BARC0053: Bartlett Architecture Skills Elective (B-Pro Courses) FINAL ASSIGNMENT

RC11_23160576

SKILLS CLASS RC11 - 23/24

※ *01 OneDrive and GITHUB*
02 OneDrive and GITHUB
03 OneDrive and GITHUB

SKILLS CLASS RC11 - 23/24

VECTORIAL ENCODINGS OF QUALITATIVE DOMAINS

FINAL ASSIGNMENT EXPLANATION AND OUTCOMES

Rc11
Tutor: Julian Besems
SN: 23160576

OneDrive:

· dataset (folder)
· painting (folder)
· palace (folder)
· book_TFIDF.pkl
· final_RC11_23160576.ipynb
· github (folder)

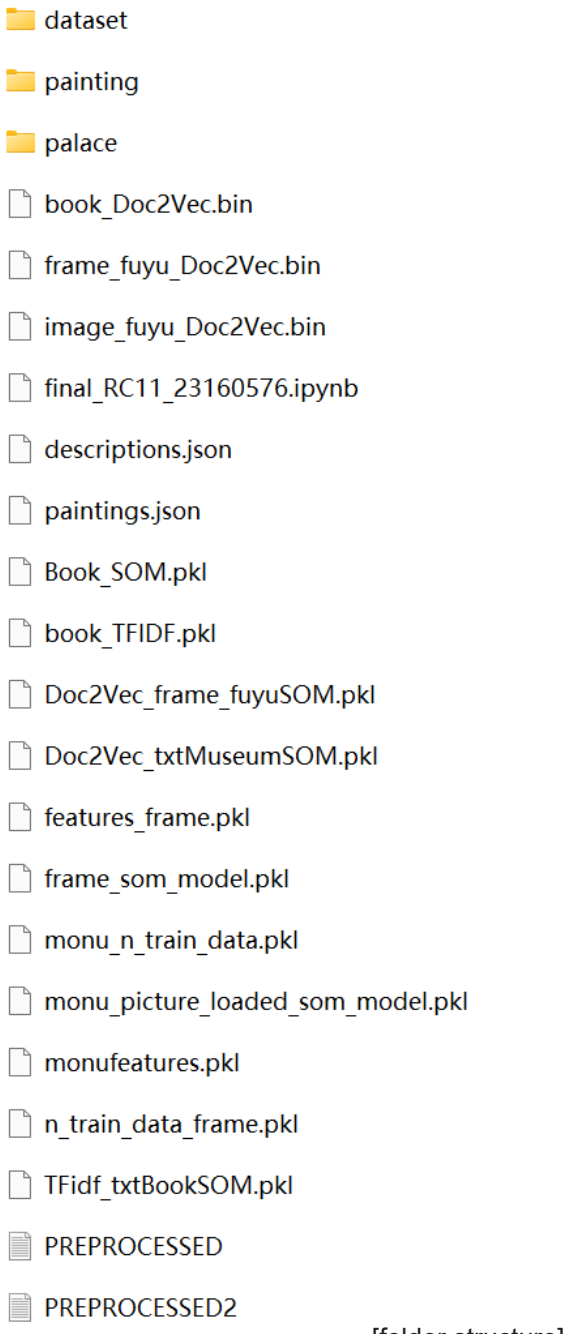
· RC11 23160576-GITHUB for Julian.zip

This OneDrive hyperlink contains the code for this assignment and all the files needed to run the code. (Including files in the FinalAssignmentJulian folder on GitHub)

Due to GitHub's size limitations for uploading files, the FinalAssignmentJulian folder on GitHub contains only .pkl, .json, and some .bin files.

I recommend downloading all the files directly from the OneDrive link, which is all the files in the RC11_23160576_FinalAssignmentJulian folder. Then, move all files from the 'github' folder to the path RC11_23160576_FinalAssignmentJulian. Keep the positions of other folders and files unchanged. Once your folder structure looks like this, you should be able to run final_RC11_23160576.ipynb smoothly.

※If the hyperlink doesn't open, please try this URL: https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlul_ucl_ac_uk/ElJ64Aqq059AlM74fHNn3k8Bhn0VawJZ1UB6mLZqEy1P0g



[folder structure]

GITHUB

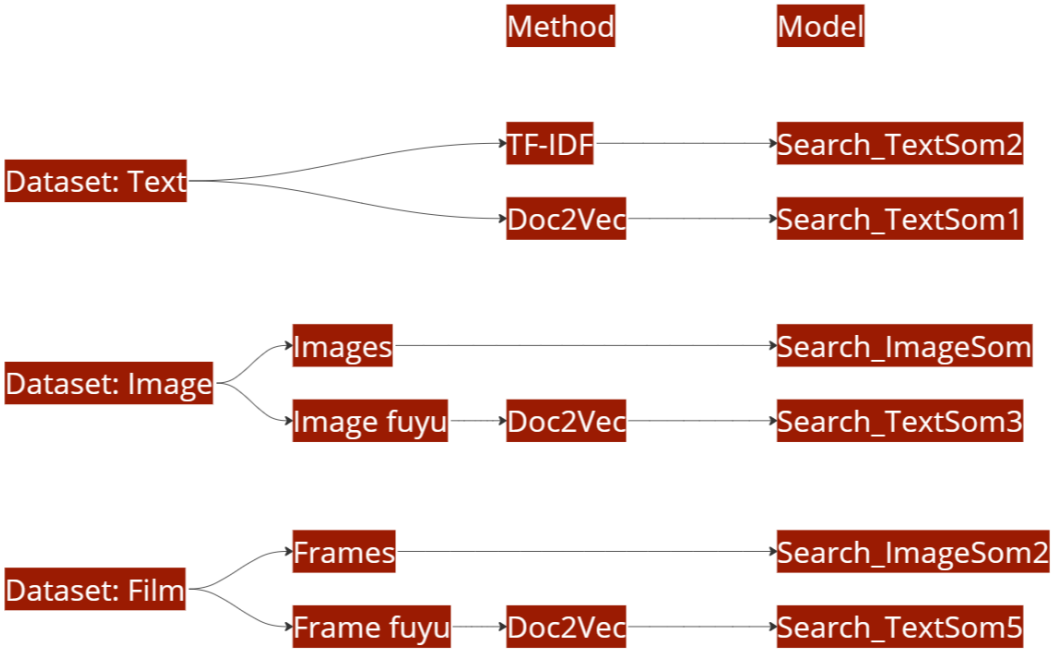
※If the hyperlink doesn't open, please try this URL: https://github.com/UD-Skills-2023-24/RC11_23160576/tree/main

Content

0 Introduce

In this task, I created a search engine that searches in three datasets, the three datasets are text, image, and film.

The first dataset is the epub files about anarchy, there are more than 2000 files. The second dataset is image files about the monument, with 1000 images. The third dataset is film files about the opposite of the monument, such as films about public space, with more than 600 files.



A. Input elements from type_text (text-text; text-image-film)

Step1: searching for matches in dataset_1_epub through TF-IDF; (search_TextSom2)
Step2: searching for matches in dataset_1_epub through Doc2Vec; (search_TextSom1)
Step3: searching for matches in dataset_2_image through Doc2Vec; (search_TextSom3)
Steo4: searching for matches in dataset_3_film through the results of 2. (search_ImageSom2)

B-1. Input elements from type_image (image-text; image-image-film)

Step1: searching for matches in dataset_1_epub through TF-IDF; (search_TextSom2)
Step2: searching for matches in dataset_1_epub through Doc2Vec; (search_TextSom1)
Step3: searching for matches in dataset_2_image; (search_ImageSOM)
Steo4: searching for matches in dataset_3_film through the results of 2. (search_ImageSom2)

B-2. Input elements from type_image (image-text; image-text-film)

Step1: searching for matches in dataset_2_image; (search_ImageSOM)
Step2: searching for matches in dataset_1_epub through TF-IDF; (search_TextSom2)
Step3: searching for matches in dataset_1_epub through Doc2Vec; (search_TextSom1)
Steo4: searching for matches in dataset_3_film through the results of 2. (search_ImageSom2)

1 Dataset_text

1.1 Epub files

In [22]:
```
PREPROCESSED = preprocess([p['paragraph'] for p in PARAGRAPHS])
```

In [23]:
```
PREPROCESSED2 = preprocess([p['paragraph'] for p in PARAGRAPHS])
```

In [21]:
```
with open('PREPROCESSED.txt', 'w') as f:
    for line in PREPROCESSED:
        f.write(line + '\n')
```

In [24]:
```
PREPROCESSED = []

with open('PREPROCESSED.txt', 'r') as f:
    for line in f:
        PREPROCESSED.append(line.strip())
```

In [23]:
```
with open('PREPROCESSED2.txt', 'w') as f:
    for line in PREPROCESSED:
        f.write(line + '\n')
```

In [25]:
```
PREPROCESSED2 = []

with open('PREPROCESSED2.txt', 'r') as f:
    for line in f:
        PREPROCESSED2.append(line.strip())
```

PREPROCESSED.txt
PREPROCESSED2.txt

1.2 Vectorise dataset

·Doc2Vec

In [46]:
```
DOC2VEC_DOCUMENTS = [TaggedDocument(doc, [i]) for i, doc in enumerate(PREPROCESSED2)]

DOC2VEC_MODEL = Doc2Vec(vector_size=250, min_count=0, alpha=0.025, min_alpha=0.025, epoch
DOC2VEC_MODEL.build_vocab(DOC2VEC_DOCUMENTS)
```

In [47]:
```
DOC2VEC_MODEL.train(DOC2VEC_DOCUMENTS, total_examples=DOC2VEC_MODEL.corpus_count, epochs=
```

In [48]:
```
DOC2VEC_MODEL.save("book_Doc2Vec.bin")
```

In [49]:
```
loaded_model = Doc2Vec.load("book_Doc2Vec.bin")
```

book_Doc2Vec.bin

·TF-IDF

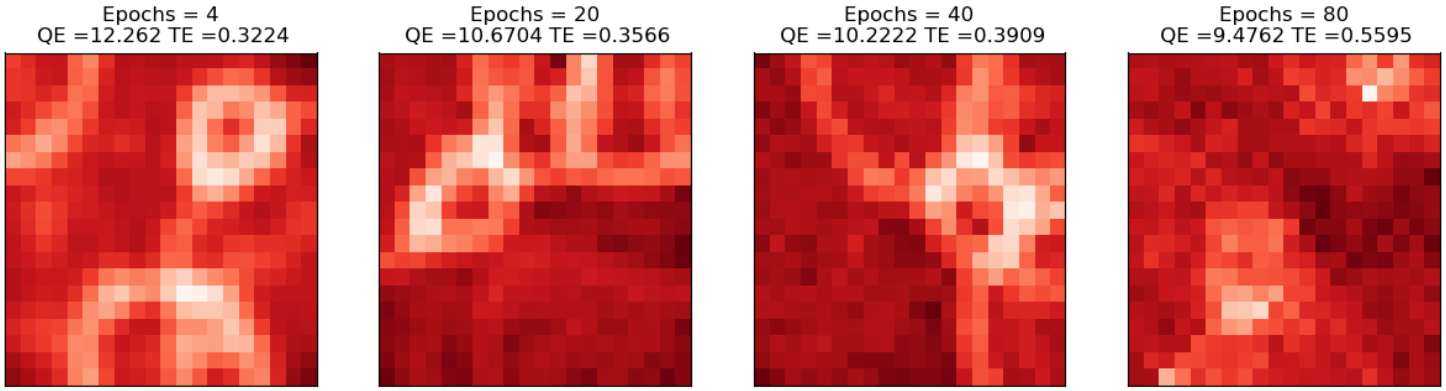
In [40]:
```
import pickle
with open('book_TFIDF.pkl', 'wb') as f:
    pickle.dump(TFIDF_MATRIX, f)
```

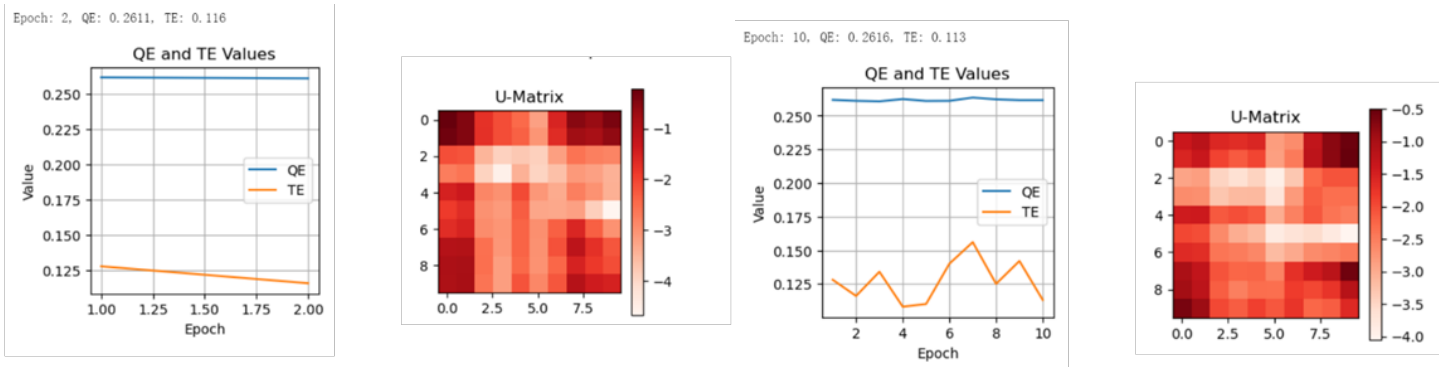
In [30]:
```
import pickle
with open('book_TFIDF.pkl', 'rb') as f:
    TFIDF_MATRIX = pickle.load(f)
```

book_TFIDF.pkl

1 Dataset_text

1.3 Epub_Doc2Vec SOM



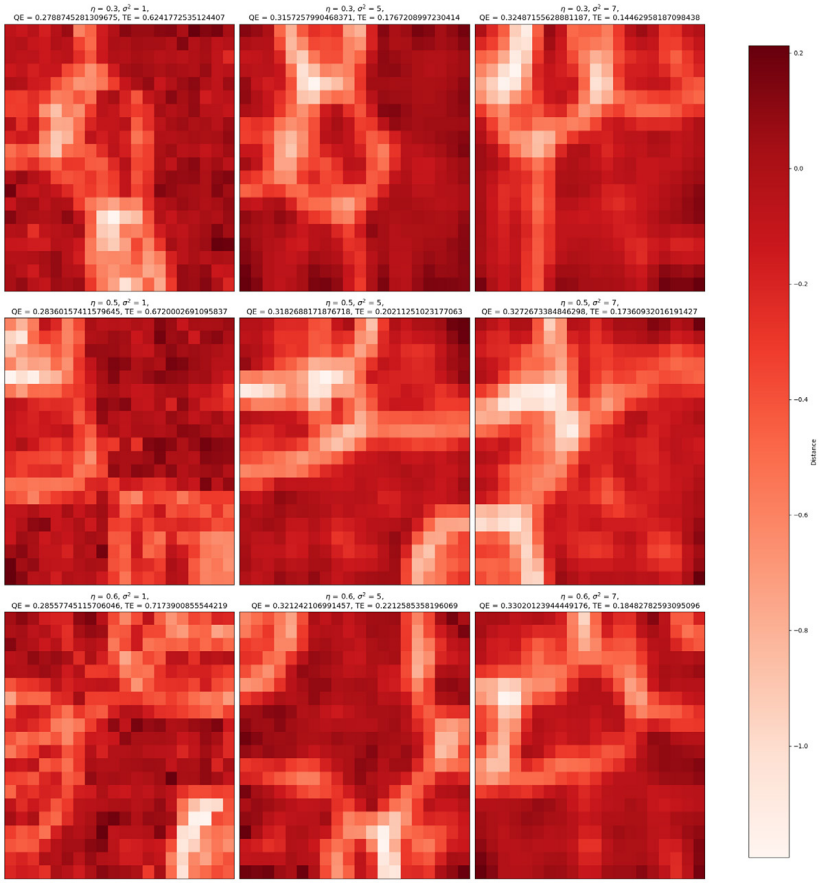


QE and TE Values with U-Matrix (Epoch 2 & Epoch 10)

TE measures how well a self-organizing map (SOM) preserves the topology of the input space. As training progresses, TE gradually decreases, eventually reaching a stable value. A smaller TE indicates that the SOM is better at preserving the topology of the input space.

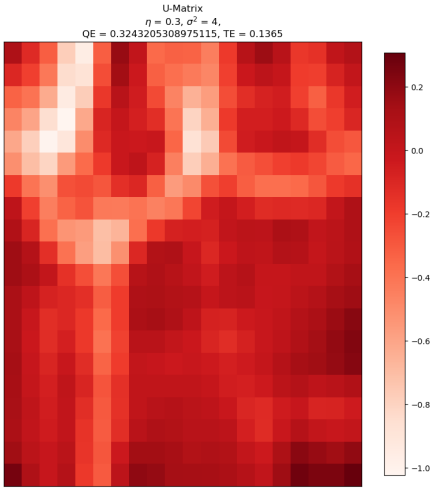
QE quantifies the discrepancy between the weight vectors of SOM neurons and the input vectors. During training, QE diminishes as training proceeds. When QE reaches a small value, it indicates that the SOM has learned the structure of the input data, and the weight vectors are sufficiently close to the input vectors. In other words, QE reflects the degree of matching between the weight vectors of the SOM and the input data.

I chose a SOM with index 2 because it has the lowest average of both TE and QE.



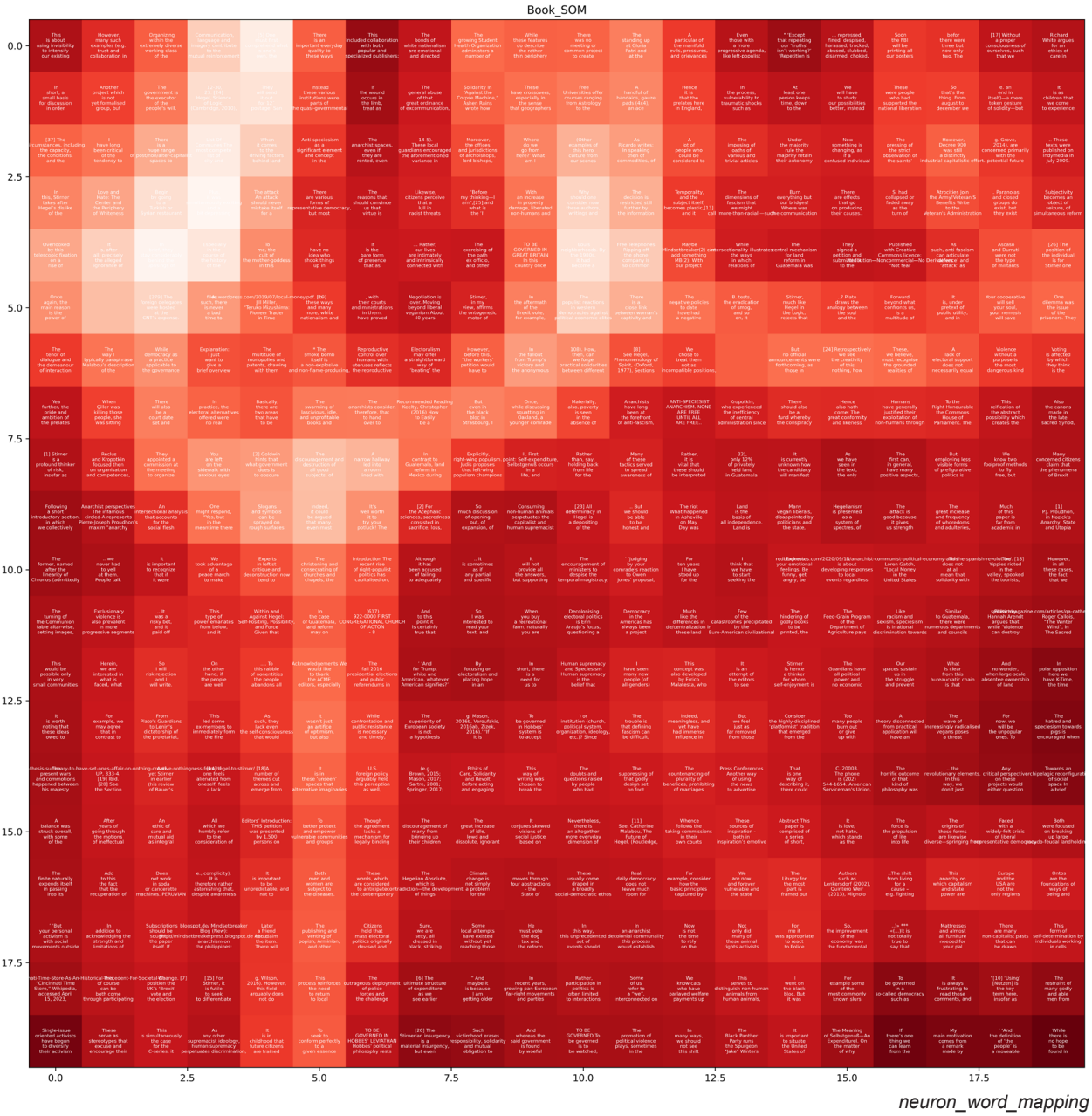
SOMS = []

1 Dataset_text



SOMS = [2]

Visualize the output of the SOM model to observe the topology among neurons and display the correlation between neurons and text data.



neuron_word_mapping
Doc2Vec

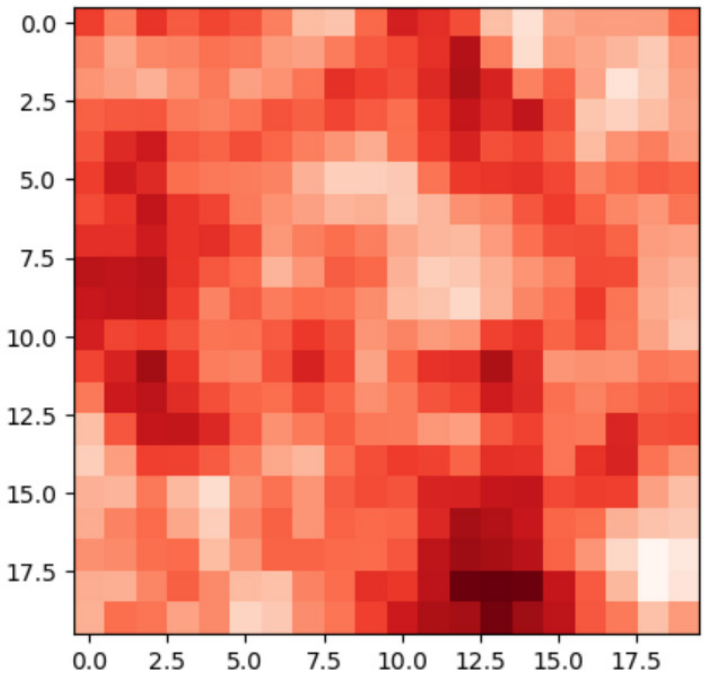
1 Dataset_text

1.4 Search function (text to text by Doc2Vec)

In [520]:
```
def search_TextSom1(SOM1, model, data_dict, query=''):
    result = []

    query = [query]
    preprocessed_query = preprocess2(query)
    query_vector = DOC2VEC_MODEL.infer_vector(preprocessed_query[0])

    print(preprocessed_query)

    fig = plt.figure()
    plt.imshow(activate1(train_data1,SOM1, query_vector), cmap=cmap)
    plt.show()

    g, h = find_BMU1(SOM1, 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']['paragraph']] = cosine_similarity(vector_array, [SOM1[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
```

In [145]: search_TextSom1(SOM1,model,data_dict,'*Anarchy reigns in the absence of governmental control.*')

```
[['anarchy', 'reign', 'absence', 'governmental', 'control']]
```

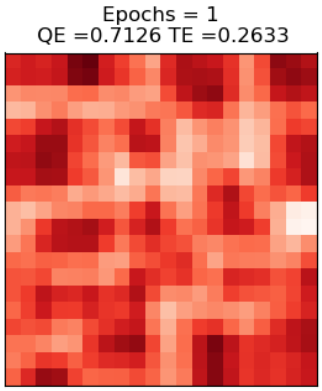


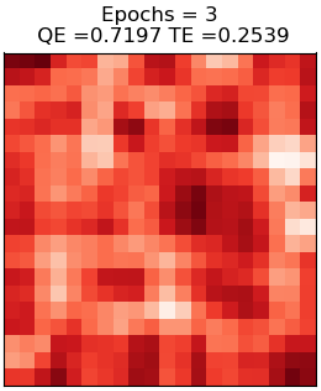

```
(18, 13)
```

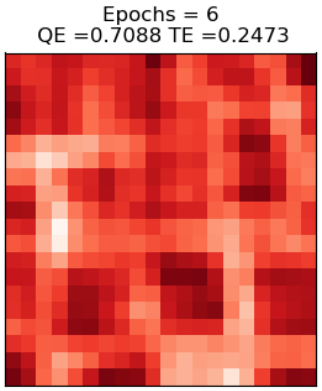
Out[145]:
```
[{'paragraph': ' This is presumably what the poet León Felipe had in mind when he wrot
e: “The nobility of Durruti’s life will inspire the birth of a legion of Durrutis in
the times to come.” Paris, April 1972 Revised in Paris, February 1977.',
```

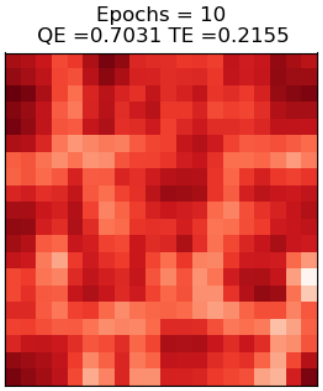
1 Dataset_text

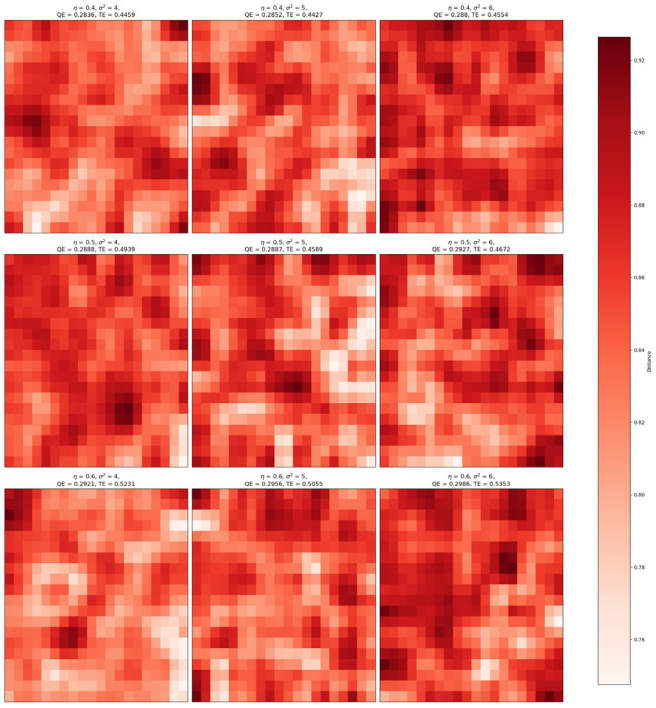
1.5 Epub_TF-IDF SOM



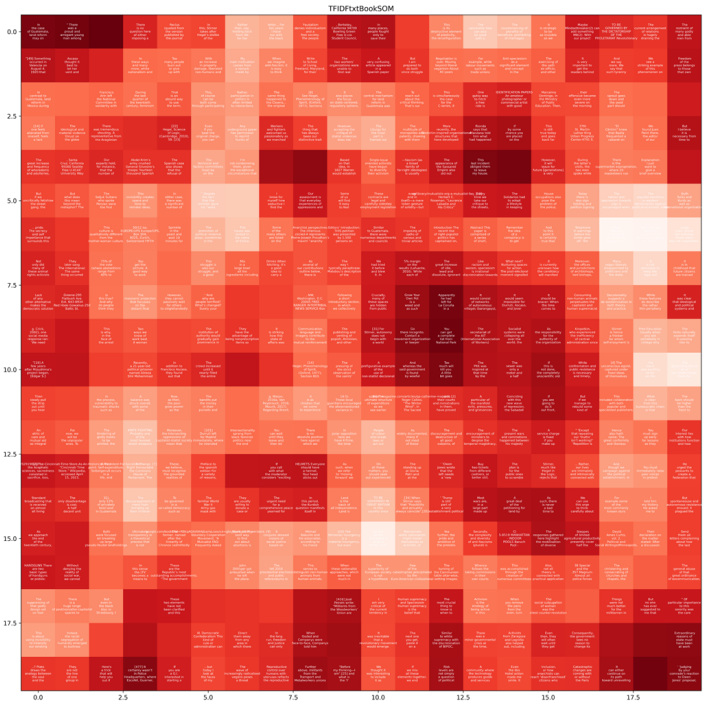








SOMS2 = []



neuron_word_mapping
TF-IDF

I chose a SOM with index 1 because it has the lowest average of both TE and QE.

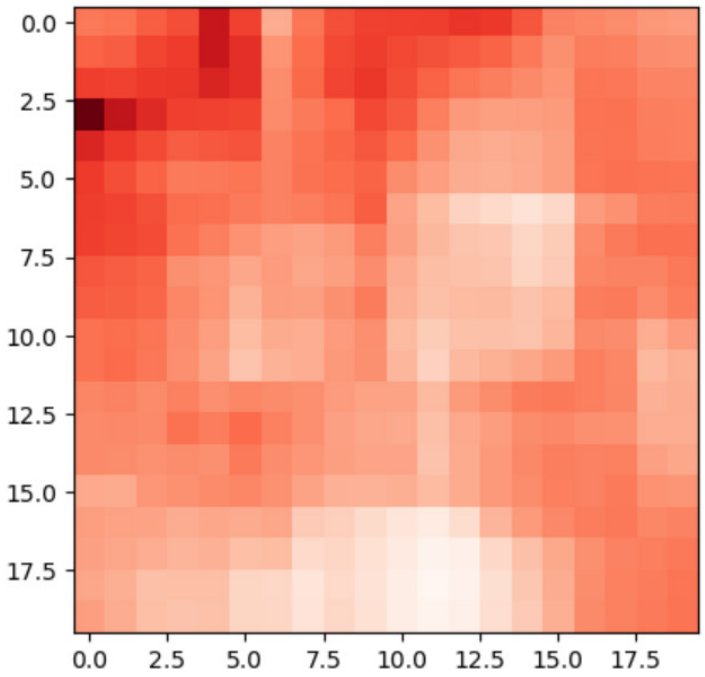
Visualize the output of the SOM model to observe the topology among neurons and display the correlation between neurons and text data.

1 Dataset_text

1.6 Search function (text to text by TF-IDF)

In [883]:
```
def search_TextSom2(SOM, model, data_dict2, query):
    result = []
    query_tfidf_vector = VECTORIZER_TFIDF.transform([query])
    query_svd_vector = svd.transform(query_tfidf_vector)
    activated_SOM = activate1(u_matrix_values, SOM2, query_svd_vector[0])

    fig = plt.figure()
    plt.imshow(activate1(u_matrix_values,SOM2, query_svd_vector[0]), cmap=cmap)
    plt.show()

    g, h = find_BMU1(SOM2, query_svd_vector[0])
    print((g, h))

    similarities = {}
    for i in data_dict2[g][h]:
        vector_array = np.array([i['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarities[i['text']['paragraph']] = cosine_similarity(vector_array, [SOM2[g][h]

    biggest = None
    max_similarity = similarities[data_dict2[g][h][0]['text']['paragraph']]
    for i in data_dict2[g][h]:
        sim = similarities[i['text']['paragraph']]
        if sim > max_similarity:
            max_similarity = sim
            biggest = i['text']
    result.append(biggest)

    tfidf_text = result[0]['paragraph']
    print("tfidf_text:", tfidf_text)

    return result
```

In [145]: search_TextSom2(SOM2, VECTORIZER_TFIDF,data_dict2,'*Anarchy reigns in the absence of governmental control.*')



Out[884]:
```
[{'paragraph': '[402] Cruells take us to the heart of the problems weighing on the Gene
ralitat at the time. They will be the cause of the events on October 6. On April 12, 19
34, the Generalitat enacted a law on agricultural contracts [ Llei de Contractes de Con
reu], which the Catalan Parliament approved.',
  'nr': 2889,
  'bookID': 'abel-paz-durruti-in-the-spanish-revolution'}]
```

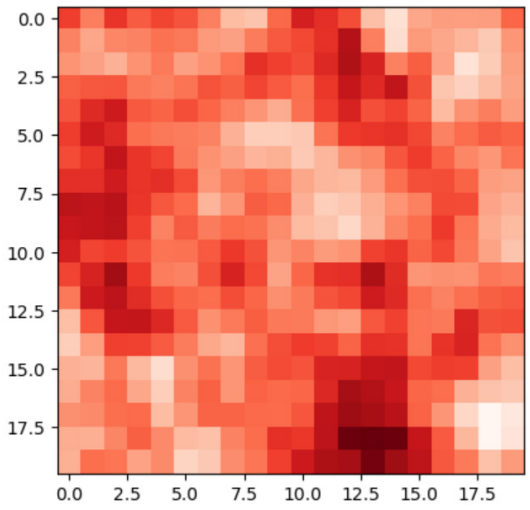
1 Dataset_text

Compare Doc2Vec & TF-IDF

In [145]: input: '*Anarchy reigns in the absence of governmental control.*'

output of search_TextSom1 (Doc2Vec)

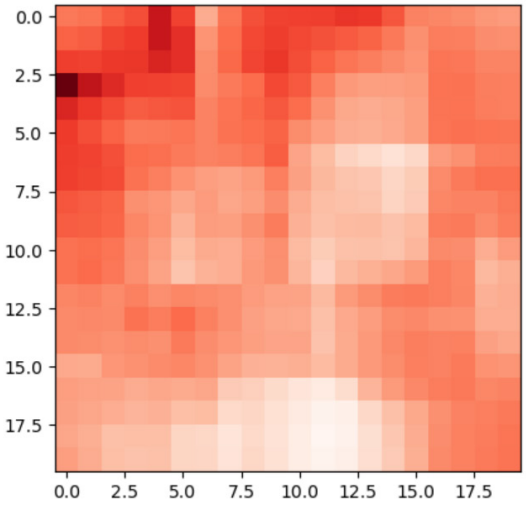
```
[['anarchy', 'reign', 'absence', 'governmental', 'control']]
```




```
(18, 13)
```

Out[145]:
```
[{'paragraph': ' This is presumably what the poet León Felipe had in mind when he wrot
e: “The nobility of Durruti’s life will inspire the birth of a legion of Durrutis in
the times to come.” Paris, April 1972 Revised in Paris, February 1977.',
```

output of search_TextSom2 (TF-IDF)



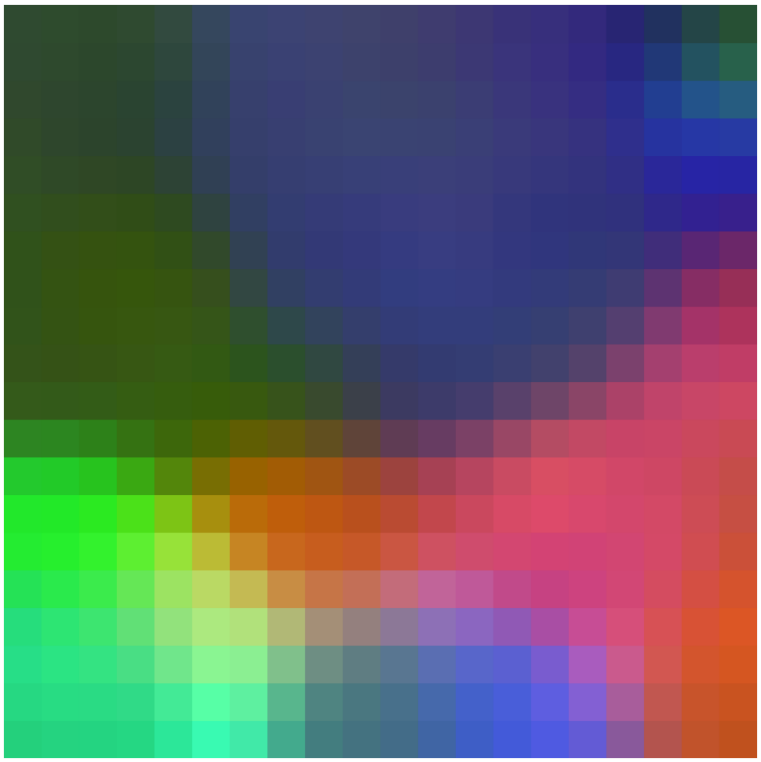
Out[884]:
```
[{'paragraph': '[402] Cruells take us to the heart of the problems weighing on the Gene
ralitat at the time. They will be the cause of the events on October 6. On April 12, 19
34, the Generalitat enacted a law on agricultural contracts [ Llei de Contractes de Con
reu], which the Catalan Parliament approved.',
  'nr': 2889,
  'bookID': 'abel-paz-durruti-in-the-spanish-revolution'}]
```

TF-IDF is suitable for simple text processing, being straightforward and intuitive, but it overlooks word order and semantics. Doc2Vec captures semantic information better, but it requires longer training time and more data.

1 Dataset_text

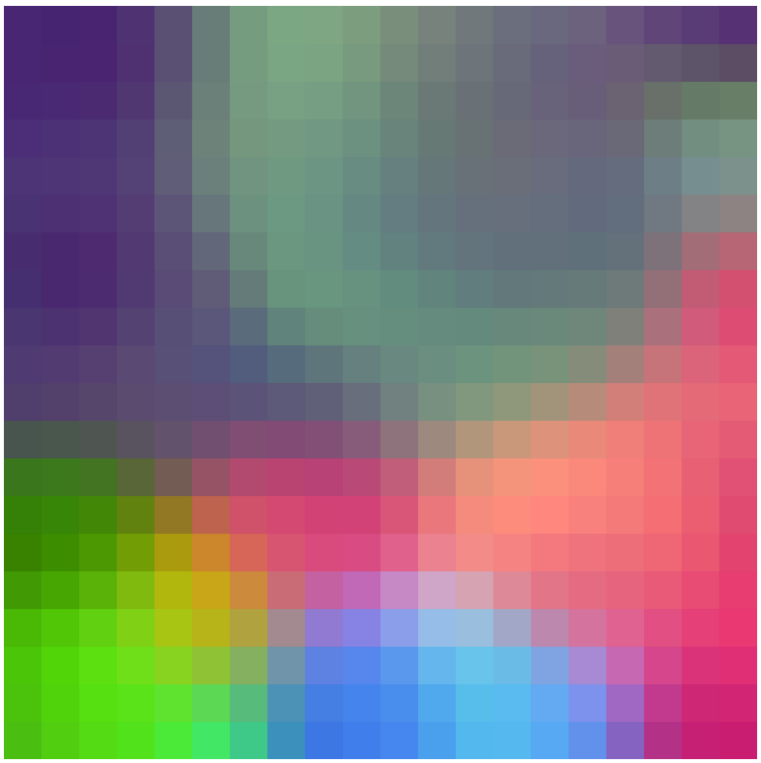
1.7 PCA

·Fit PCA on all cells



This method results in each SOM unit having its own PCA outcome, so the RGB color for each unit may differ, allowing for better capture of local features within the SOM. However, this might lead to discontinuities in colors between units.

·Fit PCA on the entire training



In this method, the entire SOM shares a single PCA outcome, ensuring consistency in color distribution across all units, resulting in more continuous and uniform colors. However, this approach may overlook some local features since PCA is fitted on the entire training set rather than on individual units.

2 Dataset_image

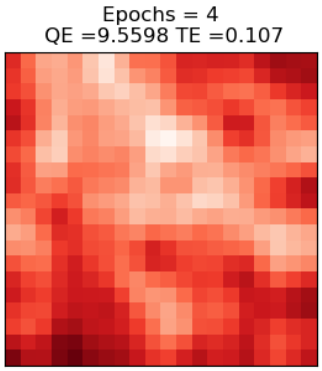
2.1 Image files

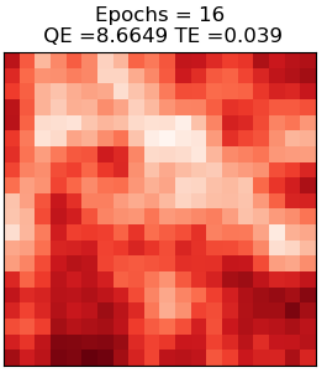
```
[773]: with open('monufeatures.pkl', 'wb') as f:
           pickle.dump(monufeatures, f)
```

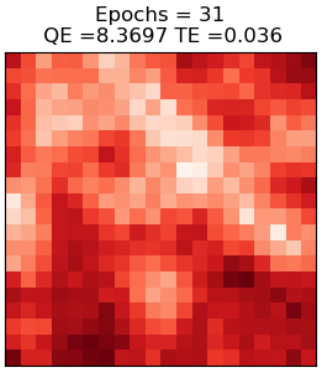
```
[774]: with open('monufeatures.pkl', 'rb') as f:
           monufeatures = pickle.load(f)
```

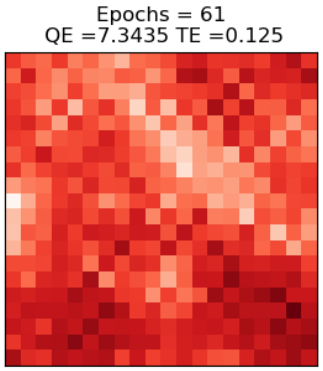
monufeatures.pkl

2.2 Image SOM





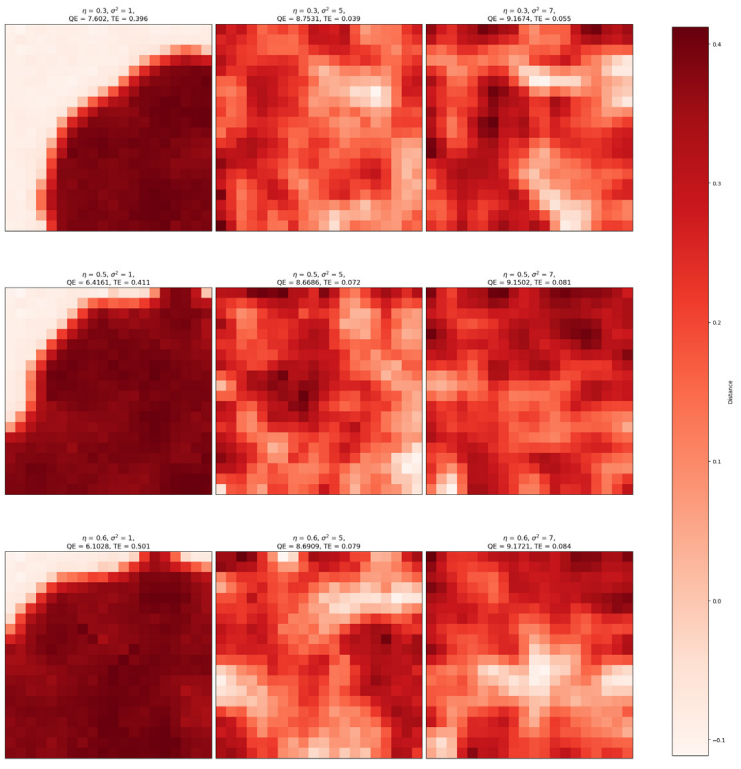




imageGrid

activatedSOM

I chose a SOM with index 1 because it has the lowest average of both TE and QE.



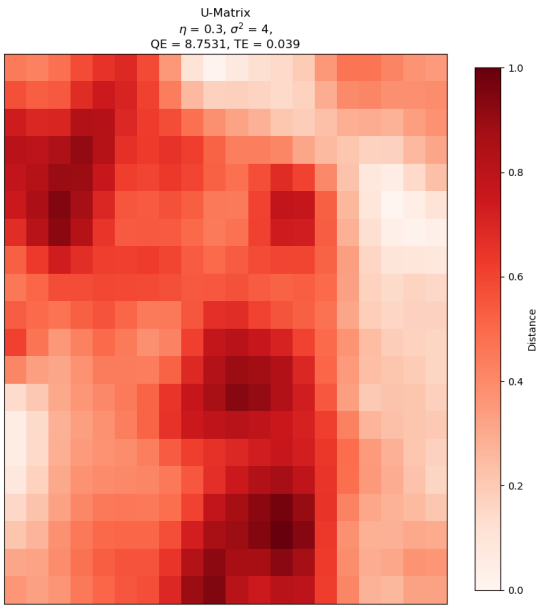
SOMS4 = []

2 Dataset_image

2.3 Search function (image to image)

In [827]:
```
def search_ImageSOM(query):
    result = []
    query_features = []
    path = query
    q_f = processImage(path, model_image)
    query_features.append(q_f)
    activatedSOM = activate(monu_n_train_data, monu_picture_loaded_som_model, query_featu

    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.Reds, 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_BMU(monu_picture_loaded_som_model, query_features)
    closest_image_index = get_closest_image(g, h)
    closest_image_path = 'D:\\RC11\\skills\\final\\dataset\\image\\' + image_data_dict[g]
    print("Closest image:", image_data_dict[g][h][closest_image_index]['image'])

    closest_image = Image.open(closest_image_path)

    plt.figure(figsize=(4, 4))
    plt.imshow(closest_image)
    plt.axis('off')
    plt.show()

    return closest_image, closest_image_path
```

In [145]:
```
closest_image, closest_image_path = search_ImageSOM('palace\\1.jpg')
print("Closest image path:", closest_image_path)
```



[Closest image path: D:\RC11\skills\final\dataset\image\2615946.]

2 Dataset_image

2.4 Image_fuyu files

In [296]:
```
def read_text_files(folder_path):
    files_list = []
    for index, filename in enumerate(os.listdir(folder_path)):
        if filename.endswith('.txt'):
            file_path = os.path.join(folder_path, filename)
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.read()
            file_info = {
                'nr': index,
                'filename': filename,
                'content': content
            }
            files_list.append(file_info)

    return files_list

folder_path = 'dataset\\texts'
texts = read_text_files(folder_path)
```

In [297]:
```
texts
```

Out[297]:
```
[{'nr': 0,
  'filename': '101632.jpg.txt',
  'content': 'The image depicts a large stone monument with an archway at night. The
archway is adorned with intricate carvings and pillars, giving it a classical appeara
nce. The monument is illuminated by street lights, creating a nighttime ambiance.\n\n
The street is bustling with traffic, as several cars are visible driving or parked al
ong the'},
 {'nr': 1,
  'filename': '101633.jpg.txt',
```

dataset\\texts

2.5 Image_fuyu SOM

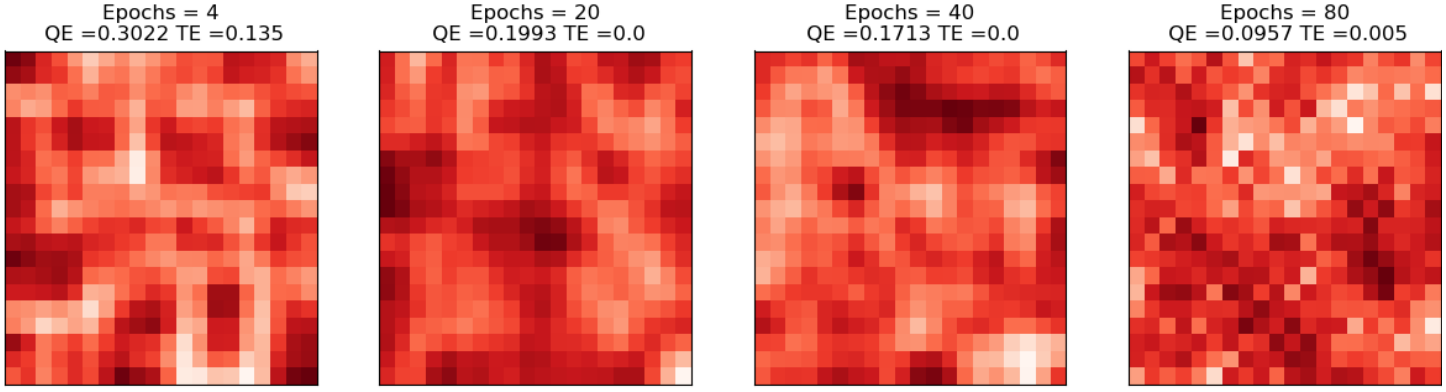
·Doc2Vec

[201]:
```
DOC2VEC_DOCUMENTS = [TaggedDocument(doc, [i]) for i, doc in enumerate(PREPROCESSED4)]

DOC2VEC_MODEL2 = Doc2Vec(vector_size=250, min_count=0, alpha=0.025, min_alpha=0.025, epocl
DOC2VEC_MODEL2.build_vocab(DOC2VEC_DOCUMENTS)
DOC2VEC_MODEL2.train(DOC2VEC_DOCUMENTS, total_examples=DOC2VEC_MODEL.corpus_count, epochs=
```

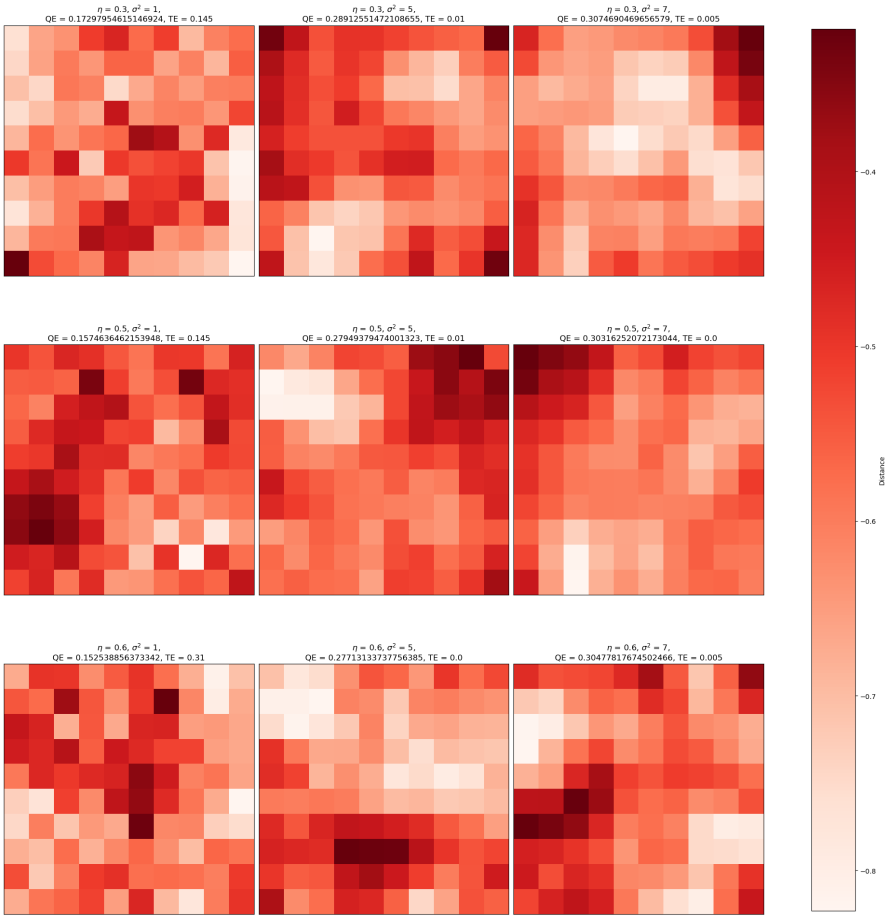
[202]:
```
DOC2VEC_MODEL2.save("image_fuyu_Doc2Vec.bin")
```

[203]:
```
DOC2VEC_MODEL2 = Doc2Vec.load("image_fuyu_Doc2Vec.bin")
```

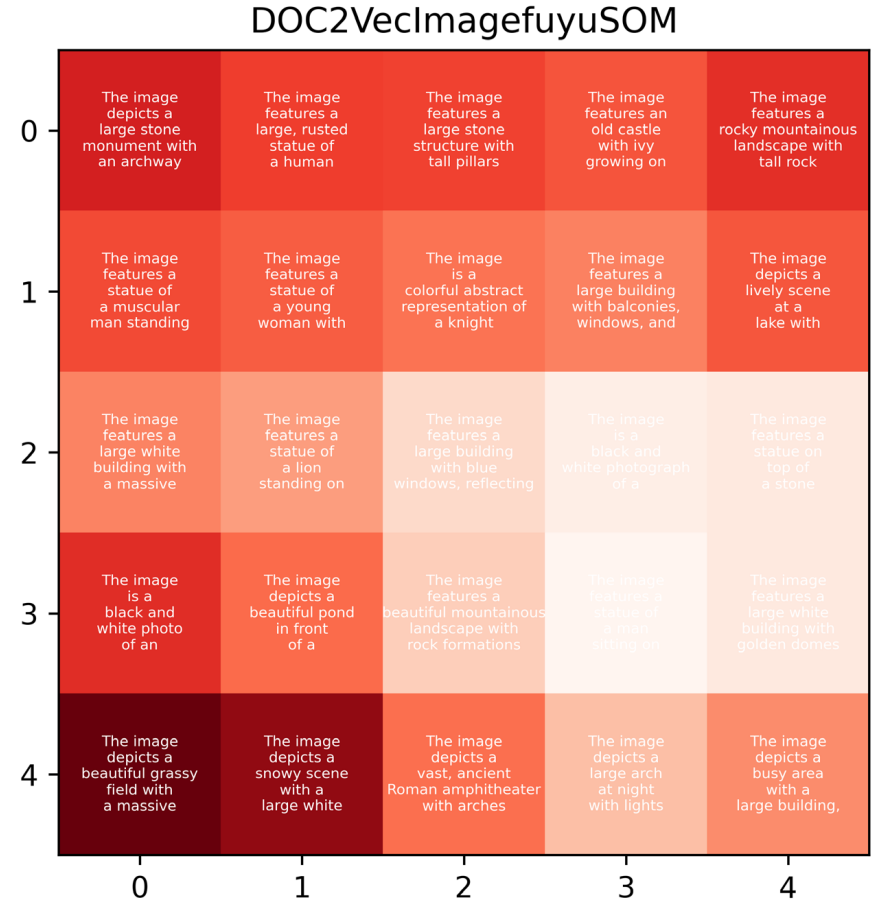
image_fuyu_Doc2Vec.bin



2 Dataset_image



SOMS3 = []



Image_fuyu _neuron_word_mapping
Doc2Vec

Visualize the output of the SOM model to observe the topology among neurons and display the correlation between neurons and text data.

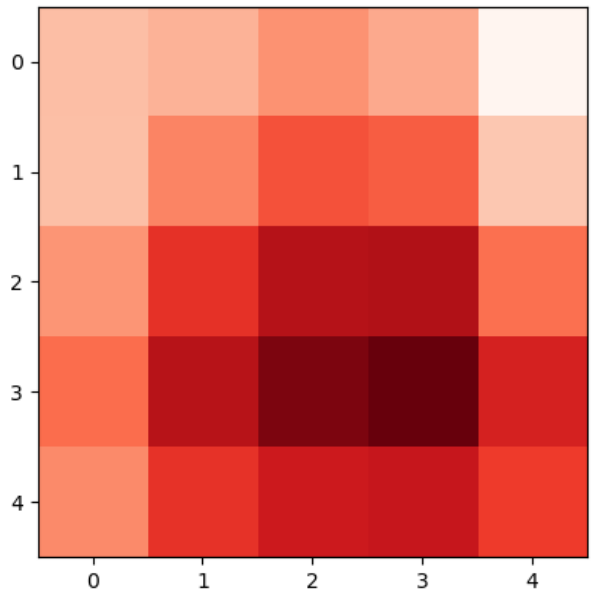
Not a lot of text information is, so a 5x5 SOM was generated to match the text.

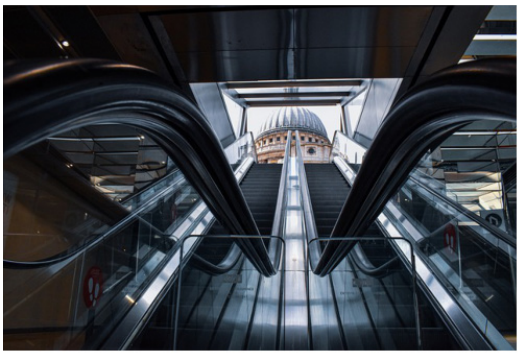
2 Dataset_image

2.3 Search function (text to image)

In [524]:
```
def search_TextSom3(SOM, model, data_dict3, query=''):
    result = []
    query = [query]
    preprocessed_query = preprocess2(query)
    query_vector = DOC2VEC_MODEL2.infer_vector(preprocessed_query[0])
    print(preprocessed_query)

    # activatedSOM
    activatedSOM = activate1(train_data2,SOM3,query_vector)
    print("Activated SOM:", activatedSOM)

    fig = plt.figure()
    plt.imshow(activate1(train_data2,SOM3, query_vector), cmap=cmap)
    plt.show()

    g, h = find_BMU1(SOM3, query_vector)
    print((g, h))

    similarities = {}
    for data in data_dict3[g][h]:
        paragraph = data['text']['content']
        vector_array = np.array([data['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarities[paragraph] = cosine_similarity(vector_array, [SOM3[g][h]])

    biggest = None
    max_similarity = max(similarities.values())
    for paragraph, sim in similarities.items():
        if sim == max_similarity:
            biggest = paragraph

    if biggest and all(isinstance(item, dict) and 'content' in item for item in result):
        if biggest not in [item['content'] for item in result]:
            for data in data_dict3[g][h]:
                if data['text']['content'] == biggest:
                    result.append({'nr': data['text']['nr'], 'filename': data['text']['fil
                    break

    return result
```

In [145]:
```
result = search_TextSom3(SOM3, model, data_dict3, query='Anarchy reigns in the absence of governmental control.')
```



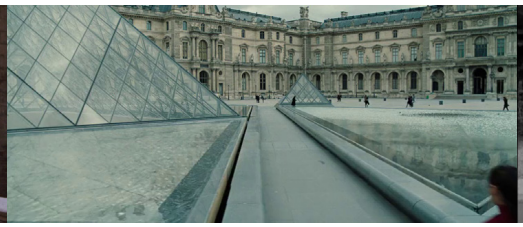


['**filename':** '6564157.jpg',
'content': 'The image depicts an escalator with escalators on both sides. The escalator spans from the top to the bottom of the picture, and the stairs are visible on both sides. The escalator appears to be moving downward, with the upper section in the middle of the picture and the lower section towards the bottom.']

3 Dataset_film

3.1 Frame files





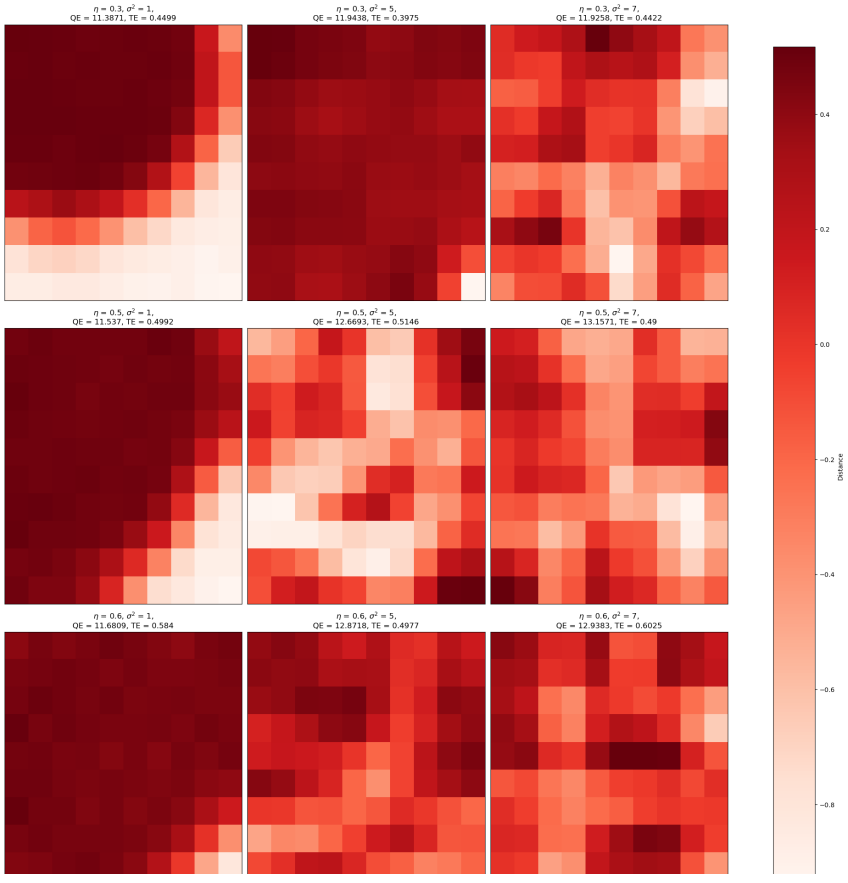





```
[513]: with open('features_frame.pkl', 'wb') as f:
           pickle.dump(features, f)

[514]: with open('features_frame.pkl', 'rb') as f:
           features = pickle.load(f)
```

features_frame.pkl

3.2 Frame SOM



I chose a SOM with index 6 because it has the lowest average of both TE and QE.

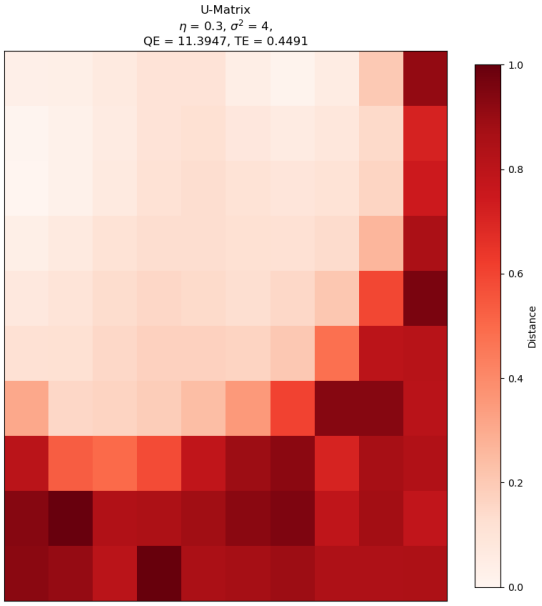
SOMS4 = []

3 Dataset_film

3.3 Search function (image to video)

In [591]:
```
def search_ImageSom2(query):
    result = []
    query_features = []
    path = query
    q_f = processImage(path, model4)
    query_features.append(q_f)
    activatedSOM = activate(n_train_data, loaded_video_som_model, query_features)

    # Visualizing the SOM
    fig = plt.figure()
    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.Reds, 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()

    unit = find_BMU(loaded_video_som_model, query_features)
    g, h = unit  # Extracting the BMU indices
    closest_image_index = get_closest_image1(g, h)  # Pass BMU indices to get closest ima

    # Extracting video name and modifying it
    video_path = video_picture_data_dict[g][h][closest_image_index]['image']
    video_name = os.path.basename(video_path)
    video_name_without_extension = os.path.splitext(video_name)[0]  # Remove extension
    video_name_parts = video_name_without_extension.split('_')
    if len(video_name_parts) > 1:  # Ensure there's at least one _
        modified_video_name = '_'.join(video_name_parts[:-1])  # Remove the last _ and nu
    else:
        modified_video_name = video_name_parts[0]
    film_name = "dataset\\film\\" + modified_video_name + ".mp4"

    # Displaying the video in the notebook
    display(Video(filename=film_name))

    result.append(film_name)
    return result
```

In [145]:
```
search_ImageSom2('dataset\\image\\780006.jpg')
```



['dataset\\film\\courtyard_58.mp4']

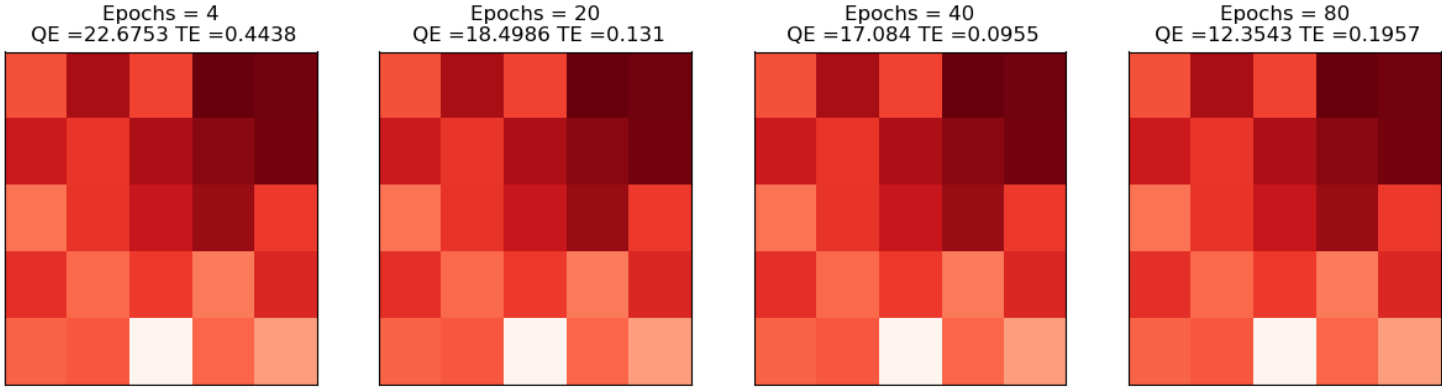
3 Dataset_film

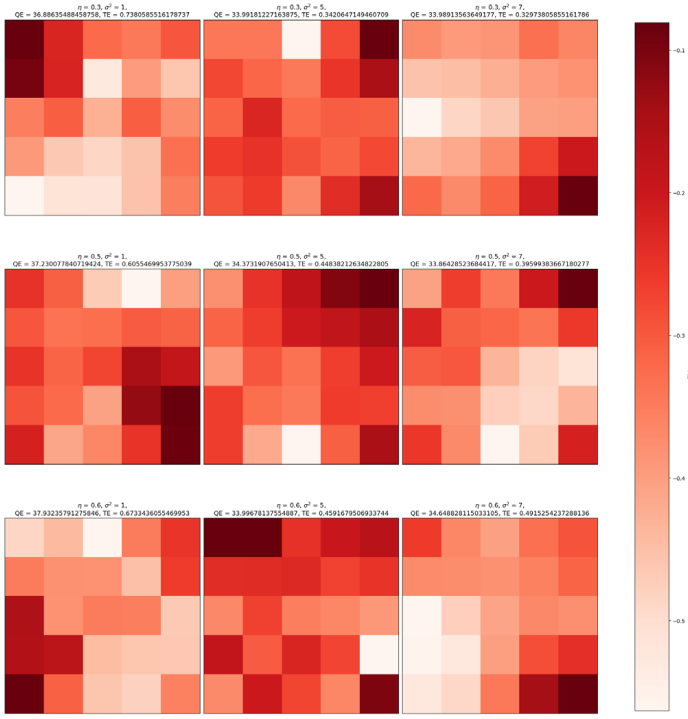
3.4 Frame_fuyu files

[{'nr': 0,
  'filename': 'amphitheater_100_1.jpg.txt',
  'content': 'How many people are there in the image?\n\nThere are four people in the image.\n'},
 {'nr': 1,
  'filename': 'amphitheater_10_1.jpg.txt',
  'content': "The image depicts a man tying a tie around a woman's neck. The woman is smiling and appears to be enjoying the interaction. There are several other people visible in the background, some standing closer to the man and others further away. A."},
  ……

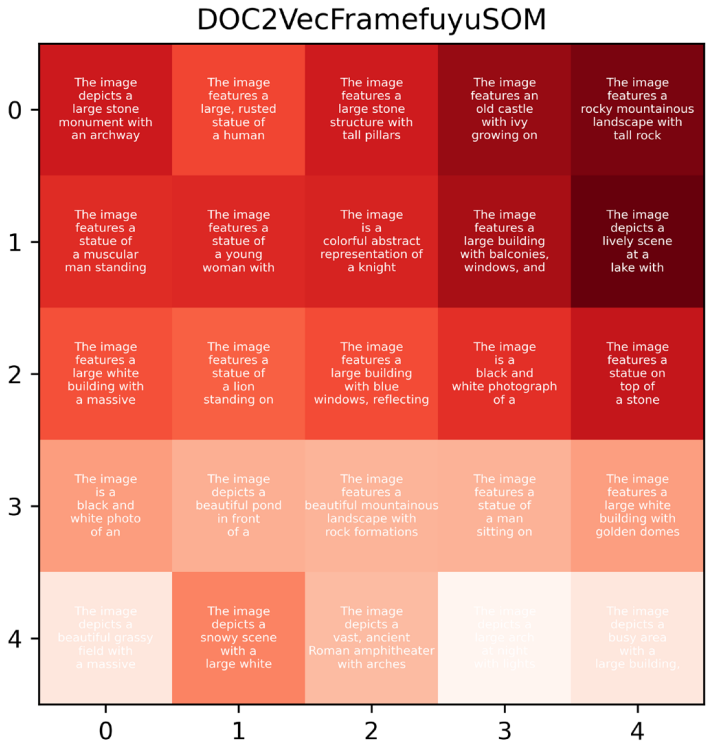
 {'nr': 648,
  'filename': 'playground_99_1.jpg.txt',
  'content': 'A young woman is standing outdoors, wearing a red shirt and holding a piece of metal up to her face. She appears to be posing for a picture.\n'}]

3.5 Frame_fuyu SOM





SOMS5 = []



Frame_fuyu _neuron_word_mapping
Doc2Vec

4 Search engine

A. Input elements from type_text

A-1. Input any sentence, find the matching content in databet1 of type text by TF-IDF and Doc2Vec, find the matching image in dataset2 of type image by Doc2Vec, and then input the obtained image into dataset3 of type video, find the matching film.

In [574]:
```
query = 'I am free.'

#text-text
output_som2 = search_TextSom2(SOM2, VECTORIZER_TFIDF, data_dict2, query)
output_som1 = search_TextSom1(SOM1, model, data_dict, query)

print("Output of search_TextSom2:")
print(output_som2)
print("------------------------")

print("Output of search_TextSom1:")
print(output_som1)
print("------------------------")

#text-image-film
result = search_TextSom3(SOM3, model, data_dict3, query)
result

image_folder = 'dataset\\image'

if result:
    data = result[-1]
    filename = data['filename'].replace('.jpg.txt', '.jpg')
    image_path = os.path.join(image_folder, filename)

    print(f"Trying to open: {image_path}")

    try:
        image = Image.open(image_path)
        plt.imshow(image)
        plt.axis('off')
        plt.show()
    except FileNotFoundError:
        print(f"File not found: {image_path}")
else:
    print("No results found.")

search_ImageSom2(image_path)
```

Steps

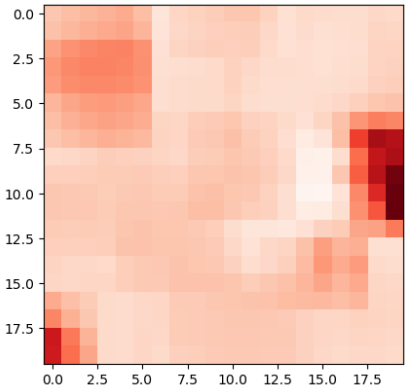
Step1: searching for matches in dataset_1_epub through TF-IDF; (search_TextSom2)
Step2: searching for matches in dataset_1_epub through Doc2Vec; (search_TextSom1)
Step3: searching for matches in dataset_2_image through Doc2Vec; (search_TextSom3)
Steo4: searching for matches in dataset_3_film through the results of 2. (search_ImageSom2)

4 Search engine

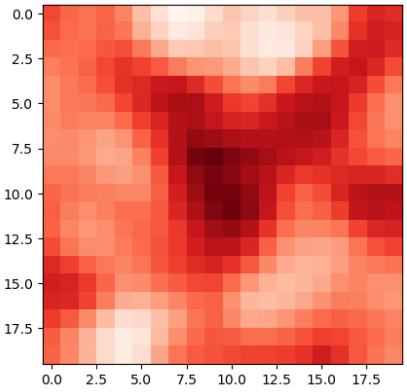
A. Input elements from type_text

A-1. **Examples**

Input: query = 'I am free.'

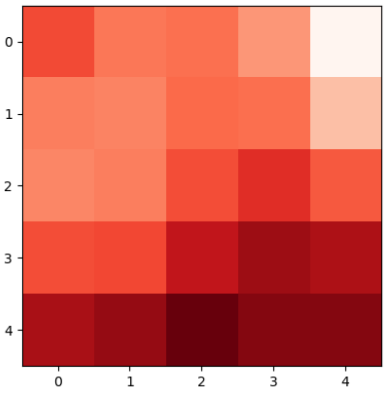


(11, 19)
[['free']]
Output of search_TextSom2:
[{'paragraph': ' Rather, this is a dynamic political process which needs direct intervention by the sovereign, the people. ……and larger congresses.', 'nr': 153, 'bookID': 'abdullah-ocalan-war-and-peace-in-kurdistan'}]

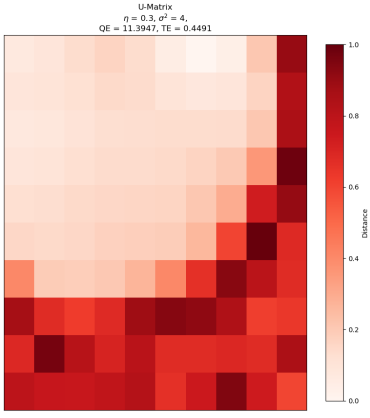


(8, 9)

Output of search_TextSom1:
[{'paragraph': ' The shootout was brief and they left the town quickly. The evening newspapers in Barcelona were……by the time they got back to the Catalan capital.', 'nr': 384, 'bookID': 'abel-paz-durruti-in-the-spanish-revolution'}]



(4, 2)
['dataset\image\1835210.jpg']
Output of search_TextSom3:
Activated SOM:
[[0.09332789 0.08999289 0.09048479 0.0873847 0.07788508]
……
 [0.10052485 0.10175006 0.10426533 0.10268149 0.102712]]



Output of search_ImageSom2:
['dataset\\film\\pavilion_6.mp4']

4 Search engine

B. Input elements from type_image

B-1. Input any image, find the matching content in databet1 of type text by TF-IDF and Doc2Vec, find the matching image in dataset2 of type image, and then input the obtained image into dataset3 of type video, find the matching film.

In [837]:
```
image_path = "palace\\1.jpg"

#image-text
with open('descriptions.json', 'r') as f:
    data = json.load(f)

description = None
for item in data:
    if item['path'] == image_path:
        description = item['description']
        break

if description:
    img = Image.open(image_path)
    display(img)
    print("Image Path:", image_path)
    print("Description:", description)
else:
    print("Image not found in the descriptions.")

output_som2 = search_TextSom2(SOM2, VECTORIZER_TFIDF, data_dict2, description)
output_som1 = search_TextSom1(SOM1, model, data_dict, description)

print("Output of search_TextSom2:")
print(output_som2)
print("------------------------")

print("Output of search_TextSom1:")
print(output_som1)
print("------------------------")


#image-image
closest_image, closest_image_path = search_ImageSOM(image_path)
print("Closest image path:", closest_image_path)

#image-video
search_ImageSom2(closest_image_path)
```

Steps

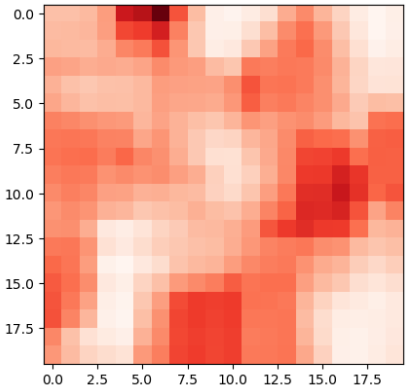
Step1: searching for matches in dataset_1_epub through TF-IDF; (search_TextSom2)
Step2: searching for matches in dataset_1_epub through Doc2Vec; (search_TextSom1)
Step3: searching for matches in dataset_2_image; (search_ImageSOM)
Steo4: searching for matches in dataset_3_film through the results of 2. (search_ImageSom2)

4 Search engine

B. Input elements from type_image

B-1. **Examples**

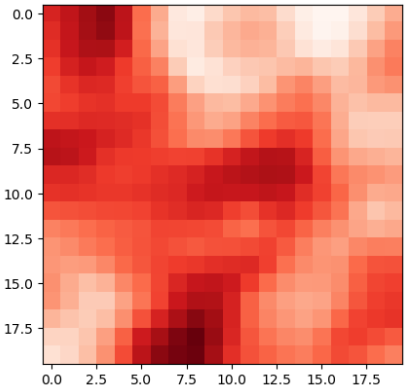
Input: image_path = "palace\\1.jpg"



(0, 6)

Output of search_TextSom2:

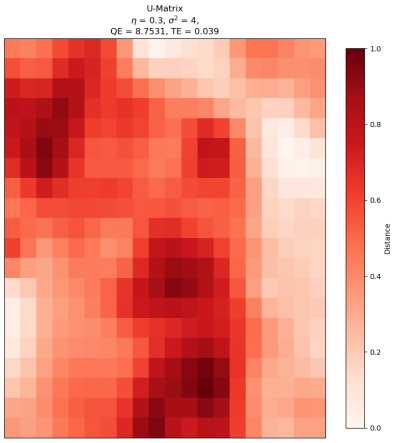
[{'paragraph': ' They would have pardoned anyone else for human weakness, but not Durruti. ……This was more convincing than his arguments.', 'nr': 3285, 'bookID': 'abel-paz-durruti-in-the-spanish-revolution'}]



(18, 8)

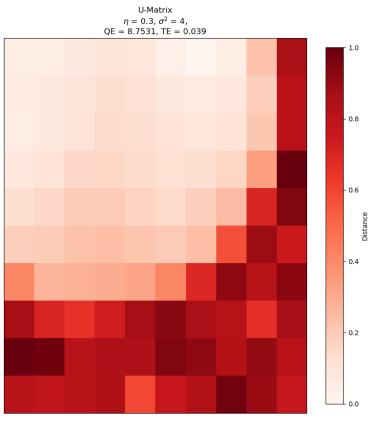
Output of search_TextSom1:

[{'paragraph': ' When he returned to the Column's headquarters, ……"The conflict will be difficult, very difficult, but we'll save Madrid if we fight well today.', 'nr': 4976, 'bookID': 'abel-paz-durruti-in-the-spanish-revolution'}]



Output of search_ImageSOM:

['Closest image: 2615946.jpg']



Output of search_ImageSom2:

['dataset\\film\\park_16.mp4']

4 Search engine

B. Input elements from type_image

B-2. Input any image, find the matching content in databet2 of image, find the matching image in dataset1 of text by TF-IDF and Doc2Vec, and then input the obtained text into dataset3 of type video, find the matching film.

In [956]:
```
def search_engine_B2(image_path):
    with open('paintings.json', 'r') as f:
        data = json.load(f)

    description = None
    for item in data:
        if item['path'] == image_path:
            description = item['description']
            break

    if description:
        img = Image.open(image_path)
        display(img)
        print("Image Path:", image_path)
        print("Description:", description)
    else:
        print("Image not found in the descriptions.")

    #image-image
    search_ImageSOM(image_path)

    #image-text
    output_som2 = search_TextSom2(SOM2, VECTORIZER_TFIDF, data_dict2, description)
    output_som1 = search_TextSom1(SOM1, model, data_dict, description)

    #text-video
    tfidf_text = output_som2[0]["paragraph"]

    result = search_TextSom3(SOM3, model, data_dict3, tfidf_text)

    image_folder = 'dataset\\image'

    if result:
        data = result[-1]
        filename = data['filename'].replace('.jpg.txt', '.jpg')
        image_path = os.path.join(image_folder, filename)

        print(f"Trying to open: {image_path}")

        try:
            image = Image.open(image_path)
            plt.imshow(image)
            plt.axis('off')
            plt.show()
        except FileNotFoundError:
            print(f"File not found: {image_path}")
    else:
        print("No results found.")

    search_ImageSom2(image_path)
```

Steps

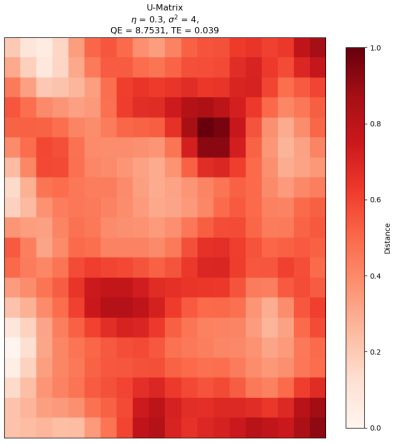
Step1: searching for matches in dataset_2_image; (search_ImageSOM)
Step2: searching for matches in dataset_1_epub through TF-IDF; (search_TextSom2)
Step3: searching for matches in dataset_1_epub through Doc2Vec; (search_TextSom1)
Steo4: searching for matches in dataset_3_film through the results of 2. (search_ImageSom2)

4 Search engine

B. Input elements from type_image

B-2. **Examples**

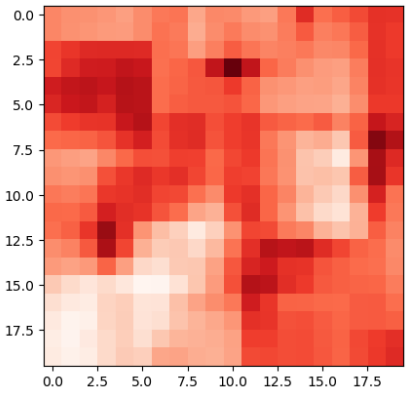
Input: image_path = "painting\\1.jpg"



→

Output of search_ImageSOM:

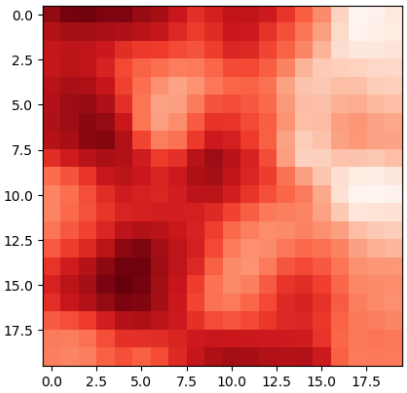
['Closest image: 7739208.jpg']



(3, 10)

Output of search_TextSom2:

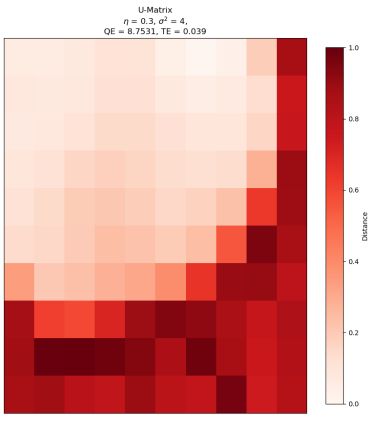
[{'paragraph': ' The many temples for goddesses and the mythological texts from this period indicate that between 4,000 and 2,000 bc the influence of the woman- mother culture on the Sumerians, who formed the centre of civilisation, was on par with that of the man.', 'nr': 72, 'bookID': 'abdullah-ocalan-liberating-life'}]



(15, 4)

Output of search_TextSom1:

[{'paragraph': ' He also said that Russian workers had organized a national support campaign through their unions, whose first remittance of money had been sent to Prime Minister Giral.[592] The response did not satisfy Durruti.', 'nr': 4215, 'bookID': 'abel-paz-durruti-in-the-spanish-revolution'}]



→

Output of search_ImageSom2:

['dataset\\film\\courtyard_8.mp4']

SKILLS CLASS RC11 - 23/24

HOUDINI'S ASSIGNMENT

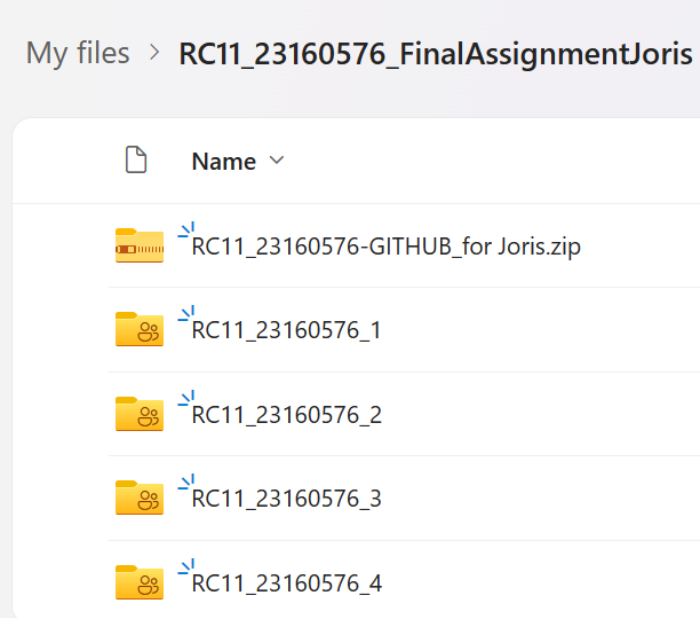
FINAL ASSIGNMENT
EXPLANATION AND OUTCOMES

Rc11
Tutor: Joris Putteneers
SN: 23160576

OneDrive:

This OneDrive link Contains houdini files for four assignments and a json file for the fourth assignment. As well as all input and output files.

※If the hyperlink doesn't open, please try this URL: https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlul_ucl_ac_uk/EvSIJOTGT_NHjuBCDDbrv1MBMiU_OWah-818eN96JjkAOw



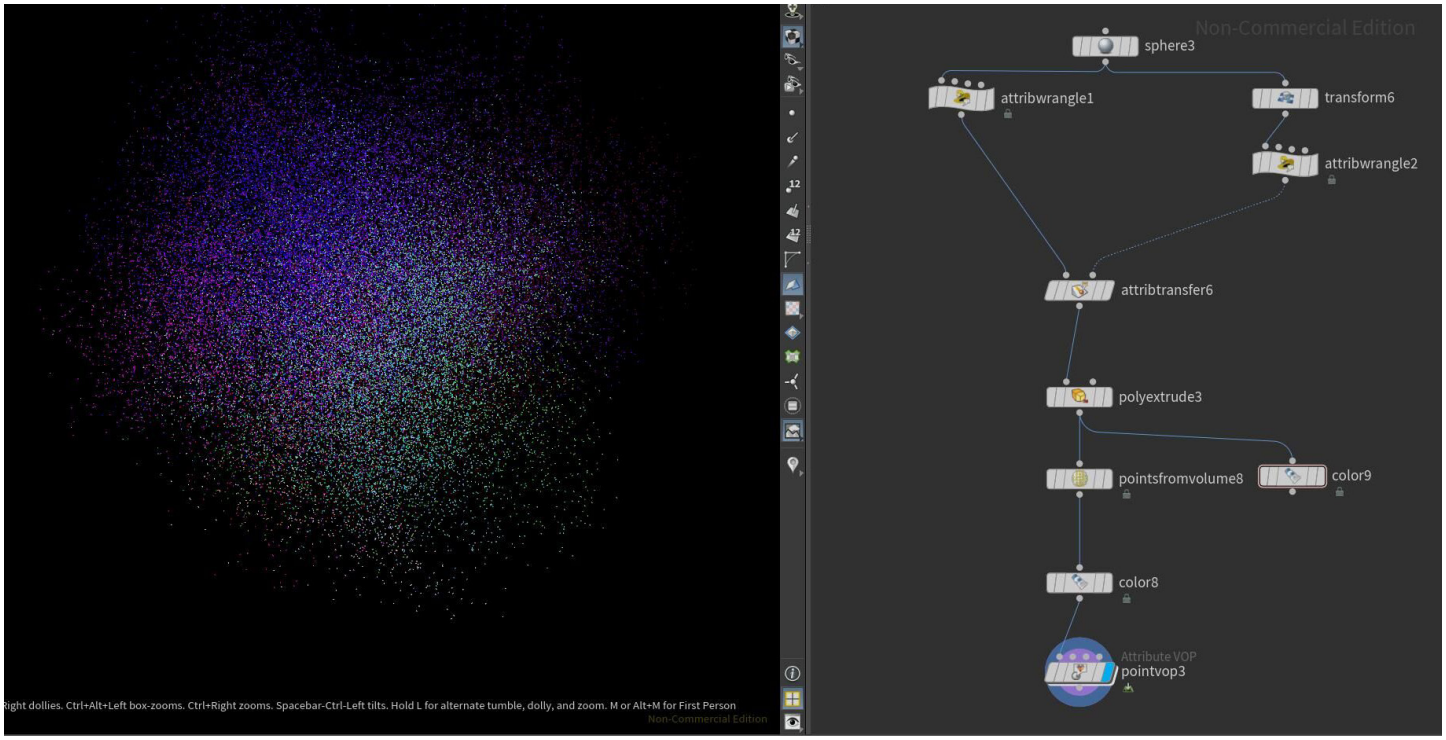
[folder structure]

GITHUB

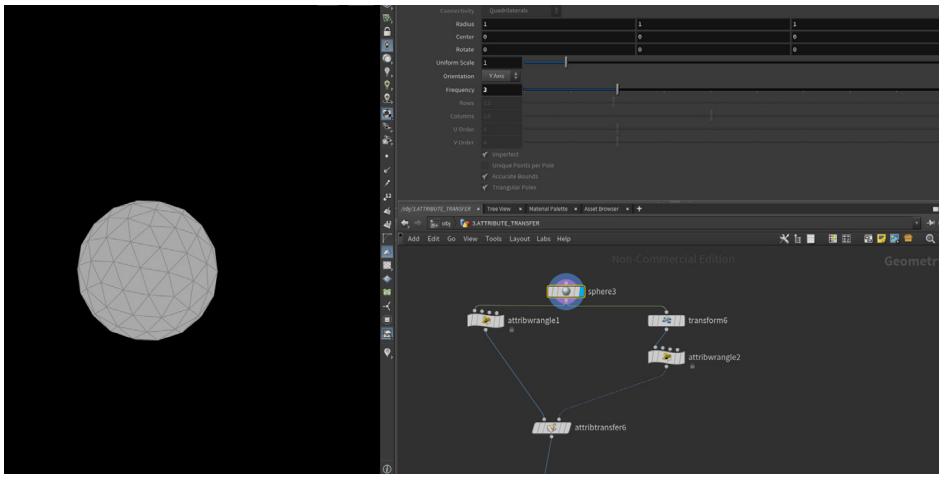
※If the hyperlink doesn't open, please try this URL: https://github.com/UD-Skills-2023-24/RC11_23160576/tree/main

Content

1 Houdini Fundamentals

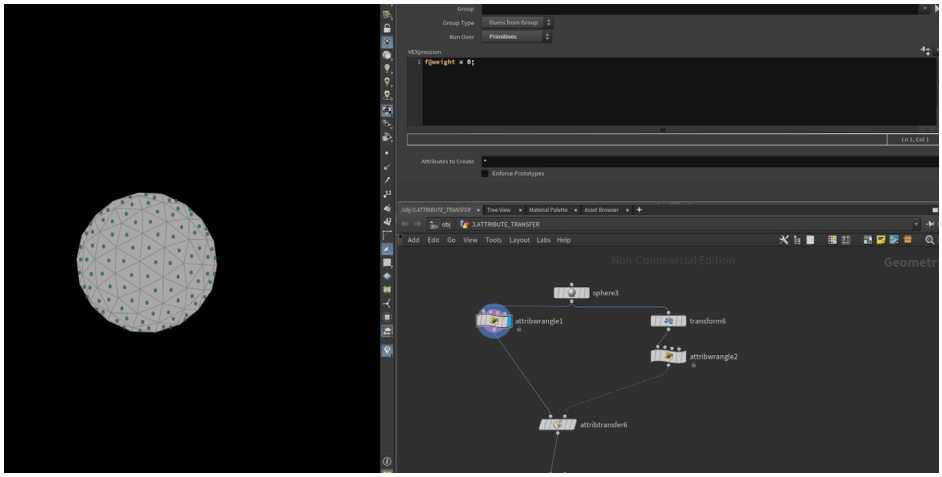


Houdini Fundamentals: Setup 1 Overview



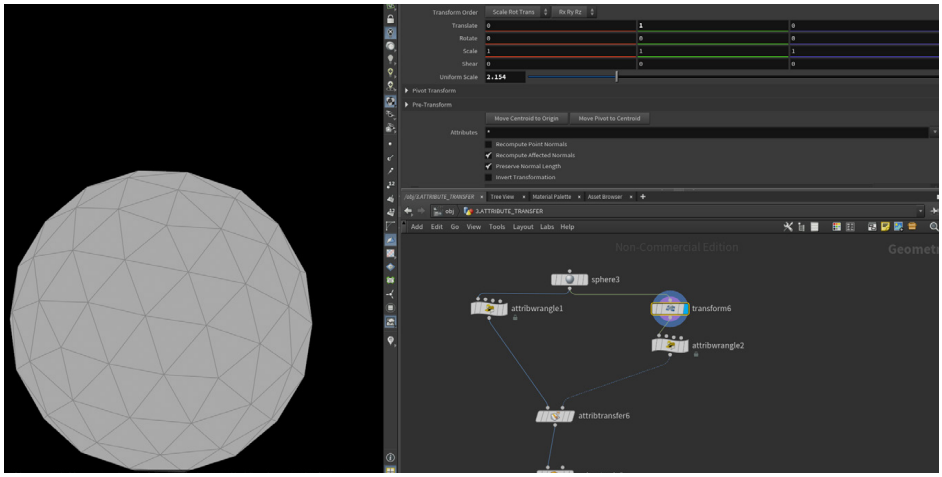
1: Sphere SOP

The "sphere" is a primitive geometric type: polygon. It's one of the fundamental geometric data types that contains points, primitives of type polygon, and vertices



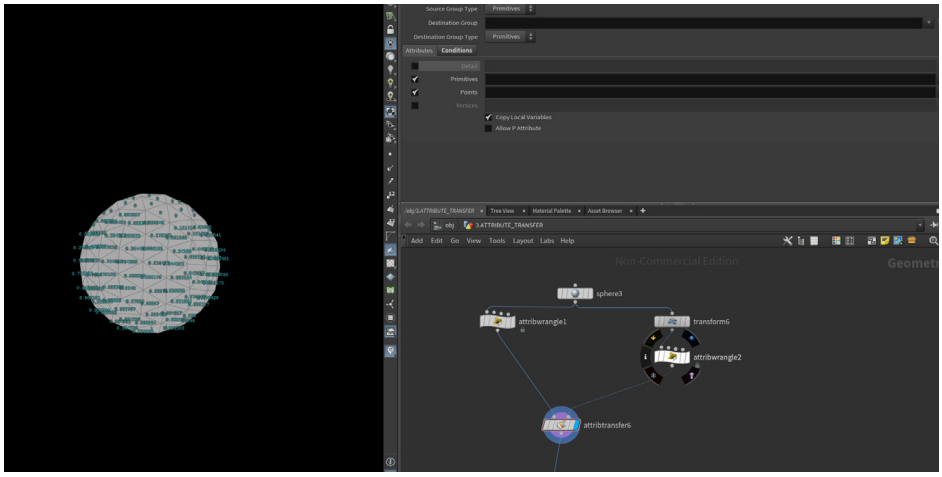
2: Attribute Wrangle SOP

In the "Attribute Wrangle" SOP, we create a "weight" attribute. This attribute, declared as "f@", is of type float and is set to run over primitives. We initialize its value to 0, as we will later use it to control the extrusion value in a PolyExtrude node setup, allowing for interpolation between 0 and 1.



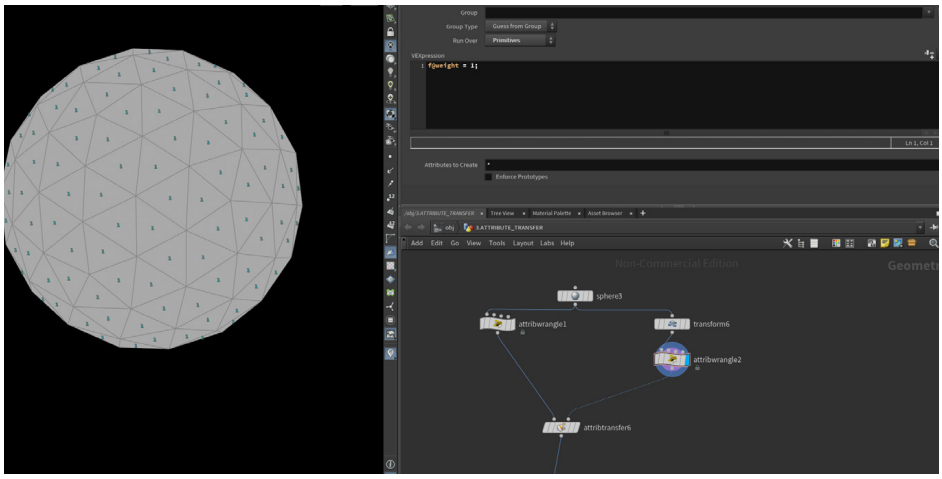
3: Transform SOP

Then, we add a "Transform" SOP. This serves as the attractor and is simply a duplicate of the original sphere. The distance between this duplicate (attractor) and the original sphere determines the final extrusion value.



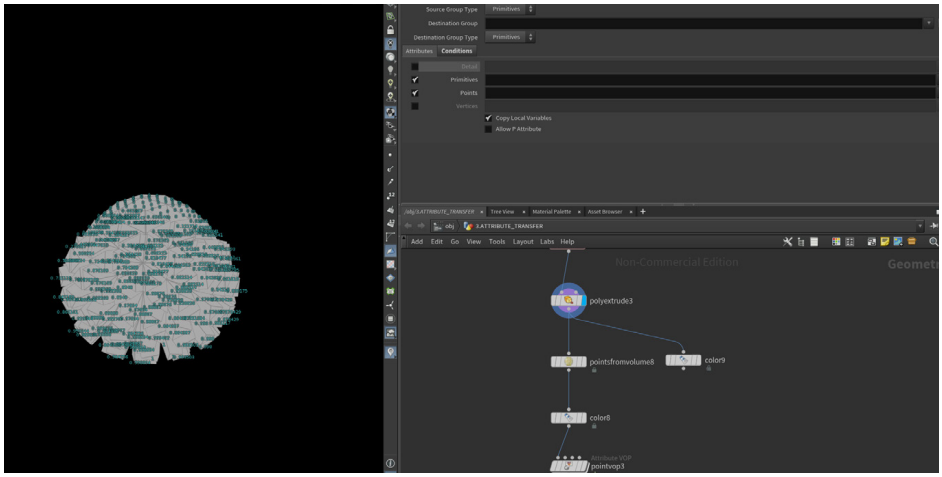
4: Attribute Transfer SOP

This SOP uses the "weight" primitive attribute to determine the distance between the original sphere and the attractor. By setting the original 'weight' attribute as a float, we enable interpolation between 0 and 1. If the weight attributes were integers, interpolation wouldn't be possible, resulting in binary output values (0 or 1).



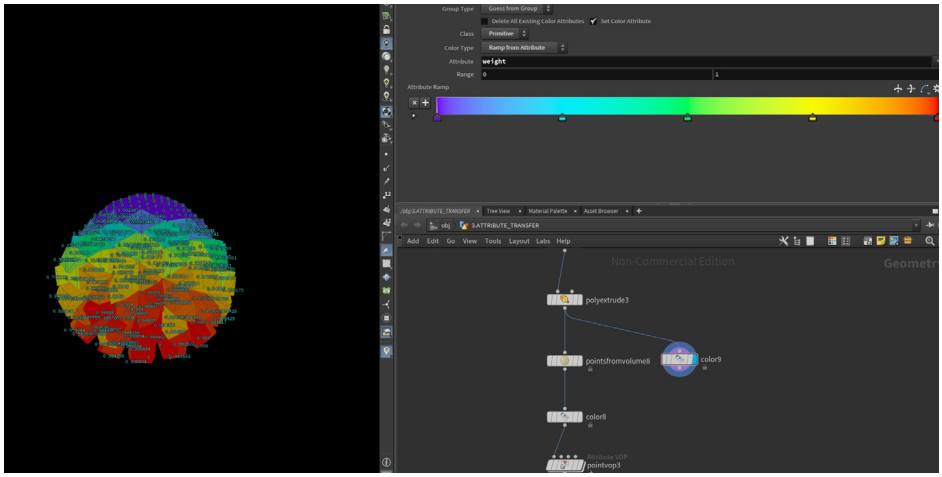
5: Attribute Wrangle SOP

Similar to the previous operation but with opposite values.
We choose to interpolate between 0 and 1 because it's more convenient to work with normalized values than arbitrary ones.



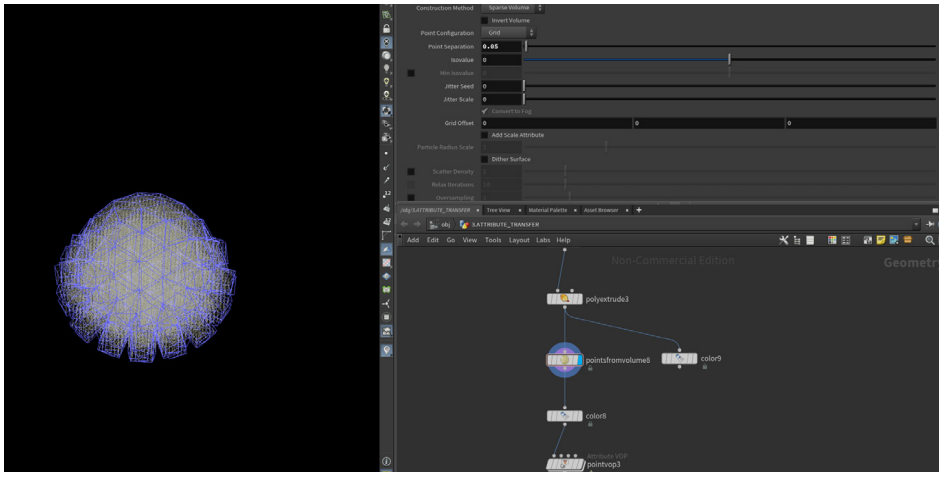
6: Poly Extrude SOP

We use the "weight" value as a multiplier for the extrusion value in the "distance scale" setting. When the distance value is set to 0, the extrusion becomes 0 (0 multiplied by the weight). When the distance value is set to 1, we obtain the full weight value as extrusion (1 multiplied by the weight).



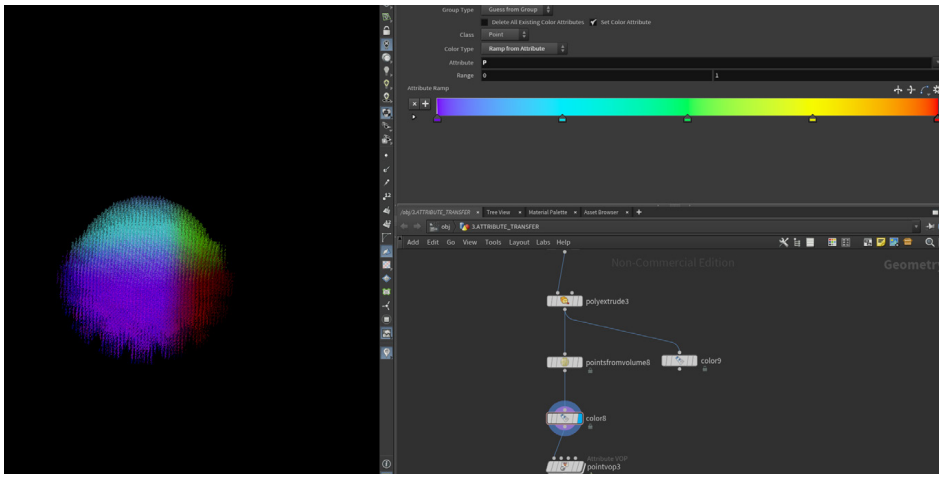
7: color SOP

We create a visualization of the "weight" primitive attribute using the "viridis" color scheme. Normalizing our values between 0 and 1 makes it more consistent if we ever modify distances or geometry inputs.



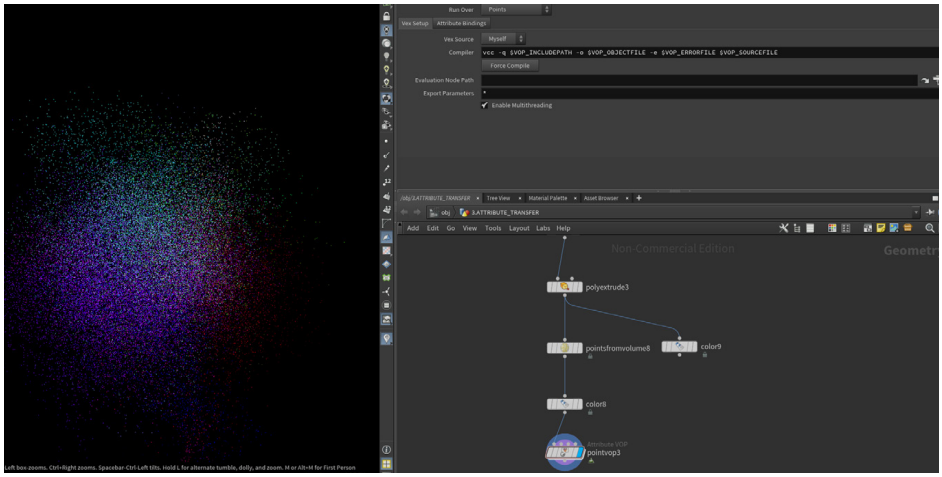
8: Points from Volume SOP

This SOP converts volume data into point cloud data. It extracts points from the given volume data and determines which points should be extracted based on certain parameters. This allows for the creation of particle systems and the generation of complex shapes from volume data.



9: color SOP

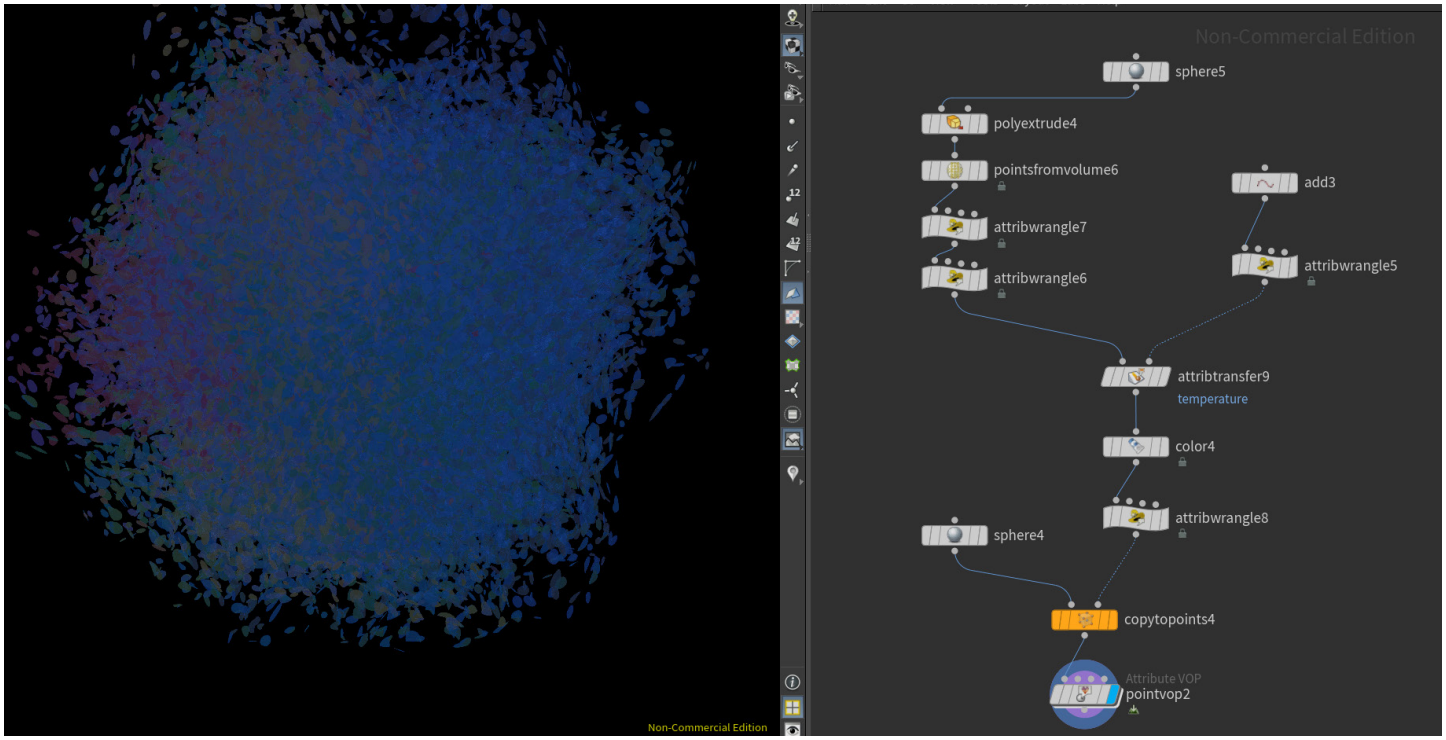
Similar to the previous operation, we create a visualization of the "weight" primitive attribute using the "viridis" color scheme. Normalizing our values between 0 and 1 makes it more consistent if we ever modify distances or geometry inputs.



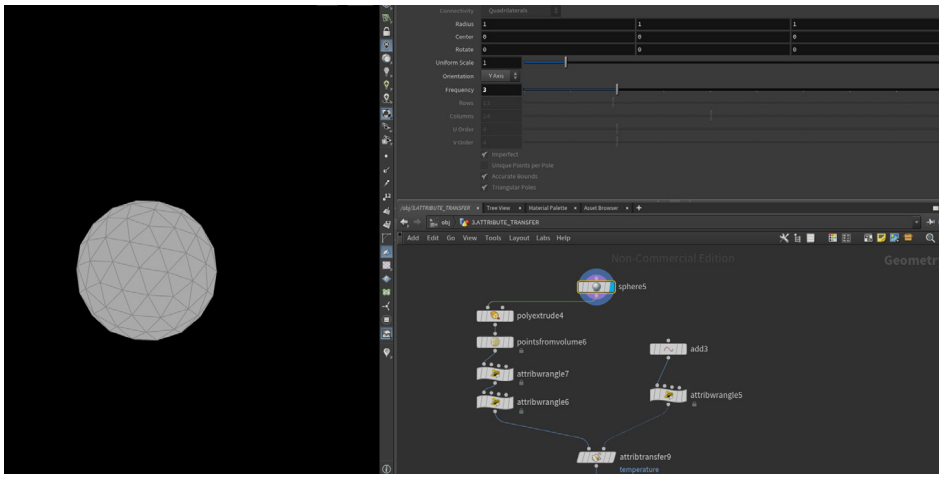
10: Point VOP SOP

This SOP is used to execute custom node programs at the point level. It calculates, transforms, or generates attributes for points. It is utilized for creating complex effects, processing point clouds, and controlling particle systems.

1 Houdini Fundamentals

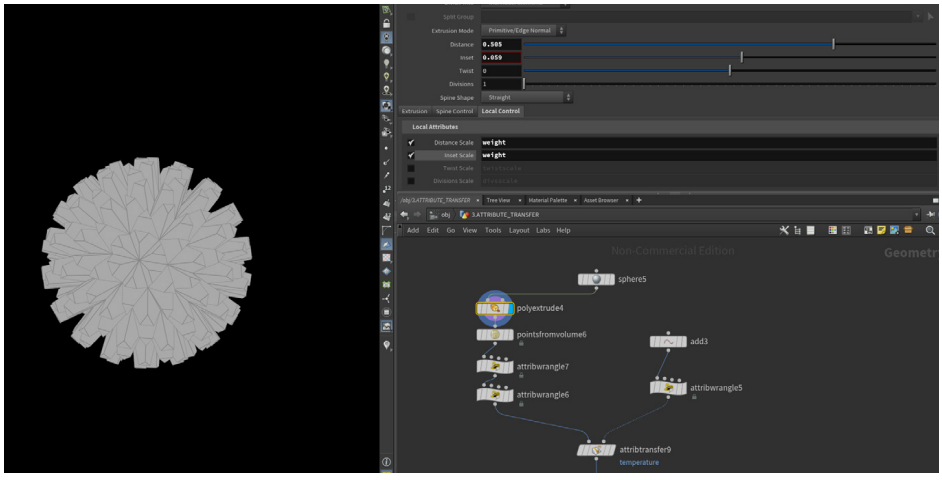


Houdini Fundamentals: Setup 2 Overview



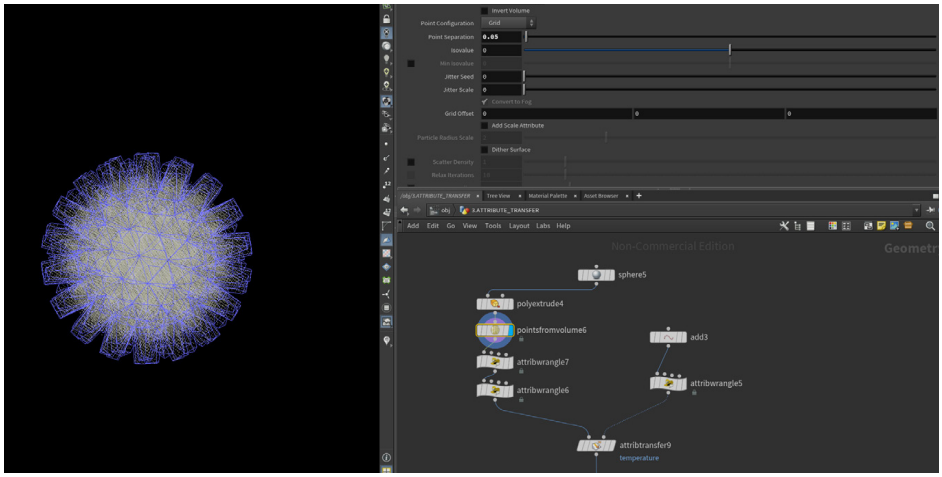
1: Sphere SOP

The "sphere" is a primitive geometric type: polygon. It's one of the fundamental geometric data types that contains points, primitives of type polygon, and vertices



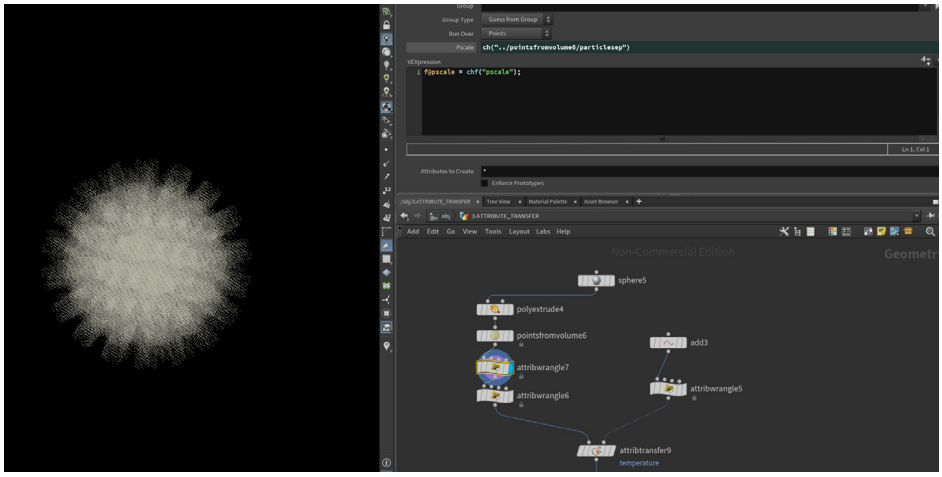
2: Poly Extrude SOP

We use the "weight" value as a multiplier for the extrusion value in the "distance scale" setting. When the distance value is set to 0, the extrusion becomes 0 (0 multiplied by the weight). When the distance value is set to 1, we obtain the full weight value as extrusion (1 multiplied by the weight).



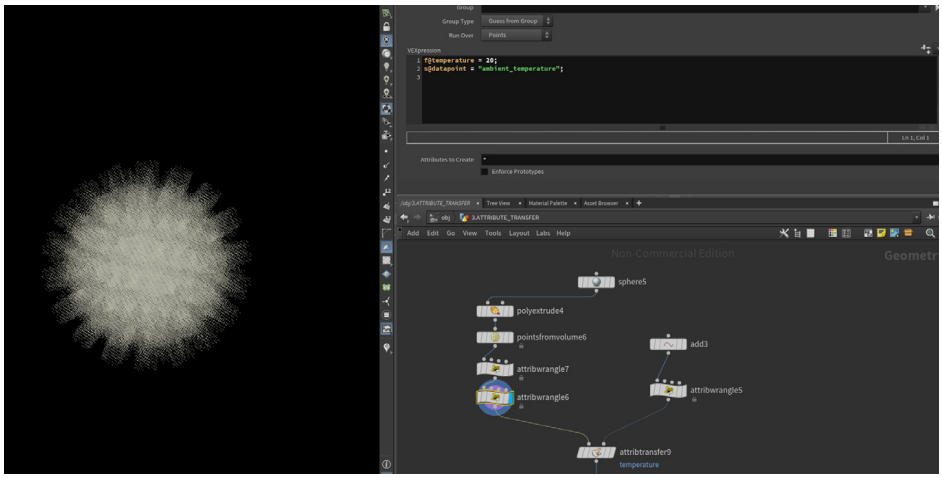
3: Points from Volume SOP

This SOP converts volume data into point cloud data. It extracts points from the given volume data and determines which points should be extracted based on certain parameters. This allows for the creation of particle systems and the generation of complex shapes from volume data.



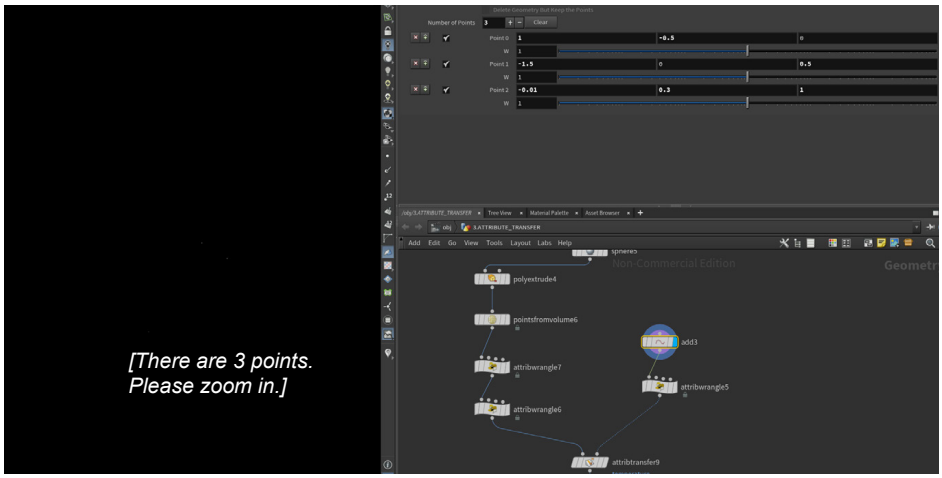
4: Attribute Wrangle SOP

In the "Attribute Wrangle" SOP, we take a channel named "pscale" as input, then assign its value to the "pscale" attribute for each point being processed. Here, the chf function is used to fetch the value of the "pscale" channel and convert it to a float before assigning it to the "pscale" attribute of the currently processed point.



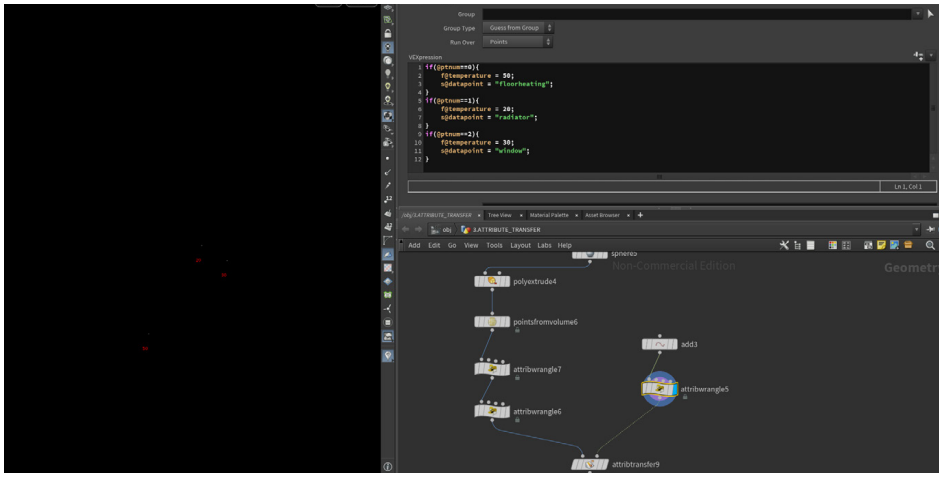
5: Attribute Wrangle SOP

In this "Attribute Wrangle" SOP, the code sets the "temperature" attribute to a constant value of 20 for each point. Additionally, it assigns the string value 'ambient_temperature' to the "datapoint" attribute.



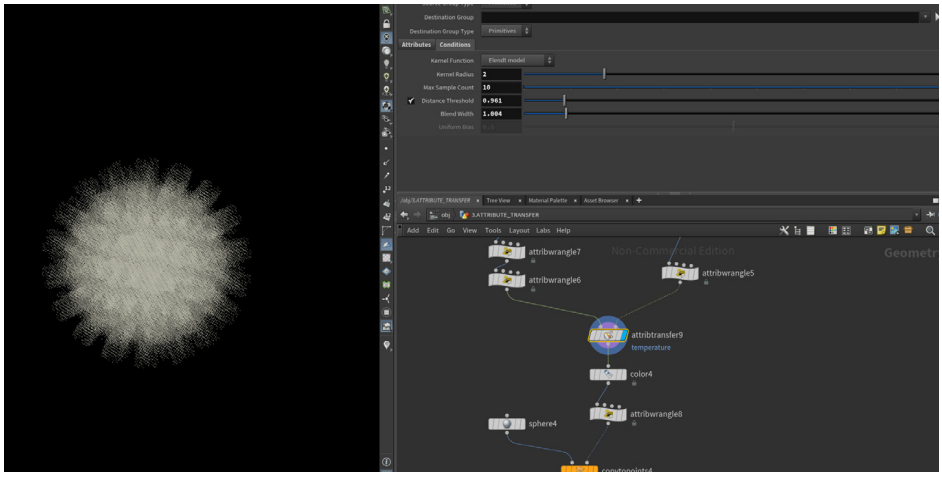
6: Add SOP

"Add" typically refers to a node or operation used to add input values together and output the result. Here, the term "add" is used to add three points.



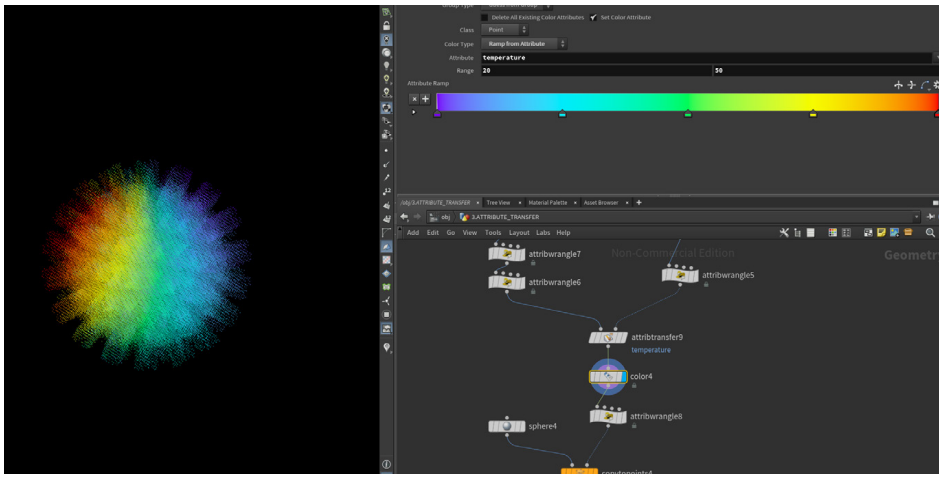
7: Attribute Wrangle SOP

This SOP assigns attribute values to the three added points. 'f@ temperature = num' sets a floating-point attribute named "temperature" with numerical values for each point's temperature. 's@datapoint = ‘floorheating’ ‘radiator' 'window" assigns a string attribute named "datapoint" with values 'floorheating', 'radiator', and 'window' respectively to each of the points.



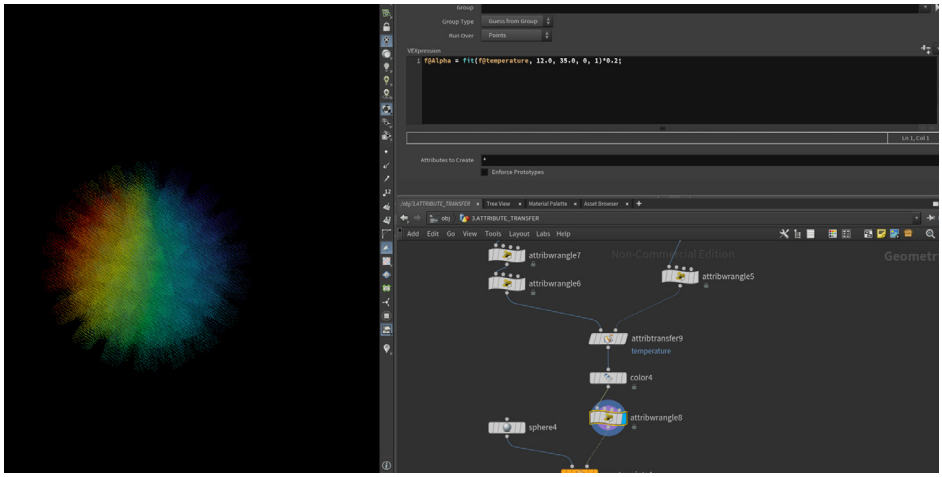
8: Attribute Transfer SOP

This SOP uses the "temperature" primitive attribute to determine the distance between the original sphere and the attractor. By setting the original 'weight' attribute as a float, we enable interpolation between 0 and 1. If the weight attributes were integers, interpolation wouldn't be possible, resulting in binary output values (0 or 1).



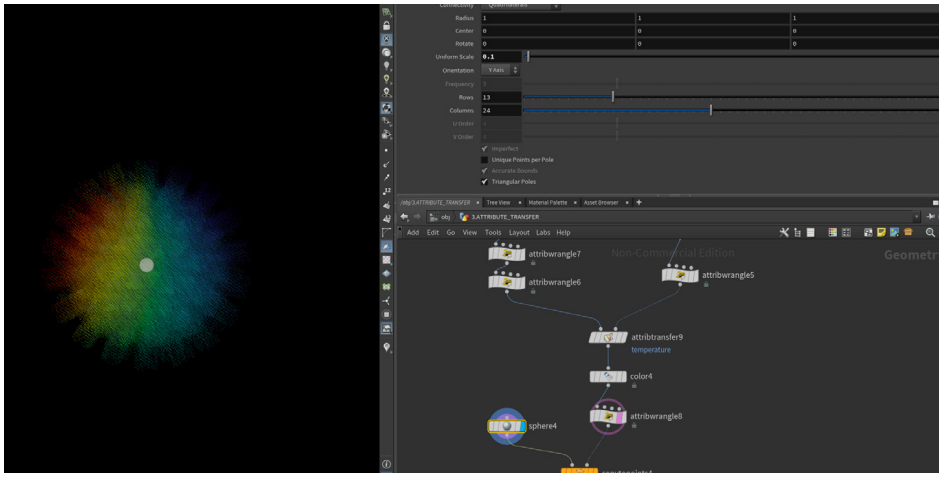
9: color SOP

Similar to the previous operation, we create a visualization of the "weight" primitive attribute using the "viridis" color scheme. Normalizing our values between 0 and 1 makes it more consistent if we ever modify distances or geometry inputs.



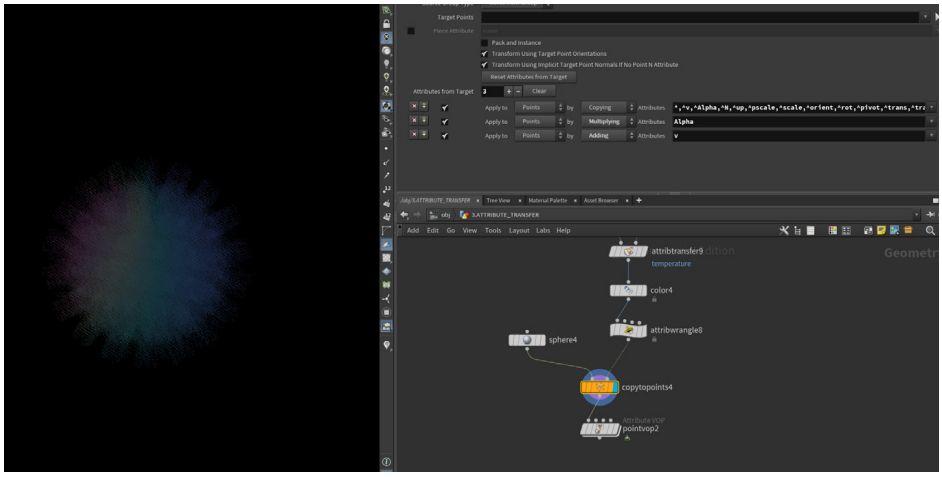
10: Attribute Wrangle SOP

This SOP computes an attribute named "alpha". The fit() function is used here to map the values of the temperature attribute f@ temperature from the range [12, 35] to the range [0, 1]. The resulting value is then multiplied by 0.2, and the final result is assigned to the "alpha" attribute.



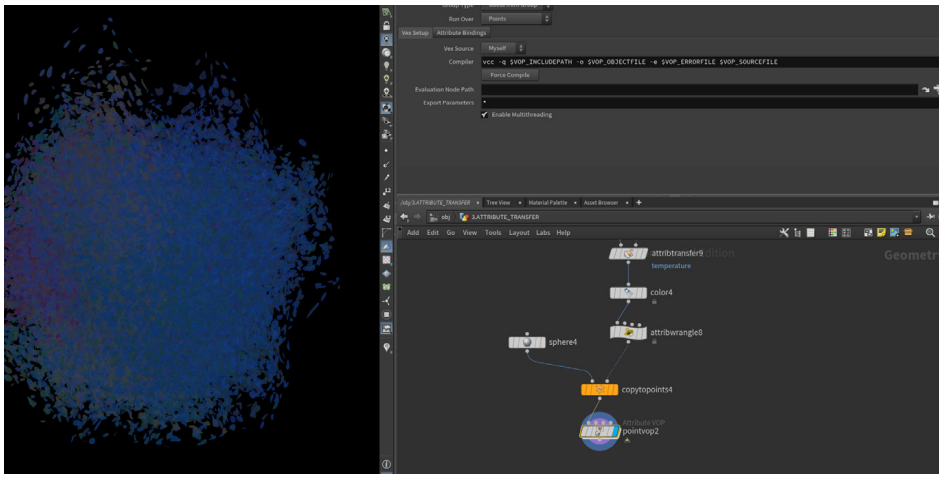
11: Sphere SOP

The "sphere" is a primitive geometric type: polygon. It's one of the fundamental geometric data types that contains points, primitives of type polygon, and vertices



12: Copy to Points SOP

This SOP is used to copy one or more geometric objects onto another set of points. Here, by inputting a spherical geometric object into the node and specifying another set of points as the target, the spherical geometry will be duplicated and arranged at each target point, effectively turning each point in the image into a sphere.

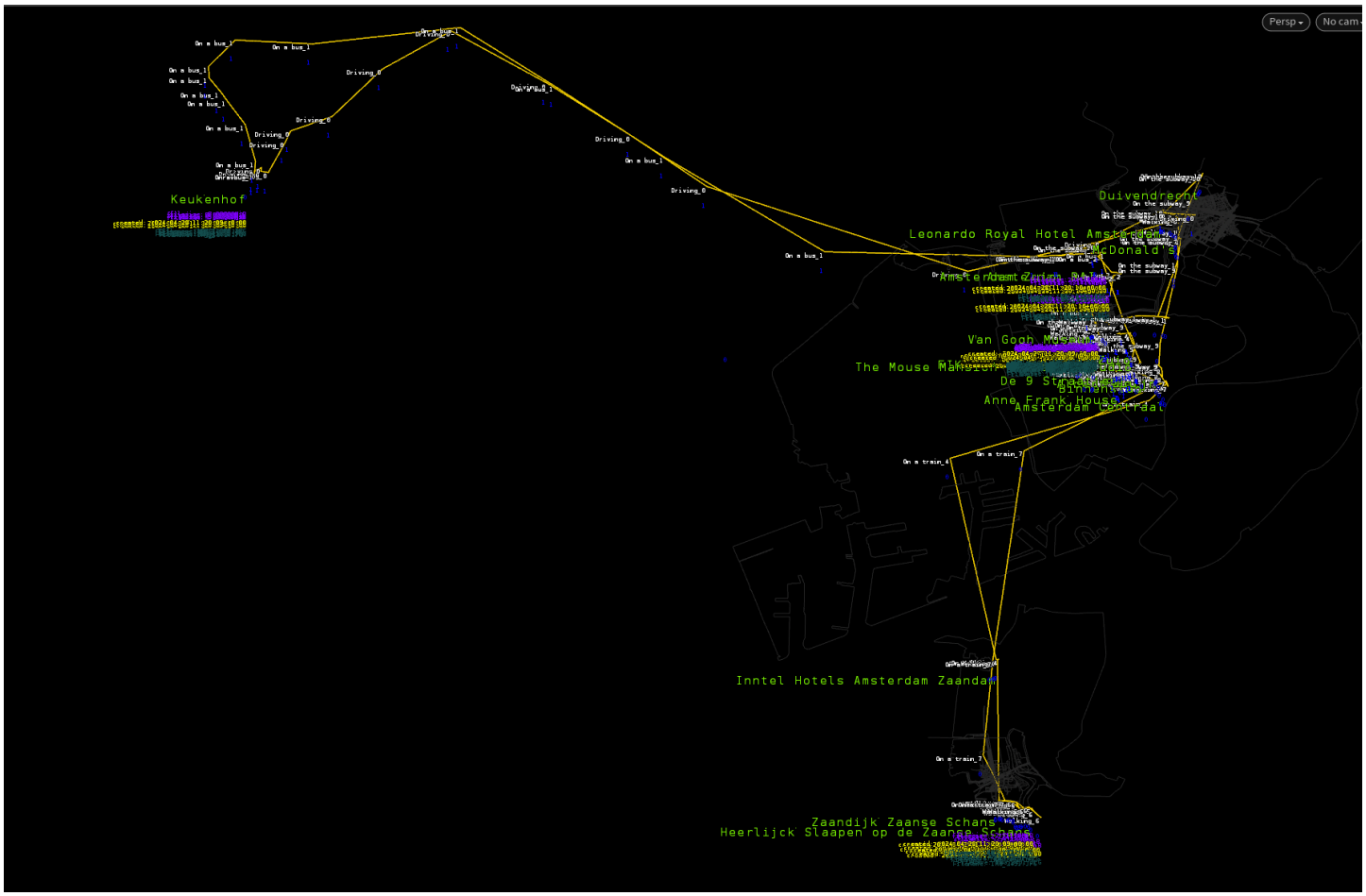


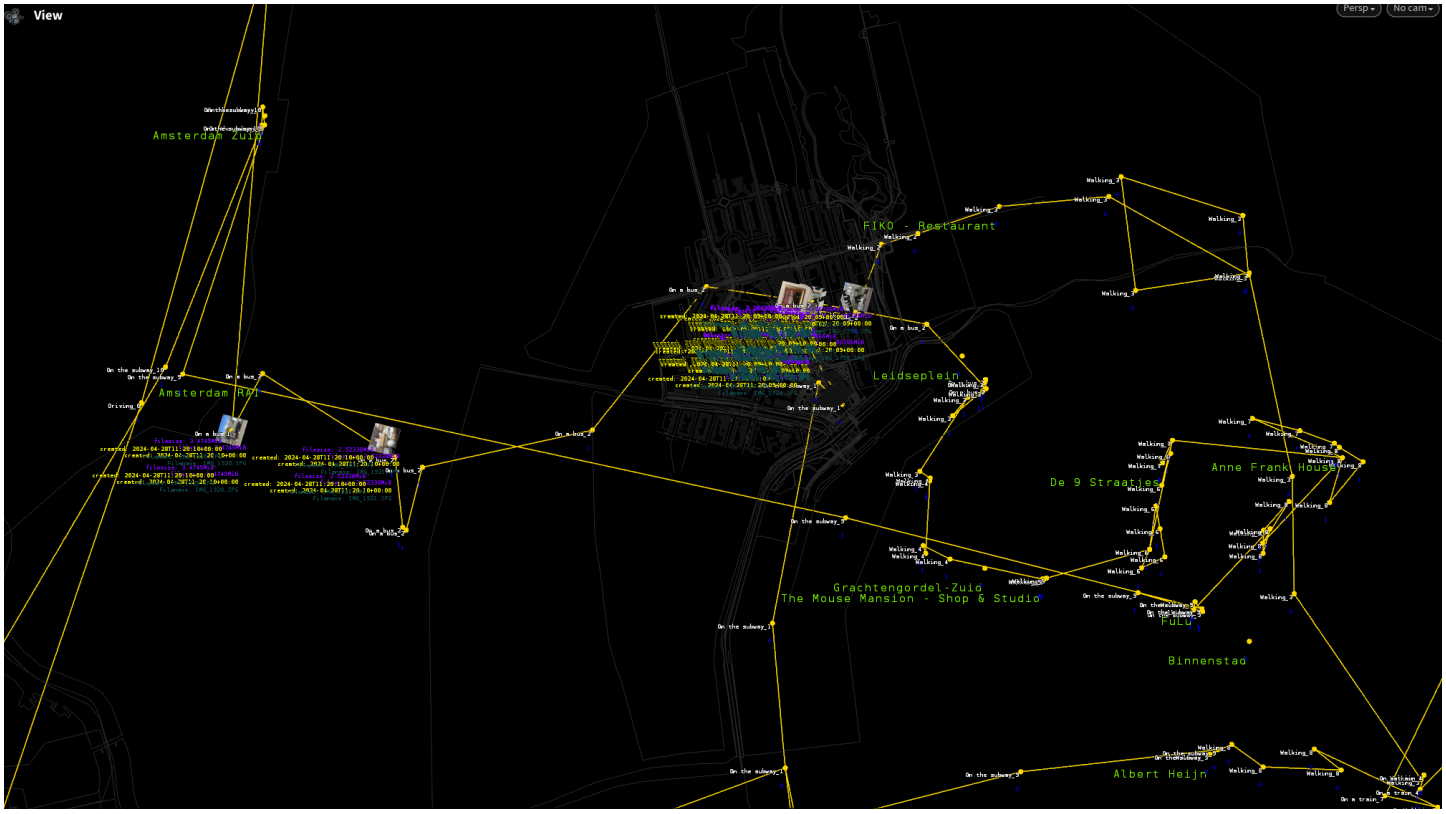
13: Point VOP SOP

This SOP is used to execute custom node programs at the point level. It calculates, transforms, or generates attributes for points. It is utilized for creating complex effects, processing point clouds, and controlling particle systems.

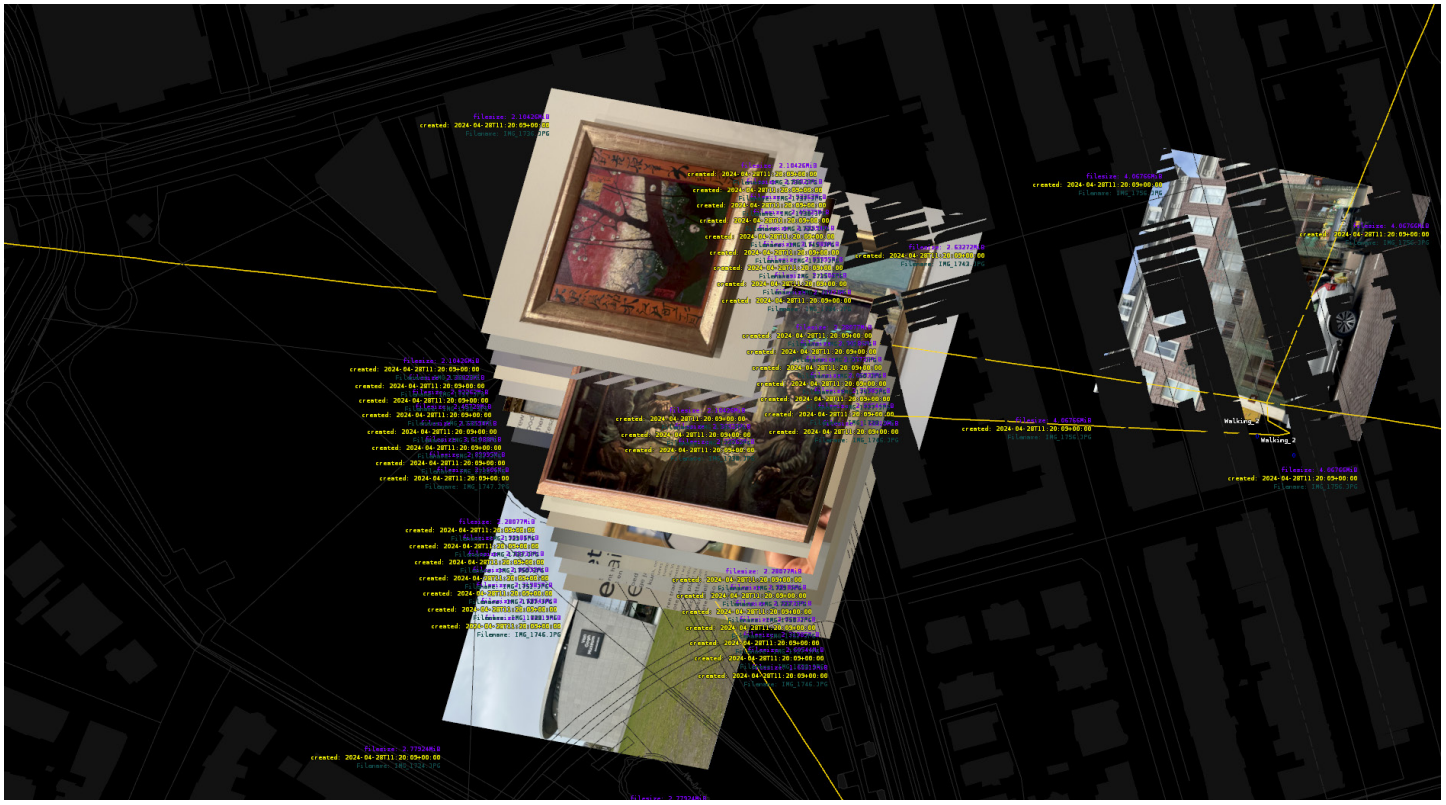
2 Visualizing GPS and Image Metadata

• Load in my own images and GPS data. (Path-Images-City)

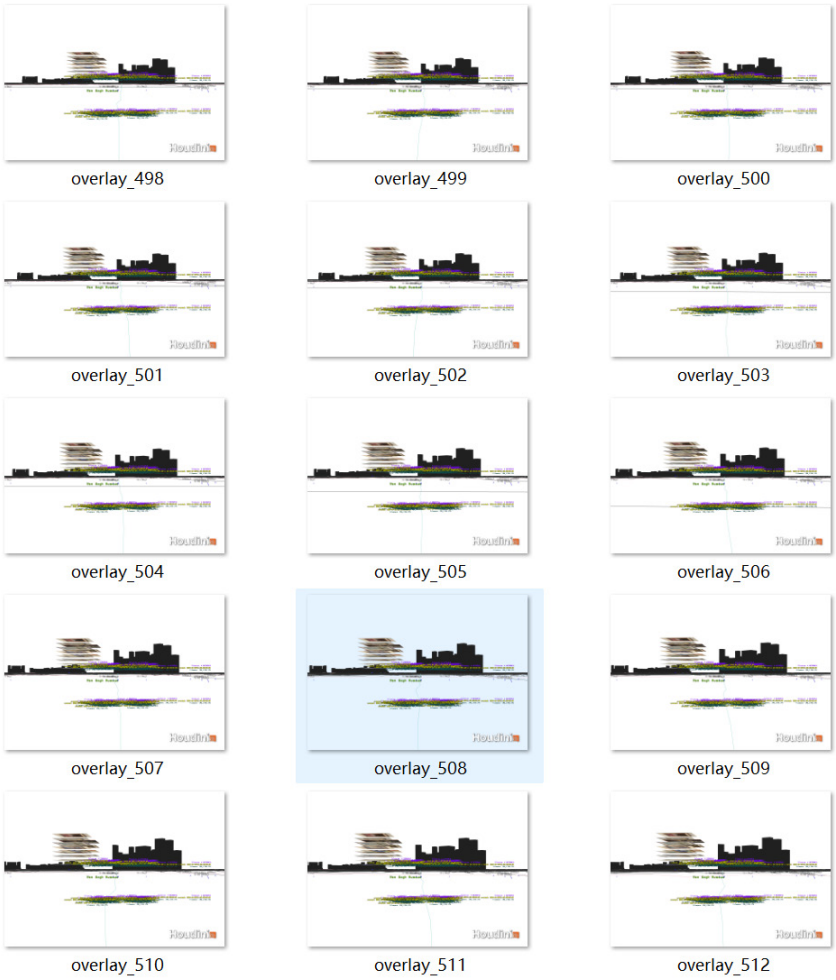




2 Visualizing GPS and Image Metadata



• Generate a series of images depicting my travels.



[Output frames: hyperlink]
※ If the hyperlink doesn't open, please try this URL:
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlul_ucl_ac_uk/EpxwRWStUj1Dh6tPP_DrmRUBfJ4DAPqFKWVqHyzkVhv2tg

3 Video to 3D Model

Description Of The Procedure:

1. Convert video to frames: Use code to convert the video into a series of image frames.

2. Import into RealityCapture: Import these image frames into RealityCapture and generate the corresponding 3D model. RealityCapture will automatically detect and reconstruct the camera path.

3. Export the model: Export the generated 3D model from RealityCapture.

4. Import into Houdini: Import the exported 3D model into Houdini.

5. Reconstruct camera path: In Houdini, create a camera path to simulate the camera movement of the video.

6. Add digital and textual information: Use Houdini's nodes and tools to add digital and textual information to the scene. This involves operations such as geometric modeling, texture mapping, and font creation.

7. Add volume effects: Use Houdini's volume modeling and rendering tools to add volume lighting, scattering, and other effects. This involves using Houdini's volume manipulation nodes and materials.

8. Add curvature and other information: As needed, calculate and add curvature and other geometric information using Houdini's nodes and tools.

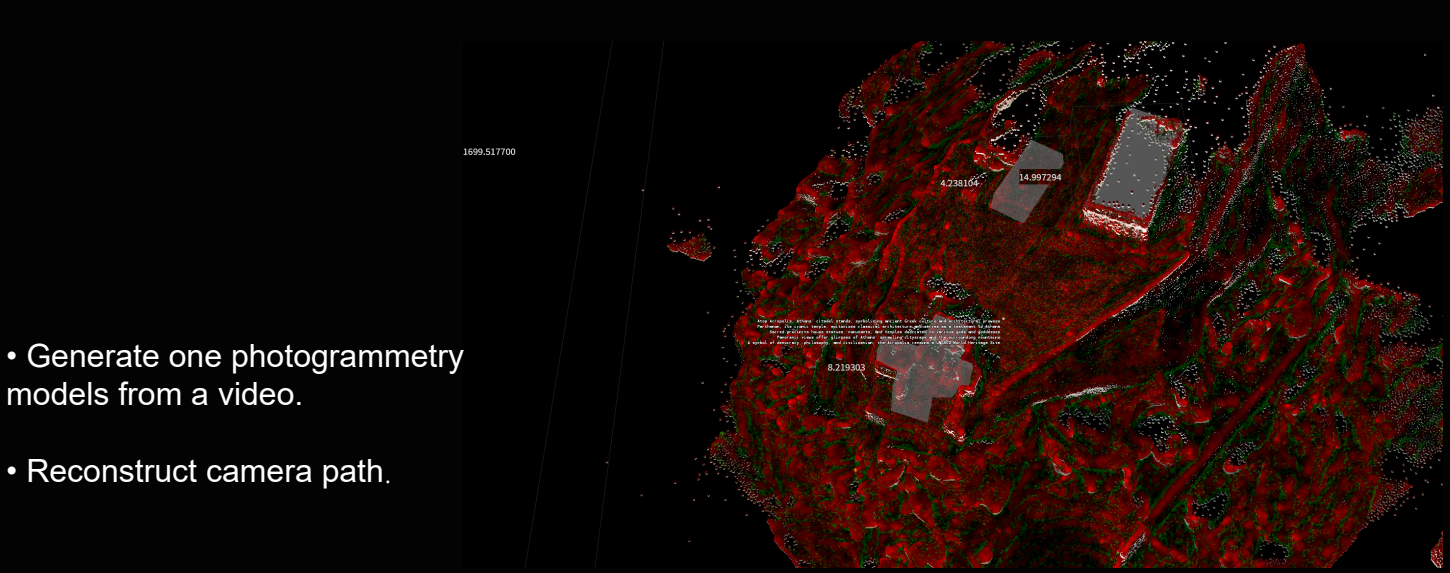
3 Video to 3D Model

• Generate one photogrammetry models from a video.

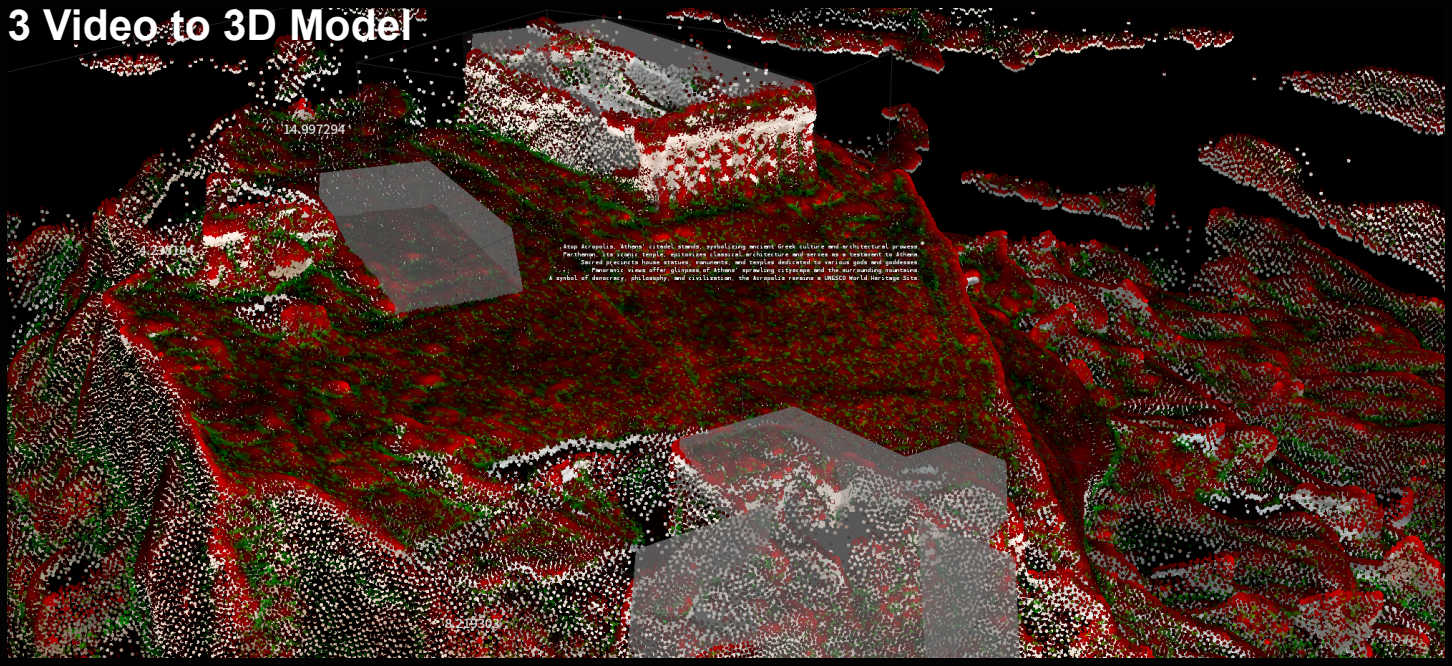
• Reconstruct camera path.

• Generate one photogrammetry models from a video.

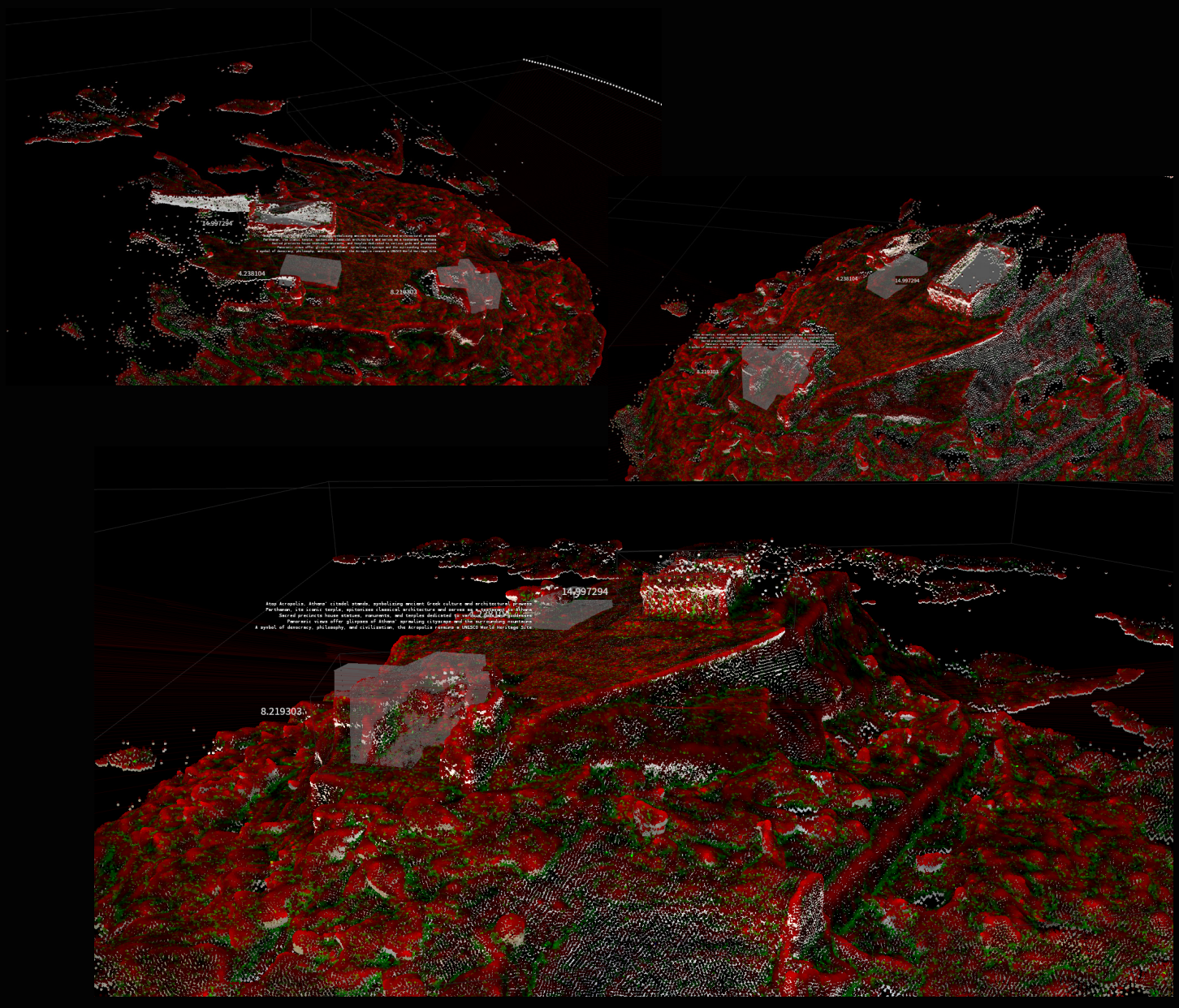
• Reconstruct camera path.



3 Video to 3D Model



• Find a way of adding information to the subject. (Volume speculation, curvature, etc.)

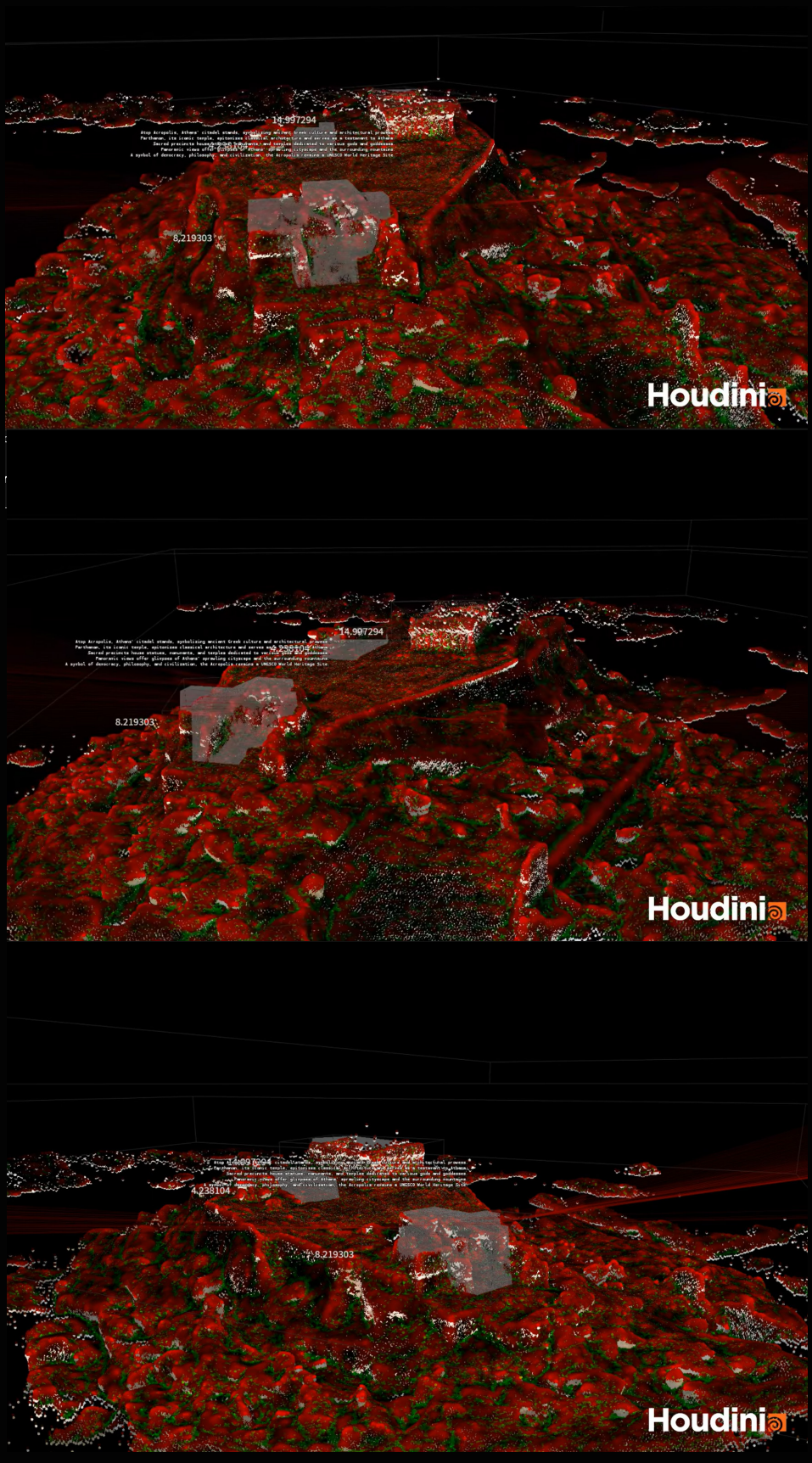


[adding information to the subject: 'Volume speculation, curvature, etc.']

3 Video to 3D Model

• Visualize model as a rendering video. (hyperlink)

※ If the hyperlink doesn't open, please try this URL:
https://liveuclac-my.sharepoint.com/:v:/g/personal/ucbvlul_ucl_ac_uk/Ecr4EHoA_PpHikVhqZJonf4BZEAmb-4f0U_uKLFmqcH_5g



[Frames from the rendering video']

4 Visualizing JSON in Houdini

Description Of The Procedure:

1.Read JSON file: First, use Houdini's Python API to read the JSON file.

2.Parse JSON data: Extract movie names, subtitles for each frame, image_n, time, film_path, etc from the JSON file.

3. Import movie frames: Import each frame of the movie into Houdini. Use Houdini nodes to load the sequence of movie frames.

4. Match JSON information: Match the information from the JSON file with the imported movie frames. Associate dialogue, image names, time, etc., with each frame based on their respective information.

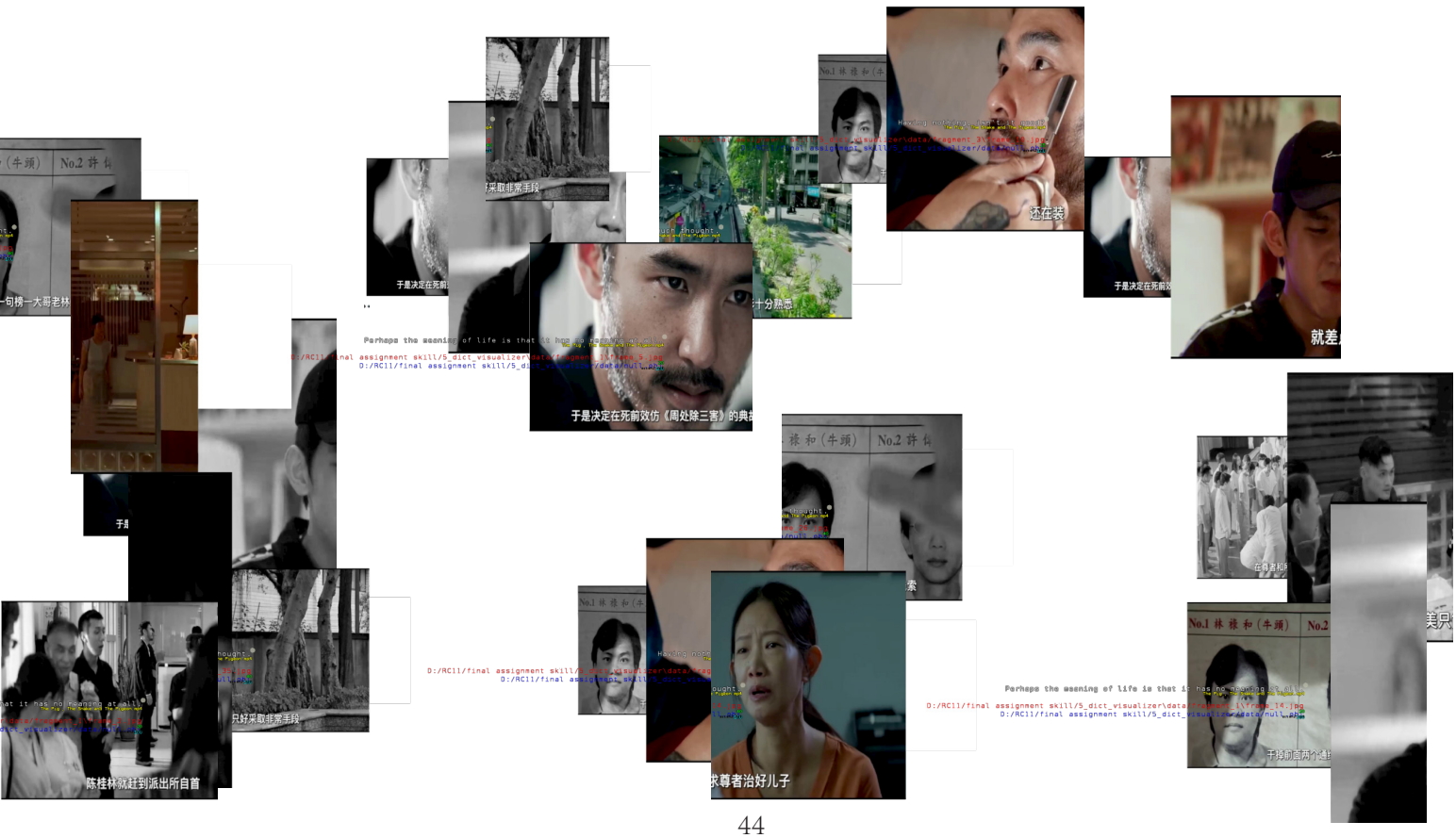
5. Visualization: Create corresponding geometry objects (such as grids) in Houdini to visualize the information from the JSON file.

6. Layout and rendering: Finally, layout the scene in Houdini and render it using rendering nodes.

{"folder": [{"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}, {"paragraph": "subtitle_1", "film_id": "The Pig , The Snake and The Pigeon.mp4", "image_name": "frame", "model_path": "", "image_n": 10, "time": 100, "film_path": "zccsh2.mp4"}]}

• Output: rendering video (camera1)

※If the hyperlink doesn't open, please try this URL:
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlul_ucl_ac_uk/ElBALOze5AROqhHWE9M5vrIB15HVxyfIrEijTiNM61Tn5Q



4 Visualizing JSON in Houdini

• Output: rendering video (camera4)

※If the hyperlink doesn't open, please try this URL: https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlul_ucl_ac_uk/ElBALOze5AROqhHWE9M5vrIB15HVxyfIrEijTiNM61Tn5Q

SKILLS CLASS RC11 - 23/24

COMPLEXITY ASSIGNMENT

FINAL ASSIGNMENT
EXPLANATION AND OUTCOMES

Rc11
Tutor: Ceel Pierik
SN: 23160576

Content

OneDrive

※If the hyperlink doesn't open, please try this URL:
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvlul_ucl_ac_uk/En2NJHMbKRJNlzh_KGVst9cBxkz5ps-flPiJf7WoGXsuxQ

GITHUB

※If the hyperlink doesn't open, please try this URL:
https://github.com/UD-Skills-2023-24/RC11_23160576/tree/main

Exercise One

Requirements:

1. Implement this algorithm in Python. Use the NumPy ndarray object for your matrices;
2. Give the asymptotic time complexity of the above algorithm or your implementation (they should be the same). Justify and explain your answer.

Suggestions:

1. Implement the algorithm with nested lists and compare the algorithms on different sizes of matrices;
2. Compare with built-in multiplication functions;
3. Look at multiplying more than two matrices, or at other operations on matrices.

In [1]:
```python
import numpy as np
import time

# Algorithm 1: Square-Matrix-Multiply using NumPy
def SMM_np(A, B):
    n = A.shape[0]  # number of rows of A
    C = np.zeros((n, n))  # initialize new n x n matrix C

    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i, j] += A[i, k] * B[k, j]  # compute the dot product

    return C

# Algorithm 2: Square-Matrix-Multiply using nested lists
def SMM_lists(A, B):
    n = len(A)  # number of rows of A
    C = [[0]*n for _ in range(n)]  # initialize new n x n matrix C

    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]  # compute the dot product

    return C

# Compare algorithms on different sizes of matrices
sizes = [10, 100, 500]  # Example sizes of matrices
for size in sizes:
    A_np = np.random.rand(size, size)
    B_np = np.random.rand(size, size)
    A_list = [[np.random.rand() for _ in range(size)] for _ in range(size)]
    B_list = [[np.random.rand() for _ in range(size)] for _ in range(size)]

    # Time the NumPy implementation
    start_np = time.time()
    result_np = SMM_np(A_np, B_np)
    end_np = time.time()
    np_time = end_np - start_np

    # Time the nested lists implementation
    start_lists = time.time()
    result_lists = SMM_lists(A_list, B_list)
    end_lists = time.time()
    lists_time = end_lists - start_lists

    print(f"Matrix size: {size}x{size}")
    print(f"NumPy implementation time: {np_time} seconds")
    print(f"Nested lists implementation time: {lists_time} seconds")

# Compare with built-in multiplication functions
start_builtin = time.time()
result_builtin = np.matmul(A_np, B_np)
end_builtin = time.time()
builtin_time = end_builtin - start_builtin
print(f"Built-in multiplication time: {builtin_time} seconds")
```

```
Matrix size: 10x10
NumPy implementation time: 0.0 seconds
Nested lists implementation time: 0.0 seconds
Matrix size: 100x100
NumPy implementation time: 0.2861495018005371 seconds
Nested lists implementation time: 0.06073737144470215 seconds
Matrix size: 500x500
NumPy implementation time: 77.07705211639404 seconds
Nested lists implementation time: 23.80222511291504 seconds
Built-in multiplication time: 0.1825664043426513 seconds
```

Answers:

For Requirement2:

To determine the asymptotic time complexity of the provided algorithms, let's analyze each one separately:

- Square-Matrix-Multiply using NumPy (SMM_np):

The algorithm consists of three nested loops iterating over the size of the input matrix (assuming square matrices). Each iteration performs a constant number of operations (multiplication and addition). Therefore, the time complexity of this algorithm is $O(n^3)$, where n is the size of the input matrix.

- Square-Matrix-Multiply using nested lists (SMM_lists):

Similar to the NumPy implementation, this algorithm also consists of three nested loops iterating over the size of the input matrix. However, accessing elements in nested lists might have slightly higher overhead compared to NumPy arrays due to memory layout differences. Nevertheless, the time complexity remains $O(n^3)$ because each operation inside the loops is constant time.

For Suggestion2:

Compare with built-in multiplication functions:

Compared to built-in multiplication functions, the manually implemented algorithm may experience performance differences, particularly for large matrices. Built-in multiplication functions are often highly optimized and utilize more efficient algorithms and low-level optimizations. As a result, for large matrices, built-in multiplication functions tend to be faster and have lower time complexity. However, for small matrices, the performance difference between the two approaches may not be significant.

Exercise Two

Requirements:

1. Describe and explain the algorithm. It should contain at least the following:
    - recursiveness: How is it recursive? What is (the criteria for) the base case? How does the recursion step reduce to the base case?
    - divide-and-conquer : How does this algorithm fit into the divide-andconquer approach? Explain each step of divide, conquer, and combine for this algorithm (as in slide 8 / pdf page 16 of the lecture slides).
2. Implement the recursive algorithm in Python. Reflect on which steps of the pseudocode were straightforward to implement and which hid a lot of complexity behind their language.
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.
    - Use slides 20-21 of the lecture slides (pages 40-45 of the pdf) of Complexity lecture 2 as a guideline.
    - You do not have to do the tree analysis (slides 22-23), but do take a guess what the final complexity is.

Suggestions:

1. Do a tree analysis;
2. Test and compare the practical speed with the non-recursive algorithm.

In [3]:
```python
def SMMRec(A, B):
    """
    Recursive algorithm for square matrix multiplication

    Parameters:
        A (list of lists): Matrix A
        B (list of lists): Matrix B

    Returns:
        C (list of lists): Resultant matrix C = A * B
    """
    n = len(A)
    C = [[0 for _ in range(n)] for _ in range(n)]

    # Base case: if the size is 1, compute the multiplication directly
    if n == 1:
        C[0][0] = A[0][0] * B[0][0]
    else:
        # Divide: Quarter matrices A, B, and C
        A11, A12, A21, A22 = quarter_matrix(A)
        B11, B12, B21, B22 = quarter_matrix(B)

        # Conquer: Recursively compute submatrices
        # Each recursive call represents a node in the recursion tree
        C11 = SMMRec(A11, B11)  # Node 1
        C12 = SMMRec(A11, B12)  # Node 2
        C21 = SMMRec(A21, B11)  # Node 3
        C22 = SMMRec(A21, B12)  # Node 4

        # Combine: Combine results
        combine_matrix(C, C11, C12, C21, C22)

    return C

def quarter_matrix(M):
    """
    Divide matrix M into four equal-sized submatrices

    Parameters:
        M (list of lists): Input matrix

    Returns:
        A11, A12, A21, A22 (list of lists): Four submatrices of M
    """
    n = len(M)
    mid = n // 2
    return M[:mid], M[:mid], M[mid:], M[mid:]
```

Answers:

For Requirement1:

Describe and explain the algorithm:

Recursiveness:

The recursive algorithm, Square-Matrix-Multiply-Recursive (SMMRec), decomposes the matrix multiplication problem by recursively dividing the matrices into smaller submatrices until reaching the base case, where the matrices become 1x1. The base case is determined when the size of the matrices becomes 1. The recursive step divides the matrices into smaller submatrices until reaching the base case.

Divide-and-conquer:

Divide: Divide the sequence of n elements to be sorted into two subsequences of n/2 elements each. In our case, the matrix multiplication algorithm divides the matrices into four equally sized submatrices.

Conquer: Sort the two subsequences recursively using merge sort (or in our case, the algorithm itself), to sort them individually. In our algorithm, we recursively apply the algorithm to each submatrix to compute their products.

Combine: Merge the two sorted subsequences to produce the sorted answer. In our algorithm, we combine the results of the recursive calls to form the final result matrix.

For Requirement3:

Do a complexity analysis for the SMMRec algorithm:

Base Case: The complexity of the base case is $0(1)$ since it involves a single multiplication operation.
Divide Step: The divide step simply computes the middle of the subarray, which takes constant time. Thus, D(n)=O(1).
Conquer Step: We recursively solve two subproblems, each of size n/2, which contributes $2 \cdot T(n/2)$ to the running time.
Combine Step: In the combine step, merging two sorted subarrays of size n/2 takes O(n) time. Therefore, C(n)=O(n).

Based on the provided recurrence equation:

$$T(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1 \\ 2T(\frac{n}{2}) + \mathcal{O}(n) & \text{if } n > 1 \end{cases}$$

Using the Master Theorem, with a=2, b=2, and f(n)=n, I find that the overall complexity of the algorithm is O(n log n).

Exercise Three

Requirements:

1. Reflect on the difference between (complexity of) addition/subtraction and multiplication on matrices.
2. Do a complexity analysis of the Strassen algorithm.
   - Instead of starting from scratch, you can also take your result from Exercise 2 and adapt to the optimisation; explain what changes in the complexity formula with these optimisations.

In [4]:

```python
def SMMStrassen(A, B):
    """
    Strassen algorithm for square matrix multiplication

    Parameters:
        A (list of lists): Matrix A
        B (list of lists): Matrix B

    Returns:
        C (list of lists): Resultant matrix C = A * B
    """
    n = len(A)
    # Base case: if the size is 1, compute the multiplication directly
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        # Divide: Quarter matrices A, B, and C
        A11, A12, A21, A22 = quarter_matrix(A)
        B11, B12, B21, B22 = quarter_matrix(B)

        # Compute the 7 auxiliary matrices
        P1 = SMMStrassen(A11, subtract_matrix(B12, B22))
        P2 = SMMStrassen(add_matrix(A11, A12), B22)
        P3 = SMMStrassen(add_matrix(A21, A22), B11)
        P4 = SMMStrassen(A22, subtract_matrix(B21, B11))
        P5 = SMMStrassen(add_matrix(A11, A22), add_matrix(B11, B22))
        P6 = SMMStrassen(subtract_matrix(A12, A22), add_matrix(B21, B22))
        P7 = SMMStrassen(subtract_matrix(A11, A21), add_matrix(B11, B12))

        # Combine: Combine results
        C11 = add_matrix(subtract_matrix(add_matrix(P5, P4), P2), P6)
        C12 = add_matrix(P1, P2)
        C21 = add_matrix(P3, P4)
        C22 = subtract_matrix(subtract_matrix(add_matrix(P5, P1), P3), P7)

        # Combine the quarters into the result matrix
        C = combine_matrix(C11, C12, C21, C22)

    return C

def add_matrix(A, B):
    """Add two matrices element-wise"""
    n = len(A)
    return [[A[i][j] + B[i][j] for j in range(n)] for i in range(n)]

def subtract_matrix(A, B):
    """Subtract matrix B from matrix A element-wise"""
    n = len(A)
    return [[A[i][j] - B[i][j] for j in range(n)] for i in range(n)]

def quarter_matrix(M):
    """
    Divide matrix M into four equal-sized submatrices

    Parameters:
        M (list of lists): Input matrix

    Returns:
        A11, A12, A21, A22 (list of lists): Four submatrices of M
    """
    n = len(M)
    mid = n // 2
    return [row[:mid] for row in M[:mid]], [row[mid:] for row in M[:mid]], \
           [row[:mid] for row in M[mid:]], [row[mid:] for row in M[mid:]]

def combine_matrix(C11, C12, C21, C22):
    """
    Combine four submatrices into one matrix
```

Answers:

For Requirement1:

Reflecting on the Difference Between Addition/Subtraction and Multiplication in Matrices:

Addition/Subtraction: These operations typically have a linear complexity, meaning their execution time scales quadratically with the size of the matrices involved. For matrices of size n x n, addition and subtraction operations have a time complexity of $O(n^2)$.

Multiplication: On the other hand, traditional matrix multiplication algorithms usually have a cubic time complexity of O(n^3). This is because each element in the resulting matrix is computed by summing the products of corresponding elements from the row and column matrices.

For Requirement2:

Complexity Analysis of the Strassen Algorithm:

Basic Idea: The Strassen algorithm aims to reduce the number of multiplications involved in matrix multiplication by recursively breaking down the problem into smaller subproblems.

Original Complexity: Traditional matrix multiplication algorithms have a time complexity of $O(n^3)$, as each element in the resulting matrix requires summation of n products.

Strassen Algorithm: By recursively dividing matrices into smaller submatrices and employing specific formulas, Strassen algorithm reduces the complexity to approximately $O(n^{2.81})$. This is achieved by decreasing the number of multiplications required and introducing additional operations.