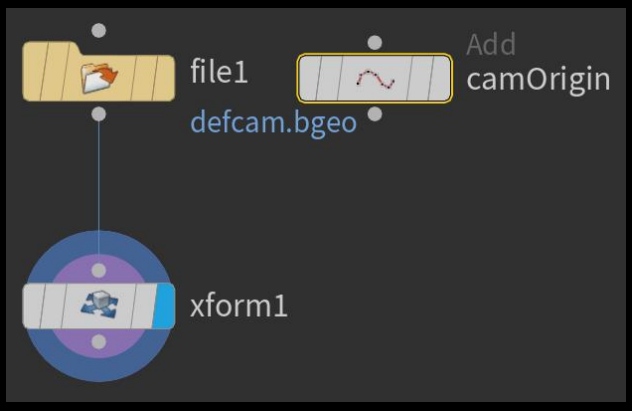
2.3 Loading images (Visualized image data/infomation)

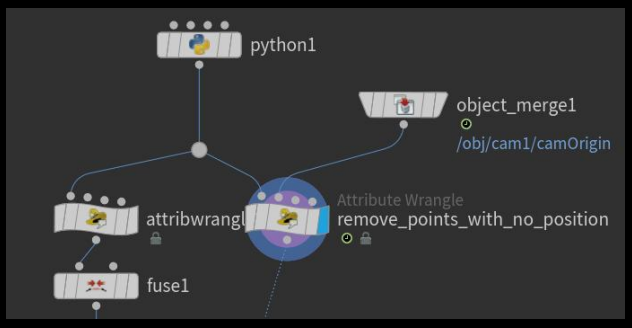
Import the folder where the image is located (in the Python module), read the image information (data) through code, and visualize the information content by adding Marker.

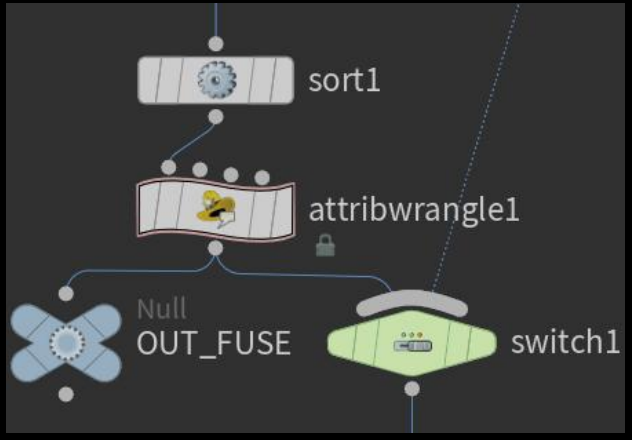
Add the Object_merge function, import camOrigin of CAM, and write a attribwangle to delete points without position. Then output OUT_FUSE which record result of fuse_1, and adjust Switch to make images oriented to CAM.

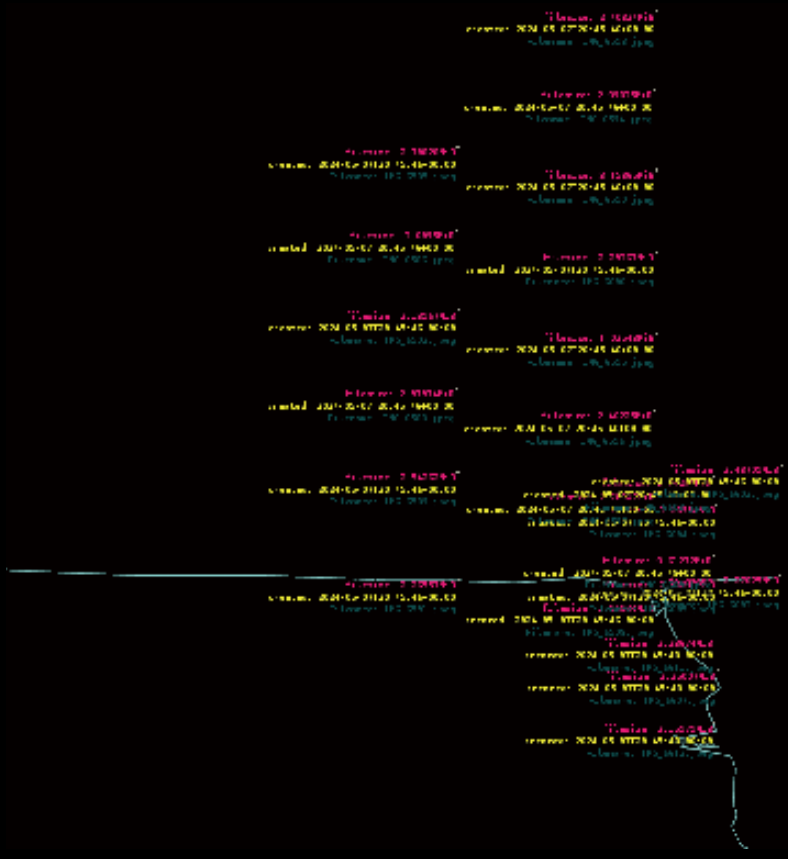








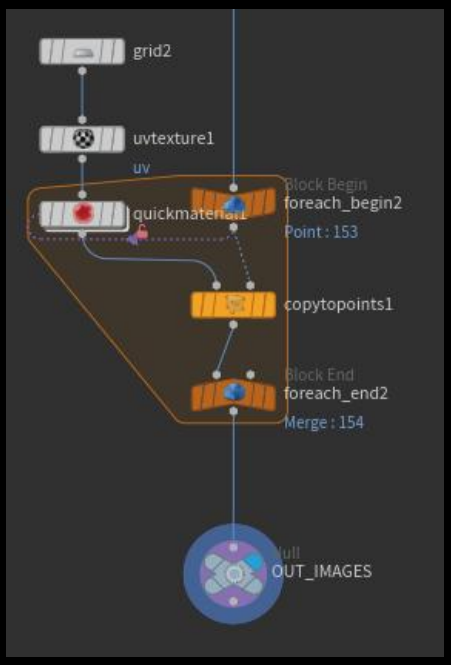




2.4 Loading images (link with points)

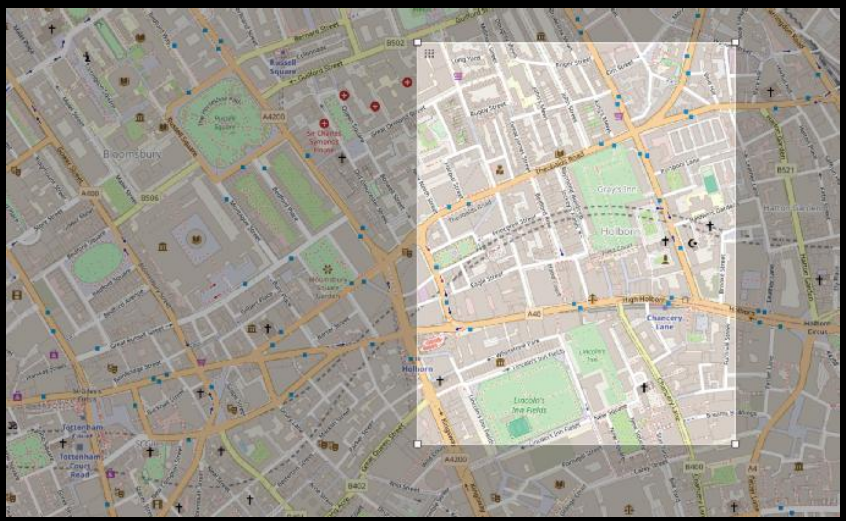
In the previous step, I have visualized the information of the image. The next step is to visualize the images. I created a Grid, connected UVtexture, and adjusted it to z-axis to ensure that the image materials on the grid can be presented correctly. Then connect it to Quickmaterial.

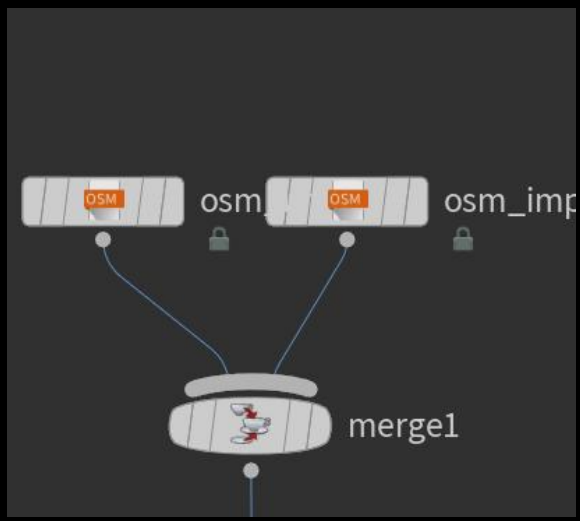
In quickmaterial, adding Spare input, and adding ForeachPoint then drag it into Spare input. After that, Copytopoint link the image infomation with each image.



2.5 Extract geographical data (city map)

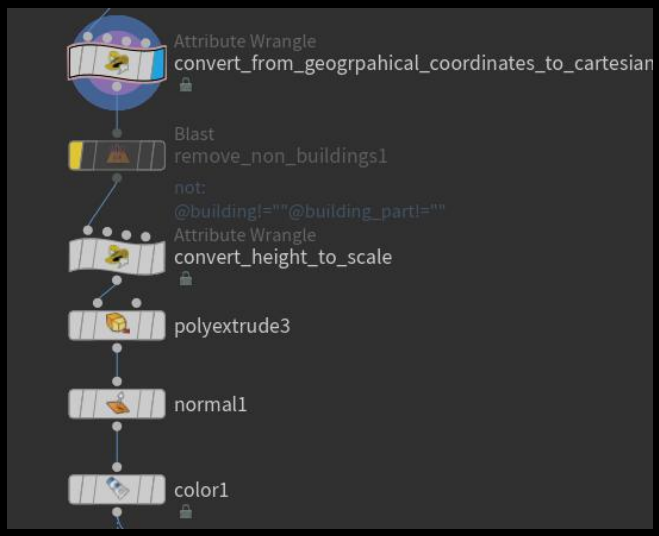
In order to obtain geographical information of the site, I exported the area where the path is located in OpenStreetMap.

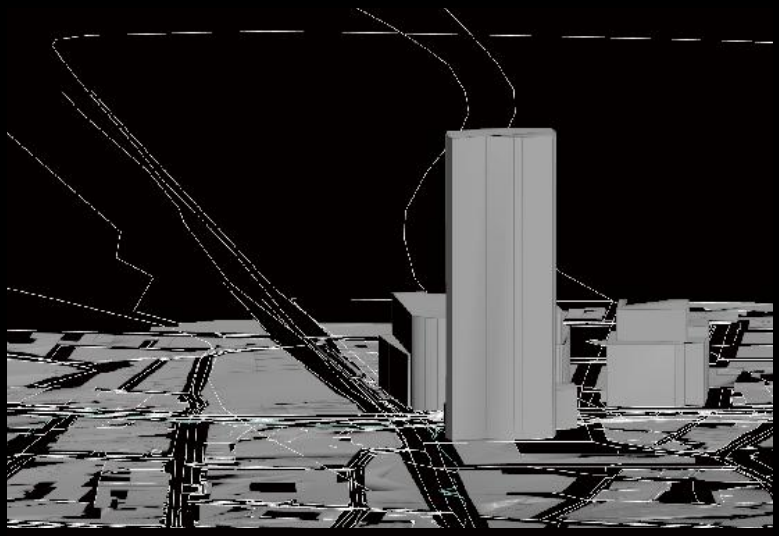


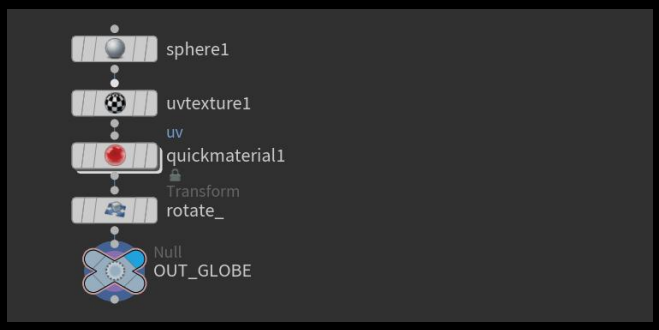


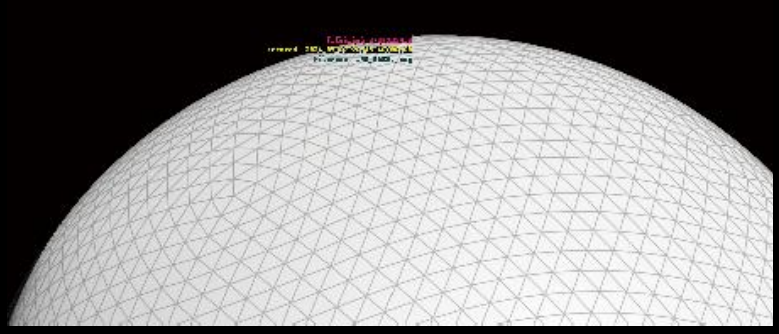
2.6 Generate maps and buildings

After merging Osm files, combine the Attribwangle input code to generate the map and obtain height information, Matching with the Earth (sphere), and then use Polyextrude to generate the volume according data in OSM.



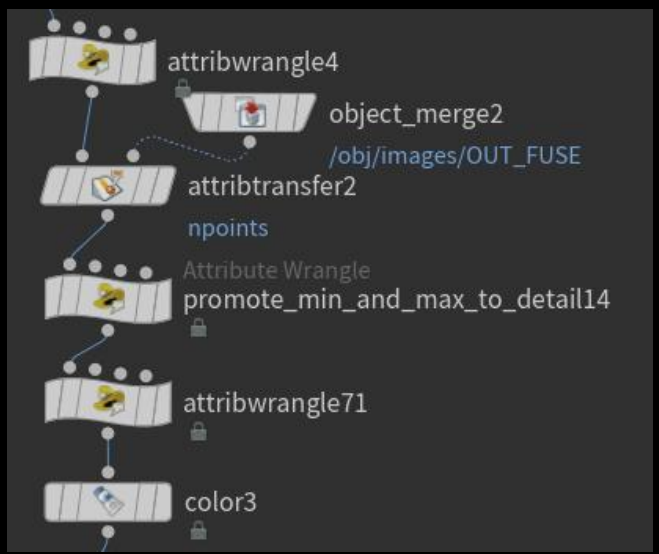






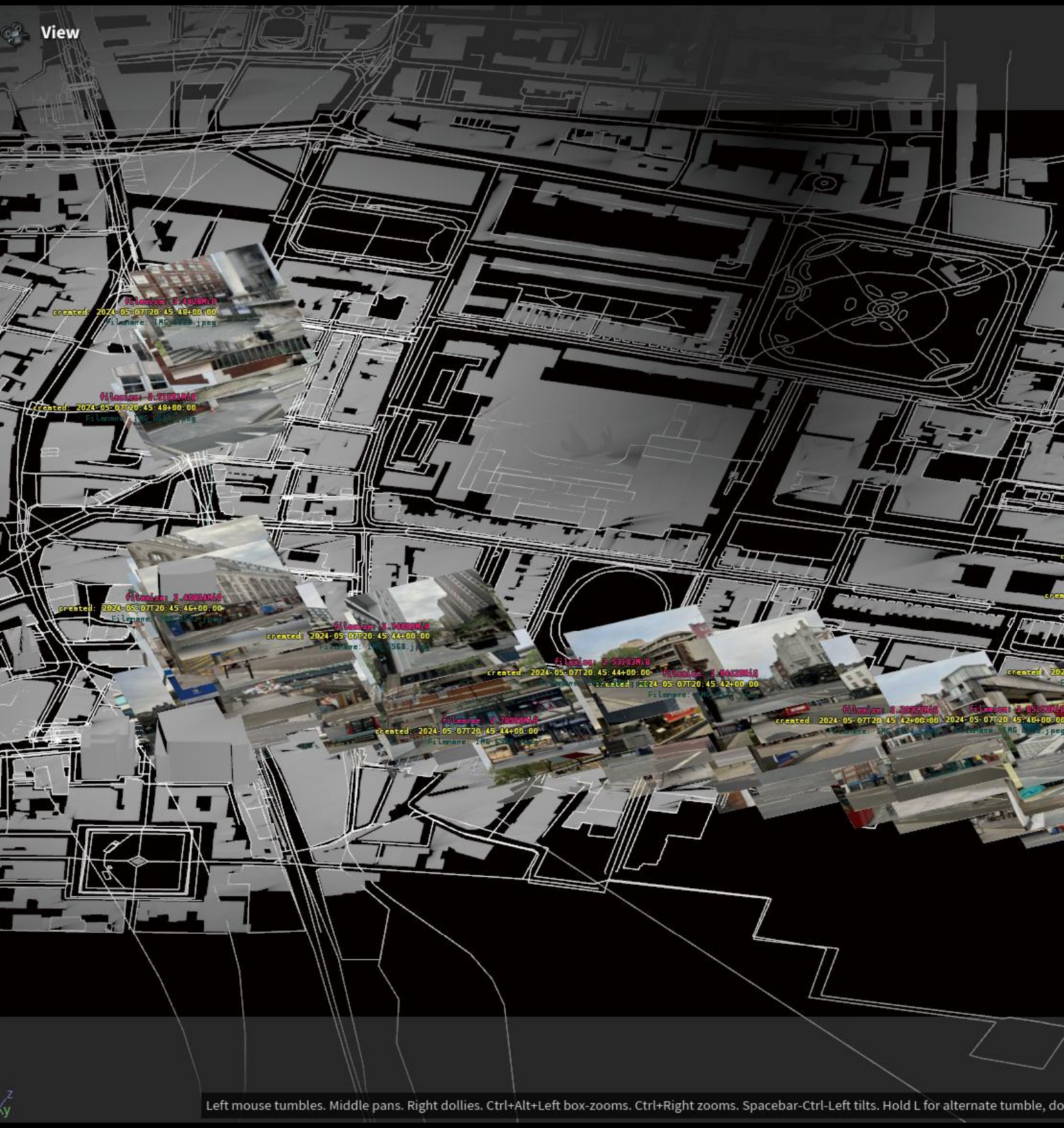
2.7 Adjusting map

In the final part of the City program, I adjusted the map generated by OSM through the settings shown in the left figure for better visibility. Afterwards, output the image frames by frame and generate a video.

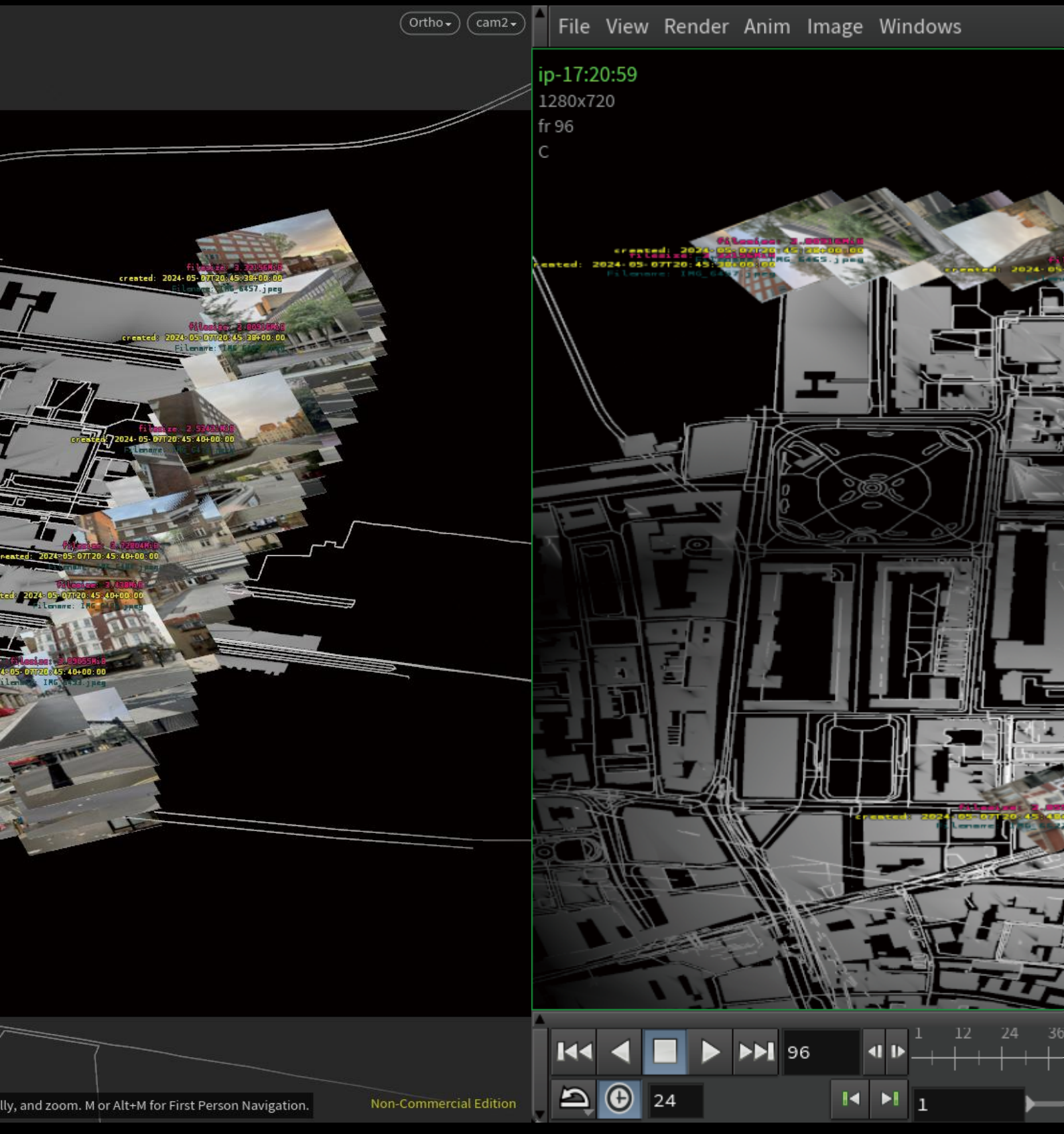


2.8 A series of images depicting the travel. (Performance/Rende

The CAM_1 plays along the walking route. In this step, I set up another camera (CAM_2), which is a orthographic overview, to review the final effect.



er)



Short video by frames


filesize: 2.54483MiB
created: 2024-05-07T20:45:42+00:00
Filename: IMG_6870.jpeg
filesize: 2.85192MiB
created: 2024-05-07T20:45:40+00:00
filesize: 3.03893MiB
created: 2024-05-07T20:45:40+00:00
filesize: 2.52506MiB
created: 2024-05-07T20:45:40+00:00
filesize: 2.15134MiB
created: 2024-05-07T20:45:42+00:00
filesize: 2.22217MiB
created: 2024-05-07T20:45:42+00:00
filesize: 2.71783MiB
created: 2024-05-07T20:45:42+00:00
filesize: 3.12268MiB
created: 2024-05-07T20:45:42+00:00
CHIPOTLE MEXICAN GRILL
J.&J. GOODYEAR
KFC
CASINO


filesize: 3.89055MiB
created: 2024-05-07T20:45:40+00:00
filesize: 3.60212MiB
created: 2024-05-07T20:45:40+00:00
filesize: 2.50503MiB
created: 2024-05-07T20:45:40+00:00
filesize: 3.65037MiB
created: 2024-05-07T20:45:40+00:00
filesize: 2.51306MiB
filesize: 4.78164MiB
2024-05-07T20:45:40+00:00
filesize: 3.18362MiB
2024-05-07T20:45:40+00:00
filesize: 2.00242MiB


1169.14m^3
Monumento a Alfonso XII
The monument is situated on the east edge of an artificial lake near the center of Retiro park.
Its center is the equestrian statue of King Alfonso XII, cast in bronze.
The three statues on the central pedestal of the monument represent 'Peace', 'Freedom', and 'Progress'.
483.95m^3
1169.14m^3

HOUDINI 3

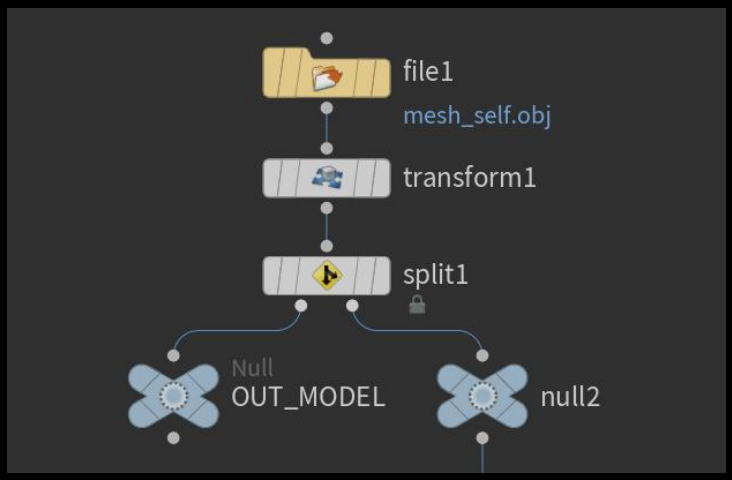
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION
3D RECONSTRUCTION

3.1 Generate photogrammetry models

Firstly, the video frames are deconstructed from the found video by code, and the images of these video frames are imported into RealityCapture to generate the reconstructed model.

For this section, I chose an overhead video of Madrid's Retiro Park from Youtube. Because our project site is in Madrid; the overhead video contains more information about the site and is more helpful for reconstruction.

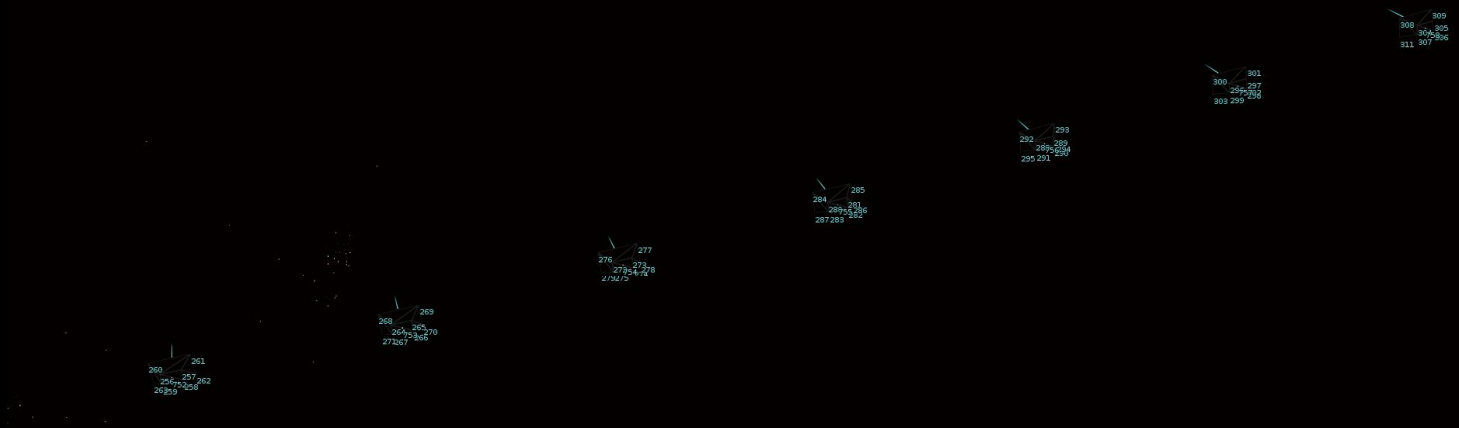
3.2 Reconstruct camera path.



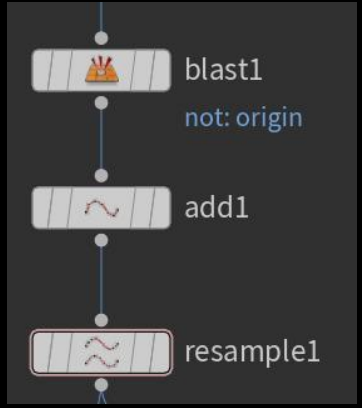
At first, after reconstructing the 3D model using RealityCapture and exporting the mesh, upload model in houdini, split it into the subsequent processed models and the Camera path to be processed now.

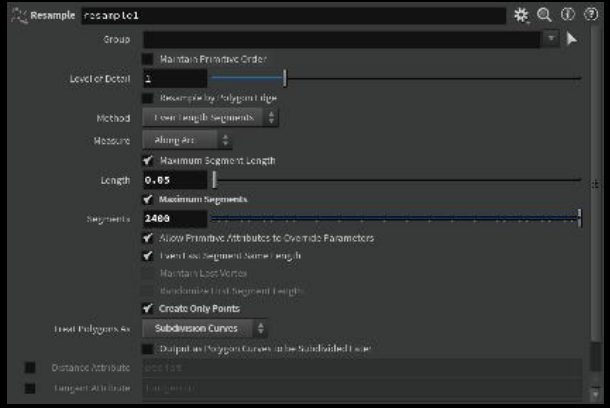
```
VEXpression
1
2  for (int pt = 0; pt<npoints(0); pt +=8){
3
4
5      float mult = 28;
6
7      vector posA = point(0, "P", pt );
8      vector posB = point(0, "P", pt +2 );
9      vector posC = point(0, "P", pt +4 );
10     vector posD = point(0, "P", pt +6 );
11
12     vector normalStart = lerp(posA, posB, 0.5);
13     vector normalEnd = lerp(posC, posD, 0.5);
14
15     int startPt = addpoint(0, normalStart);
16     setpointattrib(0, "N", startPt, -mult *(normalStart - normalEnd), "set");
17     //add a point on normalStart, and set a normal vector to that point
18     setpointgroup(0, "origin", startPt, 1, "set");
19     setpointattrib(0, "up", startPt, set(0,1,0), "set");
20 }
```

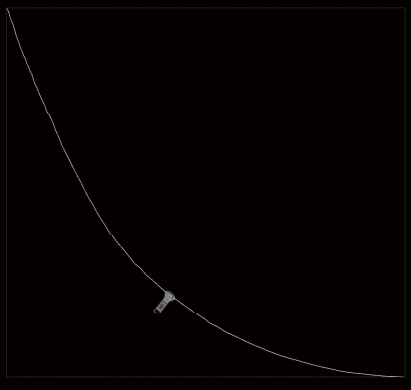
Step 2 is creating an attributewangle module and use the code shown in the figure to obtain the center point of each camera projection surface, as well as create corresponding lines for the camera facing the field.

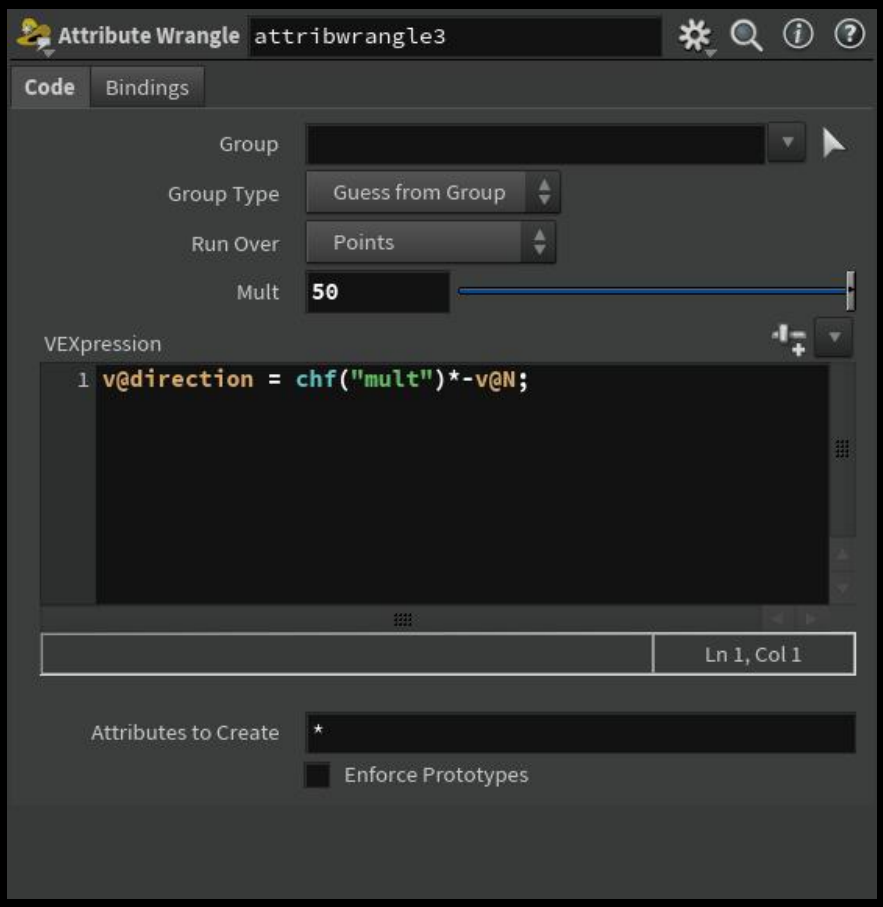


This stage is going to reconstructing the camera path, there is no need to delete points. I think the original path is good, so I did not use blast to delete the camera points. But the path itself was too short, I added the resample module, inserted more points between the original points, and extended the camera path.



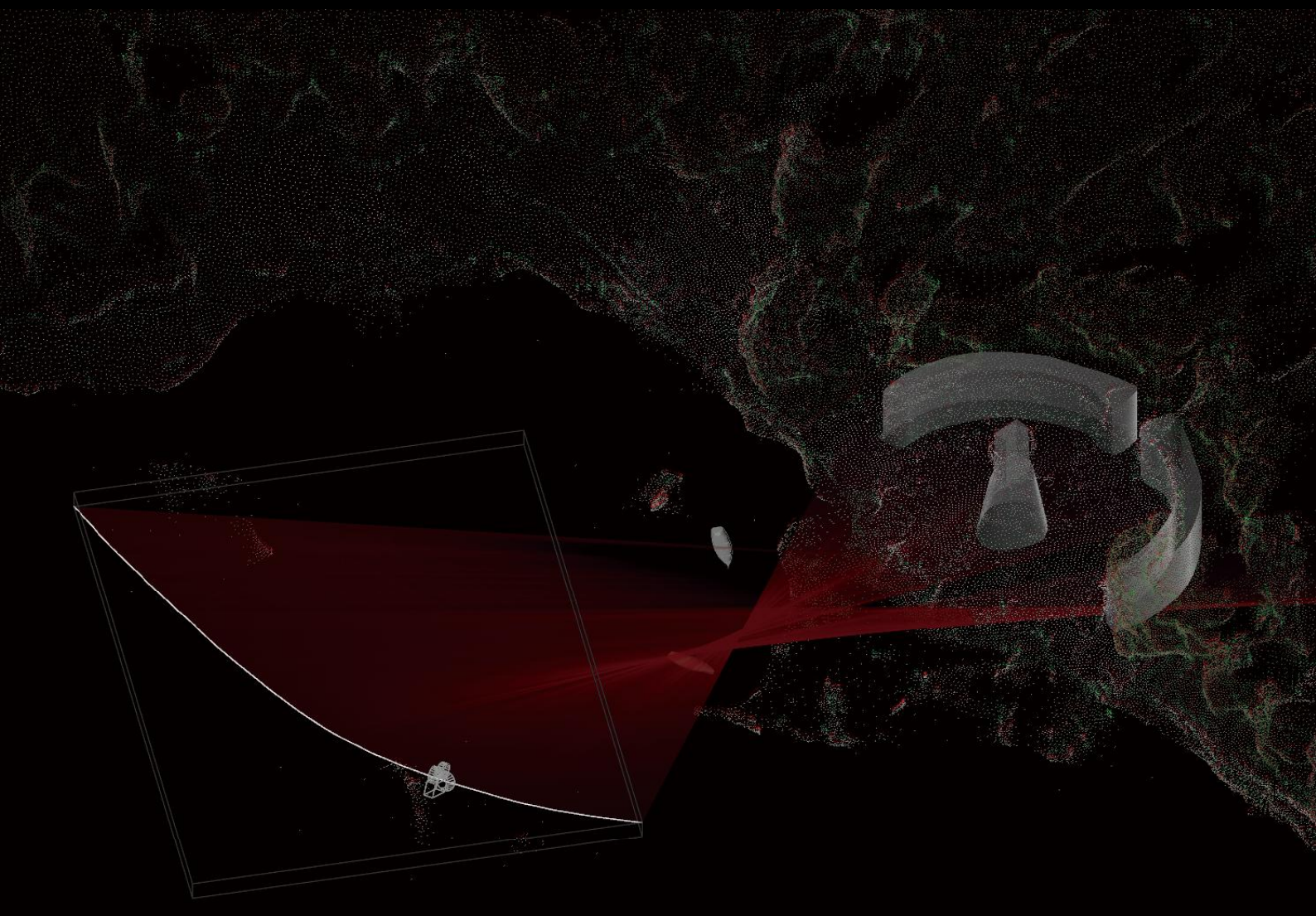






In the step 2, the corresponding lines already created for the camera facing the model. Here I add code in attribwrangle which add Mult, to control the length of those lines.

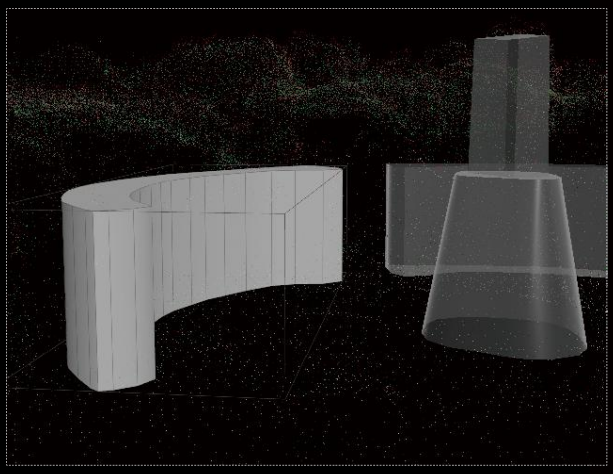
The final step in 3.2 is output the camera path' data, which will be used for the CAM part, the CAM would follow the path after this.



3.3 Volume speculation

Through the Measure curvature module, all the boundaries of the site could be seen, and based on these boundaries, I used Curve + polyextrude to generate objects. Finally, I created two Attribwrangle, one for visualizing volume data, and another attribwrangle for adjusting the transparency of volume objects.

Draw curves according to point edges



Extruded geometric block

VEXpression

```
1 s@totalvolume = "1169.14m^3";
```

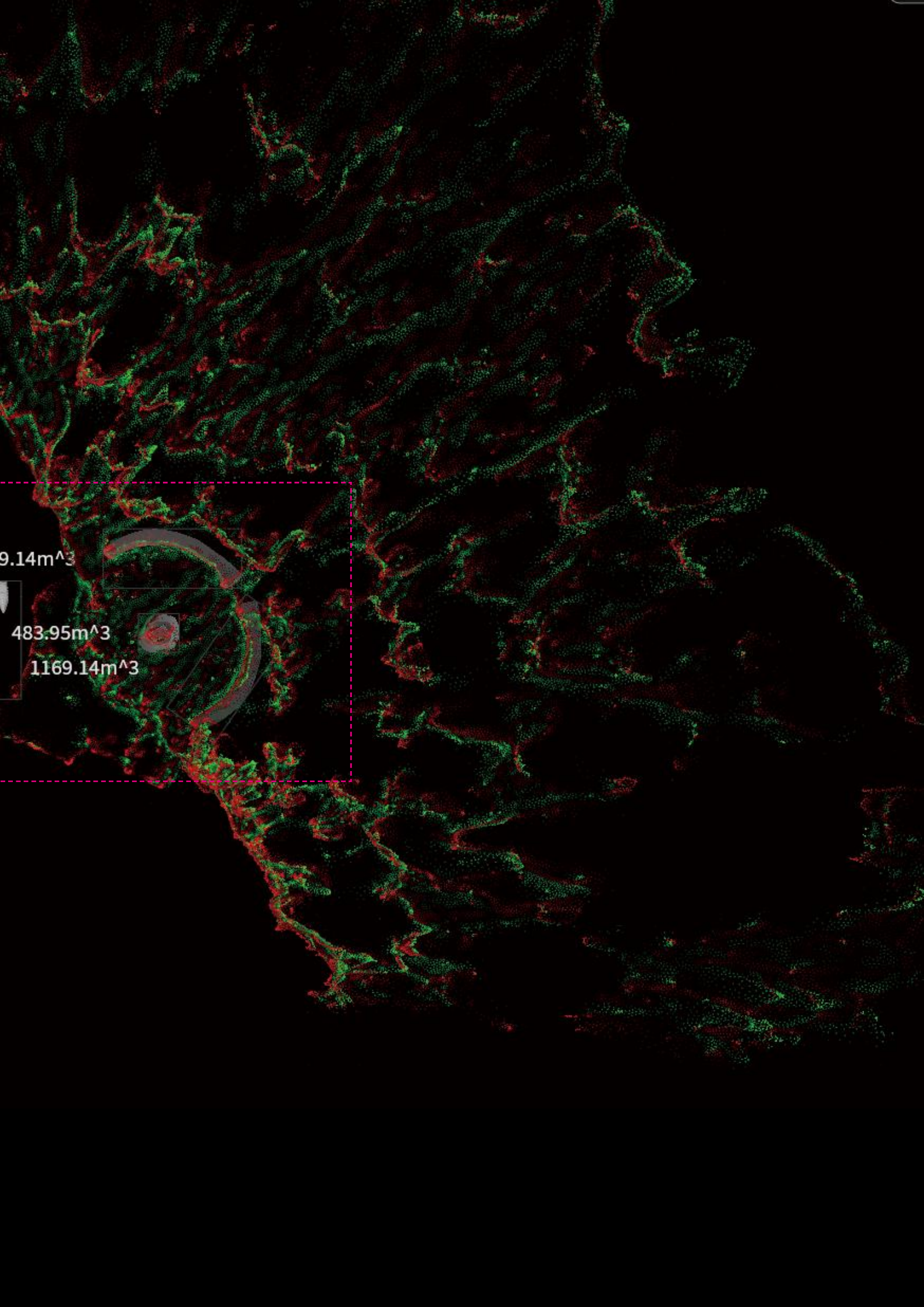
Adding a Measure module to extract value of volume, and then visualize the value through attribwrangle1

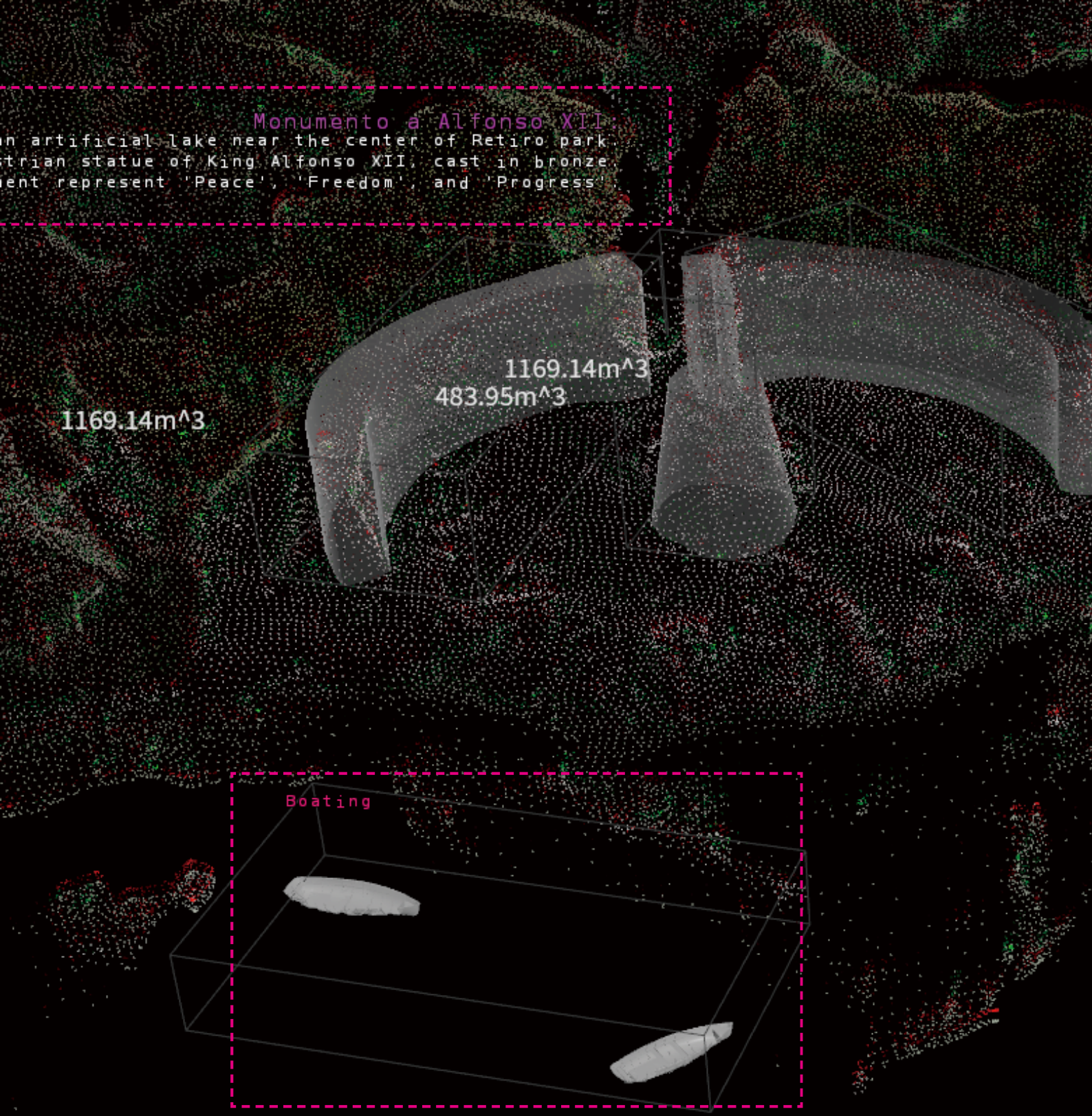
VEXpression

```
1 f@Alpha = chf("slider");
2
```

Slider 0.3

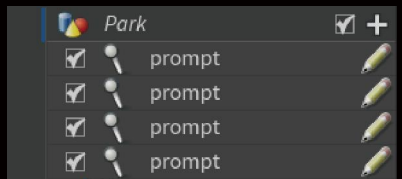
Creating slider to control transparency


9.14m^3
483.95m^3
1169.14m^3



3.4 Adding information to the subject

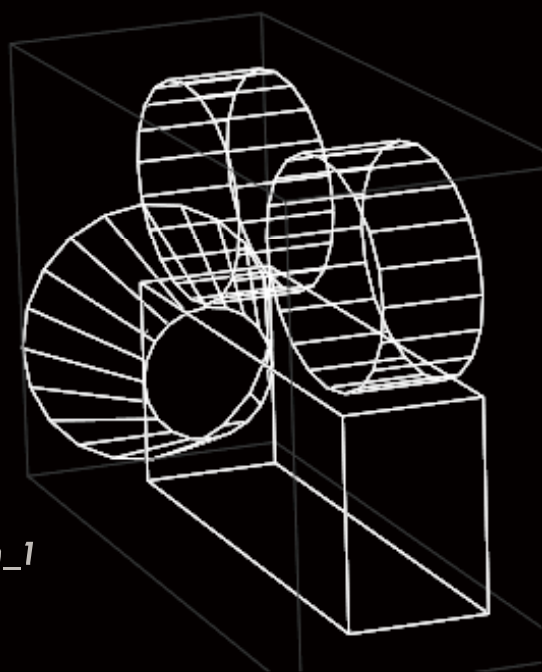
In the last part (3.3), I already add volume data to my objects.
In this part I created Markers which named prompt. Adding a attribwrangle moudle including four lines, which would attribute to the Markers. In addition, I also add a extra description for the boats models (Try to restore more scenes).



```
r of Retiro park.";
 Freedom', and 'Progress'.";
```

Short video by frames

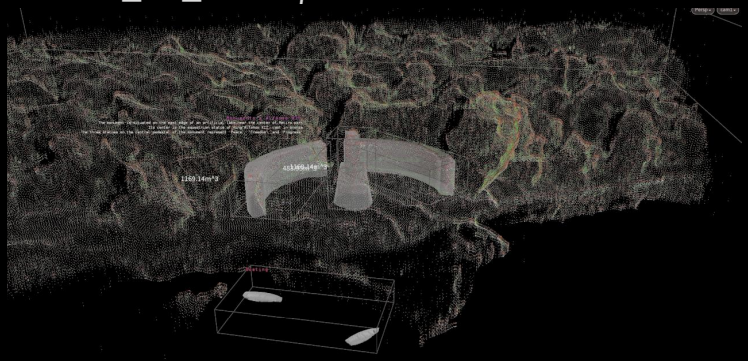
Cam_1

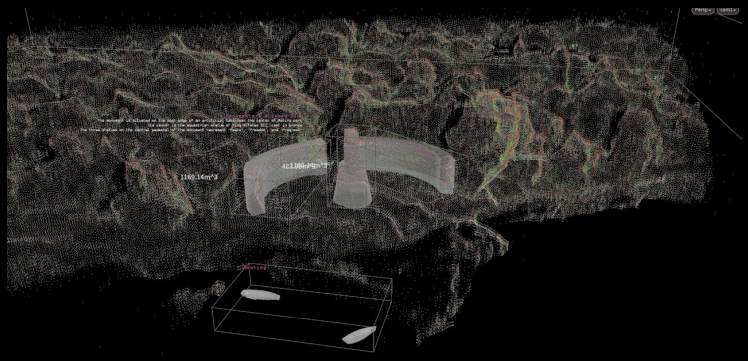


3.5 A COLLECTION OF IMAGES/RENDERS

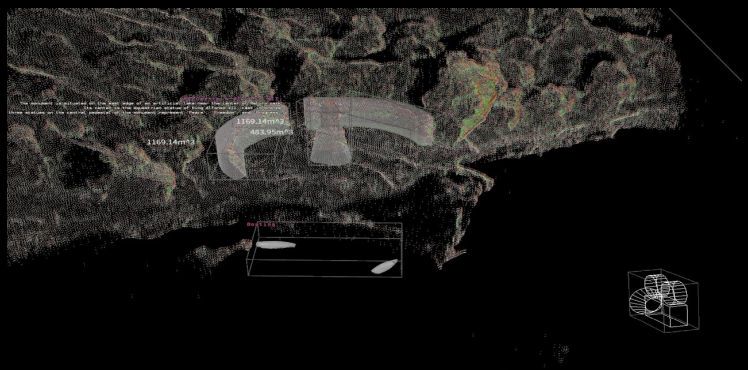
I created two camera positions, and in the second position, you can see the first camera. There are also some screenshots:

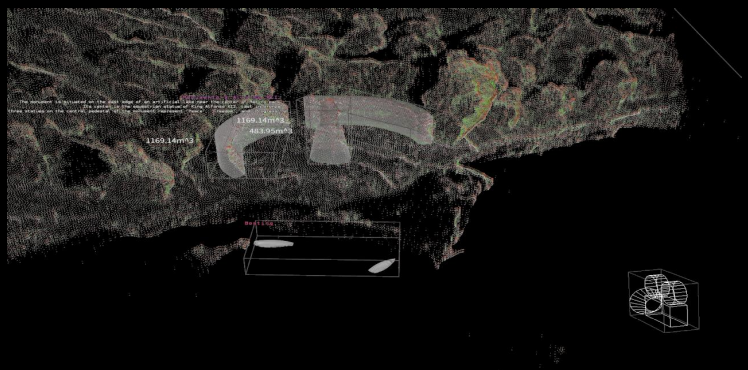
Cam_01_Examples:



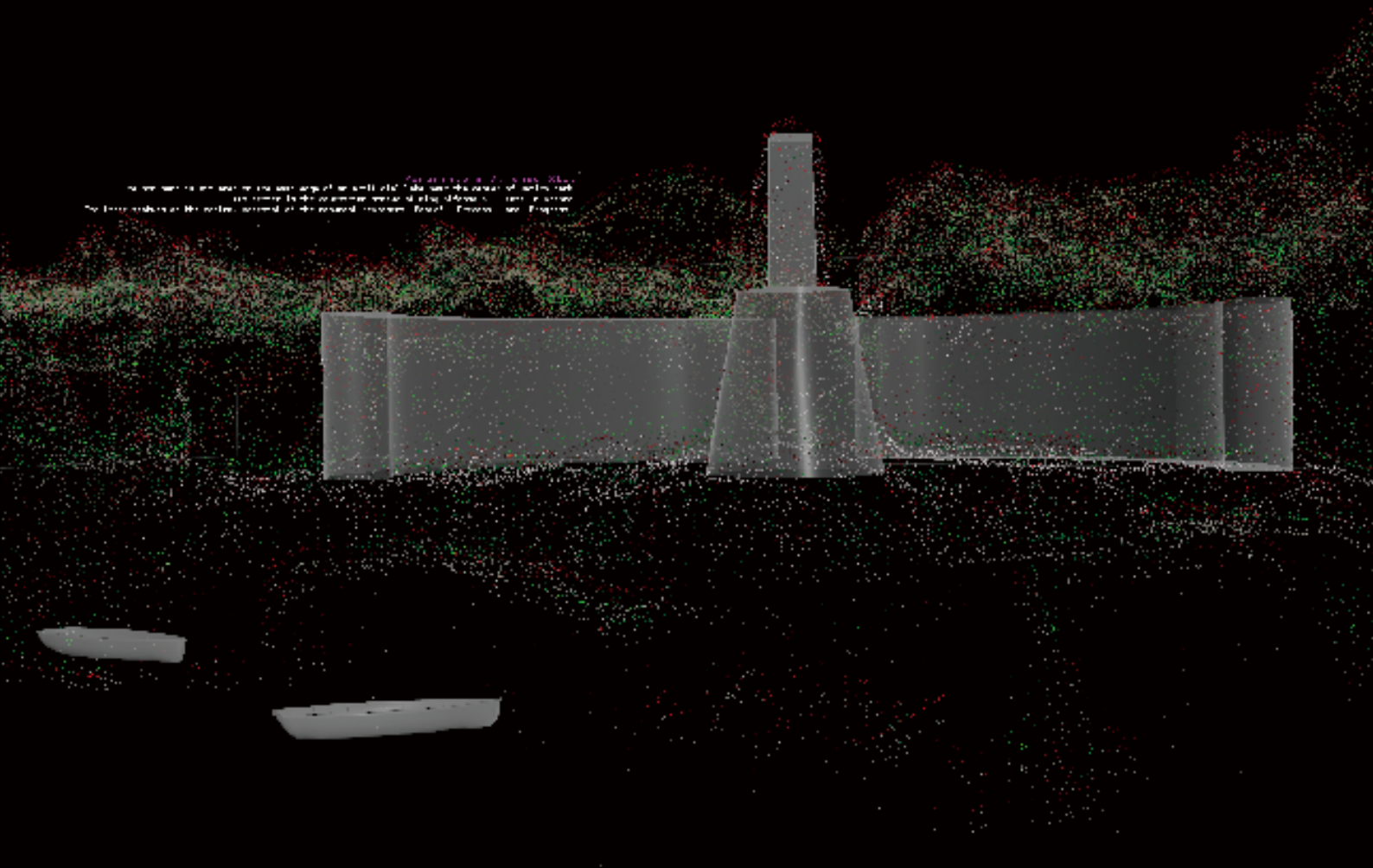


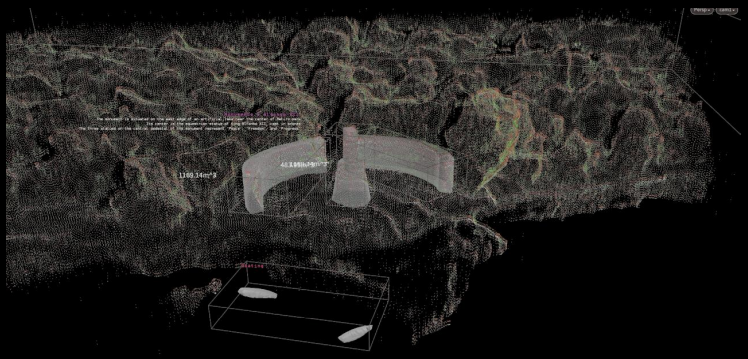
Cam_02_Examples:

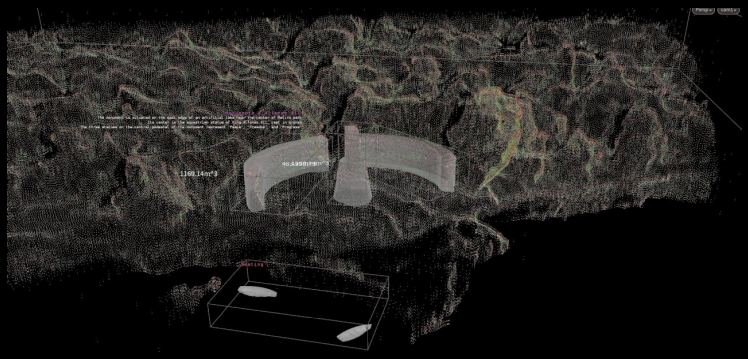


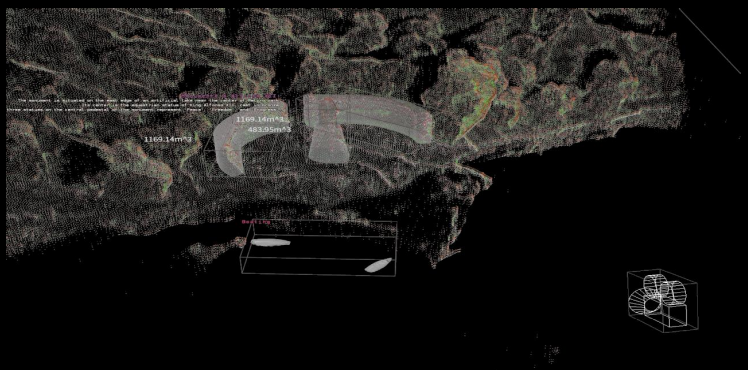


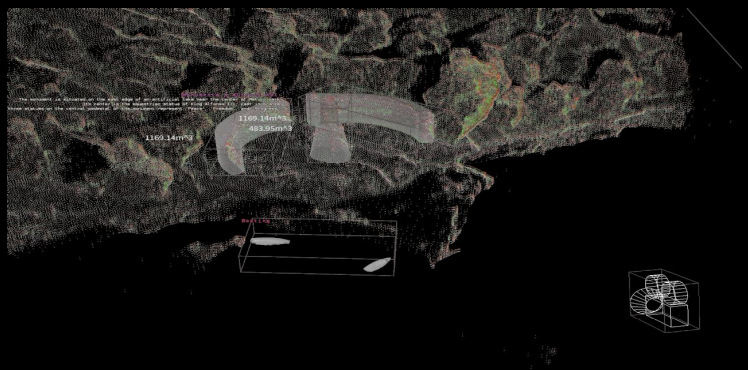
Screenshot_Examples:

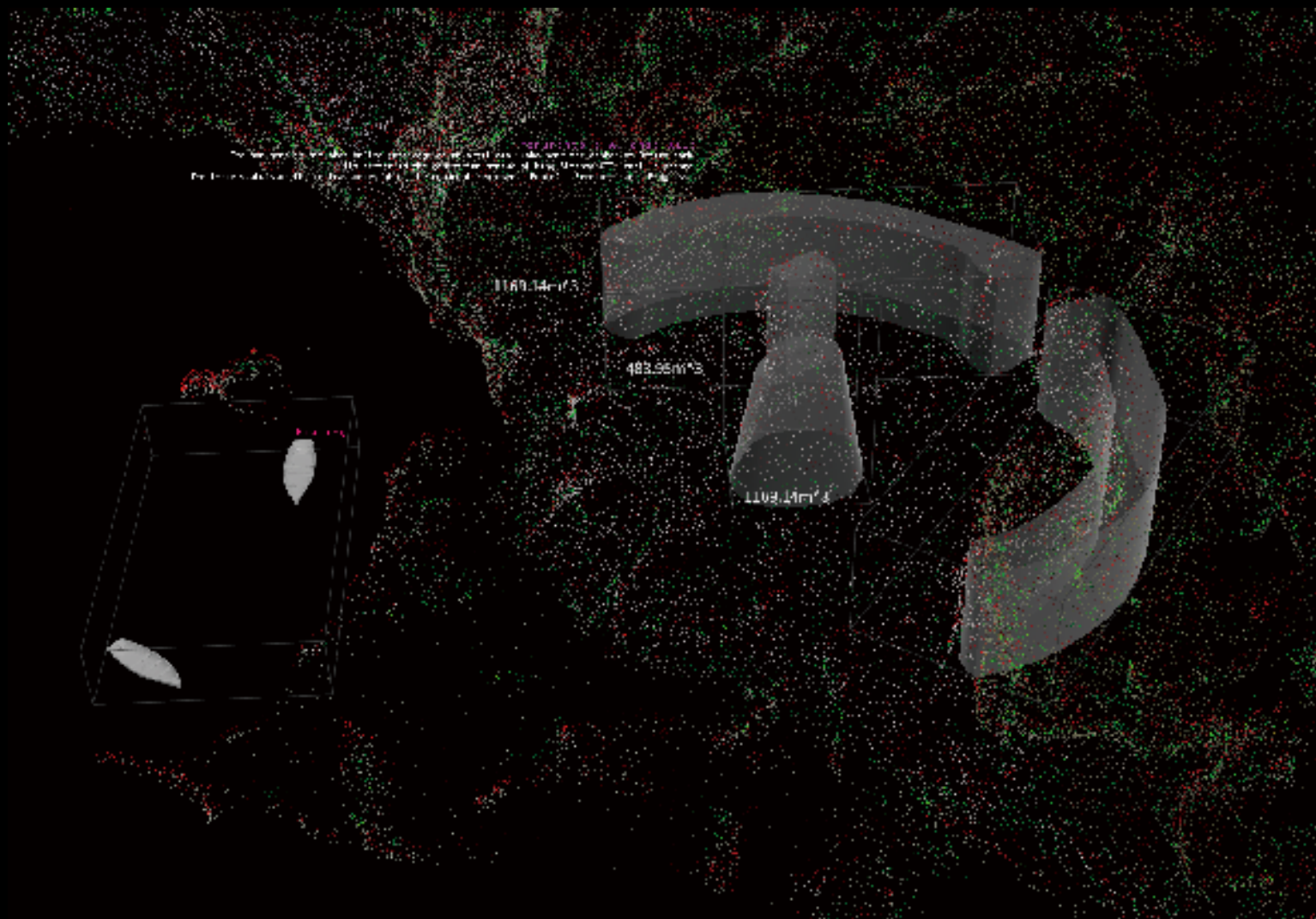






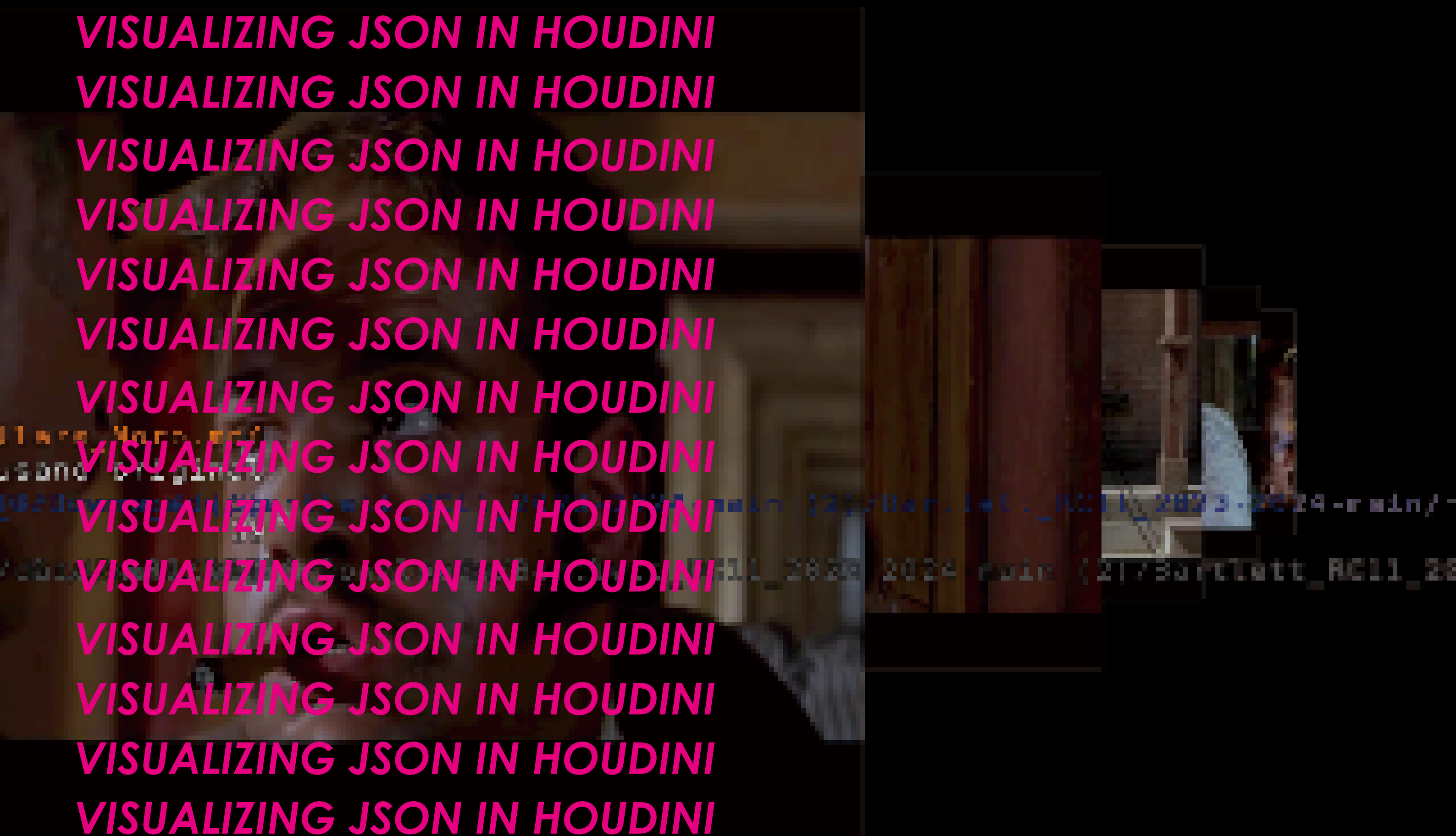





1169.14m^3
483.95m^3
1169.14m^3

HOUDINI 4

VISUALIZING JSON IN HOUDINI



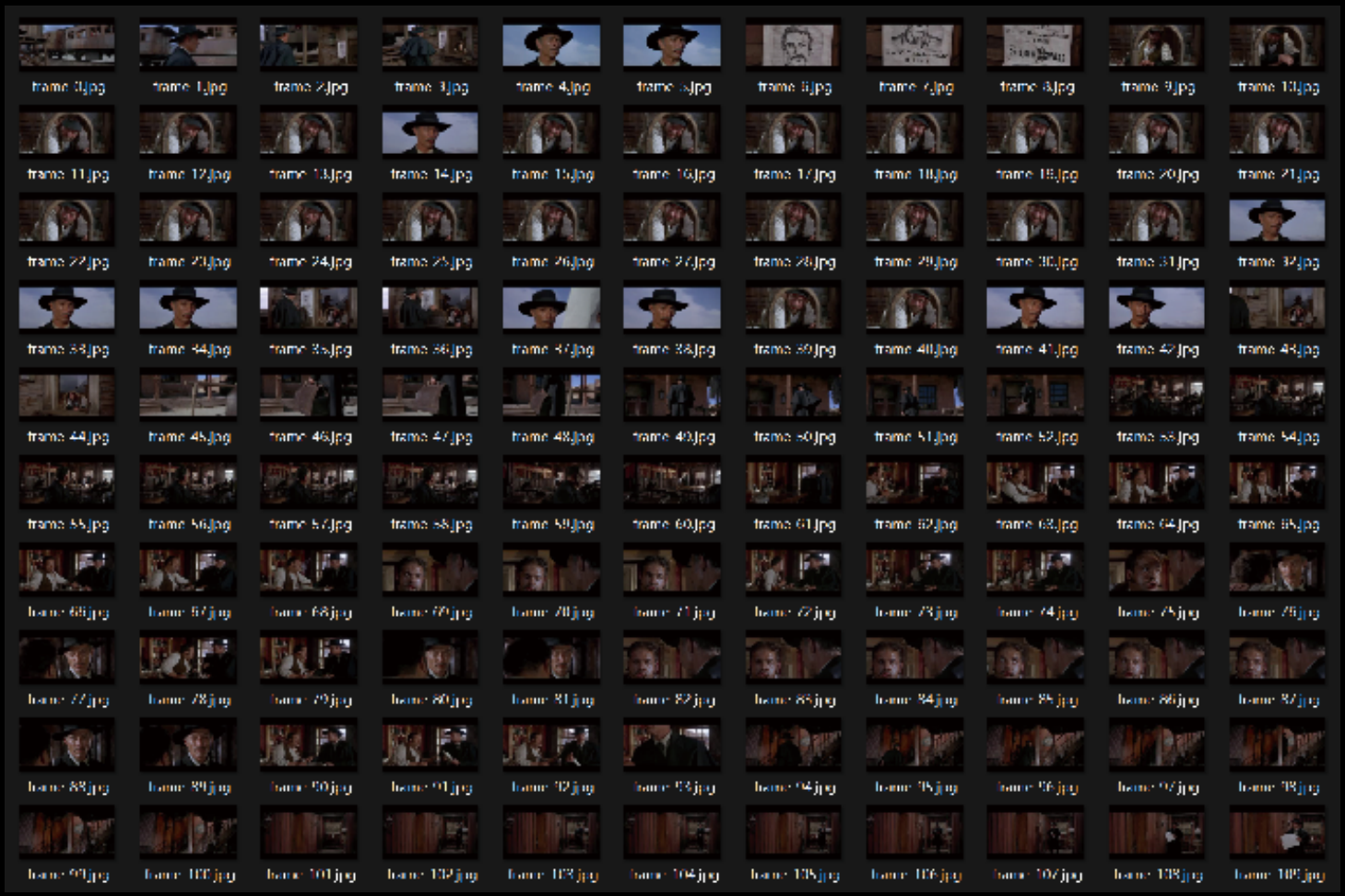
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI
VISUALIZING JSON IN HOUDINI


For_a_Few_Dollars_More.mp4
Houdini_visualizer/Data/Images/5/frame_4.jpg

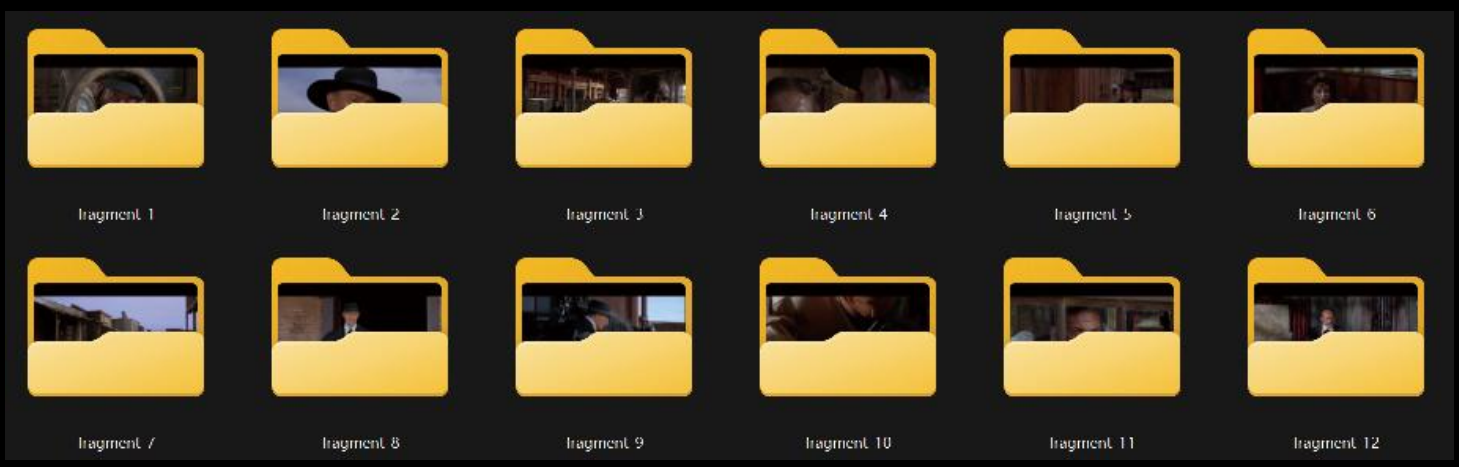
4.1 Extracting video frames

Same as assignment 3, at the beginning I used python code to first download the video from Youtube based on the url and split the video into video frames, in this assignment I split the video into 240 frames in total, which corresponds to 12 subtitles.

The selection of the video was based on our project, There are many places in Madrid that were used to shoot Spaghetti western Film in the last century. Therefore, My group members and I chose the famous one, which named 'For_a_Few_Dollars_More'.



Divide the frames into 12 subfolders for subsequent processing



Next, generate a JSON file from the pickle file using code. Here, I also attempted to manually input video information into a JSON file. The following image is the result of manual input. The information is mainly divided into: Film ID; Paragraphs; Frame_n; Corresponding model; Time; Film path...

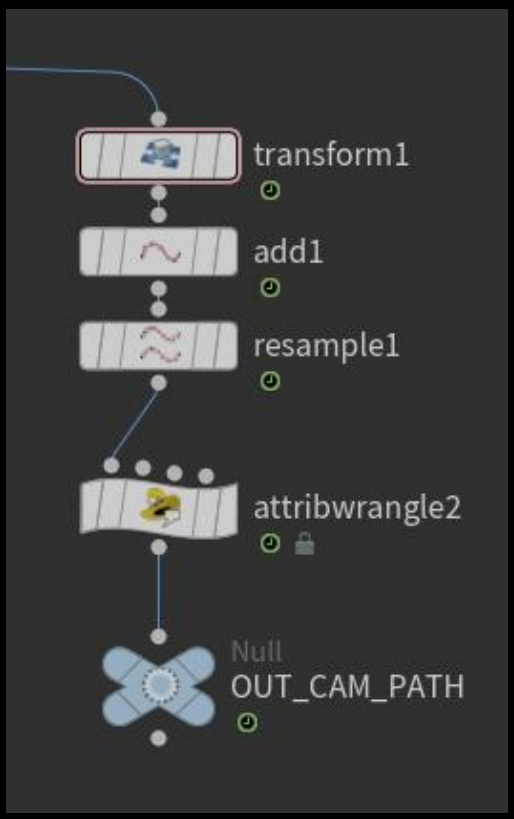

```
{
"folder": [
    {
      "paragraph" : "zeros he was spitting that one short",
      "filmID" : "For a Few Dollars More.mp4",
      "image name" : "frame",
      "model path" : "model.obj",
      "image n" : 33,
      "time" : 27,
      "film path" : "For a Few Dollars More.mp4"
    },

    {
      "paragraph" : "said a measly thousand bucks for me is",
      "filmID" : "For a Few Dollars More.mp4",
      "image name" : "frame",
      "model path" : "",
      "image n" : 17,
      "time" : 32,
      "film path" : "For a Few Dollars More.mp4"
    },

    {
      "paragraph" : "land that's right he said that many out",
      "filmID" : "For a Few Dollars More.mp4",
      "image name" : "frame",
      "model path" : "",
      "image n" : 23,
      "time" : 36,
      "film path" : "For a Few Dollars More.mp4"
    },

    {
      "paragraph" : "of the zeros and thousand original",
      "filmID" : "For a Few Dollars More.mp4",
      "image name" : "frame",
      "model path" : "",
      "image n" : 22,
      "time" : 39,
```

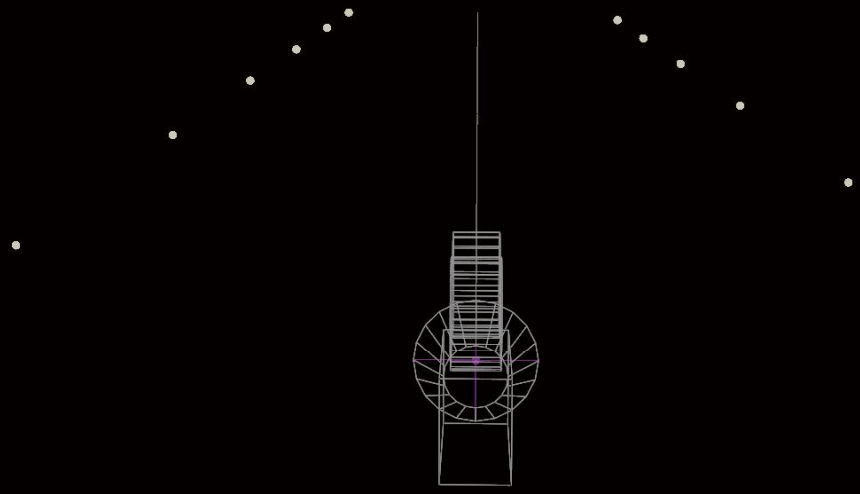
4.2 Extracting video frames

In Houdini, creating a Python modules and import Json files. Next, creating points based on the number of subtitles in the Json file and plan the positions of the points. I will divide these points into two column. In addition, writing code to define and load the infomation of json file (Film ID; Paragraphs; Frame_n...)

4.3 Creating camera path

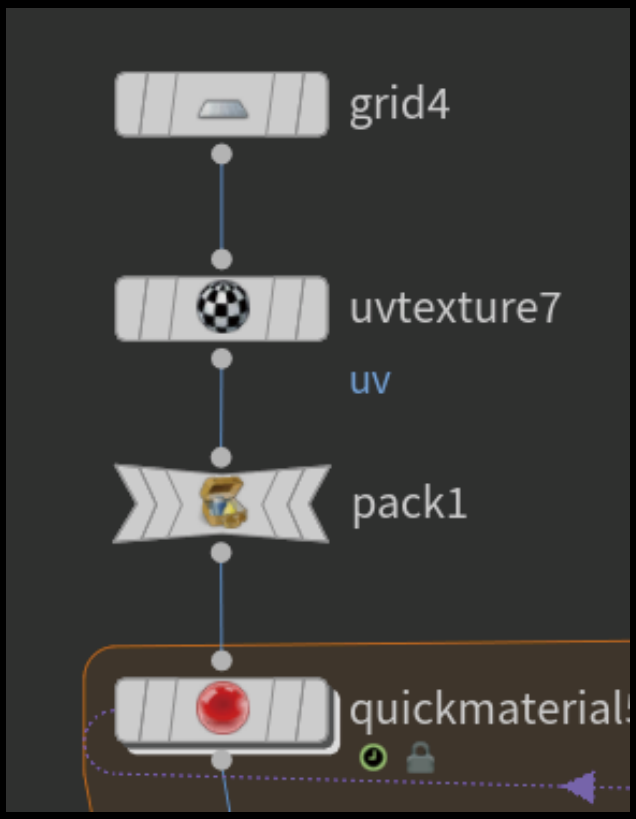
Similar to the method in assessment 3 to create a camera action trajectory. However, the code this time was used to set the route between two points lines.

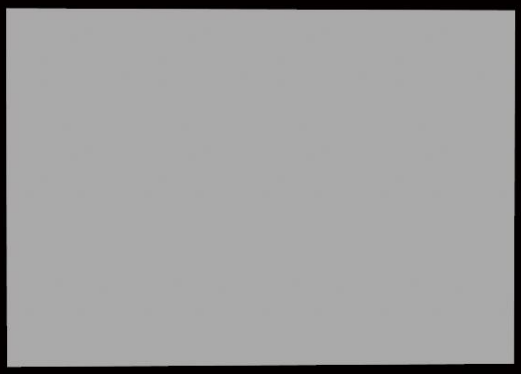




4.4 Input film frames(images)

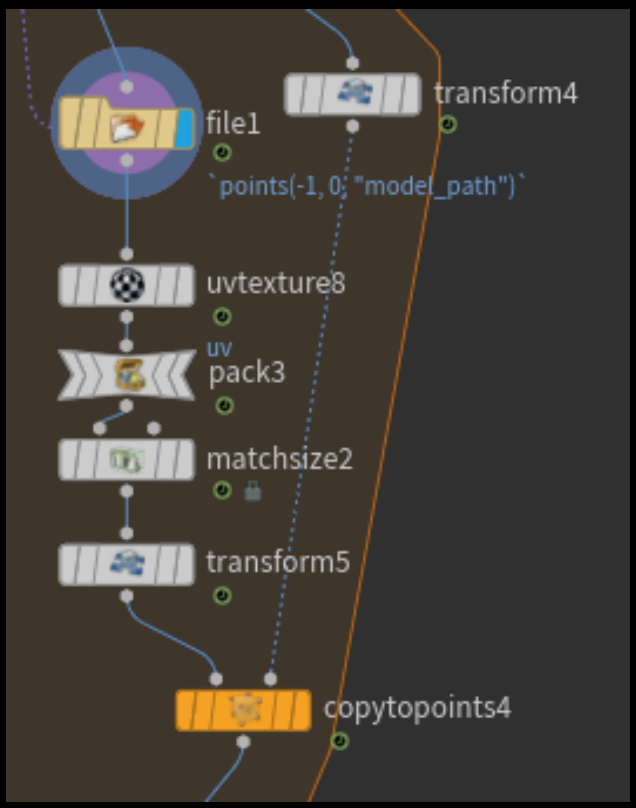
Setting a grid, the UV texture is used to ensure the correct display of images on the grid. The Pack module is used to make the information more clean (orignally, four points of a grid have the information). Then set Quickmaterial to load the frame images.

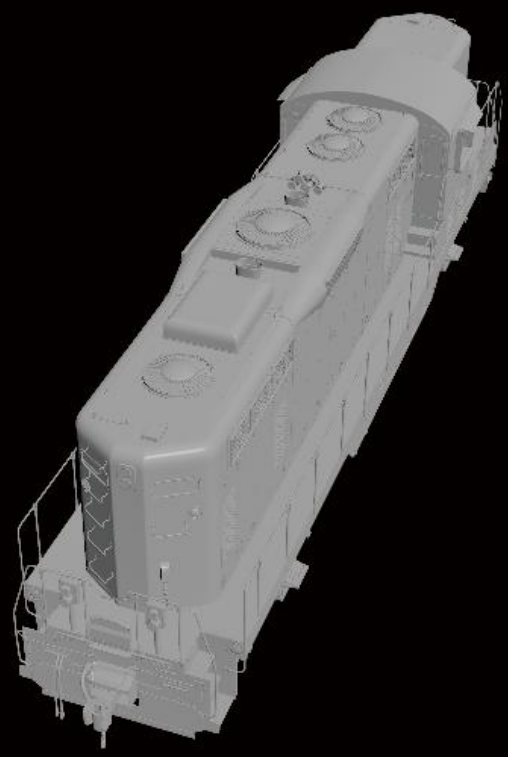




4.5 Model input

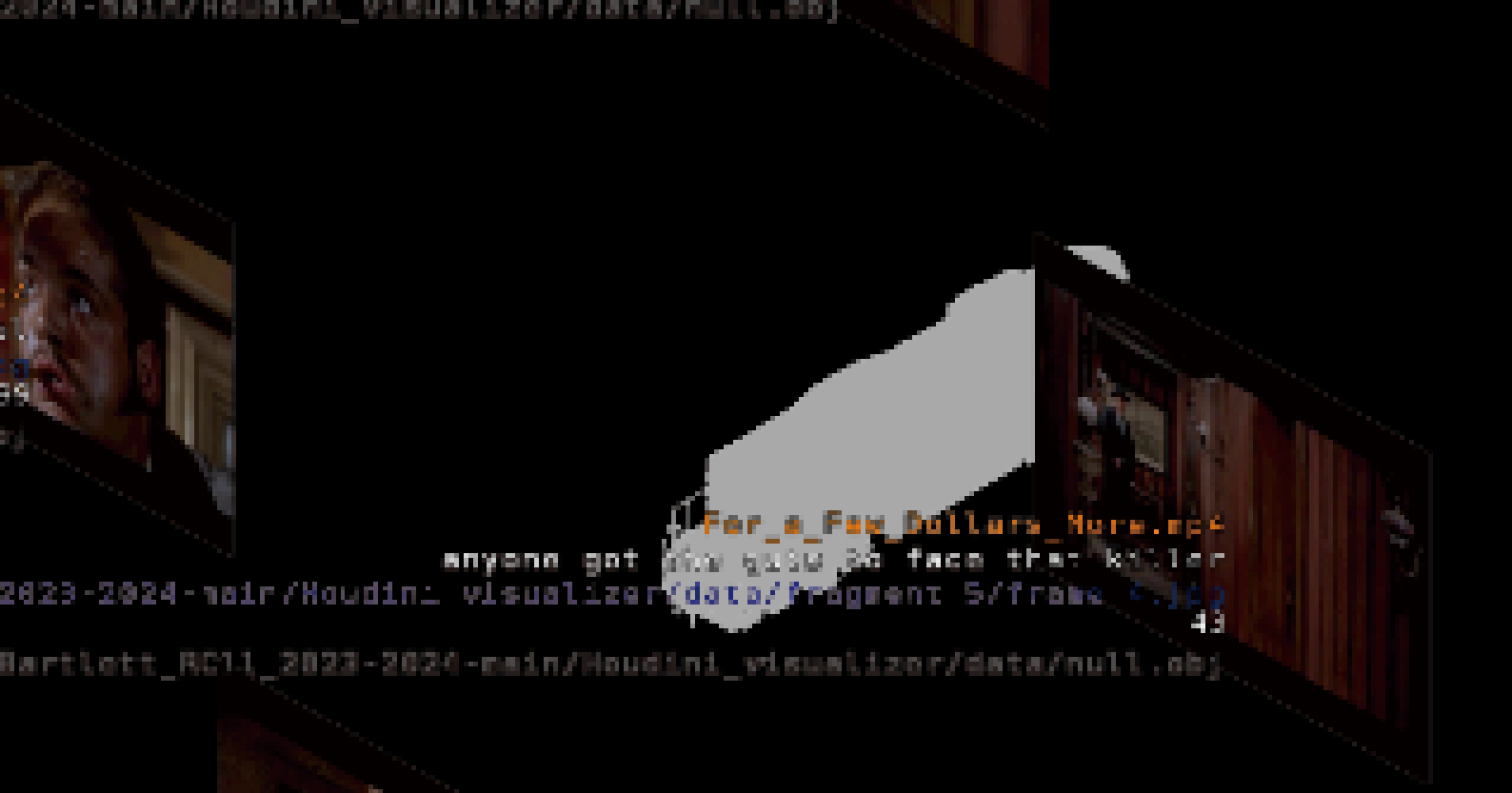
In this assessment, I also add a model by File module. Matchsize and Transform were set to adject the position of model. and the catch those points.





4.6 A COLLECTION OF IMAGES/RENDERS

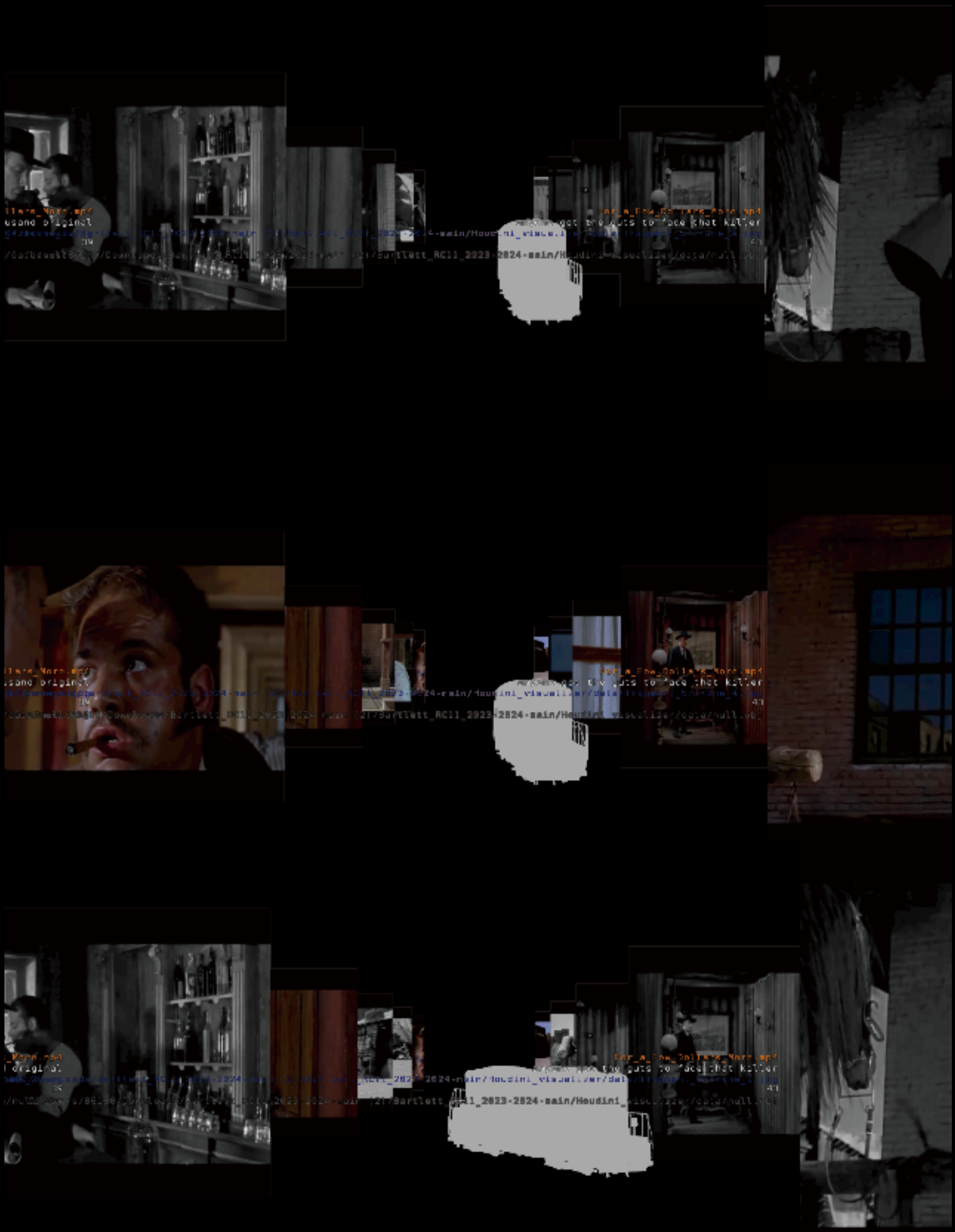
Setting another CAM to see the overview. A series of images rendered.


For_a_Few_Dollars_More.mp4
anyone got how many $ face that killer
2023-2024-main/Houdini_visualizer/data/fragment_5/frame_
43
Bartlett_RC11_2023-2024-main/Houdini_visualizer/data/null.obj


_Few_Dollars_More.mp4
e said that many out
ragment_3/frame_
35
alizer/data/null.obj

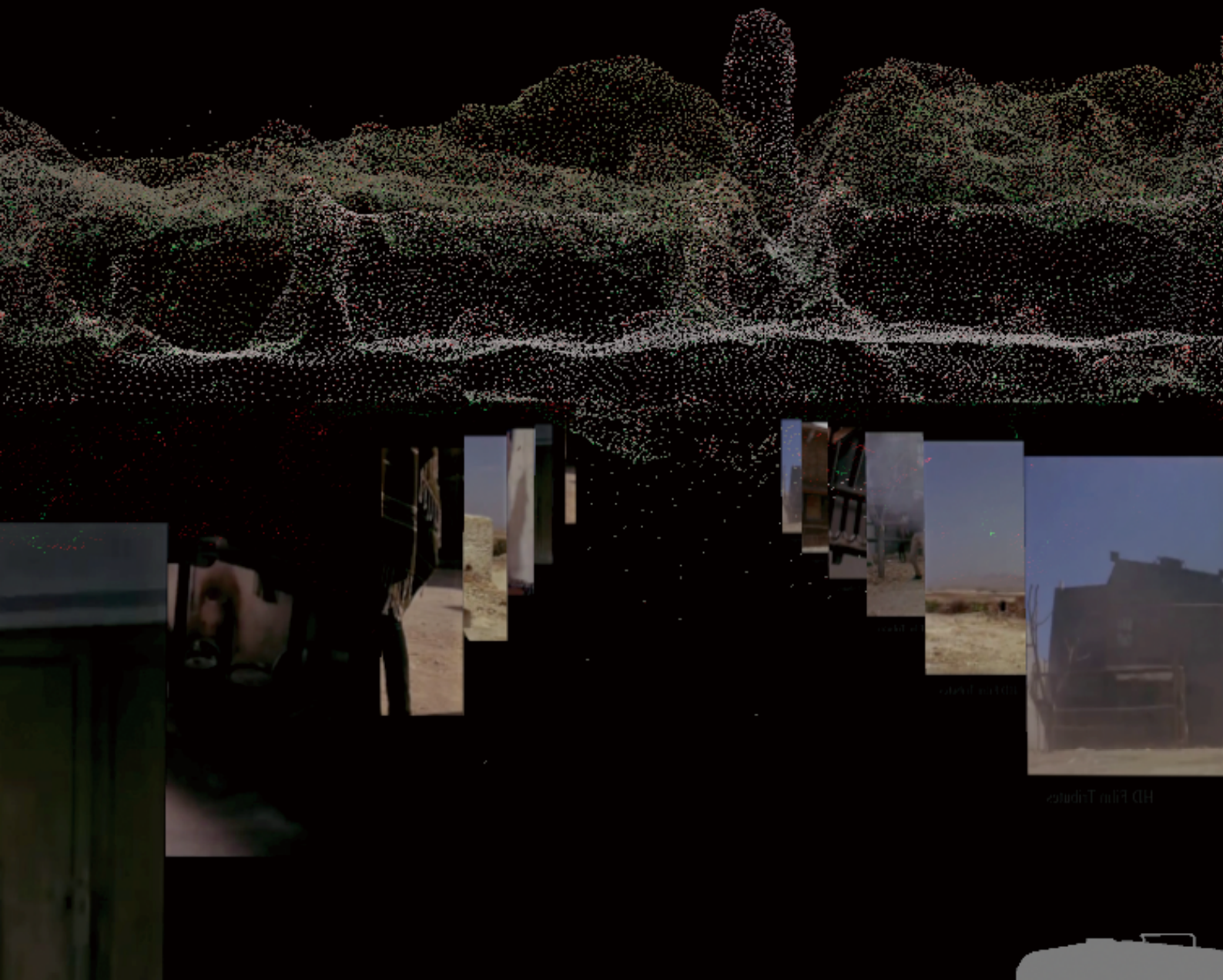
4.6 A COLLECTION OF IMAGES/RENDERS

The view of CAM_1, which following the path set firstly, between two image lines.



5. REPORT SUMMARY

In Houdini workshop, I learned how to apply basic SOP and how to write and use code in Houdini. I also learned about visualizing images, video data, and reconstructing 3D models.After completing the required tasks, I try to combine these skills together. My understanding of these technologies is that they can help designers aggregate information about a site, visualize information/data, and gain a better understanding of the location.



Computability & Complexity_Report

RC11_22161736

BARTLETT 23/24

Exercise One

1. Implement this algorithm in Python.

Use the NumPy ndarray object for your matrices;.

```
import time
import numpy as np


def multiply_matrices(A, B):
    dimension = A.shape[0]
    Result = np.zeros((dimension, dimension), dtype=int)
    for i in range(dimension):
        for j in range(dimension):
            Result[i, j] = sum(A[i, k] * B[k, j] for k in range(dimension))
    return Result
```

Implementation and Comparison

```
Matrix_A = np.array([[1, 2], [3, 4]])
Matrix_B = np.array([[5, 6], [7, 8]])
Matrix_C = np.array([[6, 3, 10, 4], [10, 3, 6, 10], [6, 9, 2, 5], [4, 2, 9, 4]])
Matrix_D = np.array([[i for i in range(1, 6)] for _ in range(5)])

print("Results from matmul with NumPy:")
start = time.time()
print('2 x 2 Matrix:', np.matmul(Matrix_A, Matrix_B))
print('4 x 4 Matrix:', np.matmul(Matrix_C, Matrix_C))
print('5 x 5 Matrix:', np.matmul(Matrix_D, Matrix_D))
run_time = time.time() - start
print('time: ', run_time)

start = time.time()
print("Results obtained by a custom matrix multiplication function:")
print('2 x 2 Matrix:', multiply_matrices(Matrix_A, Matrix_B))
print('4 x 4 Matrix:', multiply_matrices(Matrix_C, Matrix_C))
print('5 x 5 Matrix:', multiply_matrices(Matrix_D, Matrix_D))
run_time = time.time() - start
print('time: ', run_time)

print('Chained matrix multiplication:')
print(np.matmul(np.matmul(Matrix_A, Matrix_B), Matrix_A))
print(multiply_matrices(multiply_matrices(Matrix_A, Matrix_B), Matrix_A))
```

Output:

```
Results from matmul with NumPy:
2 x 2 Matrix: [[19 22]
 [43 50]]
4 x 4 Matrix: [[142 125 134 120]
 [166 113 220 140]
 [158  73 163 144]
 [114 107 106  97]]
5 x 5 Matrix: [[15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]]
time:  0.0
Results obtained by a custom matrix multiplication function:
2 x 2 Matrix: [[19 22]
 [43 50]]
4 x 4 Matrix: [[142 125 134 120]
 [166 113 220 140]
 [158  73 163 144]
 [114 107 106  97]]
5 x 5 Matrix: [[15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]]
time:  0.0009999275207519531
Chained matrix multiplication:
[[ 85 126]
 [193 286]]
[[ 85 126]
 [193 286]]
```

Comparison: custom matrix multiplication function is more efficient(fast).

Exercise Two

1. Describe and explain the algorithm. It should contain at least the following:

How is it recursive:It is recursive because it calls itself to perform the task of matrix multiplication on smaller sub-problems. Each Matrix A &B is divided into 4 sub_matrices and then these sub_matrices are used as inputs for further recursive calls.

The base case of the recursion occurs when the matrix dimension n = 1.The matrix multiplication is trivial at the point, consisting of multiplying two scalar value a11 and b11 to produce c11. This would stop the further recursive calls.

How does the recursion step reduce to the base case:Each recursive call processes a smaller sub_matrix of size n/2 * n/2, reducing the problem size with each division until the base case n = 1 is reached. The division continues recursively until the size of the matrices being multplied is just a single element.

2. Implement the recursive algorithm in Python.

```
import numpy as np
```

```
def partition_matrix(matrix):
    n = len(matrix)
    mid = n // 2
    top_left = [row[:mid] for row in matrix[:mid]]
    top_right = [row[mid:] for row in matrix[:mid]]
    bottom_left = [row[:mid] for row in matrix[mid:]]
    bottom_right = [row[mid:] for row in matrix[mid:]]
    return top_left, top_right, bottom_left, bottom_right

def add_matrices(A, B):
    return [[A[i][j] + B[i][j] for j in range(len(A[i]))] for i in range(len(A))]

def combine_matrices(top_left, top_right, bottom_left, bottom_right):
    top_half = [left + right for left, right in zip(top_left, top_right)]
    bottom_half = [left + right for left, right in zip(bottom_left, bottom_right)]
    return top_half + bottom_half

def recursive_multiply_matrices(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        A11, A12, A21, A22 = partition_matrix(A)
        B11, B12, B21, B22 = partition_matrix(B)
        M1 = add_matrices(recursive_multiply_matrices(A11, B11), recursive_multiply_matrices(A12, B21))
        M2 = add_matrices(recursive_multiply_matrices(A11, B12), recursive_multiply_matrices(A12, B22))
        M3 = add_matrices(recursive_multiply_matrices(A21, B11), recursive_multiply_matrices(A22, B21))
        M4 = add_matrices(recursive_multiply_matrices(A21, B12), recursive_multiply_matrices(A22, B22))
        return combine_matrices(M1, M2, M3, M4)
```

```
Matrix_E = np.array([[1, 2], [3, 4]])
Matrix_F = np.array([[5, 6], [7, 8]])
Matrix_G = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])

print("Results using recursive matrix multiplication:")
print('2 x 2 Matrix:', recursive_multiply_matrices(Matrix_E, Matrix_F))
print('4 x 4 Matrix:', recursive_multiply_matrices(Matrix_G, Matrix_G))

print('Chained recursive matrix multiplication (2 x 2 Matrix):')
chained_result = recursive_multiply_matrices(recursive_multiply_matrices(Matrix_E, Matrix_F), Matrix_E)
print(chained_result)
```

```
Results using recursive matrix multiplication:
2 x 2 Matrix: [[19, 22], [43, 50]]
4 x 4 Matrix: [[90, 100, 110, 120], [202, 228, 254, 280], [314, 356, 398, 440], [426, 484, 542, 600]]
Chained recursive matrix multiplication (2 x 2 Matrix):
[[85, 126], [193, 286]]
```

Test and compare the practical speed with the non-recursive algorithm

```
import numpy as np
import time

# Create a large 64x64 matrix
Large_Matrix = np.array([[i for i in range(1, 65)] for _ in range(64)])

# Measure the time taken by the basic matrix multiplication
start_time = time.time()
result_basic = multiply_matrices(Large_Matrix, Large_Matrix)
basic_mul_time = time.time() - start_time

# Measure the time taken by the recursive matrix multiplication
start_time = time.time()
result_recursive = recursive_multiply_matrices(Large_Matrix.tolist(), Large_Matrix.tolist())
recursive_mul_time = time.time() - start_time

print(f"Basic multiply time: {basic_mul_time:.4f} s, Recursive multiply time: {recursive_mul_time:.4f} s")
```

```
Basic multiply time: 0.0703 s, Recursive multiply time: 0.3292 s
```

3. Do a complexity analysis for the SMMRec algorithm.

Divide step: Partition each matrix of size n × n into four submatrices, each of size is (n/2)* (n/2).

Conquer step: Recursively multiply these smaller matrices until the base case is reached 1×1 matrices, multiplication is very straightforward at this point.

Combine: The results of these recursive multiplications are summed where necessary and then combined to form the resulting matrix C.

For the recursive matrix multiplication, each decomposition creates four subproblems, each half the size of the primitive problem. For each pair of submatrices, we perform 8 multiplications. The recursive property produces a time complexity recursive relation represented as: $T(n)=8T(2/n)+\Theta(n^2)$

Exercise Three

1. Reflect on the difference between (complexity of) addition/subtraction and multiplication on matrices.

Implement and test the algorithm

```
import numpy as np
import time

def enhanced_matrix_split(matrix):
    size = len(matrix)
    mid_point = size // 2
    quadrant1 = [row[:mid_point] for row in matrix[:mid_point]]
    quadrant2 = [row[mid_point:] for row in matrix[:mid_point]]
    quadrant3 = [row[:mid_point] for row in matrix[mid_point:]]
    quadrant4 = [row[mid_point:] for row in matrix[mid_point:]]
    return quadrant1, quadrant2, quadrant3, quadrant4

def matrix_operation_add(submatrix1, submatrix2):
    return [[submatrix1[i][j] + submatrix2[i][j] for j in range(len(submatrix1))] for i in range(len(submatrix1))]

def matrix_operation_subtract(submatrix1, submatrix2):
    return [[submatrix1[i][j] - submatrix2[i][j] for j in range(len(submatrix1))] for i in range(len(submatrix1))]

def assemble_matrix(q1, q2, q3, q4):
    upper_half = [left + right for left, right in zip(q1, q2)]
    lower_half = [left + right for left, right in zip(q3, q4)]
    return upper_half + lower_half

def optimized_strassen_matrix_multiply(A, B):
    n = len(A)
    if n <= 2:  # Use the direct method for small matrices
        return recursive_multiply_matrices(A, B)
    else:
        mid = n // 2
        A11, A12, A21, A22 = enhanced_matrix_split(A)
        B11, B12, B21, B22 = enhanced_matrix_split(B)

        S1 = matrix_operation_add(A11, A22)
        S2 = matrix_operation_add(B11, B22)
        S3 = matrix_operation_add(A21, A22)
        S4 = B11
        S5 = A11
        S6 = matrix_operation_subtract(B12, B22)
        S7 = A22
        S8 = matrix_operation_subtract(B21, B11)
        S9 = matrix_operation_add(A11, A12)
        S10 = B22
        P1 = optimized_strassen_matrix_multiply(S1, S2)
        P2 = optimized_strassen_matrix_multiply(S3, S4)
        P3 = optimized_strassen_matrix_multiply(S5, S6)
        P4 = optimized_strassen_matrix_multiply(S7, S8)
        P5 = optimized_strassen_matrix_multiply(S9, S10)
        Q1 = matrix_operation_add(matrix_operation_subtract(matrix_operation_add(P1, P4), P5), P2)
        Q2 = matrix_operation_add(P3, P5)
        Q3 = matrix_operation_add(P2, P4)
        Q4 = matrix_operation_subtract(matrix_operation_subtract(matrix_operation_add(P1, P3), P2), P5)

        return assemble_matrix(Q1, Q2, Q3, Q4)
```

```
Matrix_H = np.array([[1, 2], [3, 4]])
Matrix_I = np.array([[5, 6], [7, 8]])
Matrix_J = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])

print("Results using Strassen's algorithm:")
print('2 x 2 Matrix:', optimized_strassen_matrix_multiply(Matrix_H, Matrix_I))
print('4 x 4 Matrix:', optimized_strassen_matrix_multiply(Matrix_J, Matrix_J))
```

```
Results using Strassen's algorithm:
2 x 2 Matrix: [[19, 22], [43, 50]]
4 x 4 Matrix: [[604, 688, 110, 120], [764, 880, 254, 280], [314, 356, 136, 136], [426, 484, 72, 72]]
```

Compare the practical speed with the recursive algorithm

```
# Create a 128x128 matrix
Performance_Matrix = np.array([[i for i in range(1, 129)] for _ in range(128)])

# Start the timer for basic matrix multiplication
start_time = time.time()
result_from_basic = multiply_matrices(Performance_Matrix, Performance_Matrix)
time_for_basic = time.time() - start_time  # Calculate elapsed time

# Start the timer for recursive matrix multiplication
start_time = time.time()
result_from_recursive = recursive_multiply_matrices(Performance_Matrix.tolist(), Performance_Matrix.tolist())
time_for_recursive = time.time() - start_time  # Calculate elapsed time

print(f"Basic multiplication: {time_for_basic:.4f} s, Recursive multiplication: {time_for_recursive:.4f} s")
```

```
Basic multiplication: 0.5559 s, Recursive multiplication: 2.6781 s
```

2.Do a complexity analysis of the Strassen algorithm.

Standard Recursive Algorithm: The recurrence relation is T(n) = 8T(n/2)+Θ(n^2). This recurrence solves to O(n^3), indicating a cubic time comlexity typical for standard matrix multiplication according to the Master Theorem.

Strassen's Algorithm: In Strassen's method, the number of recursive calls is reduced to 7, instead of 8 as in the standard method. The recurrence relation for Strassen's algorithm is T(n) = 7T(n/2)+(n^2). Using the Master Theorem again, the solution to this recurrence is O(n^log7^2), which is aprroximately O(n^2.81). This demonstrates a more effcient algorithm than the standard approach, due to a lower exponent in the time complexity.

The Jupyter notebook file uploaded on Github

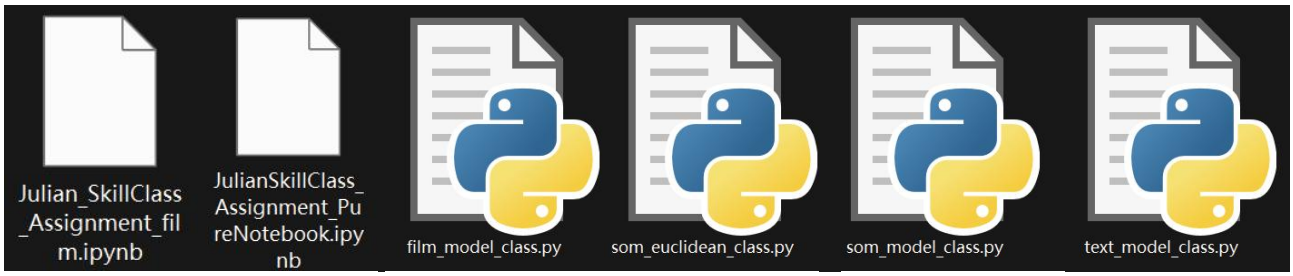
Coding Explanation

RC11_22161736

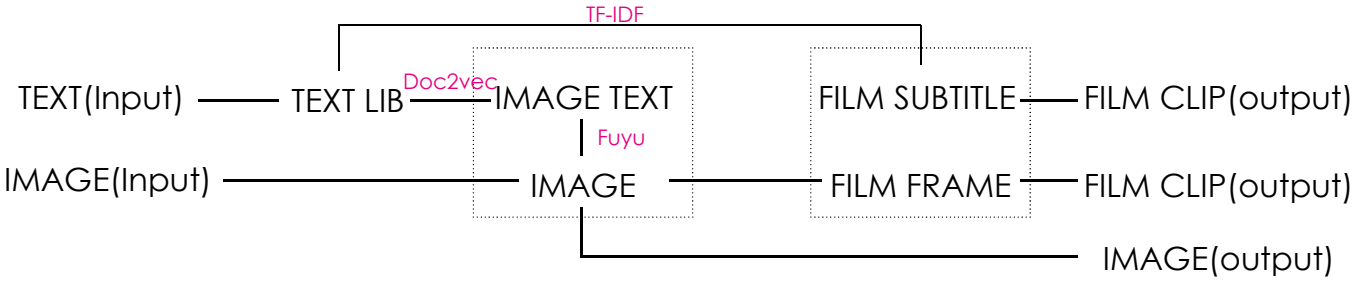
BARTLETT 23/24

In this task, I have written 2 main Jupyter notebook files and 4 integrated '. *py*'s for the running code The file package used to call functions.

The first notebook is a work notebook that contains all the initial code, defined functions, inputs, and detailed steps on how to obtain train data or train SOMs. Anther one (Quick Search) is the integrated code document of the former, which more efficiently and clearly displays the structure and output effect of the code.



In this assignment, I attempted to construct 5 different data SOMs (A-E), including "text", "FuyuImage text", "image library images", "FilmSubtitle text", and "FilmFrames images". I try to connecting while integrating these datasets within 3 domains (text, image, video). The specific vectorization process (fig.)



Input (Text):

1. Using query activates the novel text SOMs to obtain the most similar paragraphs.
2. Finding the image index and description of Fuyu text(from imagelib image) by searching for novel paragraphs in the text.
3. Searching for novel paragraphs of movie subtitles, find the most similar subtitles, and display those/that movie segment.

Input (Image):

1. Input an image to activate the trained image SOM, and then obtain the most similar image and its supporting description text (from Fuyu).
2. Search for matching images in movie frames to find the most similar frame and display the movie clip.
3. Search for matching image description texts in the novel text SOM in order to obtain the most similar novel paragraphs.

PART I: Build SOM Datasets

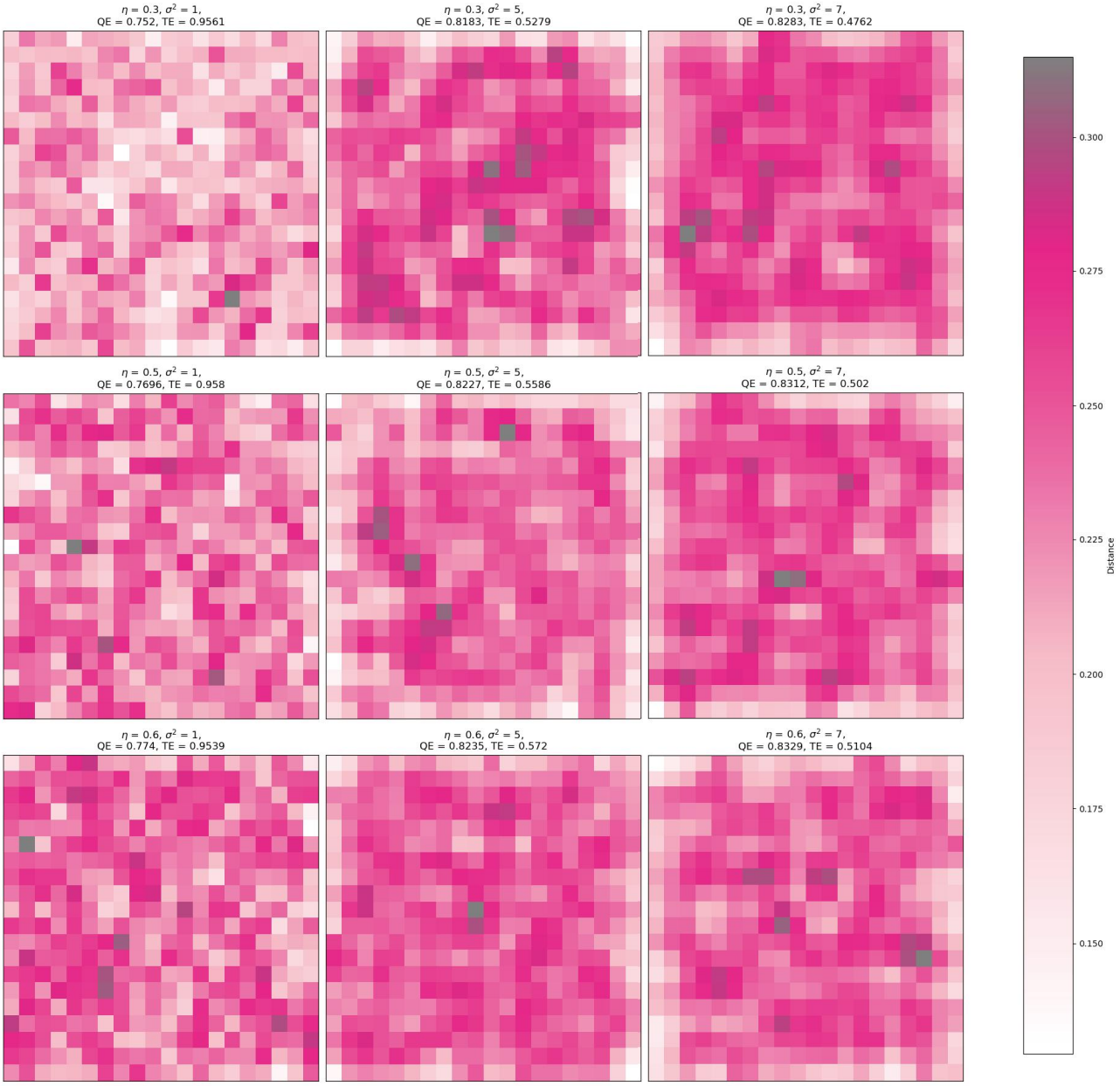
1.1 Create novel text som and searching function

At the first stage, I wrote coding to read Epub files and vectorize them. Loading novel text from the path and import it using TextModel from text_model_class. py to vectorize it, it choose using two different vectorization modes. Here, I use "doc2vec" as the vectorization mode

```
NOVEL_DIR=r'C:\Users\86180\python_Julian_assessment\image_to_text\Datasets\Text'
NOVEL_FILES = []
NOVEL_FILES.extend(glob.glob(os.path.join(NOVEL_DIR, '*.epub')))
```

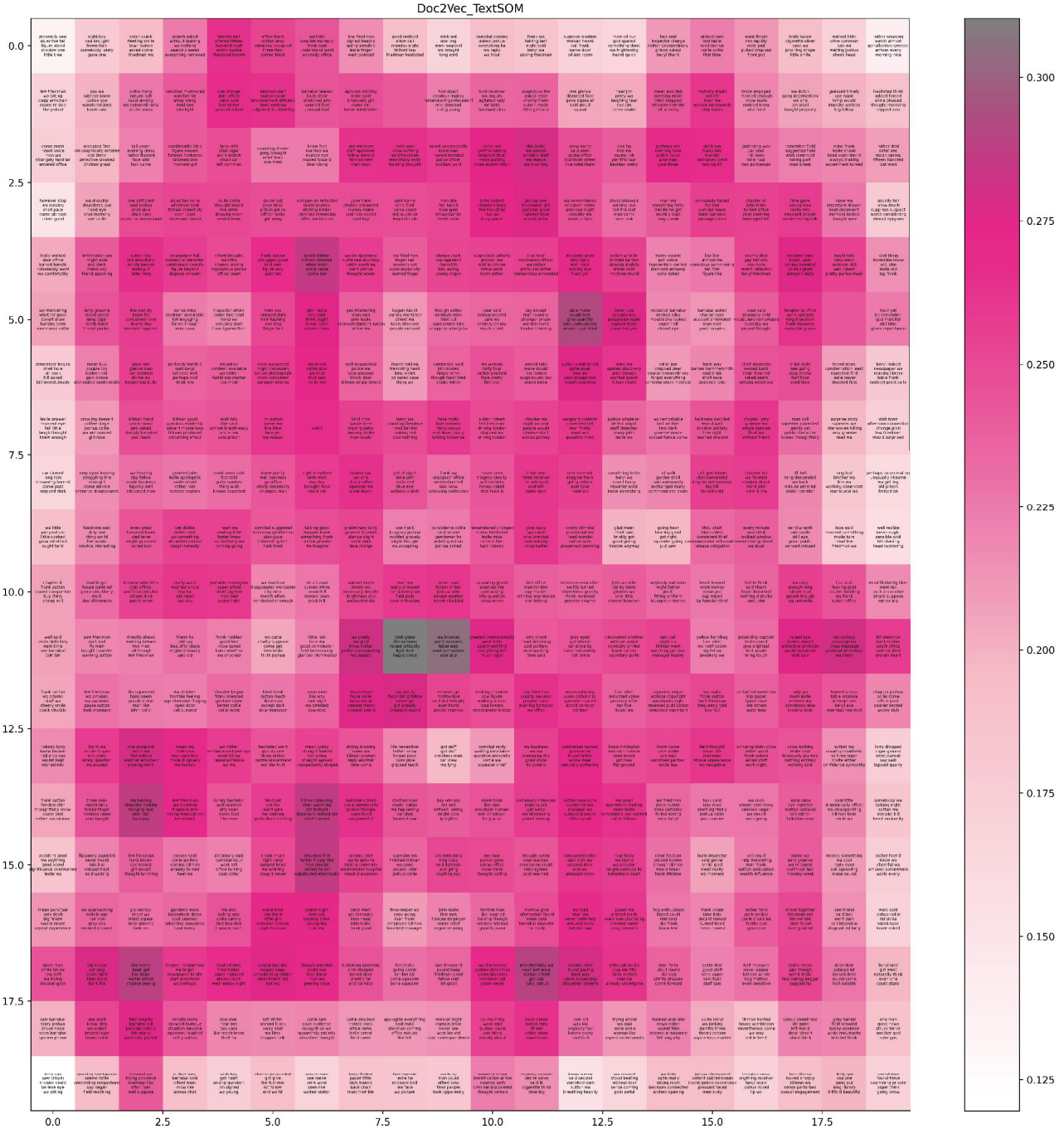
```
novel_model = TextModel(NOVEL_FILES,vectorization='lsa', dimension = 300, min_df=3)
```

Creating SOM with Doc2vec, 300 dimension:



I chose SOM [5] for further preocess, because its QE&TE are relatively low.

The U-Matrix mapping with texts of SOMs[5]



Then I am going to link paragraphs to their BMUs and create a data_dict to track the information (info). I have defined a Function here that applies to whole text datasets in this program.

```
def get_TextAndVector_dict(som, original_paragraph_list, train_data):
    data_dict=[]
    for i in range(len(som)):
        row = []
        for j in range(len(som[0])):
            row.append([])
        data_dict.append(row)

    vectortextPairs=[]
    for i in range(0,len(original_paragraph_list)):
        vectortext={}
        vectortext['text']=original_paragraph_list[i]
        vectortext['vector']=train_data[i]
        vectortextPairs.append(vectortext)
    for i in vectortextPairs:
        g,h =SOMlib.find_BMU1(som,i['vector'])
        data_dict[g][h].append(i)

    return data_dict
```

```
data_dict=get_TextAndVector_dict(som,paragraphs_list,doc2vec_train_data)
```

```
pickle_path=r'C:\Users\86180\python_Julian_assessment\image_to_text\Datasets\pickle\data_dict.pkl'

with open(pickle_path, 'wb') as f:
    pickle.dump(data_dict, f)
```

```
with open(pickle_path, 'rb') as f:
    data_dict = pickle.load(f)
```

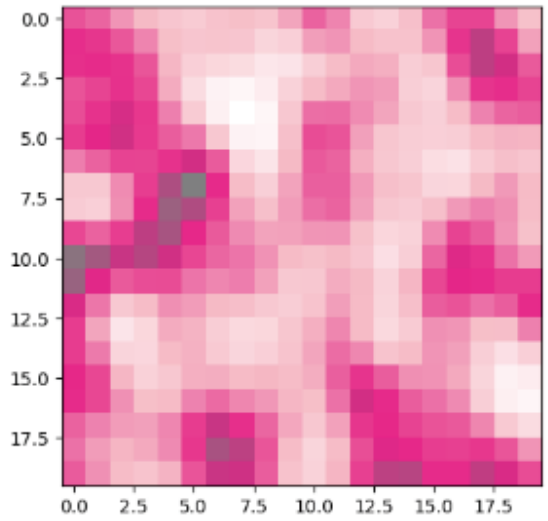
After that, I created a Searching Function, which can search text queries for SOM, returning the result with the minimum cosine distance from the lattice vector in BMU, and printing it out. Meanwhile, it will display the activated SOM.

```
def search_TextSom(som, model,data_dict,query='street',vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(train_data,som, model.vectorize(query)), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query))
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    elif vectorization =='lsa':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(train_data,som, model.vectorize(query)[0]), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query)[0])
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    similarities = {}
    for i in data_dict[g][h]:
        vector_array = np.array([i['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarities[i['text']['paragraph']] = cosine_similarity(vector_array, [som[g][h]])

    biggest = None
    max_similarity = similarities[data_dict[g][h][0]['text']['paragraph']]
    for i in data_dict[g][h]:
        sim = similarities[i['text']['paragraph']]
        if sim > max_similarity:
            max_similarity = sim
            biggest = i['text']
    result.append(biggest)
    return result
```

```
def search_TextSom(som, model, data_dict, query='street', vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(train_data, som, model.vectorize(query)), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query))
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    elif vectorization =='lsa':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(train_data, som, model.vectorize(query)[0]), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query)[0])
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    similarities = {}
    for i in data_dict[g][h]:
        vector_array = np.array([i['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarities[i['text']['paragraph']] = cosine_similarity(vector_array, [som[g][h]])

    biggest = None
    max_similarity = similarities[data_dict[g][h][0]['text']['paragraph']]
    for i in data_dict[g][h]:
        sim = similarities[i['text']['paragraph']]
        if sim > max_similarity:
            max_similarity = sim
            biggest = i['text']
    result.append(biggest)
    return result
```

```
query=query_list[10]['query']
print('The matching query is: ' + query)

train_data=doc2vec_train_data
search_TextSom(som, doc2vec_novel_model, data_dict, query, vectorization ='doc2vec')
```

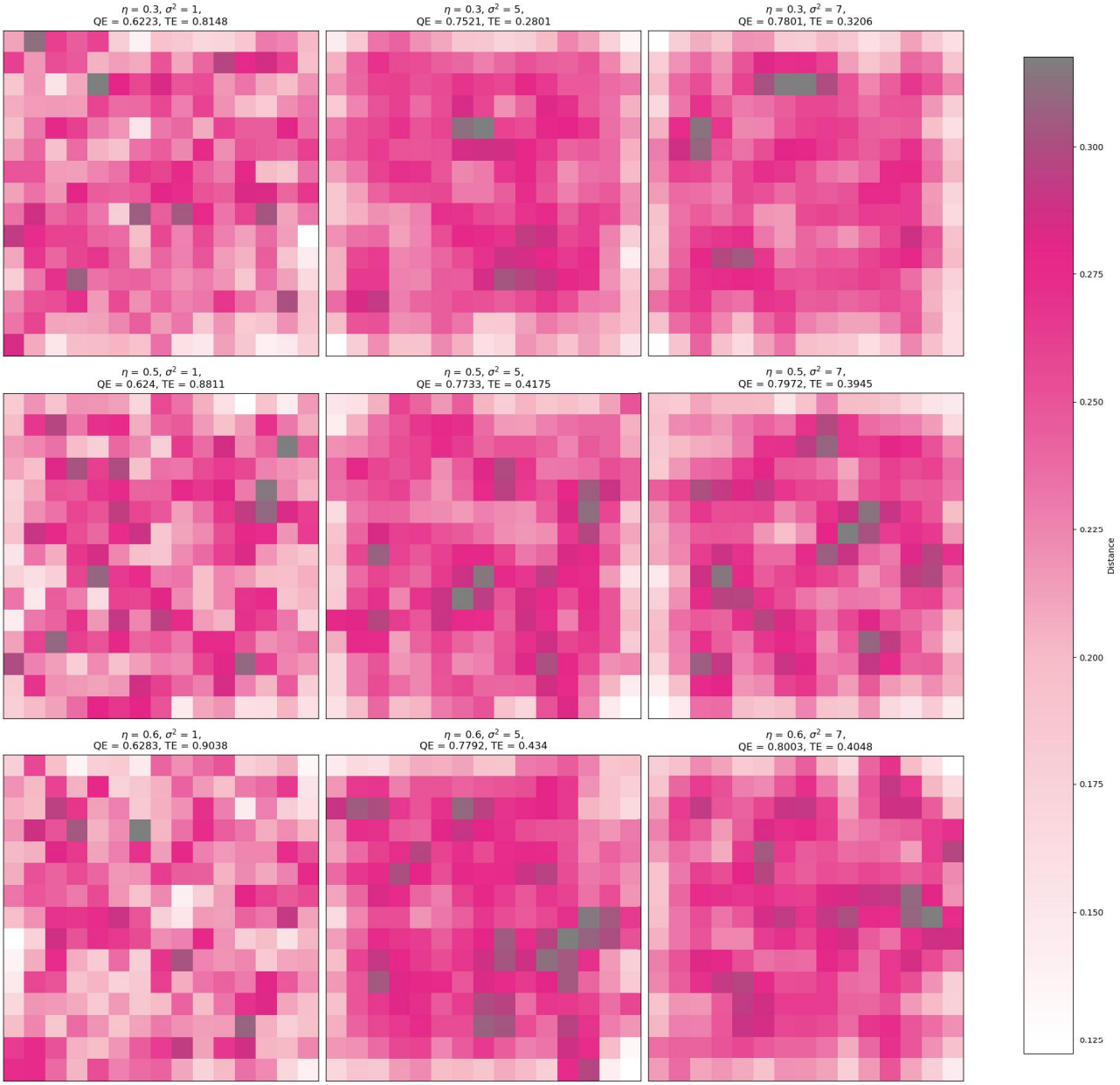
```
The matching query is: The colors nature displays at sunset are truly breathtaking.
```



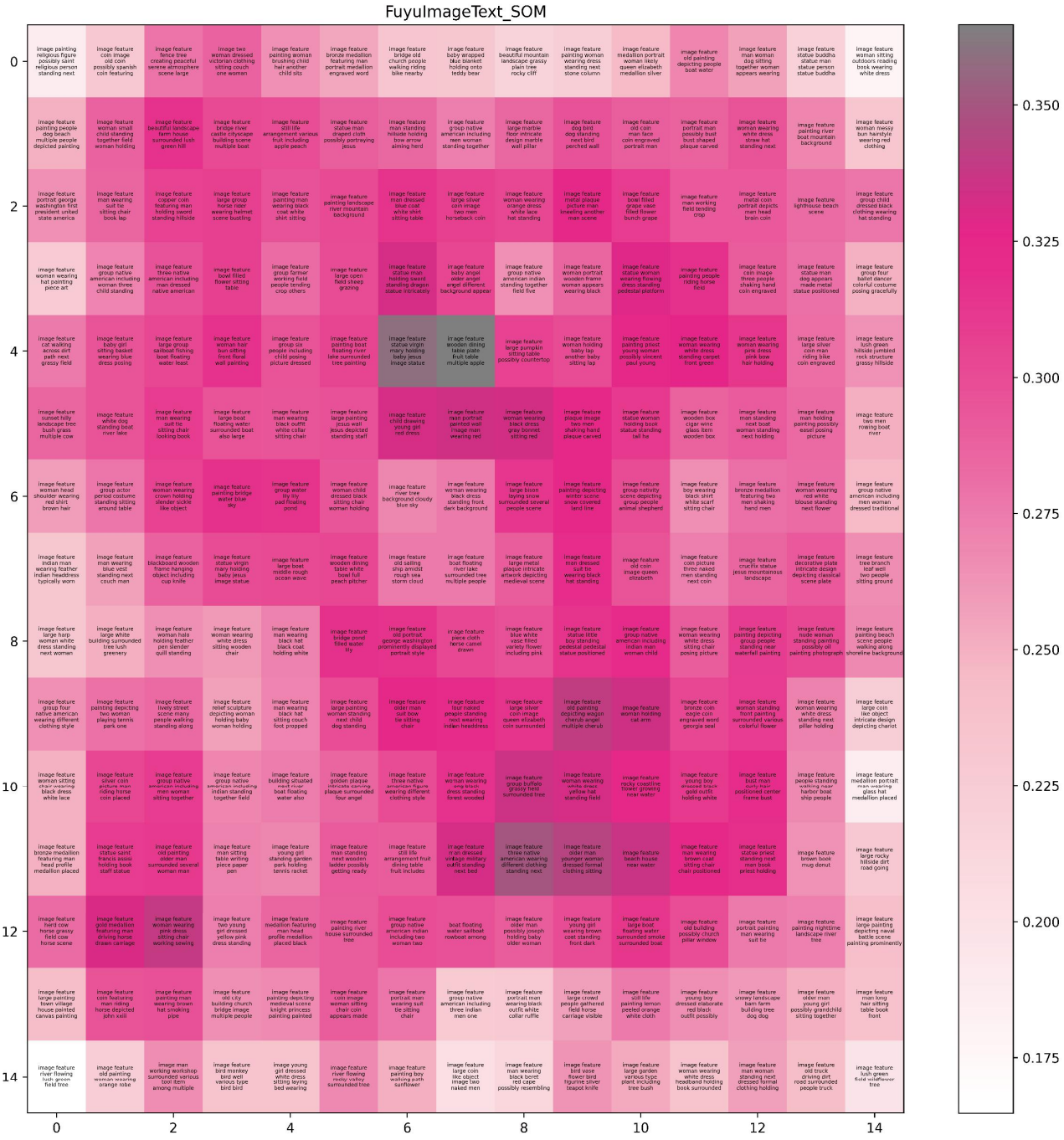

```
The best matching unit(bmu) is: (7, 5)
```

```
[{'paragraph': '  'D' you know,'  said Adam.  'I don' t believe that I' m ever going to get that fortune.'  'Well, I don' t see that you' v
e very much to complain of,'  said Archie.  'You' re no worse off than you were. I' ve lost a fiver and five bottles of champagne.',
  'nr': '1035',
  'ID': 'waugh-vile-bodies',
  'type': 'epub'}]
```

1.2 Next, I created image support text SOM and then loading ImageLibrary_Text.csv with search function. The following specific steps are similar to the foregoing steps.



The U-Matrix mapping with texts of SOMs[1]

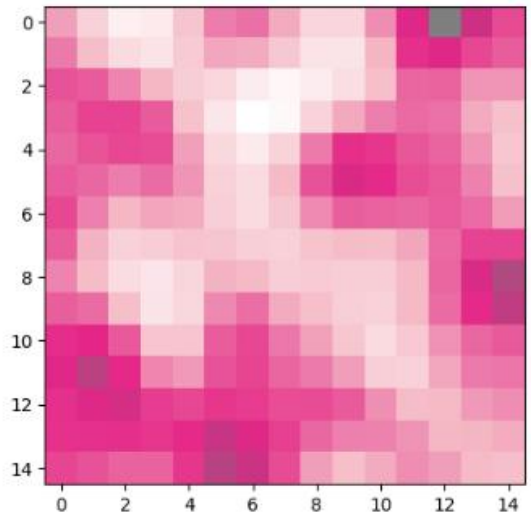


Searching function again:

```
def search_ImageTextSom(som, model,data_dict,image_folder,query='street',vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(ImageText_train_data,som, model.vectorize(query)), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query))
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    elif vectorization =='lsa':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(ImageText_train_data,som, model.vectorize(query)[0]), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query)[0])
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    similarities = {}
    for i in data_dict[g][h]:
        vector_array = np.array([i['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarities[i['text']['text']] = cosine_similarity(vector_array, [som[g][h]])

    biggest = None
    max_similarity = similarities[data_dict[g][h][0]['text']['text']]
    for i in data_dict[g][h]:
        sim = similarities[i['text']['text']]
        if sim > max_similarity:
            max_similarity = sim
            biggest = i['text']
    result.append(biggest)

    # print the image
    if result[0] is not None:
        image_name=result[0]['filename']
        image_path = os.path.join(image_folder, image_name)
        img = Image.open(image_path)
        display(img)

    return result
```

```
ImageLibrary_folder=r'C:\Users\86180\python_Julian_assessment\image_to_text\Datasets\Image'

query=query_list[10]['query']
print('The matching query is: ' + query)

search_ImageTextSom(ImageText_som, ImageText_model, FuyuText_data_dict, ImageLibrary_folder, query, vectorization ='doc2vec')
```

```
The matching query is: The colors nature displays at sunset are truly breathtaking.
```




```
The best matching unit(bmu) is: (0, 12)
```

```
[{'filename': '1800.jpg',
  'text': 'The image features a group of wild animals, including elk or deer, in a grassy field.\n\nIn the background, there is a forest wit
h trees. The scene is framed by an oval-shaped frame, adding an interesting touch to the picture. The animals are scattered throughout the f
ield, with some standing and others running. The forest in the background adds'}]
```

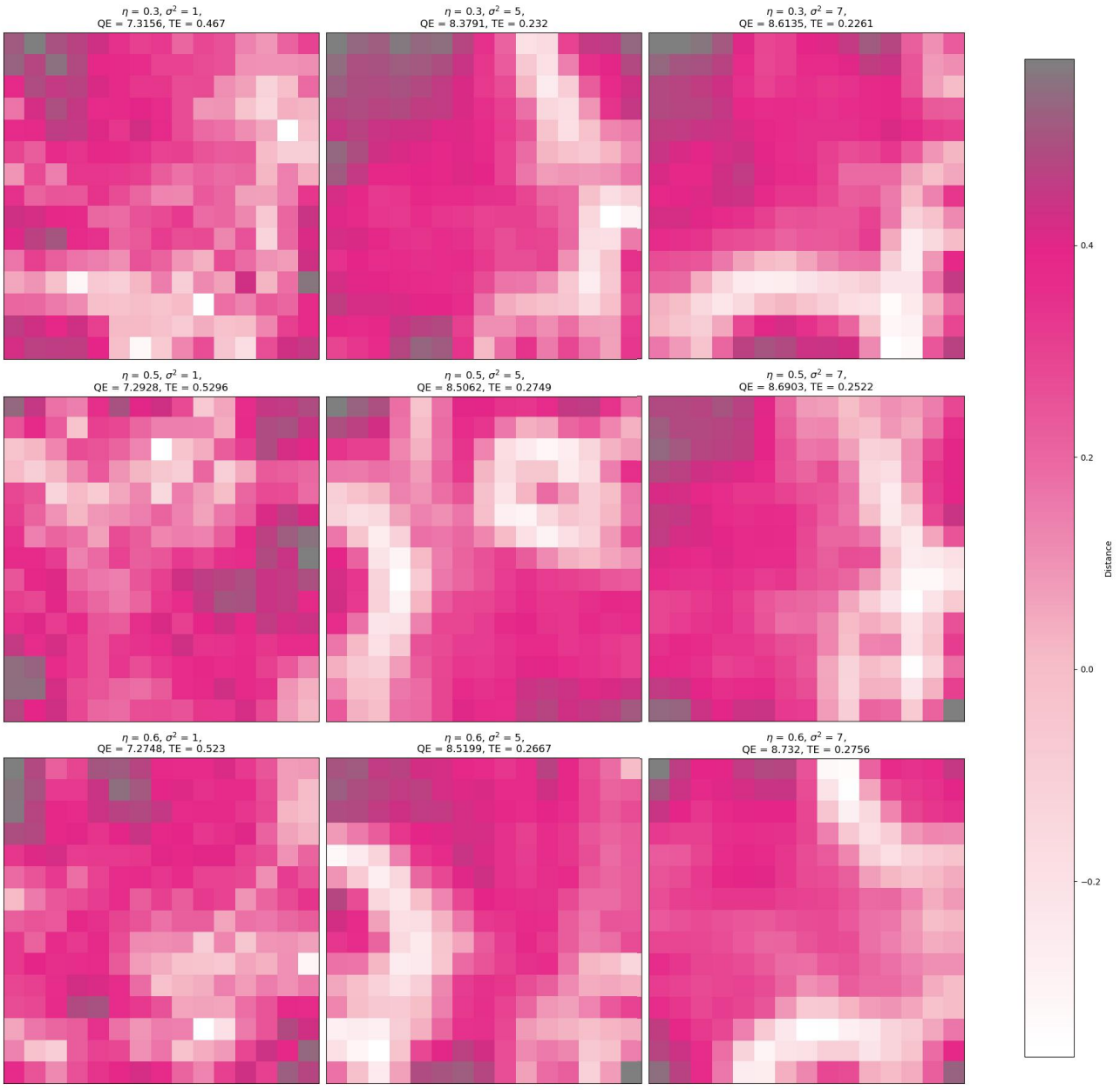
1.3 Image SOM Creation

```
ImageModel = tf.keras.applications.mobilenet.MobileNet(
# The 3 is the three dimensions of the input: r, g, b.
  input_shape=(224, 224, 3),
  include_top=False,
  pooling='avg'
)
```

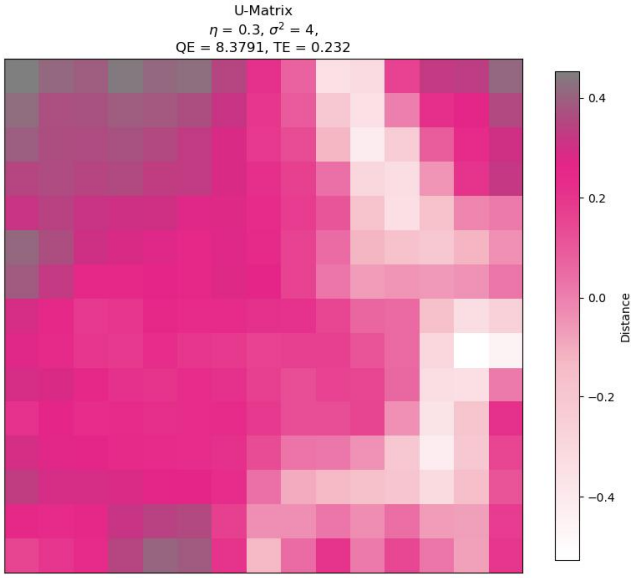
```
Image_path=r'C:\Users\86180\python_Julian_assessment\image_to_text\Datasets\Image'
ImageFiles = os.listdir(Image_path)
```

```
def processImage(imagePath, model):
    im = load_image(imagePath)
    f = model.predict(im)[0]
    return f
```

This step is creating image SOM and Searching function.
SOMs created with 1024 dimension and normalize data before training.
Euclidean distance calculation is better than cosine for 1024 dimension, so I imported the function from som_euclidean_class.py,which related with eudlidean distance.



SOMs[1] output:



The Searching function here is for searching imgae through a imge:

```
def search_ImageSom(path, model, som):
    result=[]
    query_features=[]

    #print query image
    img = Image.open(path)
    display(img)

    q_f = processImage(path, model)
    query_features.append(q_f)
    fig = plt.figure()
    plt.imshow(e_SOMlib.activate(n_train_data, som, query_features), cmap=cmap)
    plt.show()

    g,h=e_SOMlib.find_BMU(som,query_features)
    print('The best matching unit(bmu) is: {}'.format((g, h)))
    closest_image_index=e_SOMlib.get_closest_image(data_dict,som,g,h)
    result.append(image_data_dict[g][h][closest_image_index]['image'])

    #print result image
    if result[0] is not None:
        image_name=result[0]
        image_path = os.path.join(image_folder, image_name)
        matched_img = Image.open(image_path)
        display(matched_img)

    return result
```

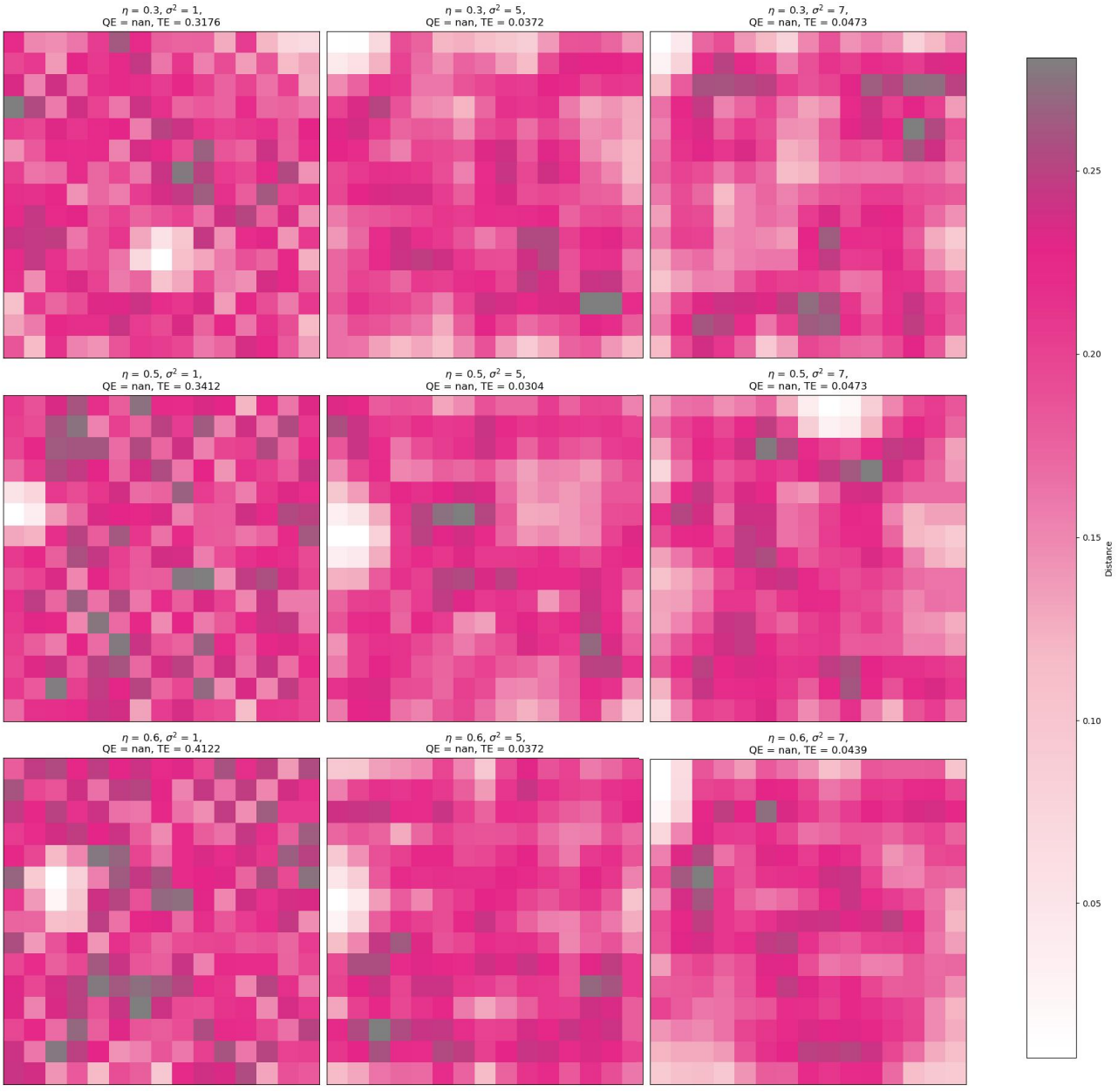
1.4. Film Subtitle SOM. The step I got the films subtitle SOM and again, created Searching function Process movie subtitles from '/.srt' subtitle files into a list(Text) and preprocess it.

```
import Code.film_model_class as Filmlib
from Code.film_model_class import FilmSubtitleModel
```

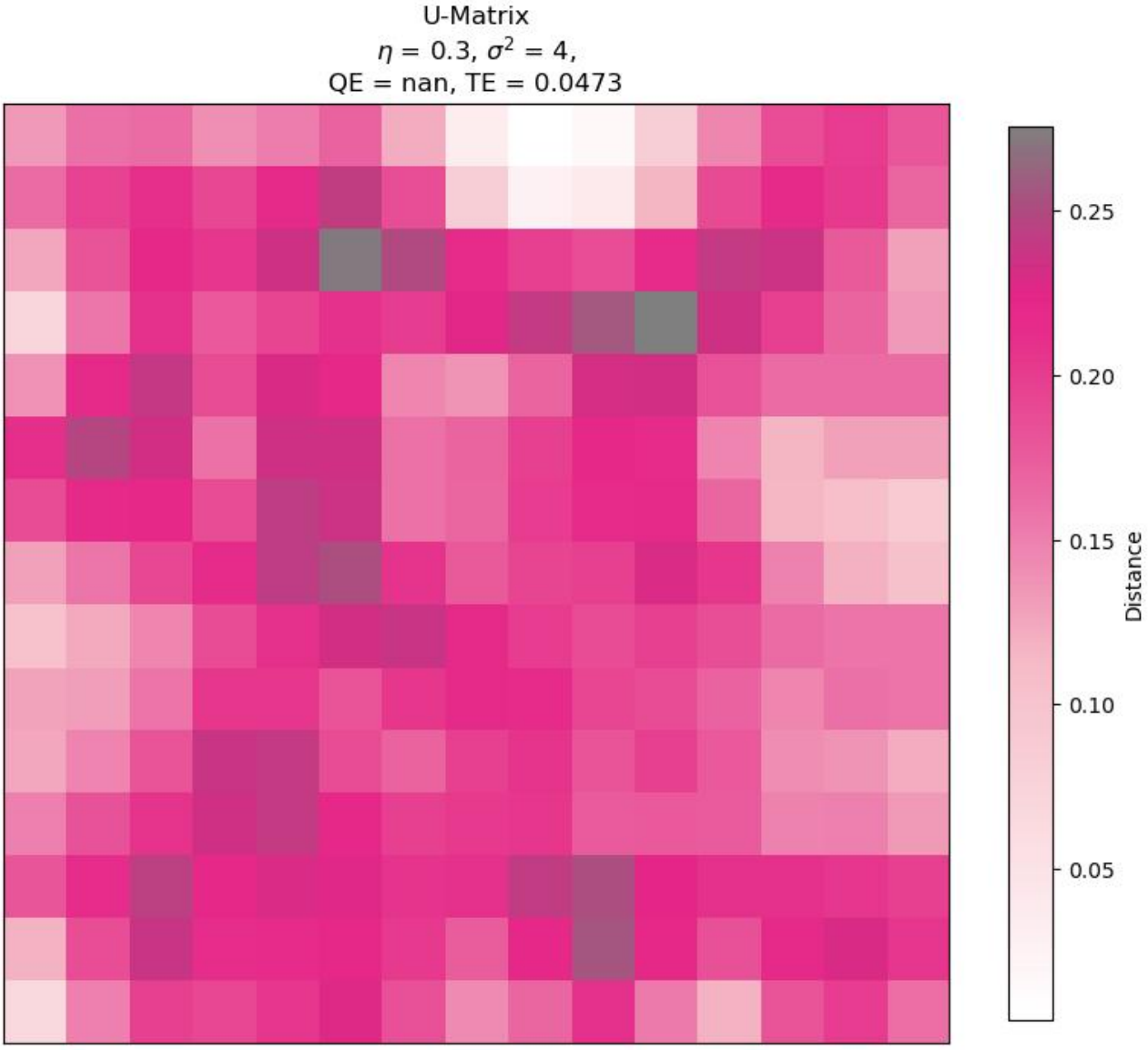
```
film_path = r'C:\Users\86180\python_Julian_assessment\image_to_text\Datasets\Film'
Filmlib.remove_spaces_in_filenames(film_path)
```

```
all_subtitles = Filmlib.process_srt_files(film_path)
merged_subtitles = Filmlib.merge_subtitles(all_subtitles)

merged_subtitles
```


```
 'text': 'tales of people who fall in or fight for or really just do anything for the sake of this enigmatic idea of love from your',
 'file_name': '500_Days_of_Summer3'},
{'index': 25,
 'start_time': '00:00:14,160',
 'end_time': '00:00:20,189',
 'text': 'Casablancas to whatever Netflix is putting out this week almost every movie has some kind of love story element',
 'file_name': '500_Days_of_Summer3'},
{'index': 26,
 'start_time': '00:00:20,189',
 'end_time': '00:00:25,350',
 'text': "whether that may be the main focus or just a way to fill time but of all the movies and all the love stories I've",
 'file_name': '500_Days_of_Summer3'},
{'index': 27,
 'start_time': '00:00:25,350',
 'end_time': '00:00:30,300',
 'text': "ever seen which is probably far too many but that's a different conversation there's only been one that's really",
 'file_name': '500_Days_of_Summer3'},
{'index': 28,
 'start_time': '00:00:30.300',
```

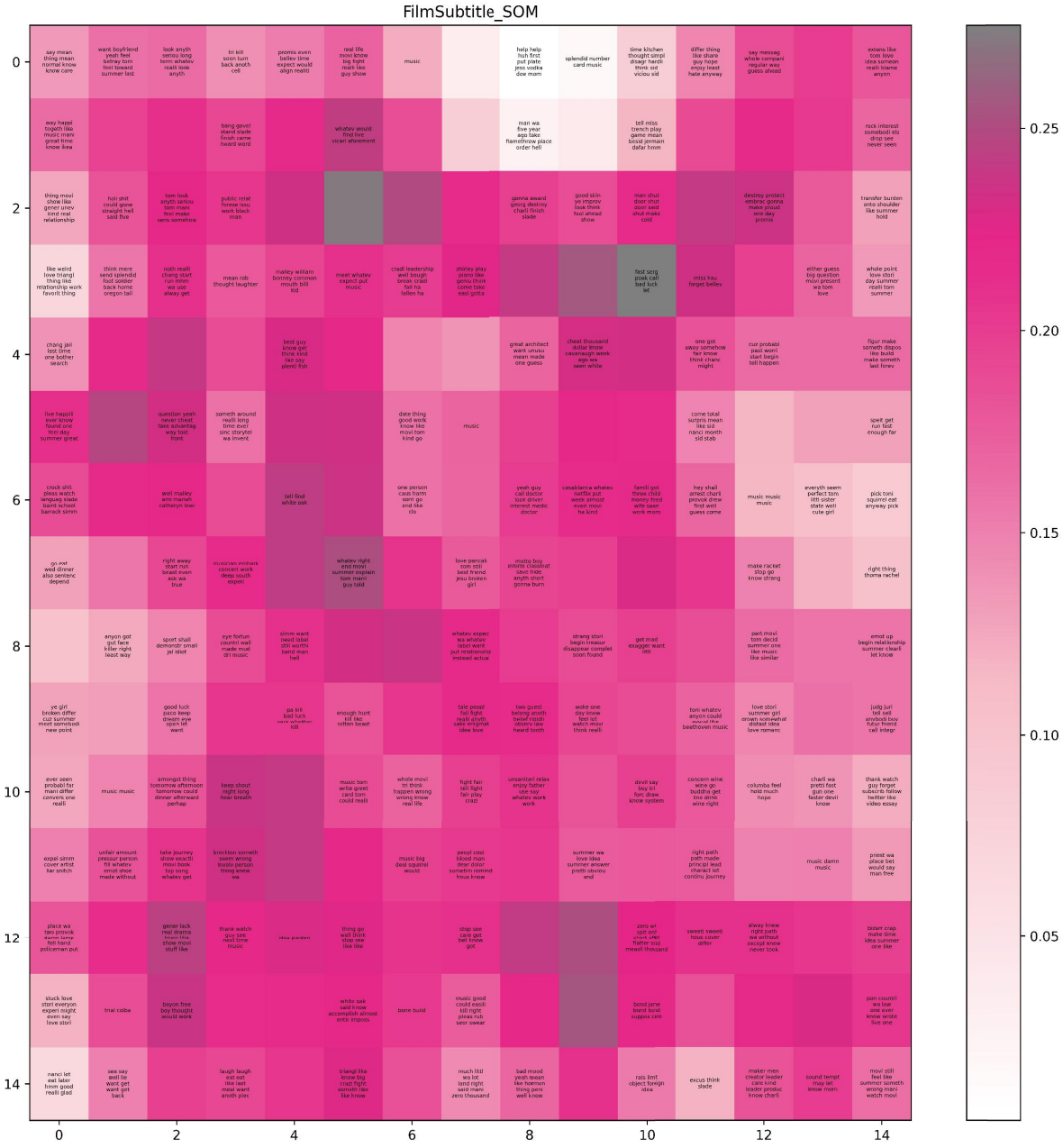
subtitle_som=soms[6]



The searching logic is similar to previous text searches, here it also using Ffmped command to show the movie clips.

```
def search_SubtitleSom(som, model, data_dict, film_folder, query='street', vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(subtitles_train_data, som, model.vectorize(query)), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query))
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    elif vectorization =='lsa':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(subtitles_train_data, som, model.vectorize(query)[0]), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query)[0])
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    similarities = {}
    for i in data_dict[g][h]:
        vector_array = np.array([i['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarities[i['text']['text']] = cosine_similarity(vector_array, [som[g][h]])

    biggest = None
    max_similarity = similarities[data_dict[g][h][0]['text']['text']]
    for i in data_dict[g][h]:
        sim = similarities[i['text']['text']]
        if sim > max_similarity:
            max_similarity = sim
            biggest = i['text']
    result.append(biggest)

    # Display the Film:
    if result[0] is not None:
        matched_film_path=Filmlib.check_film_exist(result, film_folder)
        Filmlib.display_matched_filmclip(result, matched_film_path, query)

    return result
```

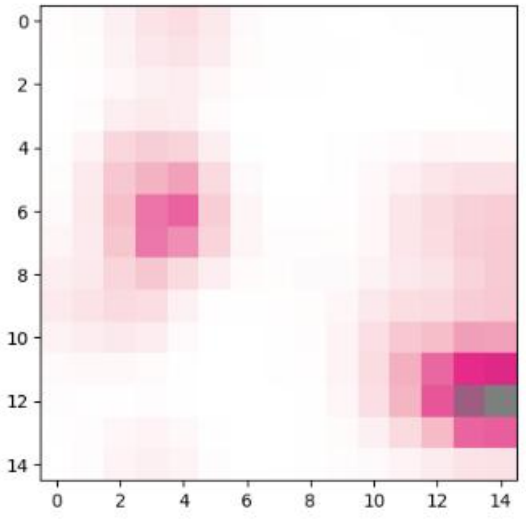
The dataset used here is the frame of the film, I obtained 200*10 frames images from my film dataset.



Searching the Subtitle by query

```
search_SubtitleSom(subtitle_som, film_model, subtitle_data_dict, film_folder, query, vectorization ='lsa')
```

```
The matching query is: today I fall in love with a western guy.
```




```
[{'index': 214,
  'start_time': '00:13:16,519',
  'end_time': '00:13:25,000',
  'text': 'go if I was in your place I'd say the same thing ehh but I wouldn't be foolish enough to expect another man to believe',
  'file_name': 'Spaghetti_Western'}]
```

```
The best matching unit(bmu) is: (12, 14)
Spaghetti_Western
Found: C:\Users\86180\python_Julian_assessment\image_to_text\Datasets\Film\Spaghetti_Western.mp4
```

PART 2: The methods to vectorise text dataset

I chose Tf-idf as the second vectorized text and used SVD for vector recombination and dimension modification, which was displayed as "lsa" in my IT. Here is a simple example of how I demonstrate these three vectorization methods using the same query.

```
query='today I fall in love with a western guy.'
```

```
tfidf_train_data=tfidf_novel_model.vector_matrix

tfidf_result=tfidf_novel_model.search(query, n=3)
tfidf_result
```

```
[{'paragraph': ' I ought not to omit the positively vast percentage of people for whom falling in love is unfortunate for no other reason th
an that their love is not returned. I suppose a man falls in love twenty times for every once that he is able to marry, even when he is not
poor or ugly, simply because it takes two to make a match'''.',
```

```
lsa_train_data=lsa_novel_model.vector_matrix

lsa_result=lsa_novel_model.search(query, n=3)
lsa_result
```

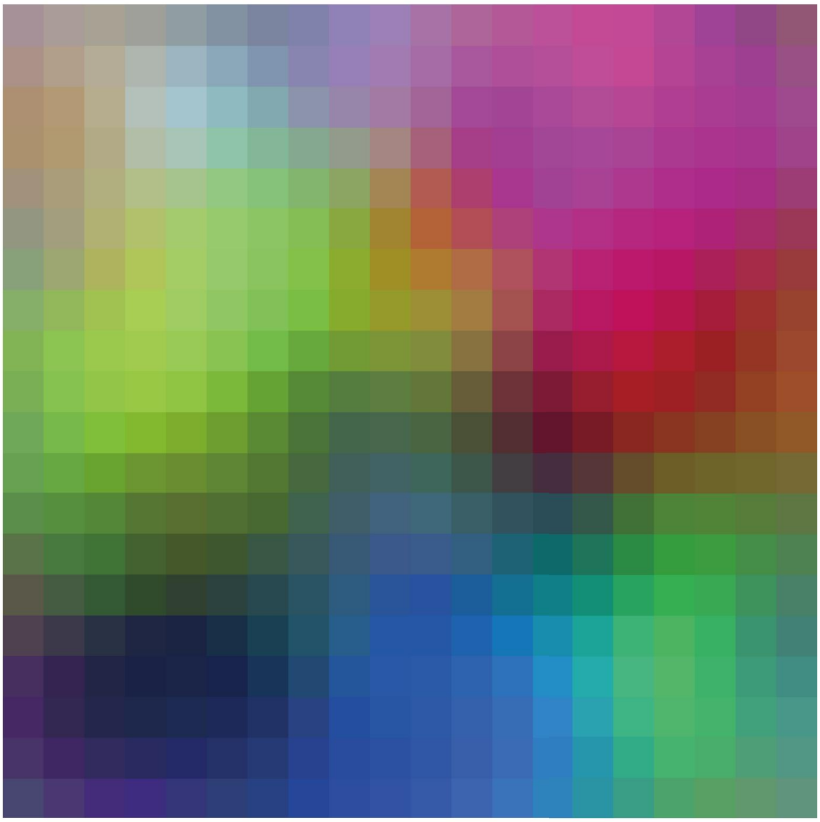
```
[{'paragraph': ' The fault of each was similar; it lay in the reference of information and judgment to the nature of man and not of the univ
erse. Both sense and intellect confused their own natures with the nature of that which was exposed to their knowledge, and this double conf
usion issued at last in sterile wrangles.',
 'nr': 1597,
```

```
doc2vec_train_data=doc2vec_novel_model.vector_matrix

doc2vec_result=doc2vec_novel_model.search(query, n=5)
doc2vec_result
```

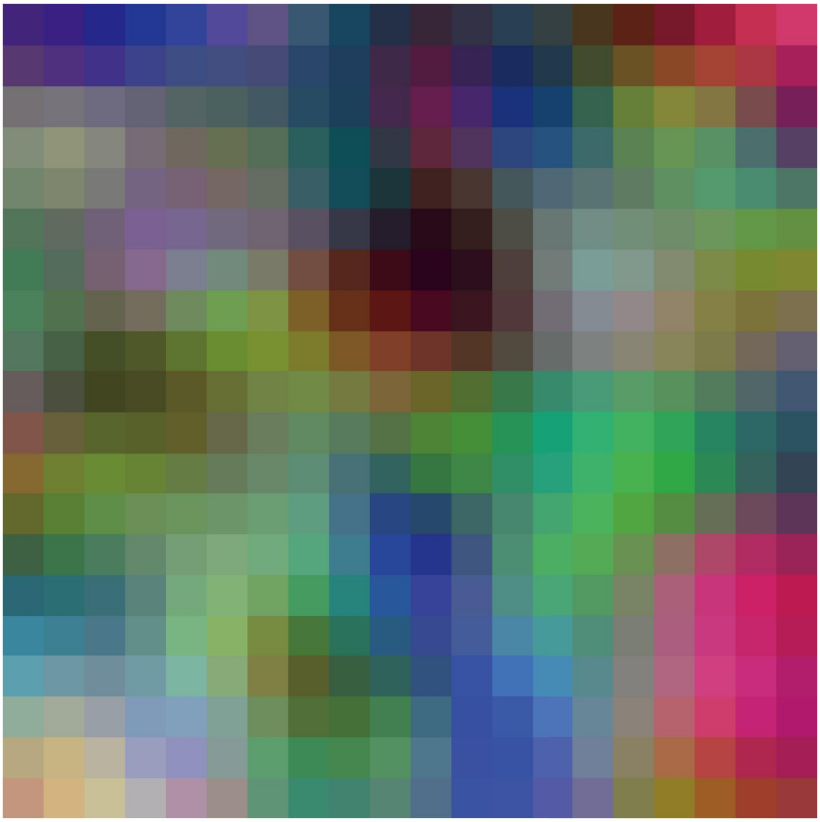
```
[{'paragraph': ' There were two kinds of Scotsmen—those born before the death of Elizabeth (the Ante-Nati) and those born since (the Post
-Nati). The Commissioners of Union had agreed to recommend two acts, each naturalizing one of these classes.',
  'nr': 827,
  'ID': 'williams-bacon',
  'type': 'epub'},
```

PART 3: fIT PCA and fit PCA in entirettraining set

PCA can reduce the vector of each unit in Doc2vec som to three dimensions and assign these three dimensions as RGB corresponding to the color to that individual unit:



The units' vector in Doc2vec SOM would be reduced down to three dimensions, which refers to the entire database using PCA, and these three dimensions are assigned to the individual unit as RGB corresponding to the color.



PART 4: Final Searching Engine

The two functions blow mainly serve as concatenation, in which the input query is converted to each other. Using video as input is also work (the logic is to convert the video into frame images and then search).

```
def search_from_inputText(query):
    print('The matching query is: ' + query)
    text_result=search_TextSom(text_som, doc2vec_novel_model, doc2vec_data_dict, query, vectorization ='doc2vec')
    print('\n'+'The best matching text result is: '+str(text_result)+'\n')

    ImageLibrary_folder=r'B:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Image'
    image_result=search_ImageTextSom(ImageText_som, ImageText_model, FuyuText_data_dict, ImageLibrary_folder, query, vectorization ='doc2vec')
    print('\n'+'The best matching image result is: '+str(image_result)+'\n')

    film_folder=r'B:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Film'
    film_result=search_SubtitleSom(subtitle_som, film_model, subtitle_data_dict, film_folder, query, vectorization ='lsa')
    print('\n'+'The best matching film clip result is: '+str(film_result))
```

```
def search_from_inputImage(img_path):
    #print query image
    print('The input image is: ')
    img = Image.open(img_path)
    display(img)

    image_folder=r'B:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Image'
    image_result=search_ImageSom1(img_path, ImageModel, image_som, image_folder)
    print('\n'+'The best matching image result is: '+str(image_result)+'\n')

    frames_folder=r'B:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Film_Frames'
    film_path=r'B:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Film'
    film_result=search_FramesSom1(img_path, FramesModel, frame_som, all_frames_info, frames_folder, film_path)
    print('\n'+'The best matching film clip result is: '+str(film_result)+'\n')

    query= get_image_text(ImageText_list, image_result)
    text_result=search_TextSom(text_som, doc2vec_novel_model, doc2vec_data_dict, query, vectorization ='doc2vec')
    print('\n'+'The best matching text result is: '+str(text_result))
```