# Python

P1-P10

# Explanation and outcomes

For this assignment, I mainly completed the linking and integration of prado text, prado images and videos scrapped from the poc website. The specific vectorisation process is shown below.


***Input begin from text:***

***1.Text input into prado text som to get most similar prado text.***

***2.Searching prado text in prado image fuyu som to find the index of image, and correspond to the image.***

***3.Prado image input in video som to search the most related video.***

***Input begin from Picture:***

***1.Picture input into prado image som to get most similar prado image, and to get corresponding fuyu text.***

***2.Searching Prado image fuyu interpretation in prado text som to get the most similar prado text.***

***3.Prado image input in video som to search the most related video.***

***Input begin from video:***

***1.Get the first frame of video to represent the video, and search it in video som, get most similar video.***

***2.Searching the frame of most similar video in prado image som to get related prado image and corresponding fuyu.***

***3.Prado image fuyu interpretation input in prado text som to search the most related prado text.***

Specific explanations and outputs are shown below.
Main code file Github link:
*https://github.com/UD-Skills-2023-24/23082976/blob/main/Python/Unitedversion%20of%20final%20skill%20work.ipynb*

## 1. Create prado text SOM and searching function

1.1 Create prado text SOM

1.1.1 Load prado text and processed them.

Load the prado text from textlist.csv as below. And run preprocess codes in preprocess part.
Preprocess text code file Github link:
*https://github.com/UD-Skills-2023-24/23082976/blob/main/Python/Code/preprocessmodel.py*

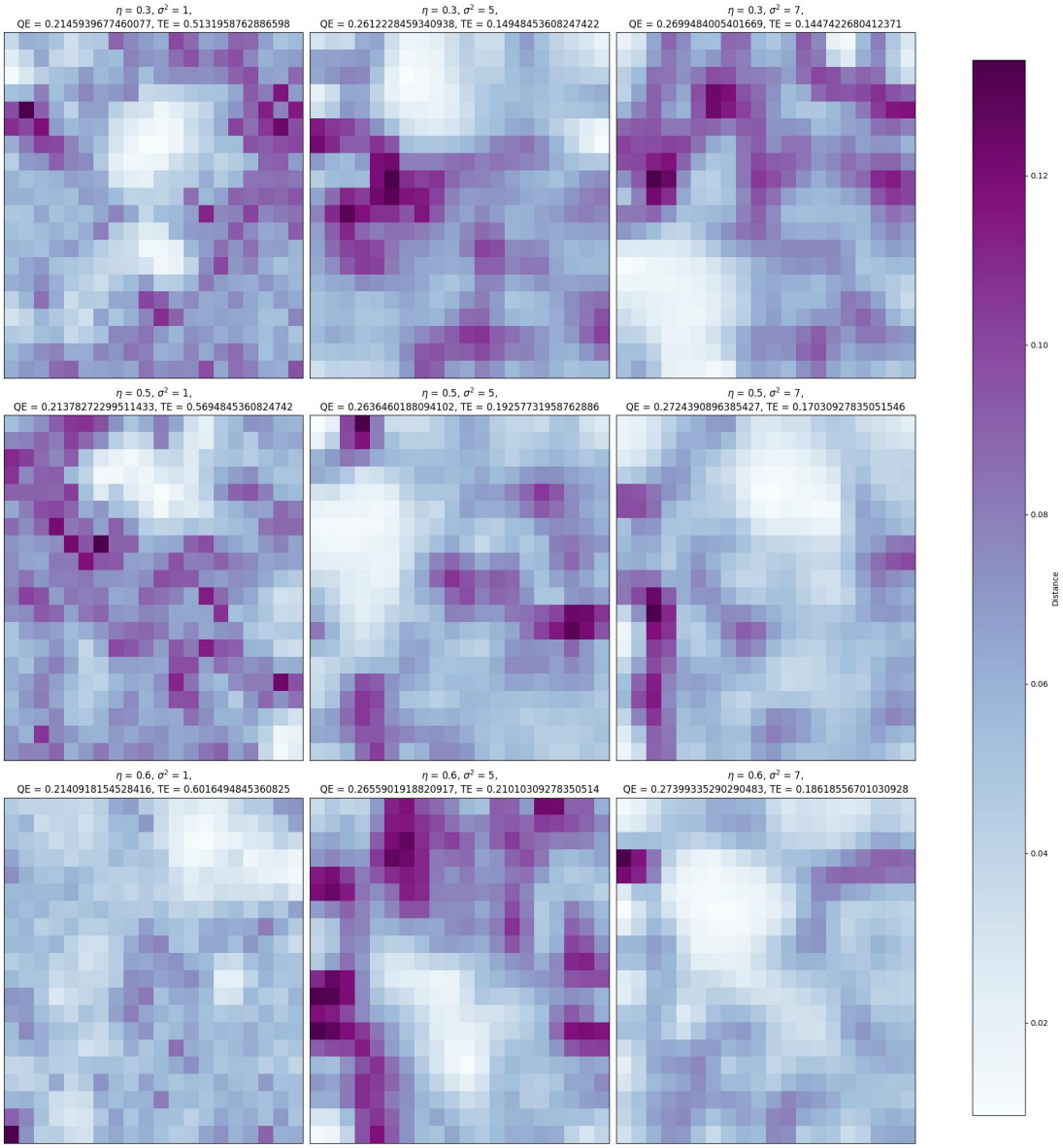
**Load text database**

```
[5]: text_list=[]
with open("textlist.csv", 'r',encoding='utf-8') as file:
    csvreader1 = csv.reader(file)
    for i in csvreader1:
        text_list.append(i[0])
```

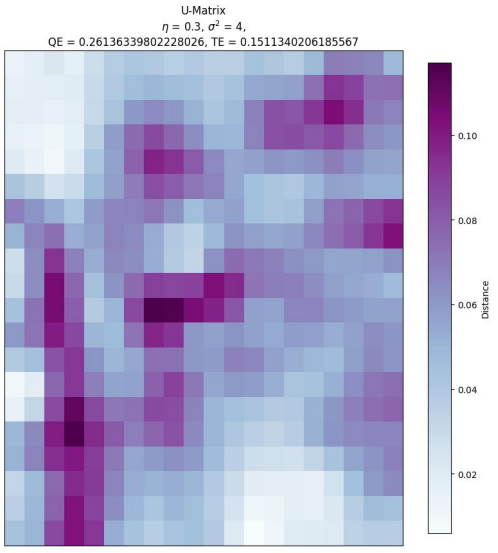
```
[6]: clear_text_list=[]
for i in text_list:
    if i not in clear_text_list:
        clear_text_list.append(i)
```

1.1.2 Create SOM

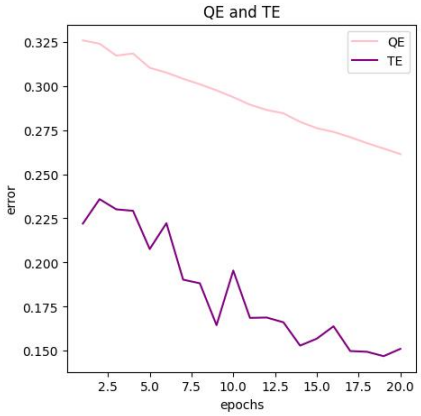
Create a SOM with Doc2vec with 300 dimension.




I chose one of the SOM with an index of 1, and the selection criterion was the one with the lowest sum of the two among the ones with the lower values of both QE and TE



This is how the QE & TE of that selected SOM changed over the 20 epochs of their training. We can see that both decrease with the number of training epochs, which means that the SOM is getting better organised with training.




1.1.3 Link the sentence with SOM unit
After that I relate the sentences to their BMUs

```
[16]: data_dict = []
for i in range(len(loaded_text_som_model)):
    row = []
    for j in range(len(loaded_text_som_model[0])):
        row.append([])
    data_dict.append(row)
vectortextPairs=[]
for i in range(0,len(clear_text_list)):
    vectortext={}
    vectortext['text']=clear_text_list[i]
    vectortext['vector']=sentence_vectors[i]
    vectortextPairs.append(vectortext)
for i in vectortextPairs:
    g,h = find_BMU1(loaded_text_som_model,i['vector'])
    data_dict[g][h].append(i)
```

```
[17]: with open('text_data_dict.pkl', 'wb') as f:
    pickle.dump(data_dict, f)
```

```
[18]: with open('text_data_dict.pkl', 'rb') as f:
    data_dict = pickle.load(f)
```

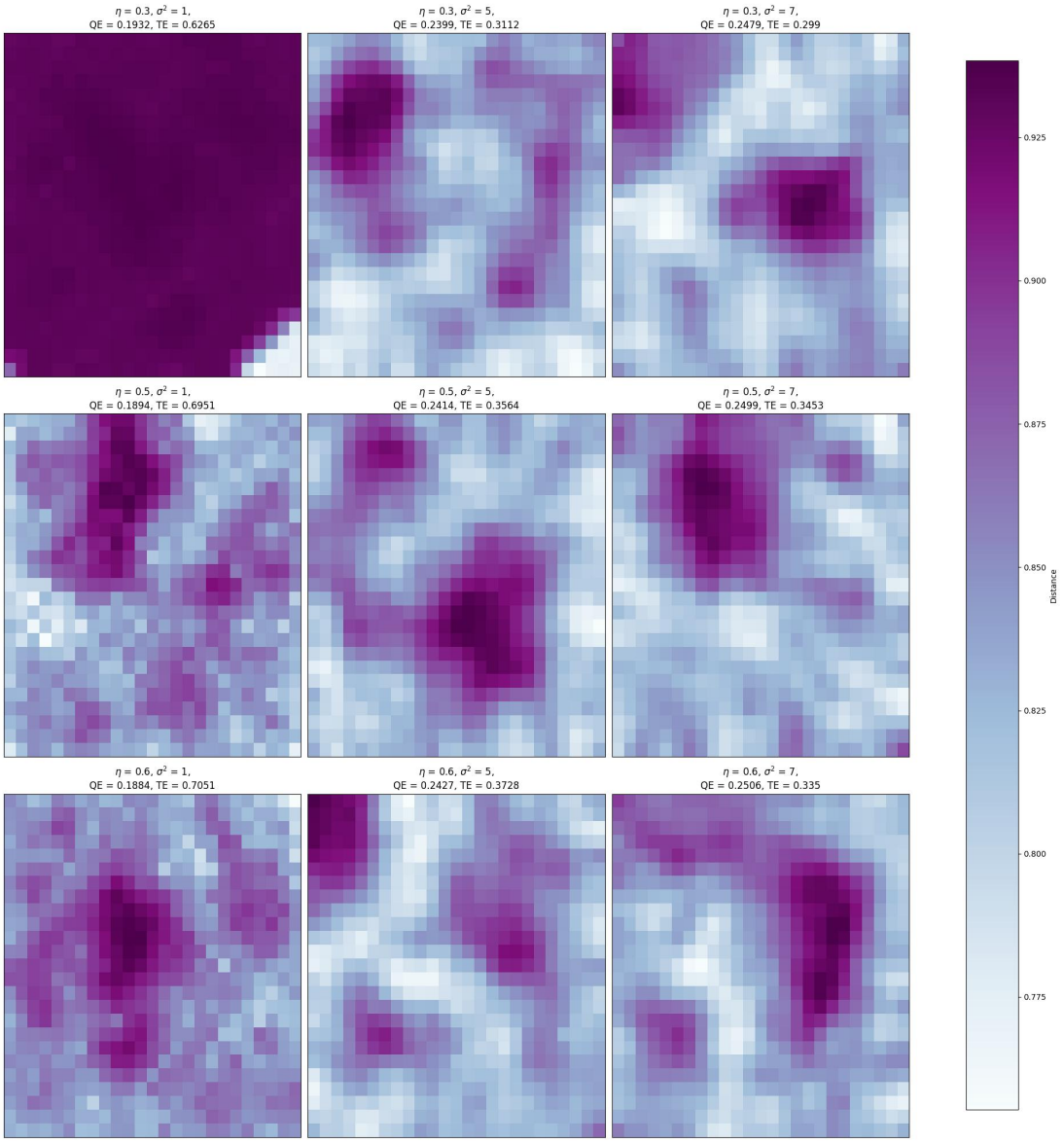
1.2 Create searching function
I defined a search function to search the text section for SOM, returning the one result in the BMU that has the smallest distance from the cosine of that lattice vector. At the same time, it will show the activated SOM.

```
[19]: def searchtextsom(query):
    result = []
    query = [query]
    preprocessed_query = preprocess(query)
    query_vector = model1.infer_vector(preprocessed_query[0].split())
    activatedSOM = activate1(sentence_vectors, loaded_text_som_model, query_vector)
    fig = plt.figure()
    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.BuPu, aspect='auto')
    plt.title(f'U-Matrix\n$\eta$ = {0.3}, $\sigma^2$ = {4},\nQE = {QE}, TE = {TE}')
    plt.colorbar(im, shrink=0.95, label='Distance')
    plt.xticks([])
    plt.yticks([])
    plt.show()
    g, h = find_BMU1(loaded_text_som_model, query_vector)
    print((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']] = cosine_similarity(vector_array, [loaded_text_som_model[g][h]])
    biggest = None
    max_similarity = similarities[data_dict[g][h][0]['text']]
    for i in data_dict[g][h]:
        sim = similarities[i['text']]
        if sim > max_similarity:
            max_similarity = sim
            biggest = i['text']
    result.append(biggest)
    return result
```

No screenshots are shown here as the return results are too long for screenshots. If necessary, you can open the source file to view.

## 2. Create prado image fuyu SOM and searching function

Load prado_fuyu.csv and the specific steps next here are similar to those above, so I won't repeat them again, and I will show the SOM results directly.




The SOM I chose is  index 2 one.

## 3.  Create prado image SOM and searching function

3.1 Create prado image SOM

3.1.1 Load prado images

Load files from 7254 prado images folders, and preprocess them with image feature model.

The OneDrive link:

*https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlz4_ucl_ac_uk/Eo6609ss1blErE7iDa6zMwUBtYiKWmyGlMzVRTo5ZoAggQ?e=QB5C1Z*

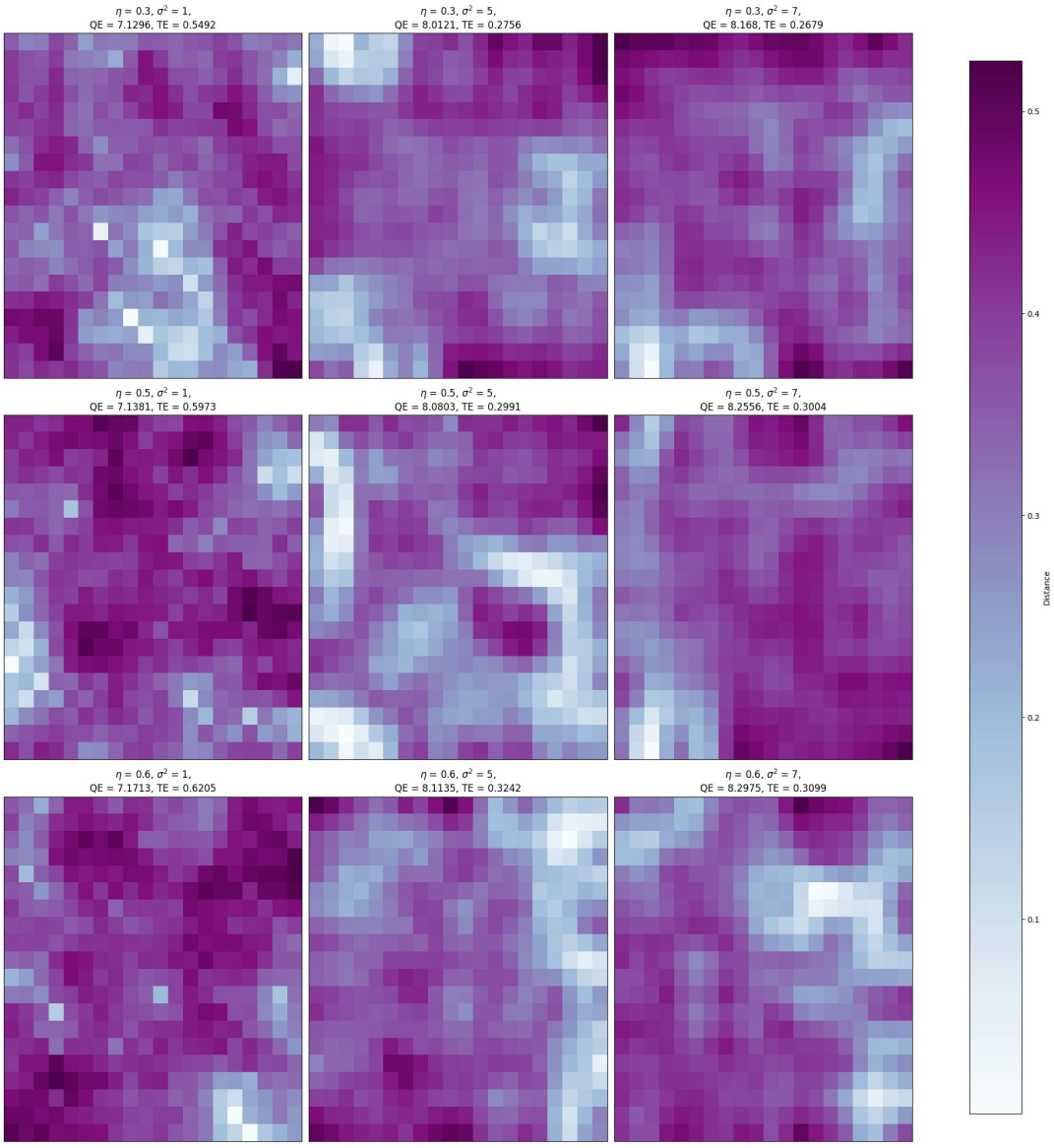
[135]:
```
model3 = tf.keras.applications.mobilenet.MobileNet(
# The 3 is the three dimensions of the input: r,g,b.
  input_shape=(224, 224, 3),
  include_top=False,
  pooling='avg'
)
```

[136]:
```
momaFiles = os.listdir('/Users/zhuolan/7254 prado images')
```

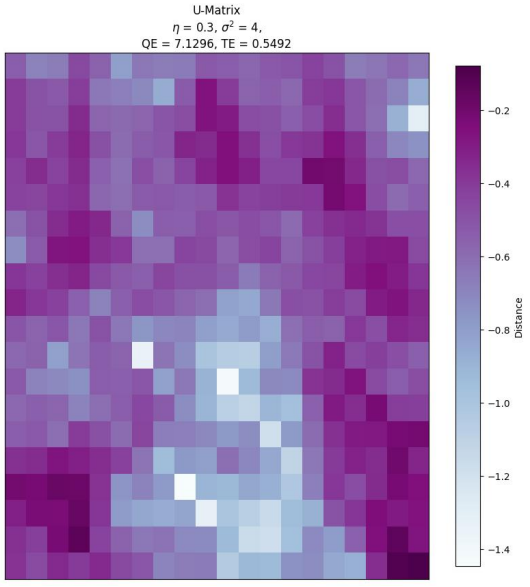
[137]:
```
def processImage(imagePath, model):
    im = load_image(imagePath)
    f = model.predict(im)[0]
    return f
```

3.1.2 Create SOM

Create a SOM with feature with 1024 dimension. I normalise the training data before training.




The SOM I chose is index 0 one.




3.1.3 Link the pictures with SOM unit

[135]:
```
model3 = 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'
)
```

[136]:
```
momaFiles = os.listdir('/Users/zhuolan/7254 prado images')
```

[137]:
```
def processImage(imagePath, model):
    im = load_image(imagePath)
    f = model.predict(im)[0]
    return f
```

3.2 Create searching function

The Euclidean distance between the unit picture vector and the unit itself is used to calculate the picture that best matches the unit. And the corresponding picture is found by searching the BMU of the new picture.

## 4. Create video image SOM and searching function

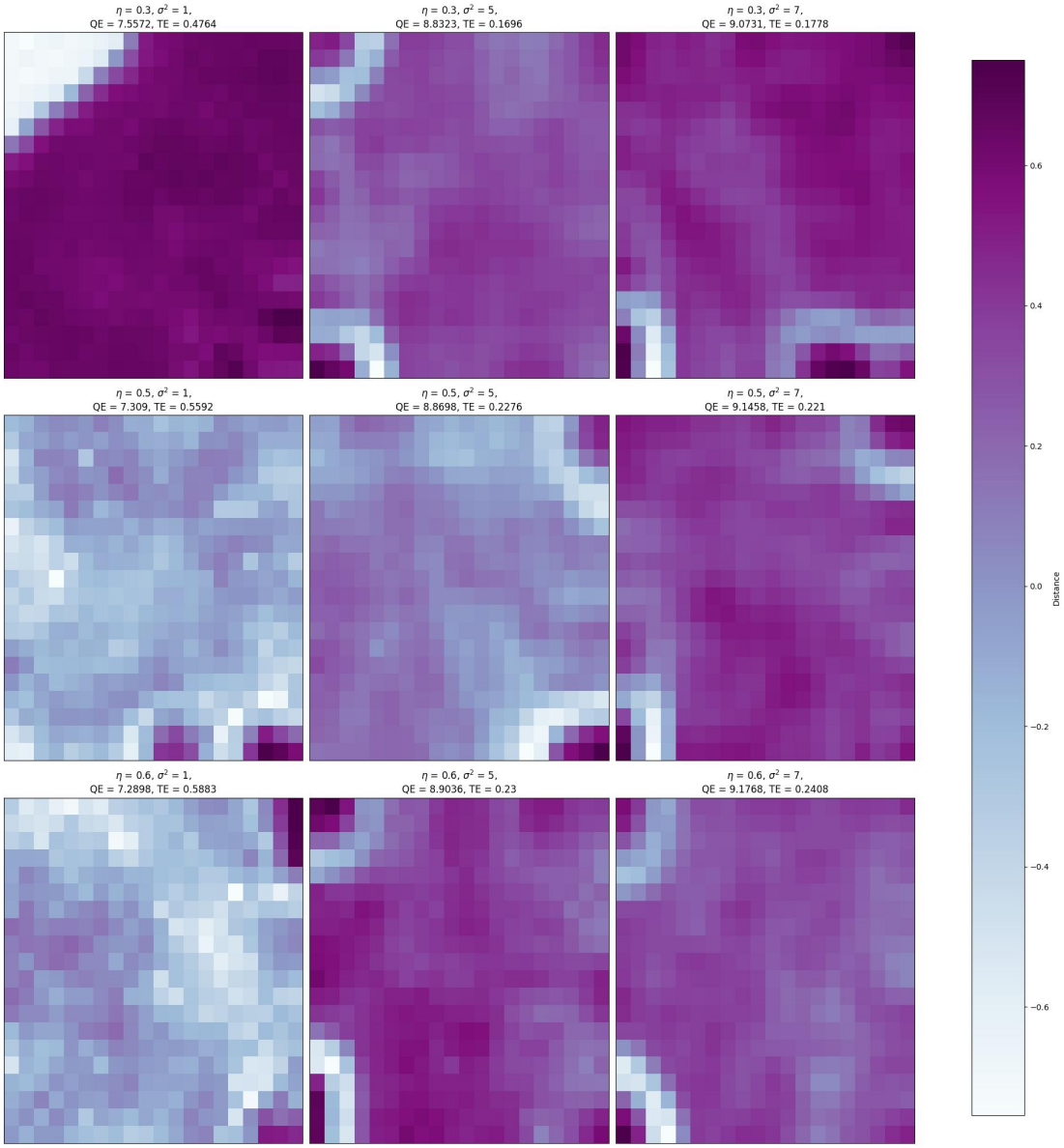
4.1 Load video images

The OneDrive link:

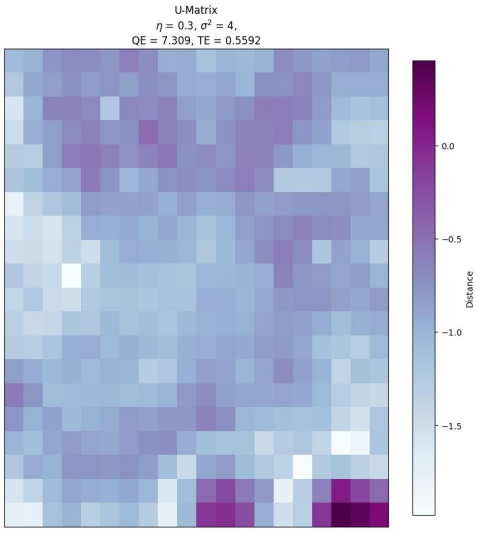
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlz4_ucl_ac_uk/EhcCP86j1tZBuuP2GTd7zc8BCbANaZroU89-g9dLpz5T2w?e=oktRqO

4.2 Create SOM

Specific implementation steps as above, the following shows the creation of SOM and selected SOM.



The SOM I chose is index 3 one.




4.3 Create searching function

This part I created a searching function to search in this SOM as well.

## 5. Create 3 searching functions to serialisation of all databases

The three functions here mainly play the part of a tandem where the top three input messages are converted into each other.And ten examples are shown below.

Here is the OneDrive link of video results:

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlz4_ucl_ac_uk/EuV1fpAu8MZKkkR1N1U0h0IBHXZ59Ztv5PZcJjGrETmQ9g?e=cqSqXF

Input text

+ 11 cells hidden

Input prado picture

+ 12 cells hidden

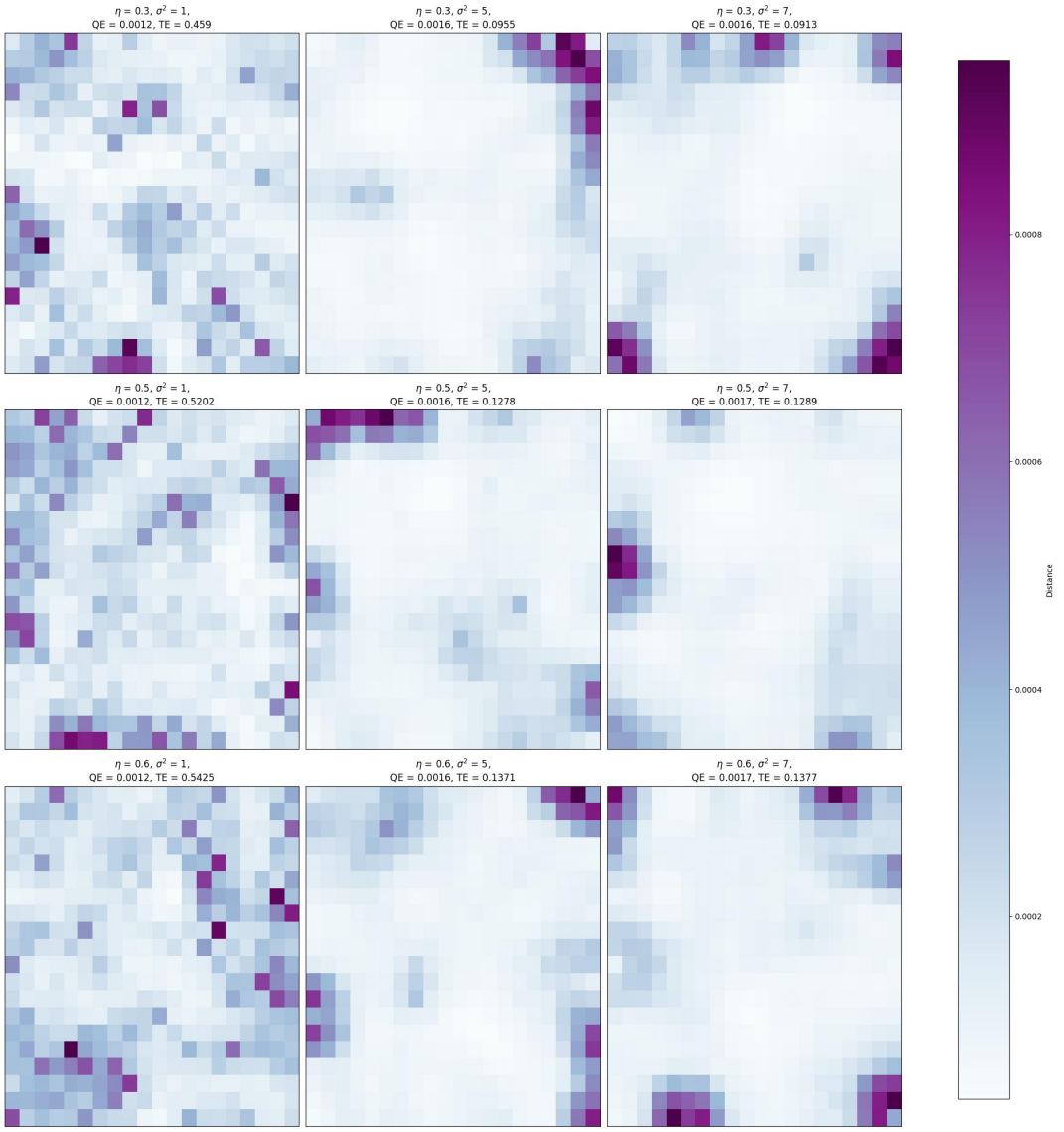
Input video

+ 13 cells hidden

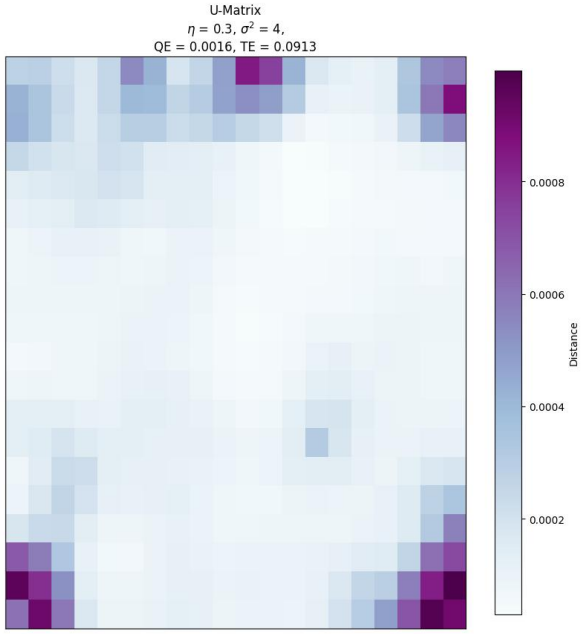
The above is the main part of the whole search engine.

## 6.Use more than one way to vectorise text databases

I chose TFidf for the second vectorised text and used svd for vector reorganisation and dimension modification.The other steps are consistent with the way Doc2vec was previously organised.




The SOM I chose is index 2 one.




And the search results according to TFidf do differ from Doc2vec's.

Like I randomly input the sentence 'I have root, but I flow.', the result from Doc2vec is this one:

['This tapestry cartoon depicts five people. The young man in the foreground is drinking from a wineskin while his companion eats a chive or tender onion. This scene has been interpreted as an allegory of gluttony, represented here by the boy with the cane and the blind drinker who are the main characters from the picaresque novel, Lazarillo de Tormes. The di sotto in s ù perspective indicates that this cartoon was intended for an over-window tapestry. The resultant tapestry was intended to hang in the dining room of the Prince and Princess of Asturias (the future Carlos IV and his wife Maria Luisa de Parma) at the Monastery of El Escorial. This work was part of a decorative series of ten··· | + This tapestry cartoon depicts five people. The young man in the foreground is drinking from a wineskin while his companion eats a chive or tender onion. This scene has been interpreted as an allegory of gluttony, represented here by the boy with the cane and the blind drinker who are the main characters from the picaresque novel, Lazarillo de Tormes. The di sotto in s ù perspective indicates that this cartoon was intended for an over-window tapestry. The resultant tapestry was intended to hang in the dining room of the Prince and Princess of Asturias (the future Carlos IV and his wife Maria Luisa de Parma) at the Monastery of El Escorial. This work was part of a decorative series of ten cartoons for tapestries on "countryside" subjects. Goya, himself, invented the specific composition of the present one. This work entered the Prado Museum Collection in 1870 by way of Madrid´s Royal Palace. Access to the series of ten tapestry cartoons destined for the dining room of the Prince and Princess of Asturias at the palace of El Pardo: The Picnic on (P00768); Dance on the Banks of the Manzanares (P00769); A Fight at the Venta Nueva (P00770); The An Avenue in Andalusia or The Maja and the cloaked Men (P00771); The Drinker (P00772); The Parasol (P00773); The Kite (P00774); The Card Players (P00775); Children blowing up a Bladder (P00776); Boys Picking Fruit (P00777). | –']

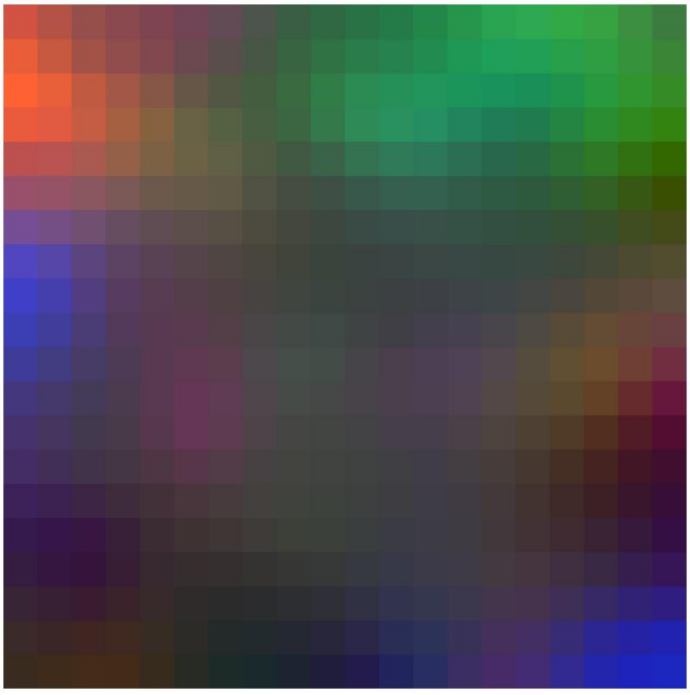
The result from TFidf is this one:

['This canvas, a companion piece to the Allegory of Smell, offers an elegant, tender, delicate treatment of a recurring theme in Western art. Both paintings belong to a transition period during which the warm, decorative spirit of late Rococo was giving way to the cold severity of early Neoclassicism.']

So we can see they output results totally different.

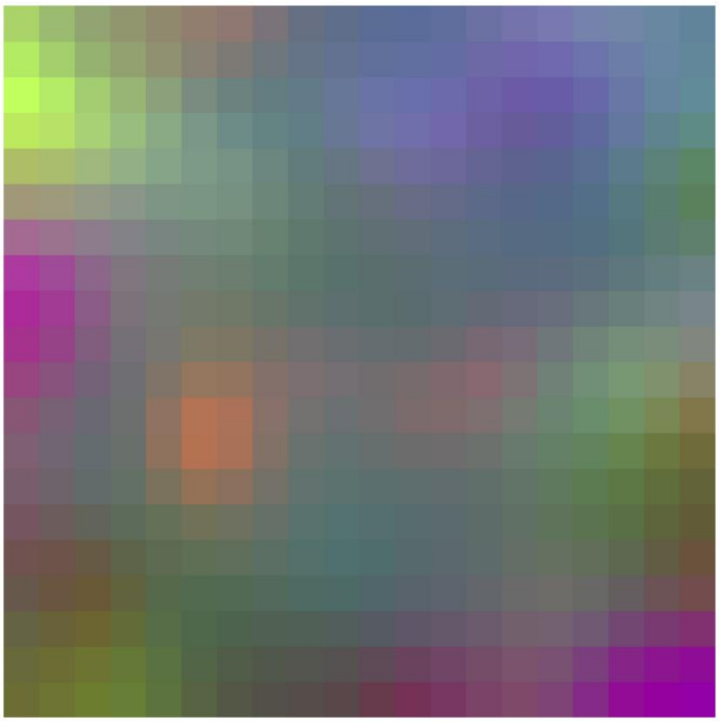
## 7.Fit PCA on all cells of the SOM

The vector of each cell in Doc2vec SOM is downscaled to 3 dimensions using PCA and the three dimensions are assigned to that individual cell as RGB corresponding to the colour. It finall showed the SOM as this,



## 8.Fit PCA on the entire training set

The vector of each cell in Doc2vec SOM is downscaled to 3 dimensions refers to the whole database using PCA and the three dimensions are assigned to that individual cell as RGB corresponding to the colour. And then when render it back to the SOM , it showed the result as follow.



# Asymptotic Complexity

# Exercise One

1.Implement this algorithm in Python. Use the NumPy ndarray object for your matrices

```
import numpy as np

def square_matrix_multiply(A, B):
    n = A.shape[0]
    C = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i, j] += A[i, k] * B[k, j]
    return C


A = np.array([[1, 2, 3], [2, 4, 6],[5, 3, 6]])
B = np.array([[5, 3, 6], [7, 9, 8],[1, 2, 3]])


result = square_matrix_multiply(A, B)


print(result)
```

```
[[22. 27. 31.]
 [44. 54. 62.]
 [52. 54. 72.]]
```

2.Give the asymptotic time complexity of the above algorithm or your implementation (they should be the same). Justify and explain your answer.

Asymptotic time complexity is O(n^3). Because ' for i in range(n):' will execute n times, so as 'for j in range(n):' and 'for k in range(n):'. So it's O(n^3).

3.Implement the algorithm with nested lists and compare the algorithms on different sizes of matrices;

```
import time


nested_list1 = np.random.rand(10, 10).tolist()


nested_list2 = np.random.rand(10, 10).tolist()


nested_list3 = np.random.rand(100, 100).tolist()


nested_list4 = np.random.rand(100, 100).tolist()


def square_matrix_multiply(A, B):
    n = len(A)
    C = [[0 for i in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    return C


start_time = time.time()
square_matrix_multiply(nested_list1, nested_list2)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 0.0009500980377197266 seconds
```

```
start_time = time.time()
square_matrix_multiply(nested_list3, nested_list4)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 0.05599665641784668 seconds
```

4.Compare with built-in multiplication functions

```
start_time = time.time()
np.matmul(nested_list3, nested_list4)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 0.026653289794921875 seconds
```

Apparently, built-in function is more efficient.

5.Look at multiplying more than two matrices

```
def multiply_matrices(matrices):
    result = matrices[0]
    for i in range(1, len(matrices)):
        result = square_matrix_multiply(result, matrices[i])
    return result


A = np.random.rand(3, 3)
B = np.random.rand(3, 3)
C = np.random.rand(3, 3)


result = multiply_matrices([A, B, C])


result
```

```
[[0.6863161063182255, 0.6797401268373633, 0.7521723126572709],
 [1.4875523118061265, 1.663080478300671, 1.503025559765348],
 [1.355042925568613, 1.4919980931756336, 1.3851072915077263]]
```

## Exercise Two

1.Describe and explain the algorithm.

Recursiveness :This algorithm keeps splitting the matrices into quarters by determining the length of each matrices until the matrices length is 1. Divide-and-conquer :This algorithm divid the complex problem of multiplying two large matrices to the base conquerable case of multiplying two number and a string of matrices added.

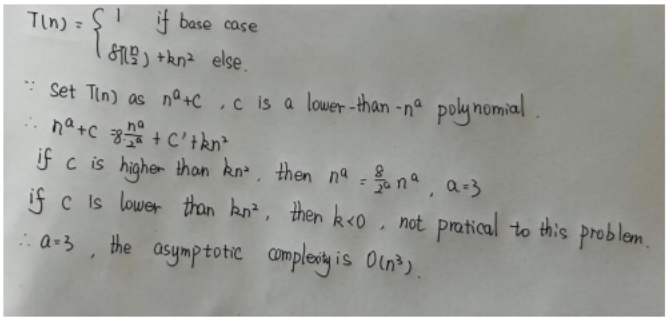
2.Implement the recursive algorithm in Python.

```
#Straightforward to implement
def split_matrix(A, n):
    A11 = [row[:n] for row in A[:n]]
    A12 = [row[n:] for row in A[:n]]
    A21 = [row[:n] for row in A[n:]]
    A22 = [row[n:] for row in A[n:]]
    return A11, A12, A21, A22


#Straightforward to implement
def matrix_add(A, B):
    n = len(A)
    return [[A[i][j] + B[i][j] for j in range(n)] for i in range(n)]


#Straightforward to implement
def combine_matrix(A, B, C, D):
    First_row = [A[i] + B[i] for i in range(len(A))]
    Second_row = [C[i] + D[i] for i in range(len(C))]
    return First_row + Second_row
```

```
def  Square_Matrix_Multiply_Recursive(A,B):
    n = len(A)
    if n==1:
        #Straightforward to implement
        return [[A[0][0] * B[0][0]]]
    else:
        A11, A12, A21, A22 = split_matrix(A, n//2)
        B11, B12, B21, B22 = split_matrix(B, n//2)
        #Hid a lot of complexity
        C11 = matrix_add(Square_Matrix_Multiply_Recursive(A11, B11),
                         Square_Matrix_Multiply_Recursive(A12, B21))
        C12 = matrix_add(Square_Matrix_Multiply_Recursive(A11, B12),
                         Square_Matrix_Multiply_Recursive(A12, B22))
        C21 = matrix_add(Square_Matrix_Multiply_Recursive(A21, B11),
                         Square_Matrix_Multiply_Recursive(A22, B21))
        C22 = matrix_add(Square_Matrix_Multiply_Recursive(A21, B12),
                         Square_Matrix_Multiply_Recursive(A22, B22))
        return combine_matrix(C11, C12, C21, C22)
```

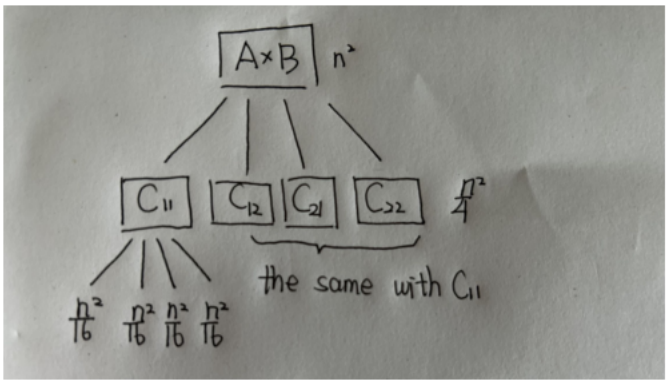
3.Do a complexity analysis for the SMMRec algorithm. First comment on the complexity of the base case, divide step, conquer step, and combine step separately, then put it all together.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('C:/Users/zl/Pictures/Screenshots/complexity analysis.jpg')
plt.imshow(img)
plt.axis('off')
plt.show()
```




4.Do a tree analysis

```
img = mpimg.imread('C:/Users/zl/Pictures/Screenshots/tree analysis.jpg')
plt.imshow(img)
plt.axis('off')
plt.show()
```




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

```
A = np.random.rand(128,128)
B = np.random.rand(128,128)
```

```
start_time = time.time()
Square_Matrix_Multiply_Recursive(A,B)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution time, "seconds")
```

```
Execution time: 2.536848783493042 seconds
```

```
start_time = time.time()
square_matrix_multiply(A, B)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 0.5864994525909424 seconds
```

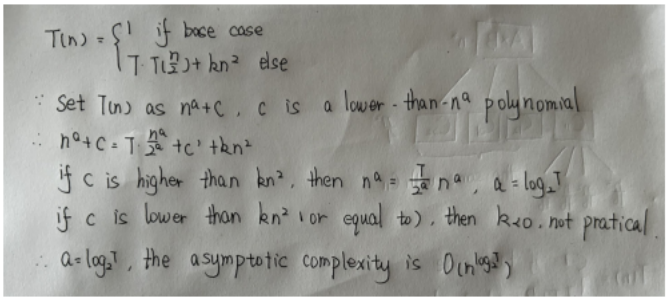
## Exercise Three

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

The aysmptotic complexity of addition/subtraction is O(n^2). Because the size of matrices is n^2. So each addition/subtraction will run n^2 times. And the aysmptotic complexity of multiplication is O(n^3). The specific analysis is in Exercise One.

2.Do a complexity analysis of the Strassen algorithm.

```
img = mpimg.imread('C:/Users/zl/Pictures/Screenshots/exercise three.jpg')
plt.imshow(img)
plt.axis('off')
plt.show()
```




3.Implement and test the algorithm

```
#Straightforward to implement
def matrix_sub(A, B):
    n = len(A)
    return [[A[i][j] - B[i][j] for j in range(n)] for i in range(n)]
#Straightforward to implement
def matrix_add(A, B):
    n = len(A)
    return [[A[i][j] + B[i][j] for j in range(n)] for i in range(n)]
#Straightforward to implement
def combine_matrix(A, B, C, D):
    First_row = [A[i] + B[i] for i in range(len(A))]
    Second_row = [C[i] + D[i] for i in range(len(C))]
    return First_row + Second_row
```

```
def strassen(A, B):
    n = len(A)
    if n == 1:
        #Straightforward to implement
        return [[A[0][0] * B[0][0]]]
    else:
        A11, A12, A21, A22 = split_matrix(A, n//2)
        B11, B12, B21, B22 = split_matrix(B, n//2)
        #Hid a lot of complexity
        P1 = strassen(matrix_add(A11, A22), matrix_add(B11, B22))
        P2 = strassen(matrix_add(A21, A22), B11)
        P3 = strassen(A11, matrix_sub(B12, B22))
        P4 = strassen(A22, matrix_sub(B21, B11))
        P5 = strassen(matrix_add(A11, A12), B22)
        P6 = strassen(matrix_sub(A21, A11), matrix_add(B11, B12))
        P7 = strassen(matrix_sub(A12, A22), matrix_add(B21, B22))
        #Straightforward to implement
        C11 = matrix_add(P1, P4)
        C11 = matrix_sub(C11, P5)
        C11 = matrix_add(C11, P7)
        C12 = matrix_add(P3, P5)
        C21 = matrix_add(P2, P4)
        C22 = matrix_add(P1, P3)
        C22 = matrix_sub(C22, P2)
        C22 = matrix_sub(C22, P6)

        C = combine_matrix(C11, C12, C21, C22)
        return C
```

4.Discuss other optimisations;

For other optimisations of the matrix algorithm, I found the Laser Method during my research. Compared to the strassen method, although the strassen algorithm reduces the computational complexity by recursion, it actually repeats the computation of each sub matrix every time after obtaining the base case. But the Laser Mathod will storage some result of sub matrix to avoid repeated calculation. So it reduce the asymptotic complexity further.

# Houdini

# 1 Houdini Fundamentals

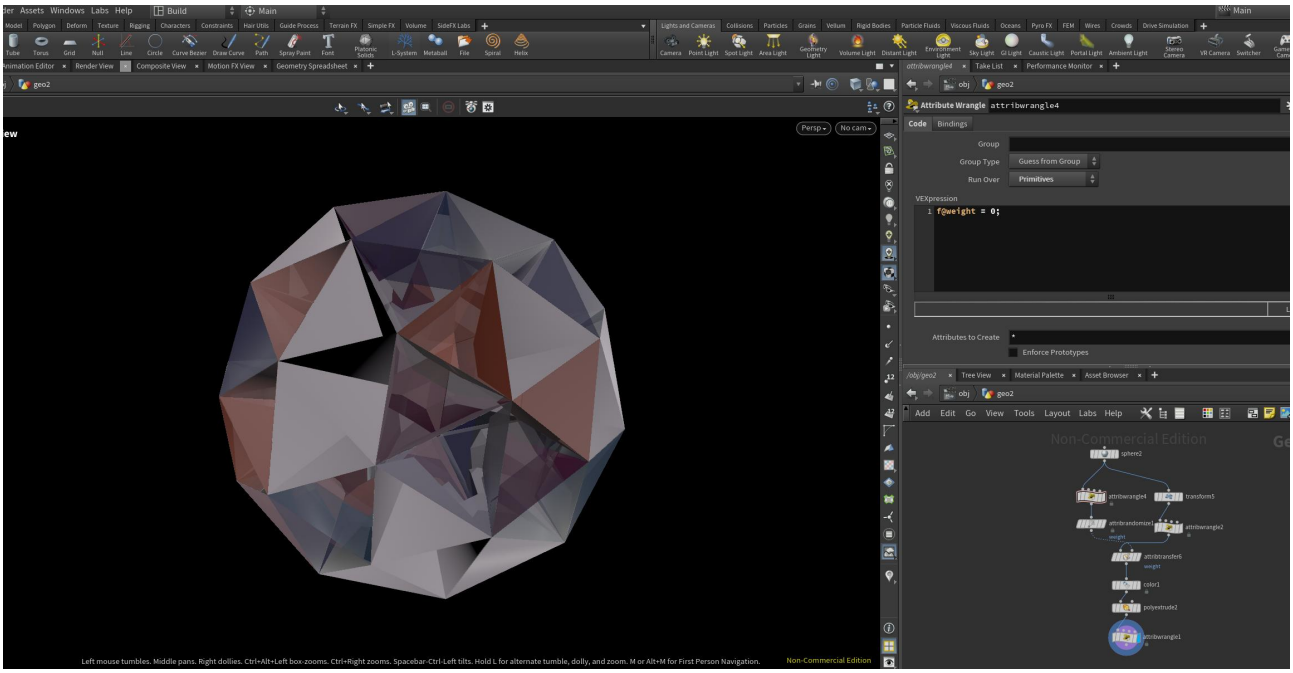
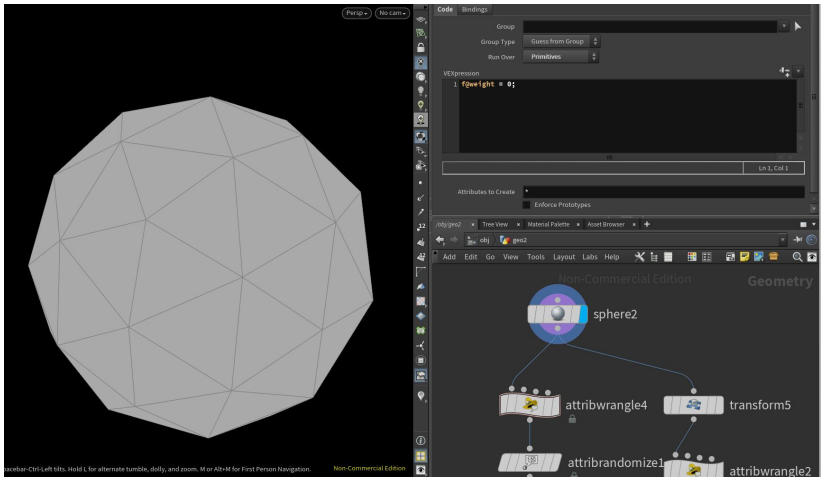
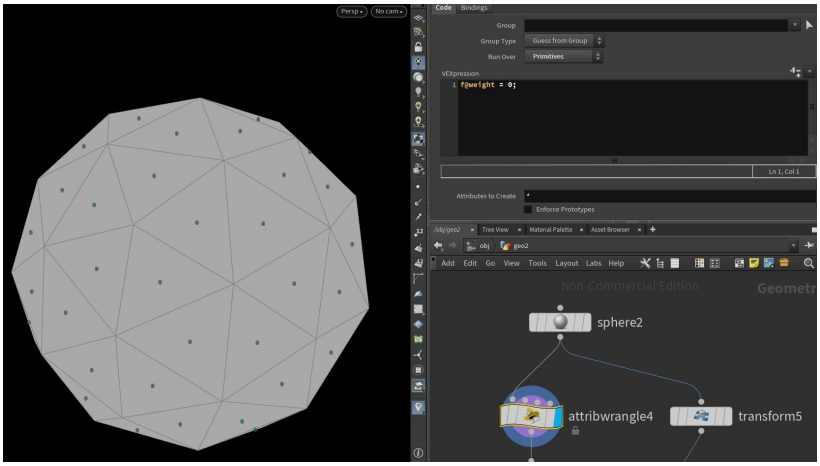


**Figure 1: Houdini Fundamentals: Setup 1 Overview**

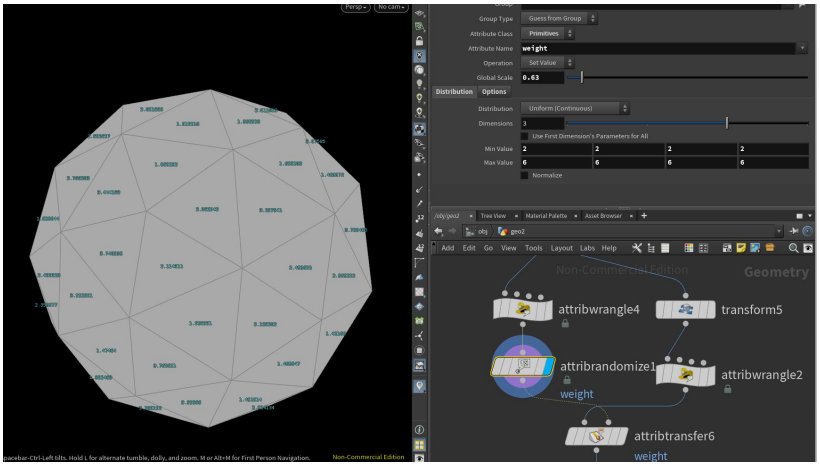
**1: Sphere SOP**
The ”sphere” is of primitive type: polygon. One of the basic geometric data types that contains points, primitives of type polygon, and vertices.



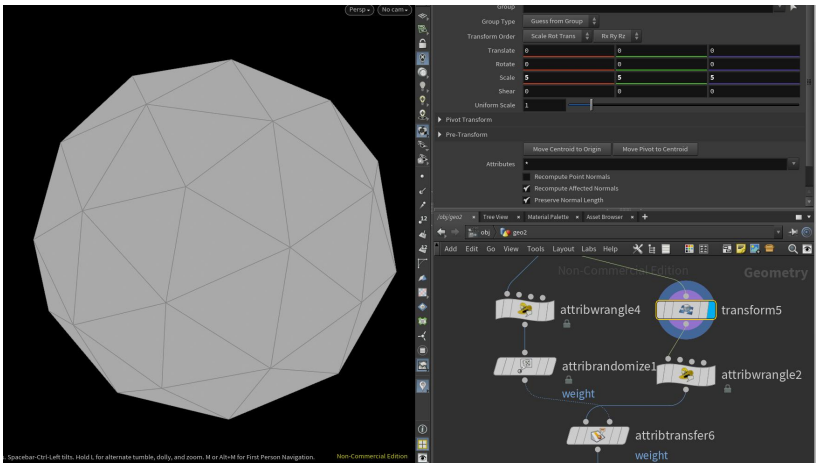

**2: Attribute Wrangle SOP**
Create an ”Attribute Wrangle” SOP, where we add a ”weight” attribute to the primitives. This one add value 0 to every primitive.



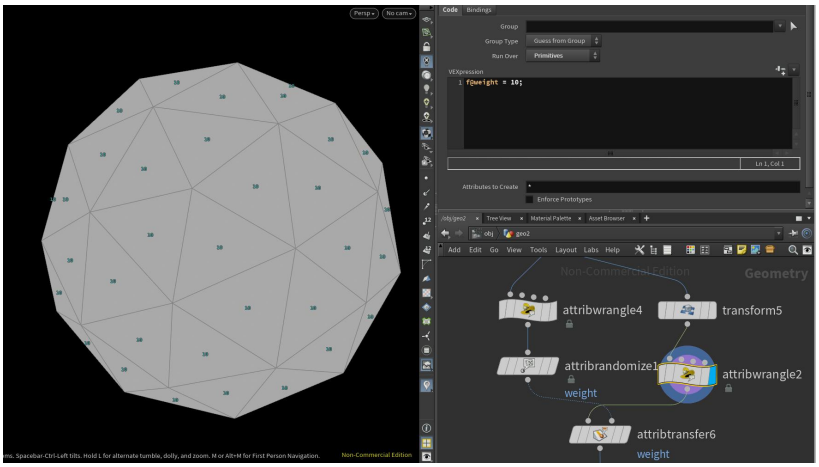
**3: Attribute Randomize SOP**
Randomized the value of weight into 2-6. The purpose of this step is to differentiate the results after attribute transfer.




**4: Transform SOP**
Create an ”Transform” SOP, and this node is used for transform subject. Such as here is to enlarge the original sphere isometrically into 5 times.

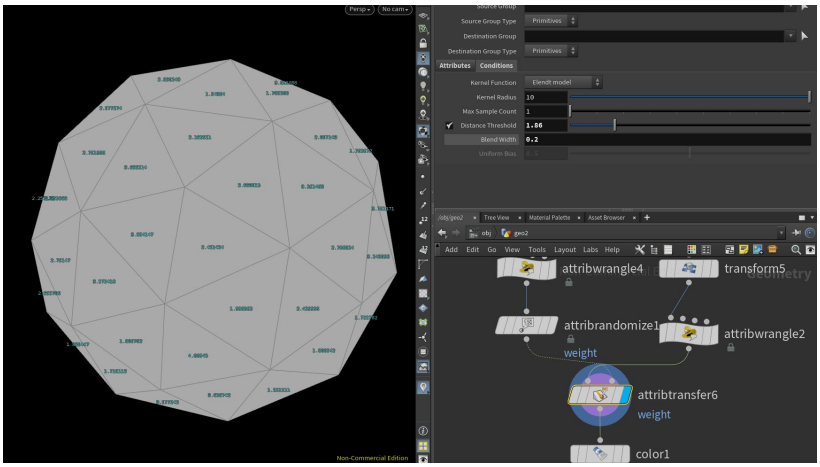



**5: Attribute Wrangle SOP**
Create an ”Attribute Wrangle” SOP, where we add a ”weight” attribute to the primitives. This one add value 10 to every primitive.



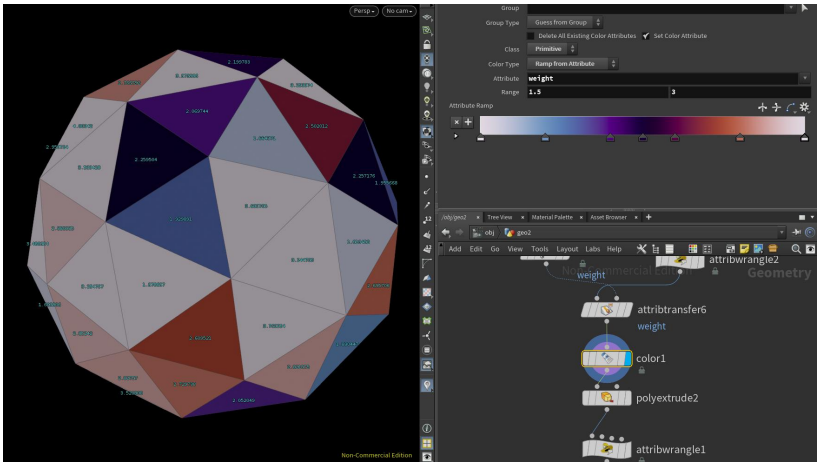
**6: Attribute Transfer SOP**

Create an ”Attribute Transfer” SOP, which used to transfer randomized weight to the enlarged sphere. By doing so, the weight on the enlarge sphere can change between 2-10 with sliding the bar Distance Threshold.



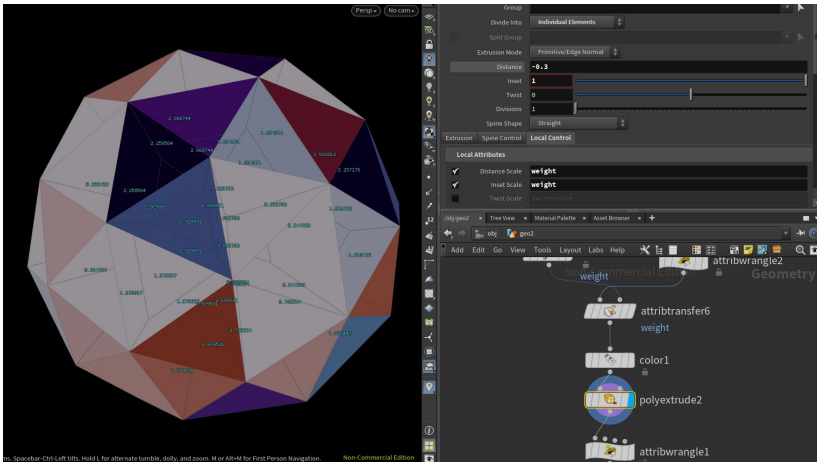

**7: Color SOP**

Create an ”Color” SOP, where we add color to the sphere based on the value of weight on each primitive.



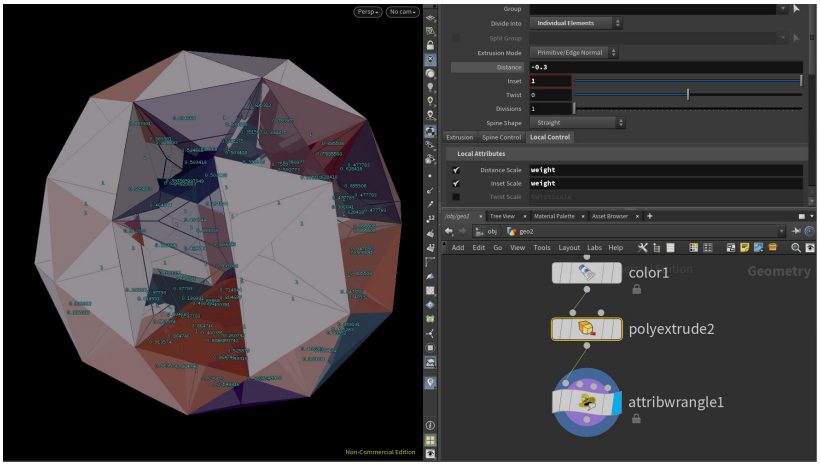

**7:Polygon Extrude SOP**

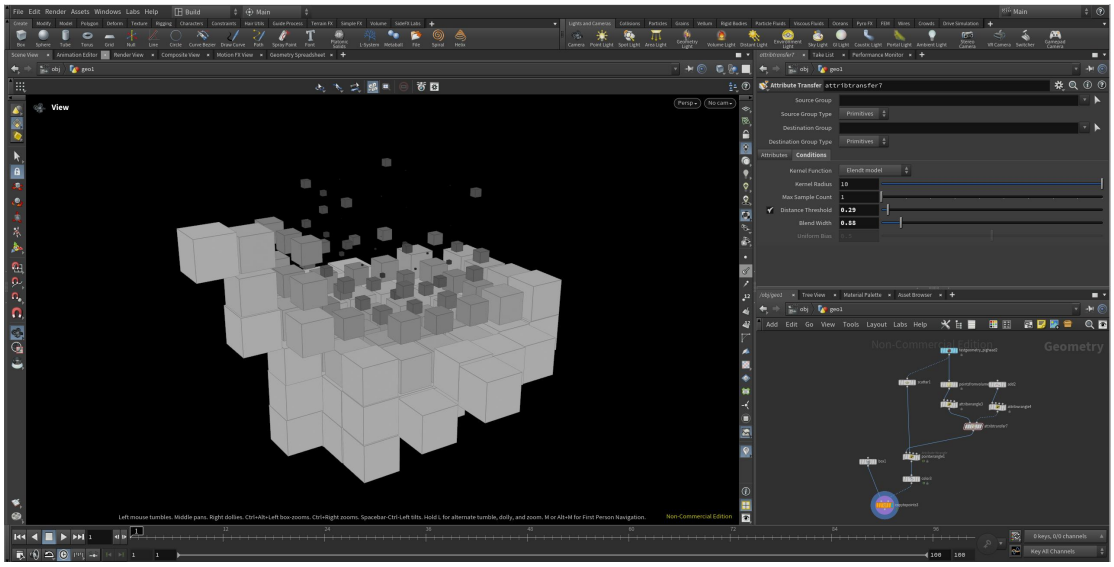
Create an ”Polygon Extrude” SOP, which used to extrude inwards each primitive on enlarged sphere generally based on the value of weight.

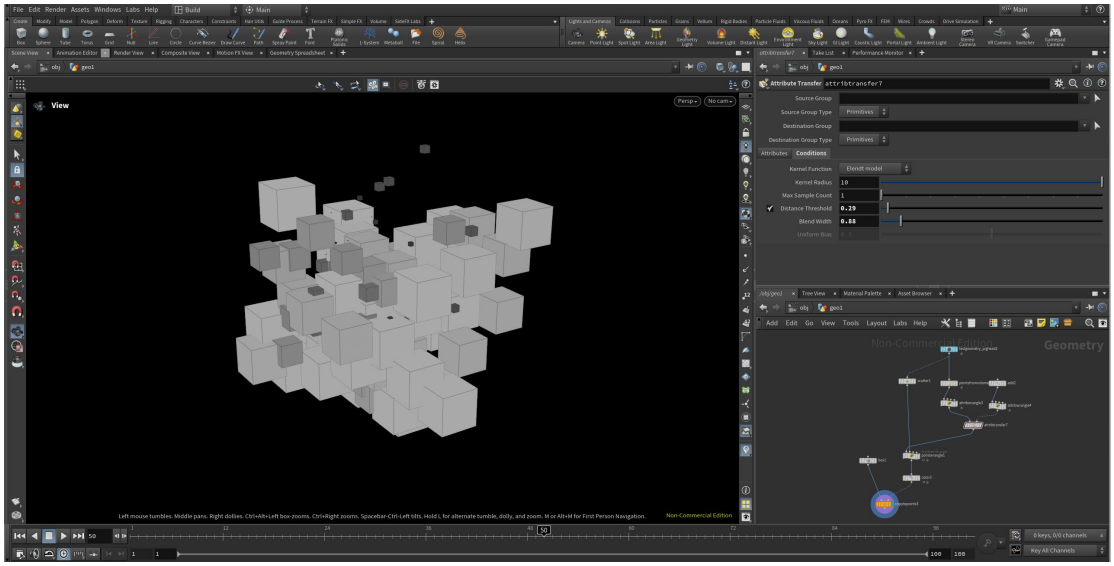


**8:Attribute Wrangle SOP**

Create an ”Attribute Wrangle” SOP, where we add a ”Alpha” attribute to the primitives. Here the alpha of each primitive will based on the value of weight.







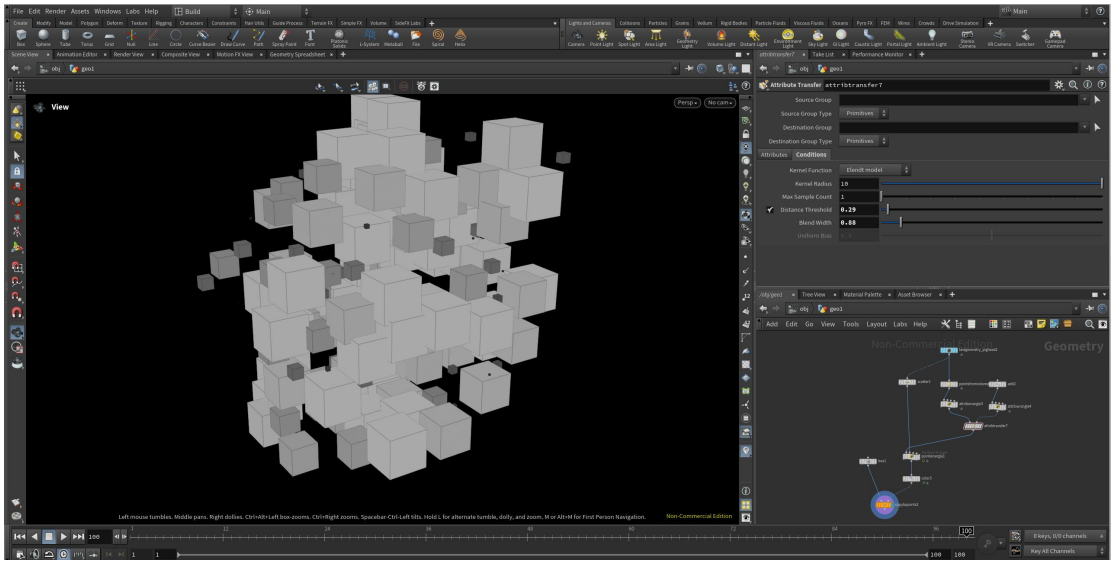
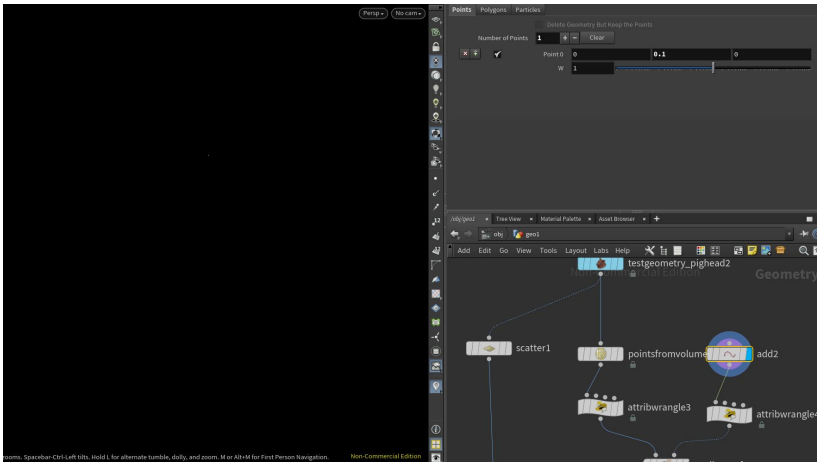


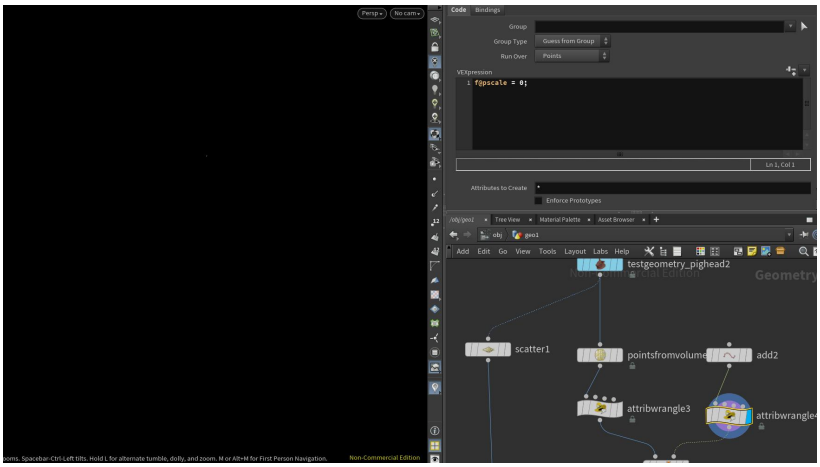
**Figure 1-3: Houdini Fundamentals: Setup 2 Overview**

**1:Test Geometry： Pig head SOP**
Create an ”Pig head” SOP. This is just a way to generate a base form to build on for subsequent operations.


**2:Add SOP**
Create an ”Add” SOP. This is generate one point at a certain place.

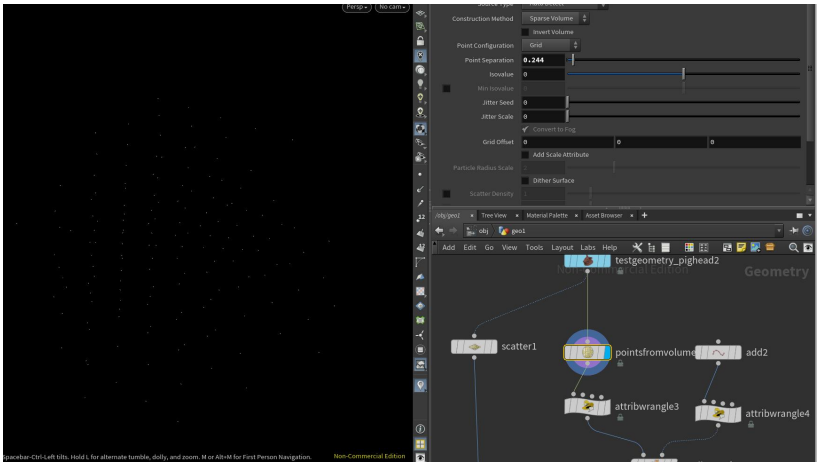



**3: Attribute Wrangle SOP**
Create an ”Attribute Wrangle” SOP, which is used to set the scale of particle into 0.



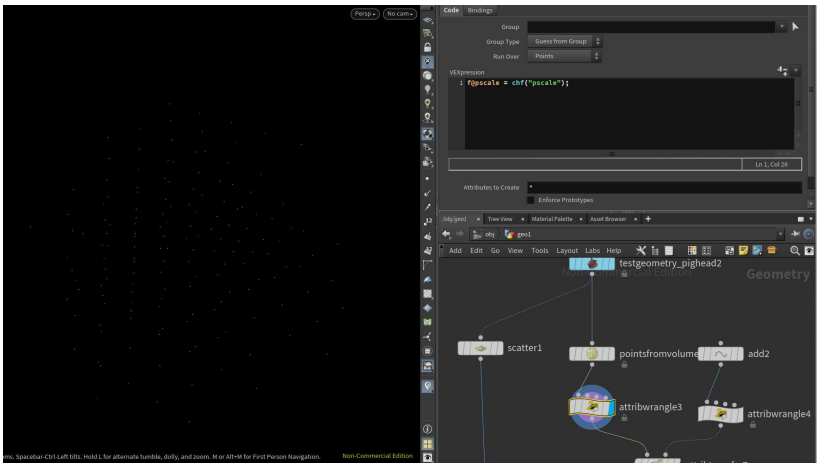
**4:Point From Volume SOP**

Create an ”Point From Volume” SOP. This is generate some point inside the volume before based on certain separation grid.



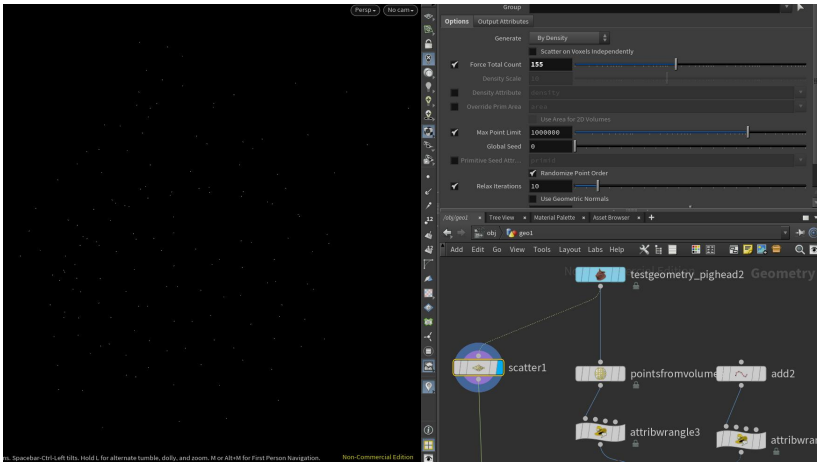

**5: Attribute Wrangle SOP**

Create an ”Attribute Wrangle” SOP, which is used to set the scale of particle refers to the value set before.



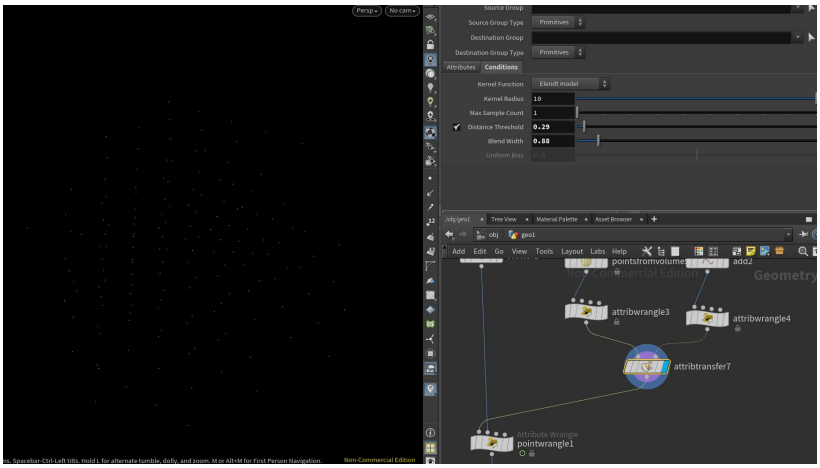

**6:Scatter SOP**

Create an ”Scatter” SOP, which is used to generate same amount of points with before on the surface of based shape.



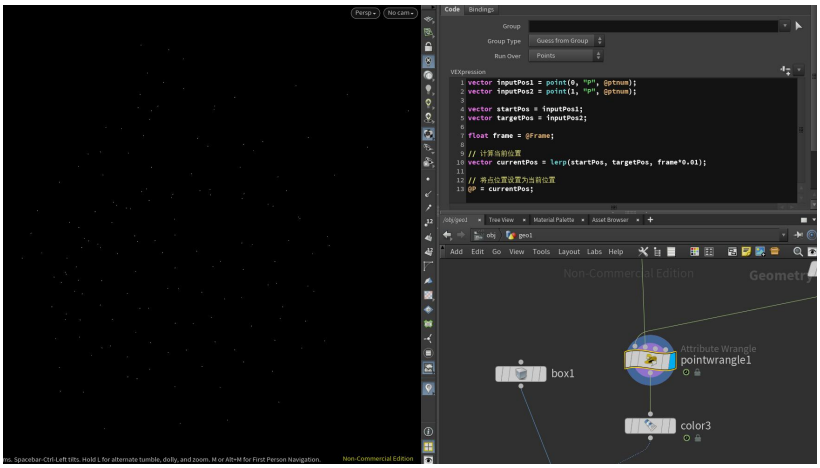
**7:Attribute Wrangle SOP**

Create an ”Attribute Transfer” SOP, which used to transfer particle size from the value set before to 0.



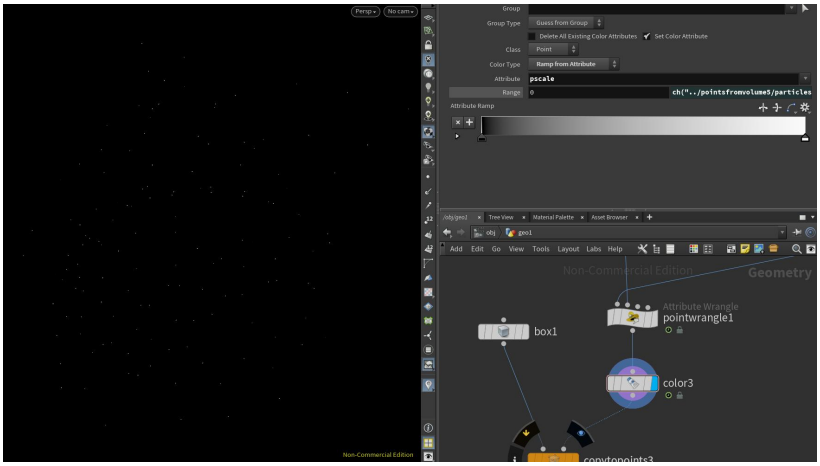

**8:Attribute Wrangle SOP**

Create an ”Attribute Wrangle” SOP, which is used to gradually change the position of point generated from volume to the position of point generated by scattering.



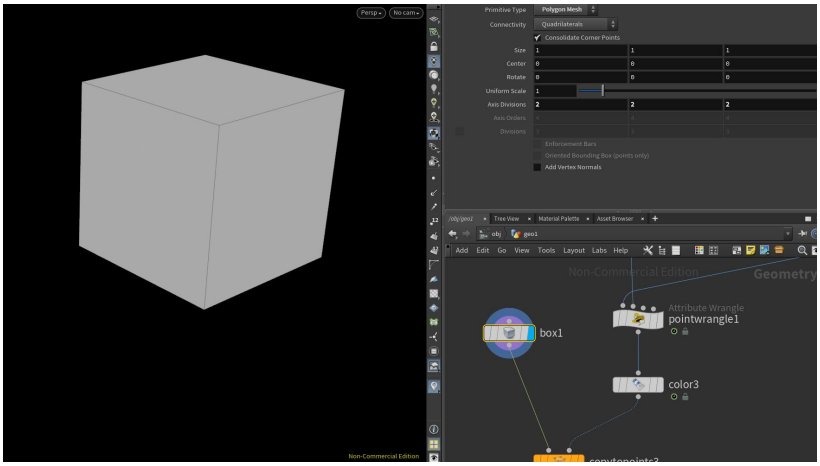

**9:Color SOP**

Create an ”Color” SOP, where we add color to the points based on the value of scale of particles.



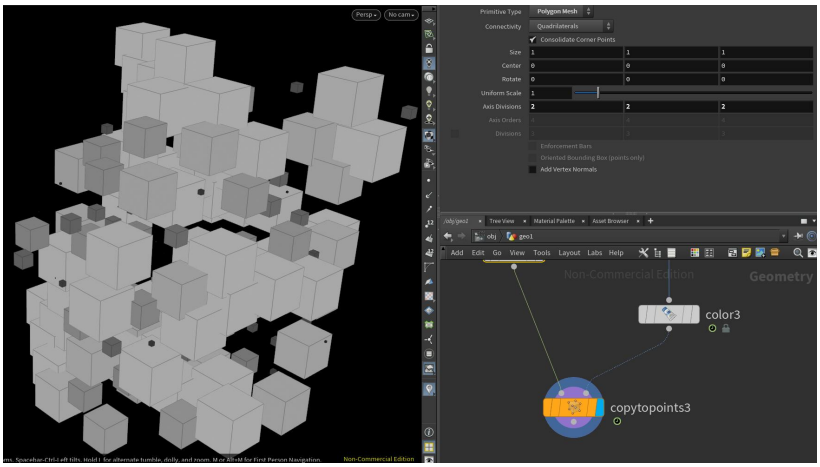
**10:Box SOP**

Create an ”Box” SOP, which is used to generate a 1*1*1 size box at certain place.



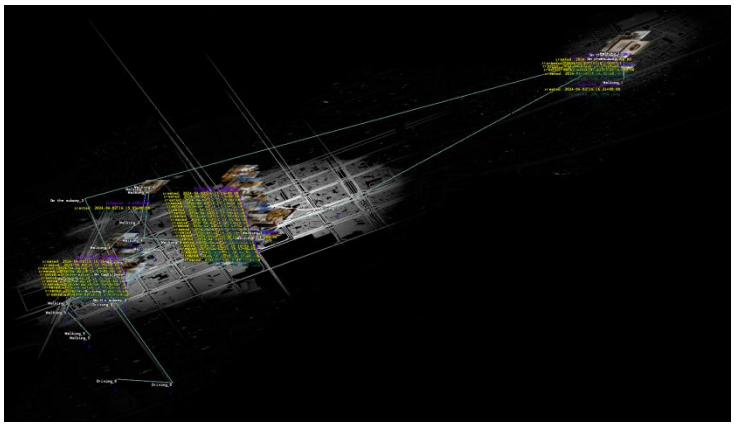

**11:Copy To Points SOP**

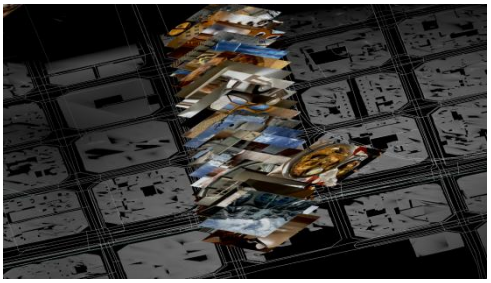
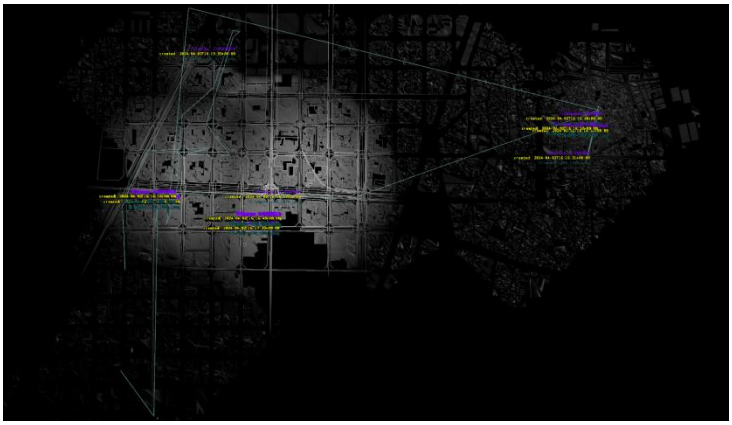
Create an ”Copy To Points” SOP, which is used to apply the generated box to generated points. Also apply all of attributes of points to the corresponding boxes.



# 2 Visulaizing_gpx

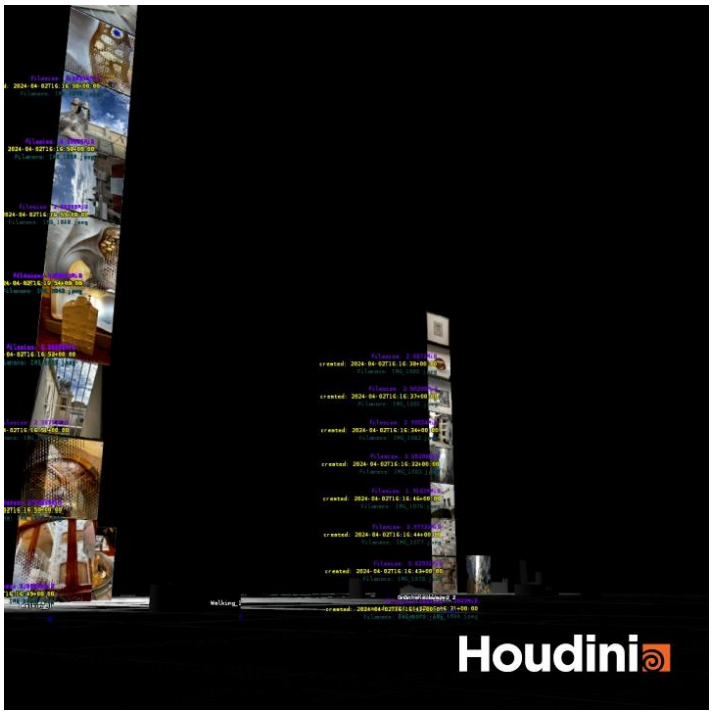
For this assignment I completed the basic data file loading



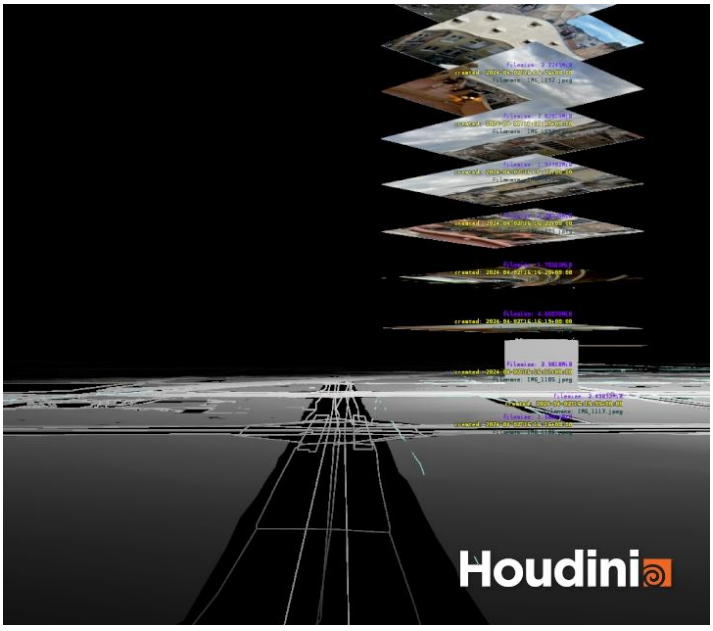

In addition, I fulfilled the requirement of generating a sequence of images according to three version

## First version




The first version produces images in the output1 folder, by rotating the photos and changing the gaps so that the viewpoint can see most of the images as it travels along the path.

## Second version



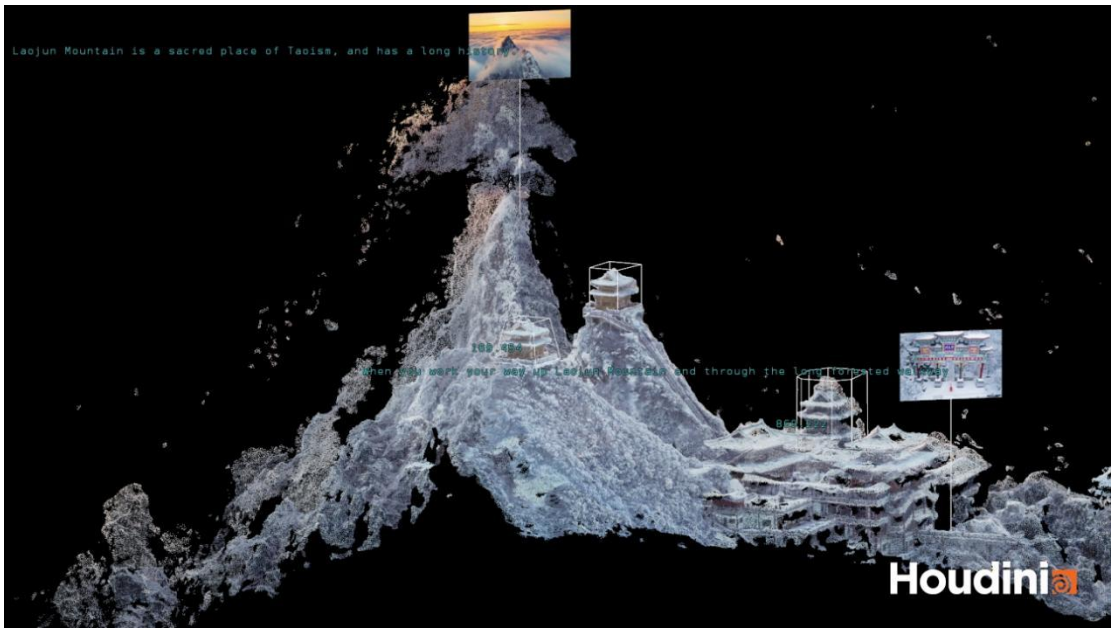
The second version is the perspective taught in the class, which is mainly based on the content of the course made annotation and picture overlap relationship adjustment. **Meanwhile with this version I tried to remove the mark by cropping inside houdini, but I found that the mark is still present in the output after cropping.**

## Third version

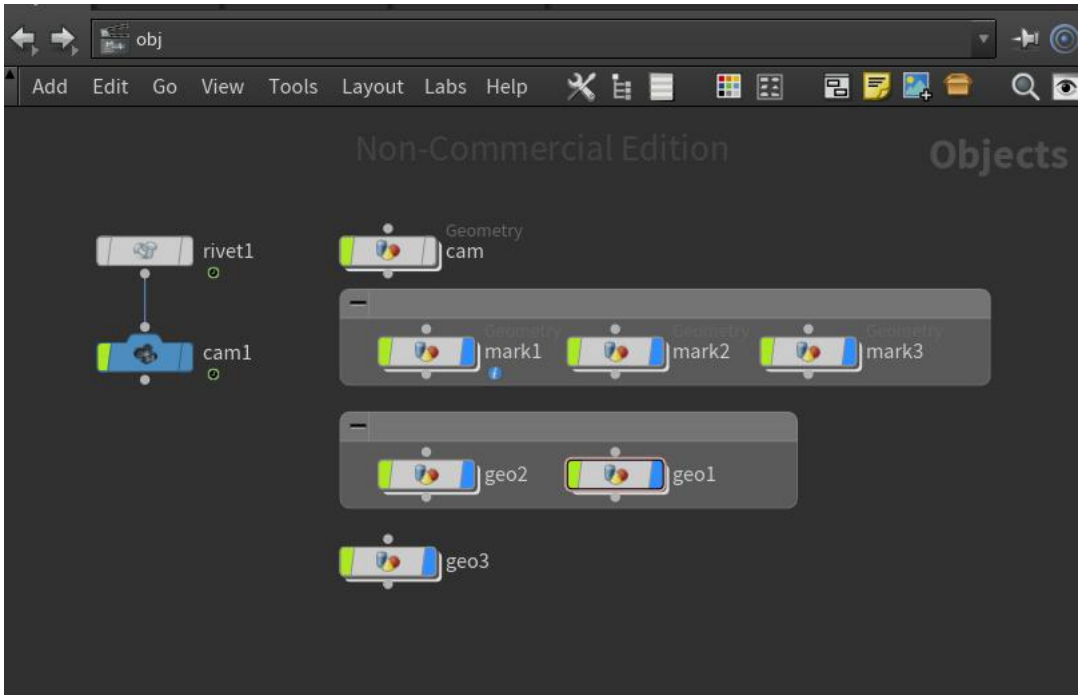

The third version focuses on pulling up the perspective, emphasising the relationship between the location and the surrounding urban environment, and helping to clarify the exact location and interrelationship of the path and images data in the city.

All three version uploaded to OneDrive, and here is the link:
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlz4_ucl_ac_uk/Emi10D34fYdLpkmyHRxJyEoBbIfA3sDBQhQ3yfx8Ut8pQg?e=klOEIa

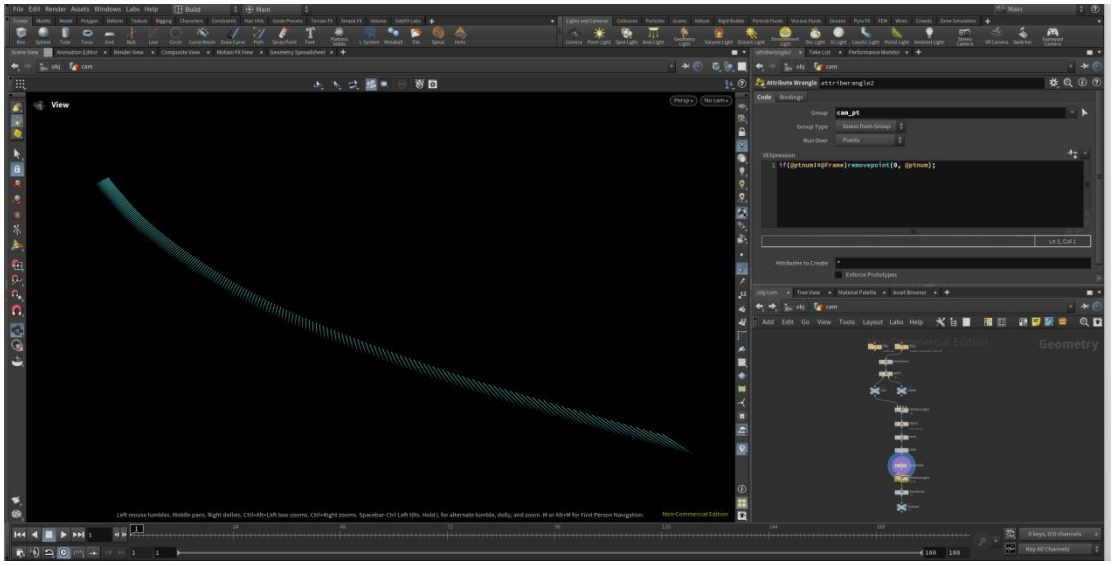
# 3 Video to 3D Model




This assignment file is divided into four parts in total, the first part is the reconstruction of the camera path, the second part is the labelling of the tower's body volume information, the third part is the labelling of the node perspective views and corresponding descriptions, and the fourth part is the point cloud of the base model.

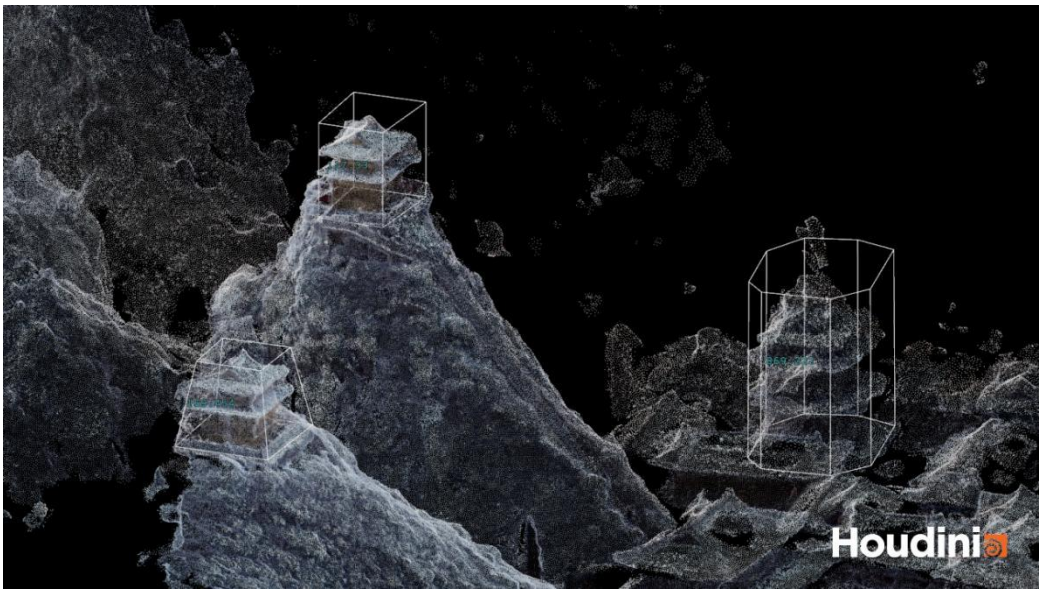


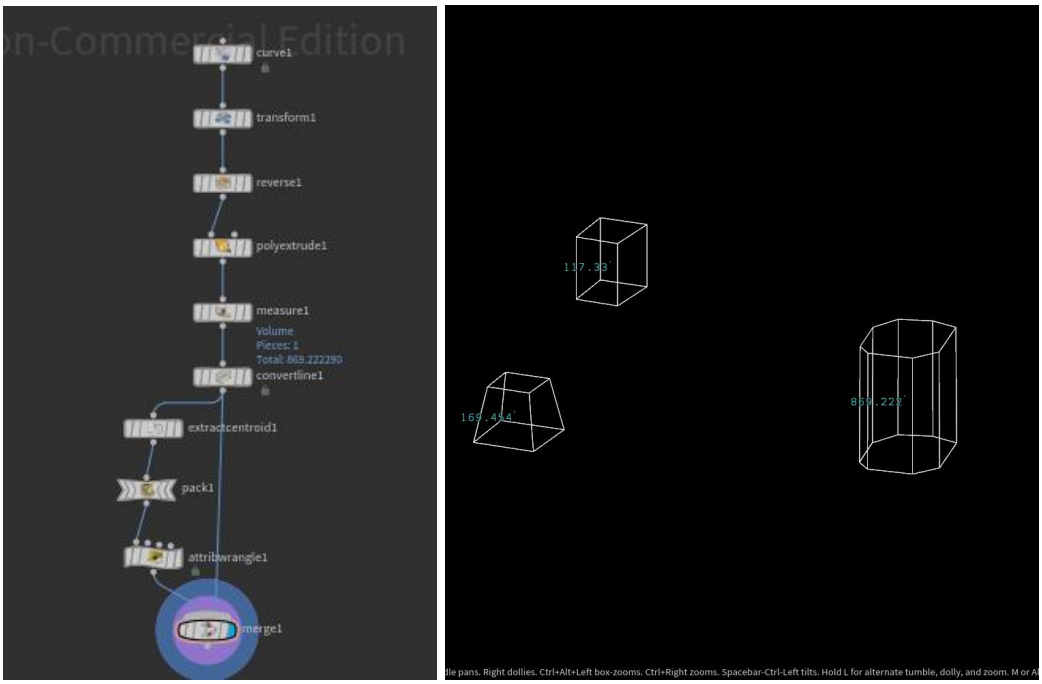
## First part: Reconstruction of the camera path



In the first part, the camera path orientation is derived by finding the centroid and connecting them as a vector for each individual in the camera path model.

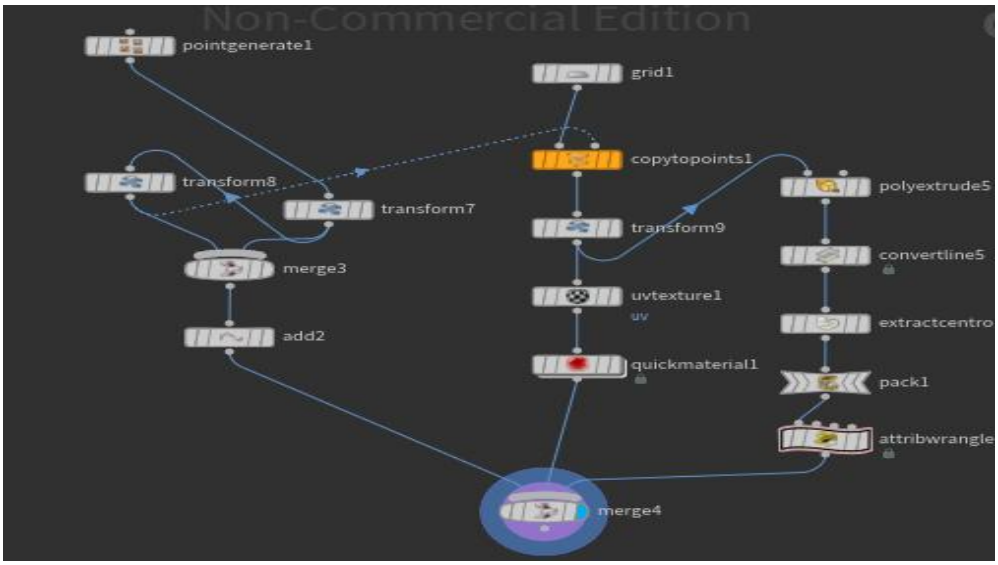
## Second part:Labelling of the tower's body volume information





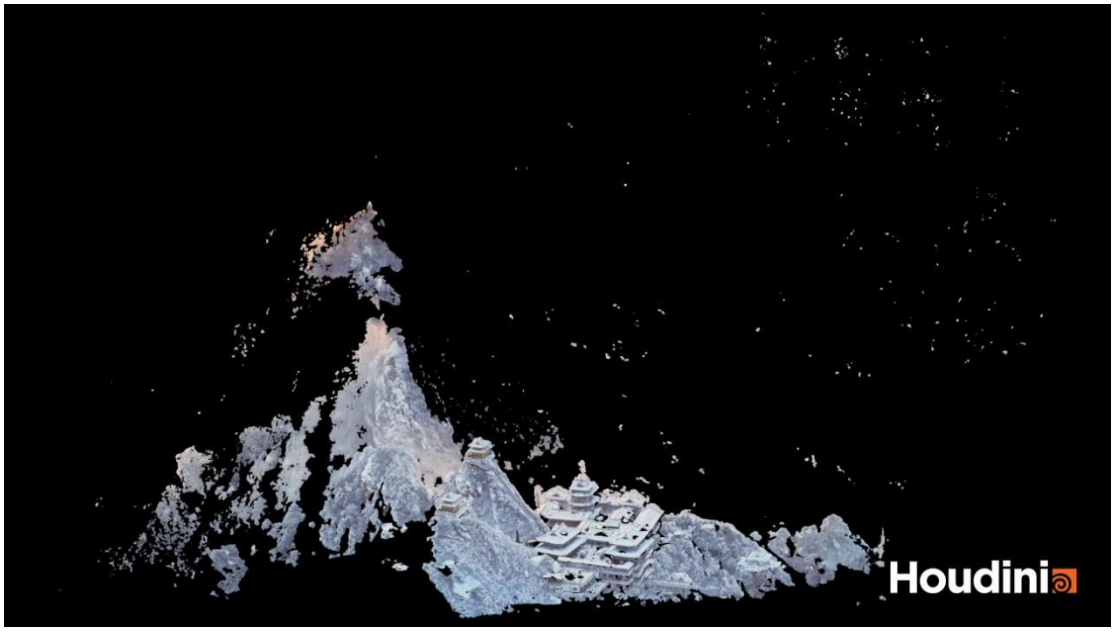

The second part determines the approximate volume of the tower by determining the base area of the tower from the top view, followed by extrusion, and then measuring the volume of the extruded volume.

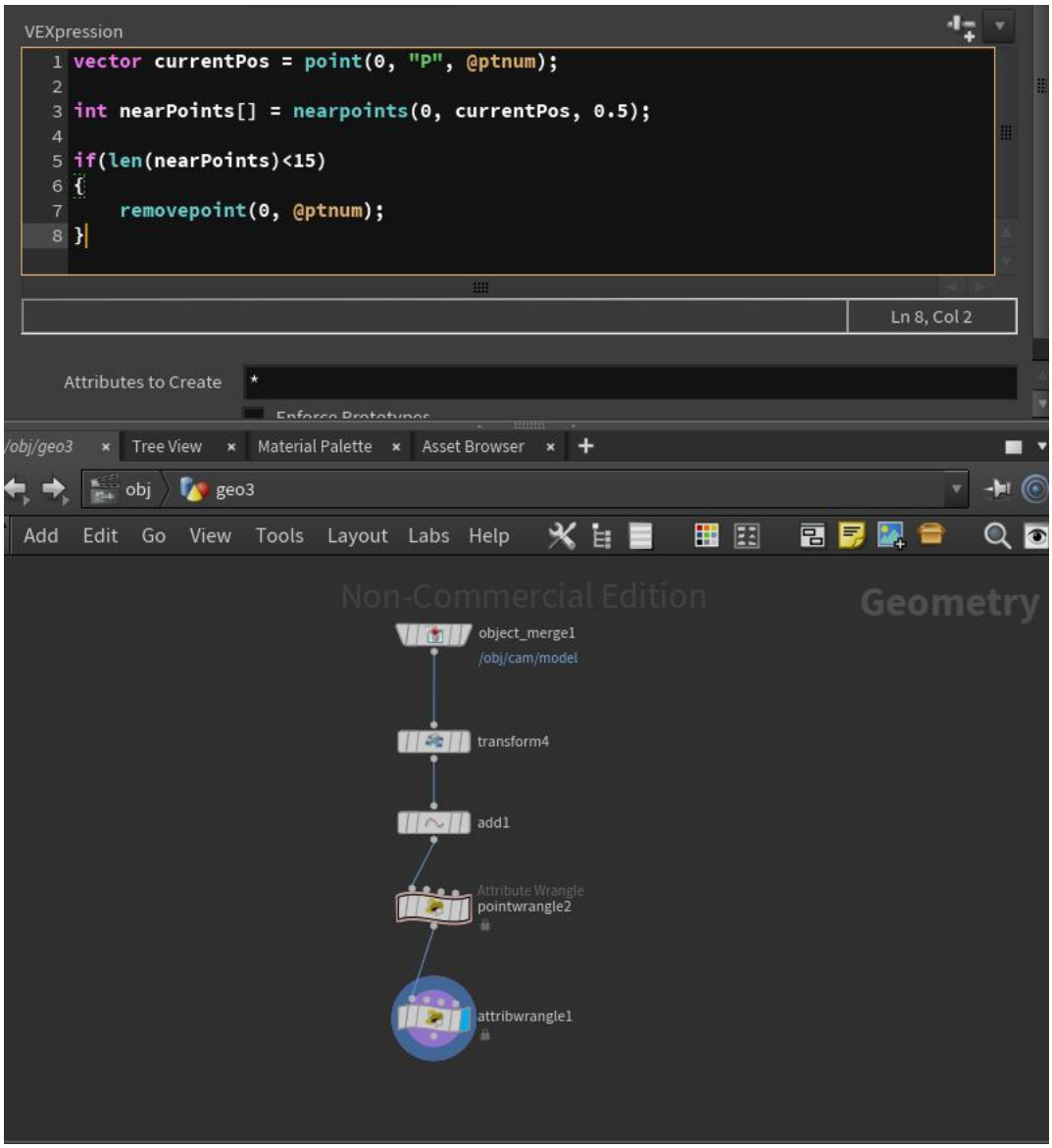
## Third part: Labelling of the node perspective views and corresponding descriptions




The third step shapes the node perspective by finding the point where the perspective is located, creating a plane above it, and giving the plane the photo where the perspective is located as a material.

## Fourth part: Transform basic model into point cloud






In the fourth step, after transforming the base model into a point cloud, the denseness of the point cloud is calculated and the sparse part of the point cloud is removed to obtain the final point cloud model.

The output frames uploaded by OneDrive:

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlz4_ucl_ac_uk/EtrE-cVA4UZGmUjC0BAhDe8BANPzGS8HNSxLtiOcZptxpQ?e=yj6qa7

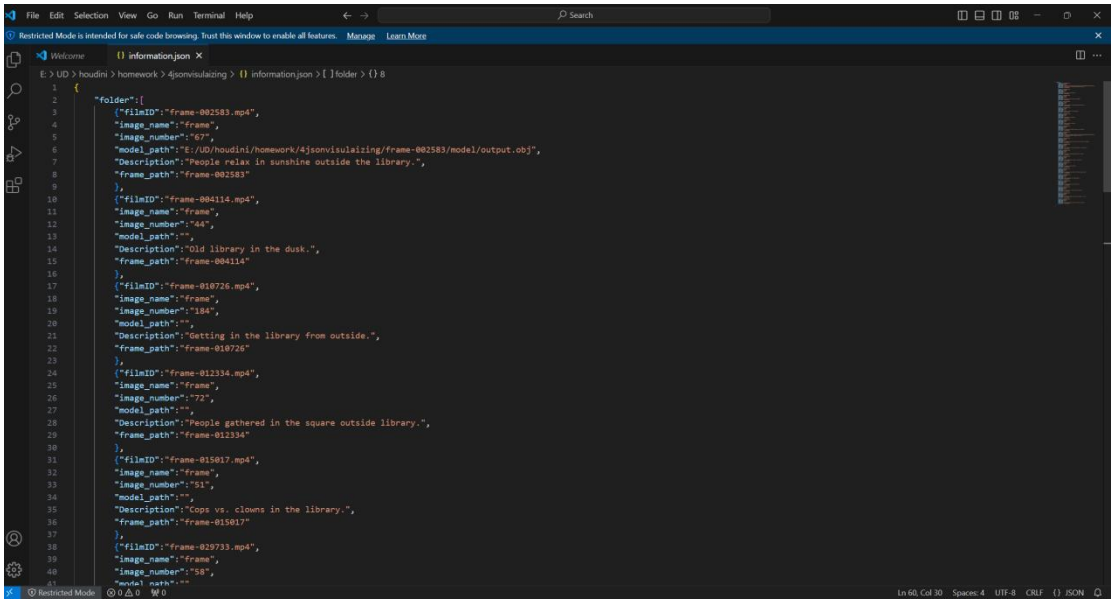
# 4 Visulaizing_.json File


Our topic in this design class focuses on the relationship between knowledge and power, so we are targeting a number of authoritative organisations that disseminate knowledge. For this visualisation, I focused on collecting videos related to libraries, obtaining their frames, transforming the buildings in individual frames into models, and describing them in relation to the content of the videos.
Here is the OneDrive link to the database:
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlz4_ucl_ac_uk/EijwJ8ER-LFDi3c4f0MuqCMBDbCq3jadY348Y6zPF-Orag?e=mhcfaF

## First part: Generate .json file

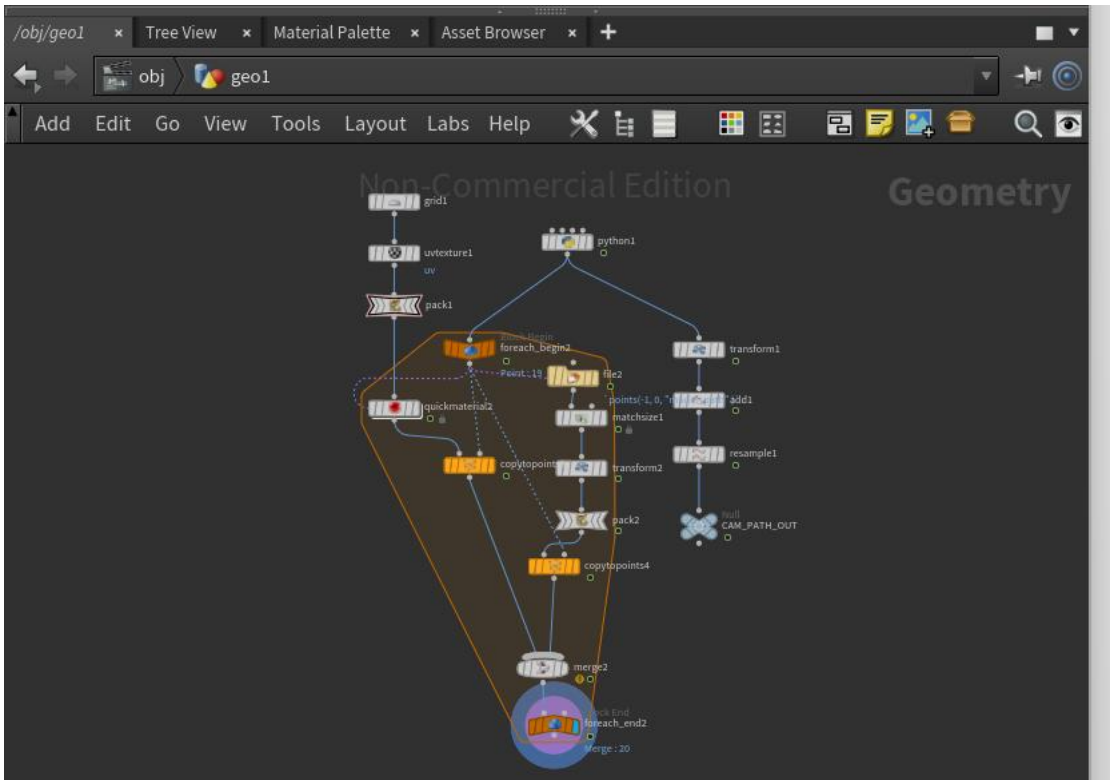
```
In [7]: cd E:\UD\houdini\homework
        E:\UD\houdini\homework

In [12]: folder_path = "E:\\UD\\houdini\\homework\\4 json visulaizing\\film"

         files = os.listdir(folder_path)

         filelist=[]
         for file in files:
             filelist.append(file)

In [13]: filelist
Out[13]: ['frame-002583.mp4',
          'frame-004114.mp4',
          'frame-010726.mp4',
          'frame-012334.mp4',
          'frame-015017.mp4',
          'frame-029733.mp4',
          'frame-032073.mp4',
          'frame-036861.mp4',
          'frame-051479.mp4',
          'frame-071300.mp4',
          'frame-078970.mp4',
          'frame-089422.mp4',
          'frame-089751.mp4',
          'frame-095379.mp4',
          'frame-098222.mp4',
          'frame-106193.mp4',
          'frame-108999.mp4',
          'frame-121526.mp4',
          'frame-129444.mp4',
          'frame-402528.mp4']

In [17]: cd "E:\\UD\\houdini\\homework\\4 json visulaizing\\film"
         E:\UD\houdini\homework\4 json visulaizing\film

In [22]: for i in filelist:
             imagepaths "E:\\UD\\houdini\\homework\\4 json visulaizing\\"+str(i)
             print(imagepath)
             frame_no = 0
             cap = cv2.VideoCapture(str(i))
             while (cap.isOpened()):
                 try:
                     ret, frame = cap.read()
                     if(frame_no % 10 == 0):
                             name =  f"frame_{frame_no}.jpg"
                             target = os.path.join(imagepath,name)
                             cv2.imwrite(target, frame)
                     frame_no +=1
```

Extract the frames from the downloaded video by python and store them in different folders.




Afterwards, create a .json file based on its contents through visual studio code.

## Second part: Visualization



```
#make point string attributes
try:
    with open(json_file_path, "r") as f:
        data = json.load(f)

        Description = geo.addAttrib(hou.attribType.Point, "Description", "")
        filmID = geo.addAttrib(hou.attribType.Point, "filmID", "")
        image_name = geo.addAttrib(hou.attribType.Point, "image_name", "")
        model_path = geo.addAttrib(hou.attribType.Point, "model_path", "")
        image_number = geo.addAttrib(hou.attribType.Point, "image_number", 0)
        frame_path = geo.addAttrib(hou.attribType.Point, "frame_path", "")



        for i, item in enumerate(data["folder"]):
            point = geo.createPoint()
            position = hou.Vector3((i%2)*2,1,i)
            point.setPosition(position)


            Description_value = item["Description"]
            filmID_value = item["filmID"]
            image_number_value = int(item["image_number"])
            frame_path_value = item["frame_path"]
            frame = int(hou.frame())
            image_index = (frame // 10)*10
            frame_path = "E:/UD/houdini/homework/4jsonvisulaizing/" + str(frame
            image_name_value = item["image_name"]


            point.setAttribValue(Description, Description_value)
            point.setAttribValue(filmID, filmID_value)
            point.setAttribValue(image_name, frame_path)
            point.setAttribValue(image_number, image_number_value)


            model_path_value = "E:/UD/houdini/homework/4jsonvisulaizing/null.ob
            if item["model_path"] != "":
                model_path_value = item["model_path"]
                point.setAttribValue(model_path, model_path_value)
            else:
```

By reading all kinds of information in the .json file, creating a new attribute for the point and displaying it, and giving the picture as material to the generated grid at each point, and loading the model under the corresponding model path and placing it to the corresponding position, the following effect is completed.