Explaination and Outcomes

RC11_23090277

01 Python:

OneDrive Link

3 Datasets

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvyh4_ucl_ac_uk/EnLnK1dTiT1PugemuGsjTM4B3AQ3R9zqHNfC6wJwQIdpOg?e=sI39Fd

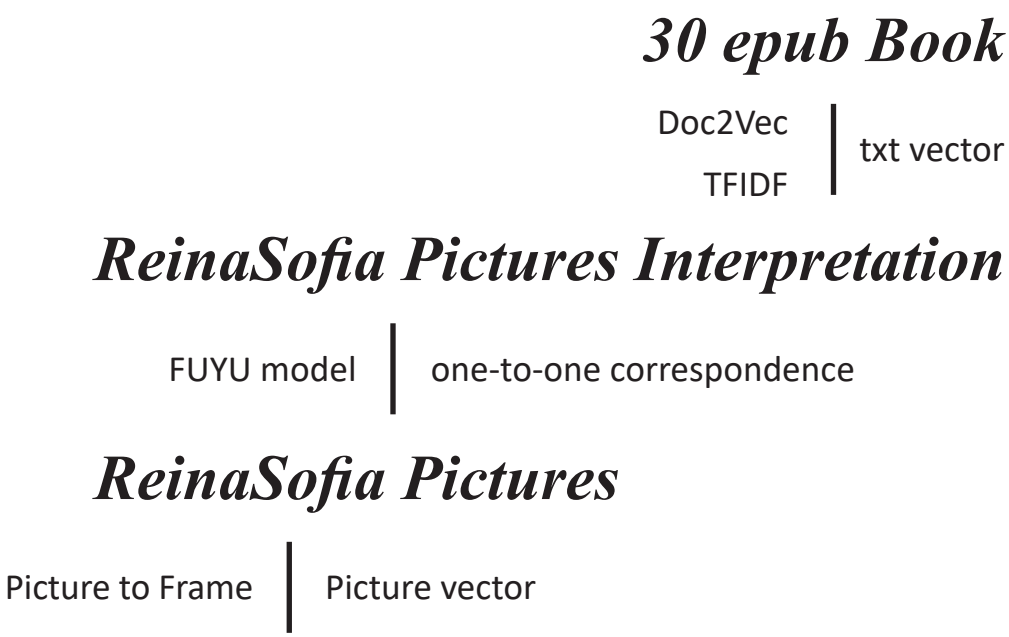
Files to import

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvyh4_ucl_ac_uk/EjZU1TSdFO9FkpVJzZxNPkEBEwfEk9iV-GSVHT8RVPlEDg?e=p0ef3d

Code

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvyh4_ucl_ac_uk/Ev47yxqjZTVCk4F4kBrS2L8B6K2J-apH2EqrlH0BP_cA-w?e=GzkWkk

In this task, I mainly completed the vectorisation and cross-searching of 30 book epubs about Spain, 1400 images from the ReinaSofia Museum and 150 videos from poc in the following way:



Input from text:

1.input text into 30 Books SOM to get a most related paragraph.
2.the paragraph as input into ReinaSofia pictures interpretation text SOM to get picture description.
3.through picture description get the best matching picture.
4.Extract a frame from the video for 10s to train the SOM, then match it with the image input and extend the most matched frame for 5s to output the video.

Input from picture:

1.Getting images directly from the image SOM by inputting an image.
2.Getting paragraph in Text SOM by using the description of the picture as input.
3.The input image is matched with the SOM of the video frame to obtain the video.

The same goes for **input video**, from video frames to images, image descriptions to text

The actual process:

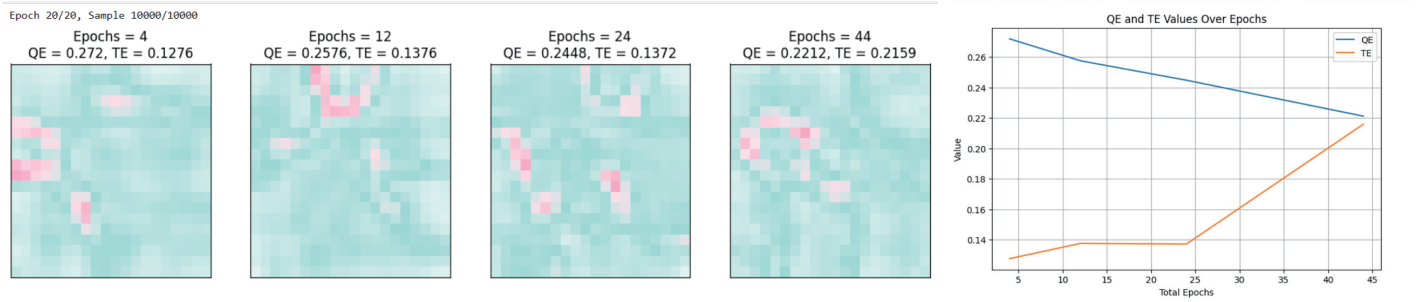
1.0 Text to text

1.1.1 Text pre-processing, giving TFIDF and Doc2Vec training vectors respectively

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

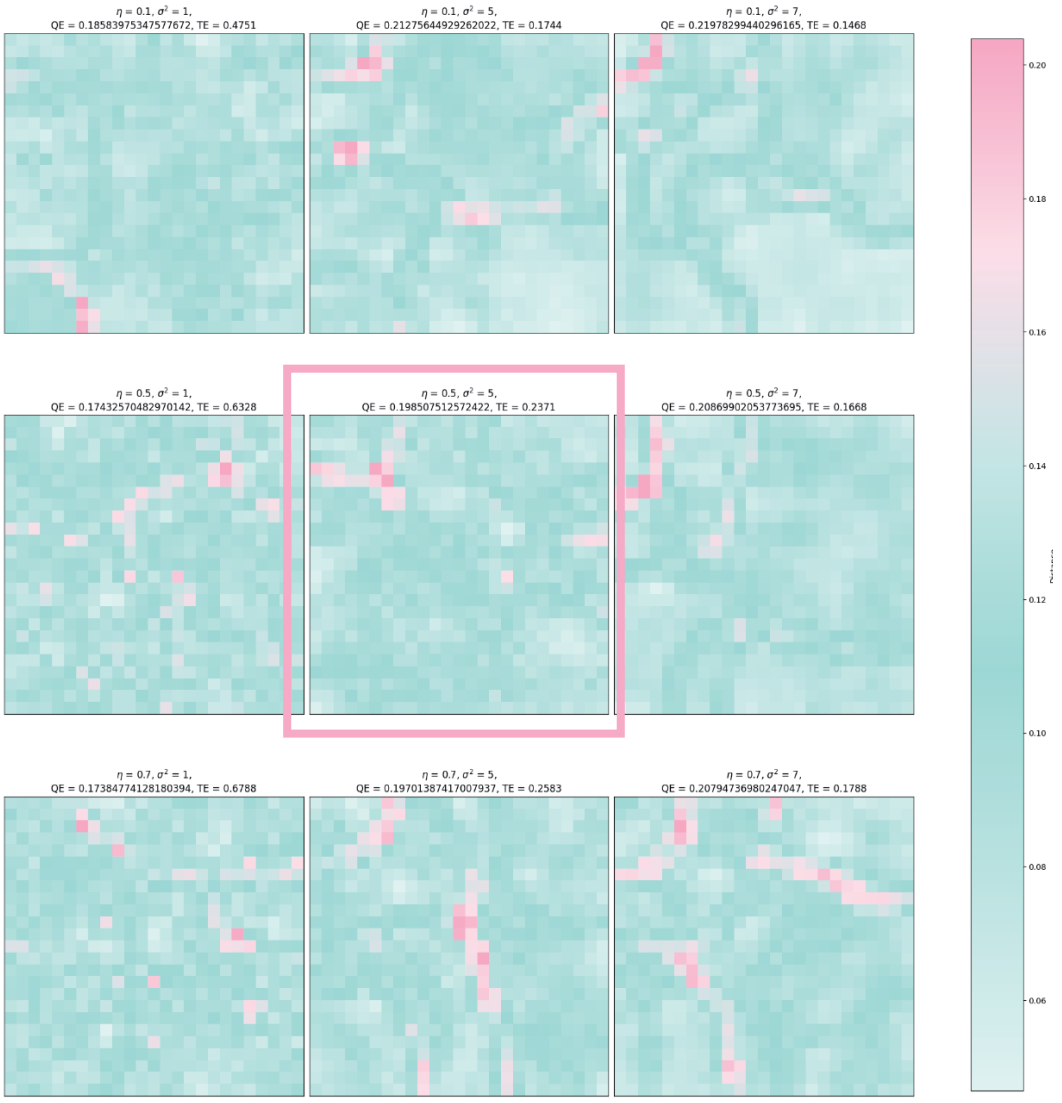
```
PREPROCESSED2 = preprocess2([p['paragraph'] for p in PARAGRAPHS])
```

1.1.2 Training SOM with doc2vec



TE : Measures how well the SOM maintains the topological relationships of the input space. It assesses if the two closest neurons (winner and runner-up) to an input vector are neighbors in the map. A high TE indicates poor topology preservation.

QE : Quantifies the average distance between each input vector and its nearest neuron in the SOM. A lower QE indicates better accuracy in representing the input data on the map, showing the effectiveness of the SOM in capturing the data's variability.

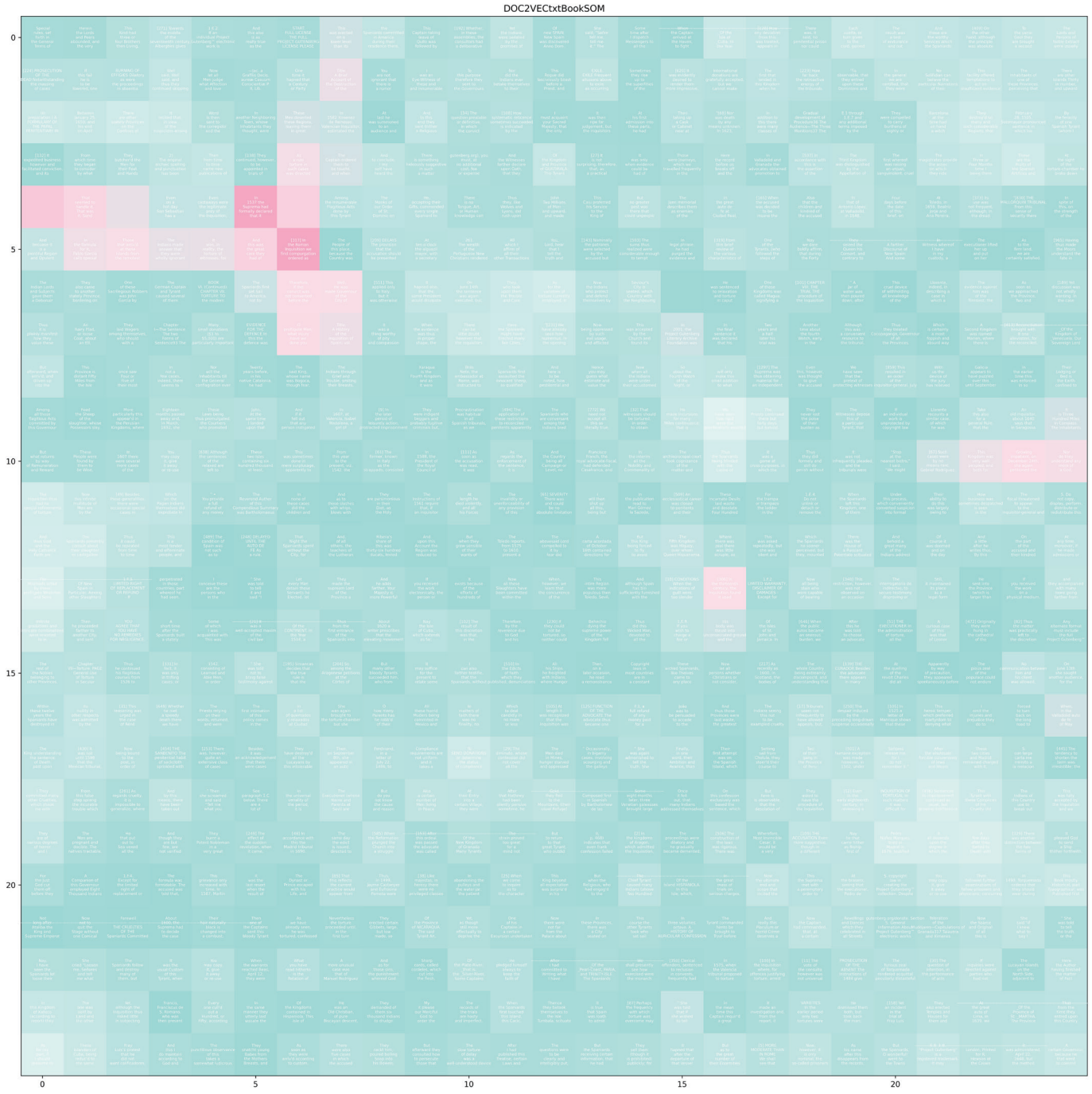


I chose SOM [4] because it has the lowest average of both TE QEs

1.1.3 Assigning data to SOM units

```
data_dict1 = []
for i in range(len(SOM1)):
    row = []
    for j in range(len(SOM1[0])):
        row.append([])
    data_dict1.append(row)
vectortextPairs=[]
for i in range(0,len(PARAGRAPHS1)):
    vectortext={}
    vectortext['text']=PARAGRAPHS1[i]
    vectortext['vector']=train_data1[i]
    vectortextPairs.append(vectortext)
for i in vectortextPairs:
    g,h = find_BMU(SOM1,i['vector'])
    data_dict1[g][h].append(i)
```

Text passages are read from the PARAGRAPHS1 list and the corresponding vectors are read from train_data1. Each paragraph and its corresponding vector are encapsulated into a dictionary and added to the vectortextPairs list, and the BMUs are found using the find_BMU function and added to the corresponding positions in data_dict1.



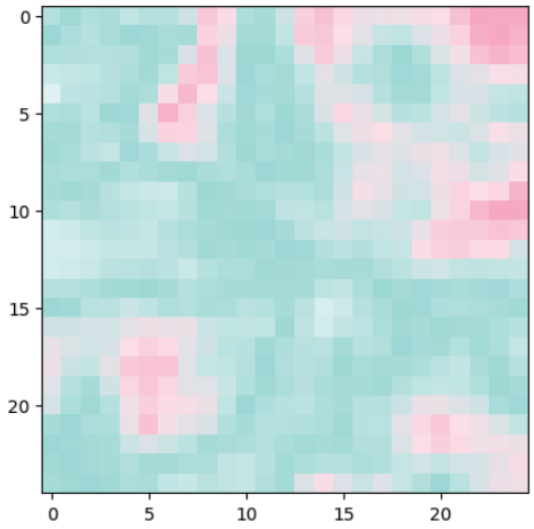
1.1.4 Creating Search Functions

The function processes the query words and transforms them into vectors through the Doc2Vec model, then finds the most matching cell in the SOM and calculates the similarity of all the text within that cell to the query vector, eventually returning the text passage with the highest similarity.

```
def search_TextSom1(SOM1, model, data_dict1, query='street'):
    result = []

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

    fig = plt.figure()
    plt.figure(figsize=(5, 5))
    plt.imshow(activate1(train_data1, SOM1, query_vector), cmap=cmap)
    plt.show()
    g, h = find_BMU1(SOM1, query_vector)
    print((g, h))

    if g >= len(data_dict1) or h >= len(data_dict1[g]) or not data_dict1[g][h]:
        print("No data found for this BMU.")
        return []

    similarities = {}
    for i in data_dict1[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]])[0][0]

    if similarities:
        biggest = max(similarities, key=similarities.get)
        result.append(biggest)
    else:
        print("No similarities found.")

    return result
```

```
search_TextSom1(SOM1,DOC2VEC_MODEL,data_dict1,'Documents were sent by men to overseas colonies, and treasures were found on islands ')
```

```
<Figure size 640x480 with 0 Axes>
```




```
(0, 22)
[' They gained their living chiefly by hawking goods around the city; this at length aroused the shopkeepers, and the corregidor represented to the trib
unal that scandals were occasioned by their entering houses and committing indecencies; there was loss to the king for, as penitents, they were not subj
ect to the alcavala and other imposts; thus favored they undersold the shopkeepers, who had lost half of their trade, while the penitents grew rich, for
they came almost naked from the secret prison and, in a short time, they were well clothed and enriched.']
```

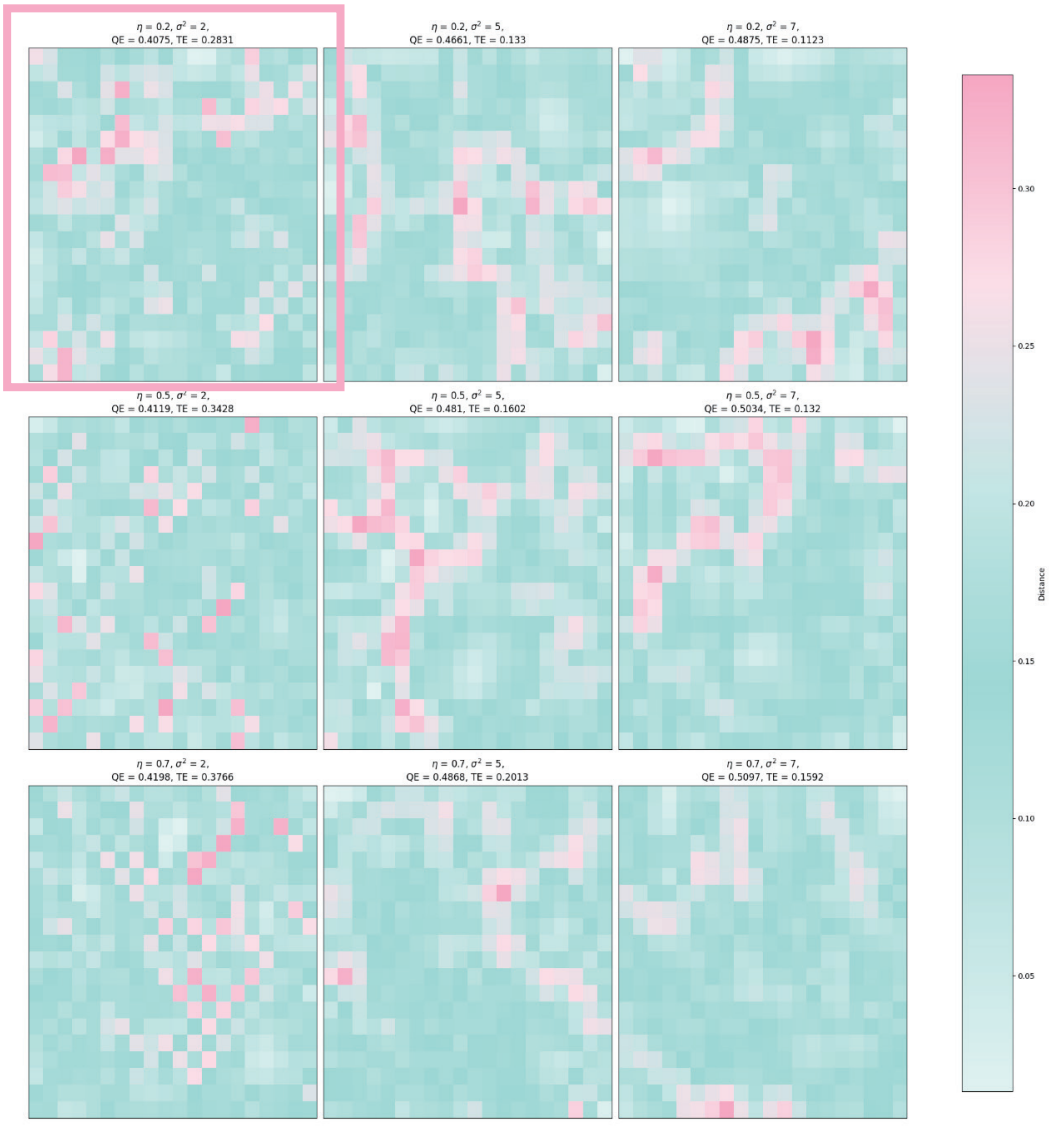
The function processes the query words and transforms them into vectors through the Doc2Vec model, then finds the most matching cell in the SOM and calculates the similarity of all the text within that cell to the query vector, eventually returning the text passage with the highest similarity.

1.1.5 Training SOM with TFIDF

Vectorise text with tfidf, dimensionalise with SVD and train som

```
VECTORIZER_TFIDF = TfidfVectorizer(min_df=2)
TFIDF_MATRIX = VECTORIZER_TFIDF.fit_transform(PREPROCESSED3)

non_zero_count = TFIDF_MATRIX.getnnz(axis=1)

non_zero_indices = [i for i, count in enumerate(non_zero_count) if count > 0]
filtered_tfidf_matrix = TFIDF_MATRIX[non_zero_indices, :]

svd = TruncatedSVD(n_components=250)
svd_u_matrix = svd.fit_transform(filtered_tfidf_matrix)

print(f"Number of non-zero rows: {len(non_zero_indices)}")
print(f"Filtered SVD Matrix Shape: {svd_u_matrix.shape}")
```

```
Number of non-zero rows: 9943
Filtered SVD Matrix Shape: (9943, 250)
```



I chose SOM [0] because it has the lowest average of both TE QEs

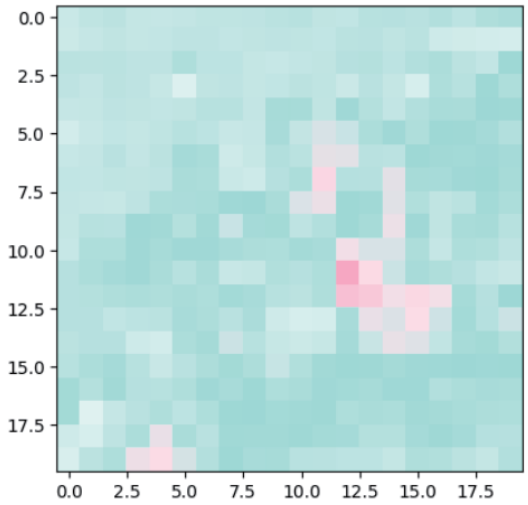
1.1.6 Creating Search Functions

After excluding the zero vectors due to the absence of valid information in the document, the TF-IDF representation of the text is mapped to the SOM 2D lattice, after which the questioning is vectorised by the similarity logic to return the activated som and bmu.

```
non_zero_indices = [i for i, count in enumerate(TFIDF_MATRIX.getnnz(axis=1)) if count > 0]
data_dict2 = [[[] for _ in range(len(SOM2[0]))] for _ in range(len(SOM2))]

vectortextPairs = []
for i in non_zero_indices:
    vectortext = {
        'text': PARAGRAPHS1[i],
        'vector': svd_u_matrix[non_zero_indices.index(i)]
    }
    vectortextPairs.append(vectortext)

for i in vectortextPairs:
    g, h = find_BMU(SOM2, i['vector'])
    data_dict2[g][h].append(i)
```

```
def search_TextSom2(SOM, model, data_dict2, query='street'):

    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)
    return result
```

71]:
```
search_TextSom2(SOM2, VECTORIZER_TFIDF,data_dict2,'Documents were sent by men to overseas colonies, and treasures were found on islands')
```



(11, 12)

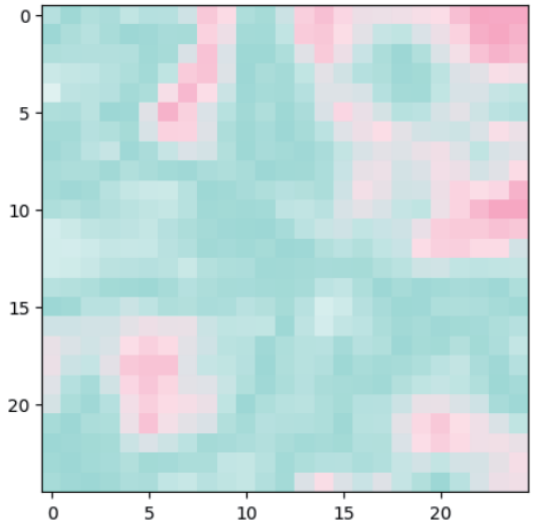
71]:
```
[{'paragraph': '[701] Meanwhile, Francisco Botelho, who had been sent with João's letter, was conferring with the pope, who blandly assured him that Li
pomano's mission was only to notify the king of the approaching convocation of the Council of Trent.',
  'nr': 1677,
  'bookID': 'A History of the Inquisition of Spain; vol. 3'}]
```

1.1.7 Doc2Vec VS TFIDF

Doc2Vec

```
search_TextSom1(SOM1,DOC2VEC_MODEL,data_dict1,'Documents were sent by men to overseas colonies, and treasures were found on islands ')
```

```
<Figure size 640x480 with 0 Axes>
```



Pros:

Doc2Vec captures the semantic relationships between words and documents, and therefore works better for semantic similarity computation.

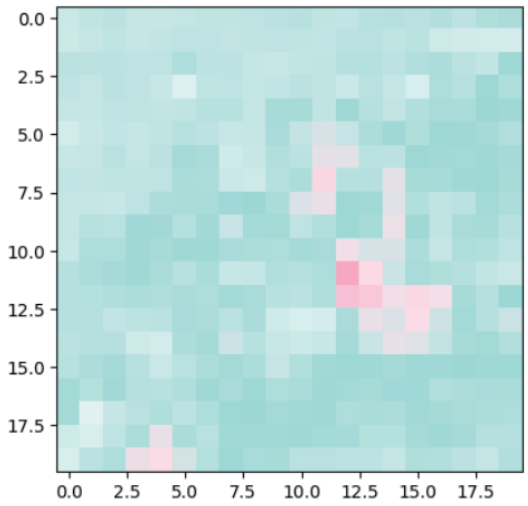
Cons:

Compared to TF-IDF, Doc2Vec models usually require more computational resources and time, and need a large amount of training data to get good performance, which may not be effective enough for small-scale datasets.

```
(0, 22)
[' They gained their living chiefly by hawking goods around the city; this at length aroused the shopkeepers, and the corregidor represented to the trib
unal that scandals were occasioned by their entering houses and committing indecencies; there was loss to the king for, as penitents, they were not subj
ect to the alcavala and other imposts; thus favored they undersold the shopkeepers, who had lost half of their trade, while the penitents grew rich, for
they came almost naked from the secret prison and, in a short time, they were well clothed and enriched.']
```

TFIDF

```
71]: search_TextSom2(SOM2, VECTORIZER_TFIDF,data_dict2,'Documents were sent by men to overseas colonies, and treasures were found on islands')
```



Pros:

TF-IDF pairs are simple and intuitive, suitable for information retrieval and keyword pairing sentences, faster computation than Doc2Vec.

Cons:

TF-IDF only considers the frequency of words and the frequency of documents, and ignores the semantic relationship between words, so it may not be able to capture the semantic similarity between words.

```
(11, 12)
71]: [{'paragraph': '[701] Meanwhile, Francisco Botelho, who had been sent with João's letter, was conferring with the pope, who blandly assured him that Li
pomano's mission was only to notify the king of the approaching convocation of the Council of Trent.',
  'nr': 1677,
  'bookID': 'A History of the Inquisition of Spain; vol. 3'}]
```

Since my project involves the need to react to the views of different groups of people on events, and it is more important to consider semantic connections rather than keywords, I chose doc2vec as a way to vectorise the text.

1.2 Text to image

1.2.1 Generate descriptive text for each drawing by fuyu model

```python
def read_csv_files(folder_path):
    files_list = []
    global_index = 0

    for index, filename in enumerate(os.listdir(folder_path)):
        if filename.endswith('.csv'):
            file_path = os.path.join(folder_path, filename)

            with open(file_path, 'r', encoding='utf-8') as csvfile:
                reader = csv.DictReader(csvfile)

                for row_num, row in enumerate(reader, start=1):
                    file_info = {
                        'nr': global_index,
                        'filename': row.get('Filename', ''),
                        'content': row.get('Text', ''),
                        'file_index': index,
                        'row_index': row_num
                    }
                    files_list.append(file_info)
                    global_index += 1

    return files_list
```

```
[94]: texts
```


```
[94]: [{'nr': 0,
  'filename': 'ReinaSofia_9.jpg',
  'content': 'In the image, there is a painting of a man wearing sunglasses standing next to a tall building. The painting is surrounded by various o
bjects, including a car and a potted plant. Additionally, there is a blue car parked nearby. The scene likely depicts a man standing next to a buildi
ng, possibly near a parking lot or in front of a tall building. The painting itself captures the',
  'row_index': 1},
 {'nr': 1,
  'filename': 'ReinaSofia_77.jpg',
  'content': 'What type of image is shown in the image?\n\nIn the image, there is a large piece of art depicting a brown dot pattern.\n',
  'row_index': 2},
 {'nr': 2,
  'filename': 'ReinaSofia_7.jpg',
  'content': 'In the image, there is a dark background with a yellow and blue abstract design. The artwork is painted on a wall, and the wall appears
to be rusted.\n',
  'row_index': 3},
```

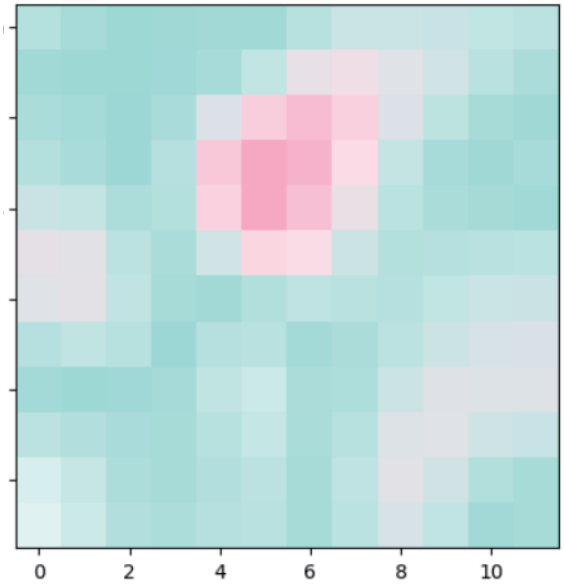

1.2.2 Train description text SOM

1.2.3 Find the image according to the file description

Since the textual SOM search part of the process is similar, it is omitted here and only the output results are shown, i.e., the corresponding image is opened by describing the corresponding file name.

```
ult3 = search_TextSom3(SOM3, DOC2VEC_MODEL2, data_dict3,featureTextPairs2, query='man take phone talk to woman about flower')
```

```
gure size 640x480 with 0 Axes>
```




```
5)
t similar content: In the painting, there is a portrait of a man wearing a tie and a suit. The man is wearing a tie, which signifies that h
ssed and likely posing for a formal portrait. The suit suggests that he is dressed in formal attire, likely for a professional or formal oc
ainting captures the man's appearance in excellent detail, showcasing
```

```
ename3 = [item['filename'] for item in result3]
```

```
ge_folder = r'D:\YeqinHUANG_Collection\ucl\Term3\Portfolio\week3\museumReinaSofia'
ge_path = os.path.join(image_folder, filename3[0])

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

ge = Image.open(image_path)
.imshow(image)
.axis('off')
.show()
```

```
ing to open: D:\YeqinHUANG_Collection\ucl\Term3\Portfolio\week3\museumReinaSofia\ReinaSofia_1161.jpg
```

1.3 Text to video

1.3.1 Extract Video frames

Processing video files and training SOM by pumping one frame every ten seconds and vectorising the frame images

```python
def process_all_films(folder_path, interval, model):
    film_features = []
    ensure_folder_exists('Frames')  # Ensure "Frames" folder exists

    for file in os.listdir(folder_path):
        if file.endswith(".mp4"):
            film_path = os.path.join(folder_path, file)
            features = processFilmByFrames(file, film_path, interval, model)
            film_features.extend(features)

    return film_features

def processFilmByFrames(filmName, filmPath, interval, model):
    features = []
    film = VideoFileClip(filmPath)
    nrFrames = int(film.duration / interval)

    ensure_folder_exists('Frames')

    for i in range(nrFrames):
        s = i * interval
        frame = film.get_frame(s)
        frame_image = Image.fromarray(frame, 'RGB')
        temp_frame_path = f'Frames/{filmName}_{s}s.jpg'
        frame_image.save(temp_frame_path)

        im = load_image(temp_frame_path)
        f = model.predict(im)[0]
        features.append({'film': filmPath, 'second': s, 'features': f, 'frame_path': temp_frame_path})

    return features
```

```python
model = 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'
)
```

```python
film_features = process_all_films('database03_video', 10, model)
```

```
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 47ms/step
```

1.3.2 Train SOM

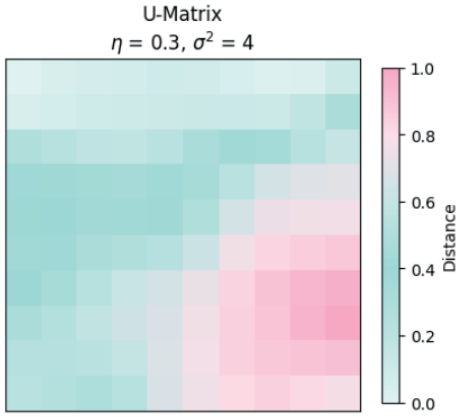
1.3.3 Search

Activate the SOM with the image input output from 1.2.3

[221]:
```
def searchvideopicturesom4(query):
    result = []
    query_features = []
    path = query
    q_f = processImage(path, model)
    query_features.append(q_f)

    print(f"Query features shape: {np.array(query_features).shape}")
    activatedSOM = activate(n_train_data4, SOM4, query_features)

    g, h = find_BMU(SOM4, query_features[0])
    print(f"BMU location: ({g}, {h})")

    plt.figure(figsize=(5, 4))
    im = plt.imshow(activatedSOM, cmap=cmap, aspect='auto')
    plt.title(f'U-Matrix\n$\eta$ = {0.3}, $\sigma^2$ = {4}')
    plt.colorbar(im, shrink=0.95, label='Distance')
    plt.xticks([])
    plt.yticks([])
    plt.show()

    closest_image_index = get_closest_image1(g, h)
    result.append(video_picture_data_dict[g][h][closest_image_index]['image'])
    return result
```

[294]:
```
query = image_path
result4 = searchvideopicturesom4(query)
```

```
1/1 [==============================] - 0s 33ms/step
Query features shape: (1, 1024)
BMU location: (7, 9)
```



[295]:
```
result4
```

[295]:
```
['database03_video\\video_16.mp4']
```

Find the position corresponding to the video and frame through the video path, intercept the video clip 5s after the corresponding frame to save and output the first frame image as a check

```
import os
from moviepy.video.io.VideoFileClip import VideoFileClip
import matplotlib.pyplot as plt

base_name = os.path.splitext(os.path.basename(query))[0]
video_path = result4[0]

matching_feature = next((f for f in film_features if f['film'] == video_path), None)

if matching_feature:
    start_time = matching_feature['second']
    end_time = start_time + 5

    clip = VideoFileClip(video_path).subclip(start_time, end_time)

    output_directory = "output_videos"
    os.makedirs(output_directory, exist_ok=True)
    output_path = os.path.join(output_directory, f"{base_name}_trimmed_video.mp4")
    clip.write_videofile(output_path, codec='libx264', fps=24)
    print(f"saved to: {output_path}")

    first_frame = clip.get_frame(0)
    plt.imshow(first_frame)
    plt.axis('off')
    plt.title("First Frame of Video")
    plt.show()

else:
    print(f"no `film_features` in {video_path} ")
```

```
Moviepy - Building video output_videos\ReinaSofia_1161_trimmed_video.mp4.
MoviePy - Writing audio in ReinaSofia_1161_trimmed_videoTEMP_MPY_wvf_snd.mp3

MoviePy - Done.
Moviepy - Writing video output_videos\ReinaSofia_1161_trimmed_video.mp4

Moviepy - Done !
Moviepy - video ready output_videos\ReinaSofia_1161_trimmed_video.mp4
saved to: output_videos\ReinaSofia_1161_trimmed_video.mp4
```

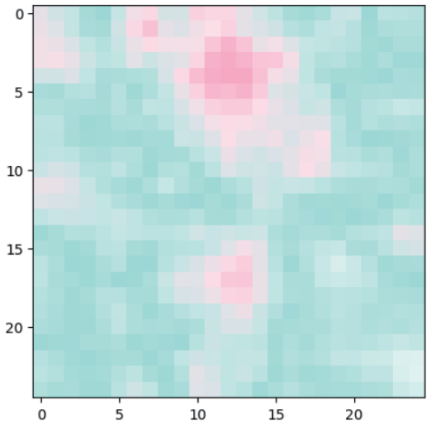
1.4 search together

Asked by ChatGPT randomly given, unabridged to test the results of the programme, because 10 examples too much in the pdf only show the first 2, the remaining results can be viewed in the github-Python_Skillwork_search from text file

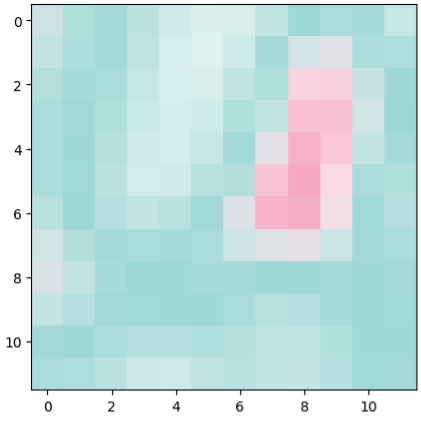
[297]:
```
query_texts = [
    "Shopping malls are bustling with people looking for the latest trends.",
    "Eating out at a new restaurant offers a taste of different cuisines.",
    "Buying fresh groceries at the local market is a weekly ritual for many.",
    "Traveling to exotic destinations provides a break from daily routines.",
    "Sports involve athletes training for competition with strategic teamwork.",
    "Scenic mountain views offer a breathtaking backdrop for hiking adventures.",
    "City parks are ideal for picnics and relaxing on a sunny day.",
    "Visiting museums and galleries is a popular weekend activity.",
    "Catching a movie at the cinema is a favorite pastime for families.",
    "Attending a live concert brings excitement to an otherwise quiet evening."
]


image_folder = r'D:\YeqinHUANG_Collection\ucl\Term3\Portfolio\week3\museumReinaSofia'
```

[298]:
```
def process_query_and_video(query, SOM1, DOC2VEC_MODEL, data_dict1, SOM3, DOC2VEC_MODEL2, data_dict3, featureTextPairs2, image_folder, searchv
    results = {}

    # Search SOM1
    result1 = search_TextSom1(SOM1, DOC2VEC_MODEL, data_dict1, query)
    print(result1)

    # Search SOM3
    result3 = search_TextSom3(SOM3, DOC2VEC_MODEL2, data_dict3, featureTextPairs2, query)
    filename3 = [item['filename'] for item in result3]
    image_path = os.path.join(image_folder, filename3[0])
    image = Image.open(image_path)
    plt.imshow(image)
    plt.axis('off')
    plt.show()

    # Search for video
    print(image_path)
    result4 = searchvideopicturesom4(image_path)

    base_name = os.path.splitext(os.path.basename(image_path))[0]
    video_path = result4[0]
    matching_feature = next((f for f in film_features if f['film'] == video_path), None)

    if matching_feature:
        start_time = matching_feature['second']
        end_time = start_time + 5
        clip = VideoFileClip(video_path).subclip(start_time, end_time)

        output_directory = "output_videos"
        os.makedirs(output_directory, exist_ok=True)

        output_path = os.path.join(output_directory, f"{base_name}_trimmed_video.mp4")
        clip.write_videofile(output_path, codec='libx264', fps=24)
        print(f"saved to: {output_path}")

        first_frame = clip.get_frame(0)
        plt.imshow(first_frame)
        plt.axis('off')
        plt.title("first frame of video")
        plt.show()
```

01 Input: "Shopping malls are bustling with people looking for the latest trends."

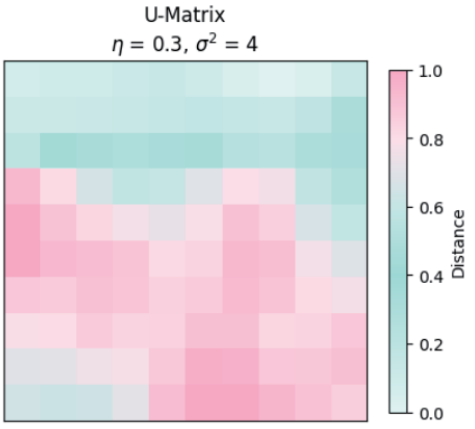



```
(4, 13)
[' exaltar t exalt; –do impassioned, passionate. examinar t examine. exánime lifeless. exceder t or de exceed, go beyond. excelente excellent, the best
of. exceptuar t except. excesivo excessive; adv. –mente.']
```




```
(5, 8)
Most similar content: In the image, there is a painting depicting a classical scene with men and women. The painting depicts a gathering of people, incl
uding nymphs and cherubs, in a park-like setting. The painting is surrounded by various vases and urns, adding to the aesthetic of the scene.

There are multiple
```

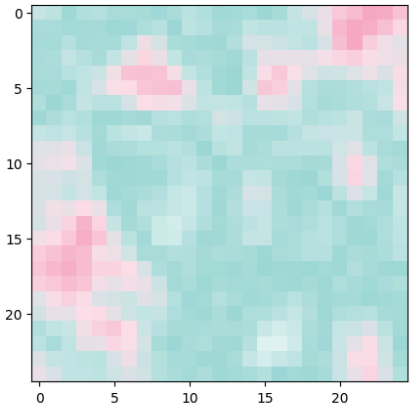
```
D:\YeqinHUANG_Collection\ucl\Term3\Portfolio\week3\museumReinaSofia\ReinaSofia_822.j
1/1 [==============================] - 0s 31ms/step
Query features shape: (1, 1024)
BMU location: (9, 6)
```



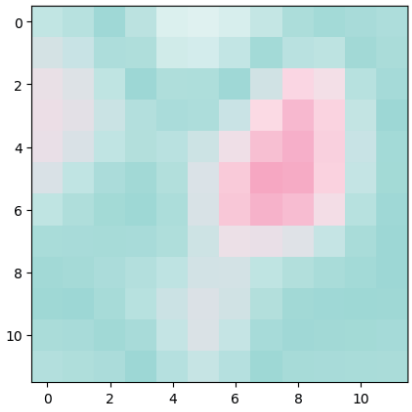

```
Moviepy - Building video output_videos\ReinaSofia_822_trimmed_video.mp4.
MoviePy - Writing audio in ReinaSofia_822_trimmed_videoTEMP_MPY_wvf_snd.mp3

MoviePy - Done.
Moviepy - Writing video output_videos\ReinaSofia_822_trimmed_video.mp4

Moviepy - Done !
Moviepy - video ready output_videos\ReinaSofia_822_trimmed_video.mp4
saved to: output_videos\ReinaSofia_822_trimmed_video.mp4
```

first frame of video

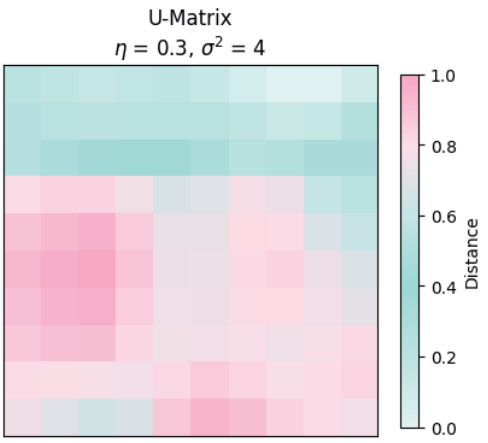
02Input: ""Eating out at a new restaurant offers a taste of different cuisines.""




```
(1, 21)
[' Gregory responded with a brief of June 28, 1583 in which he renewed the grant, at the same time reducing it to one-half of a prebend in Lisbon, Evora
and Coimbra and one-third in the other sees, nor is it likely that, under the stern rule of Philip, the grant was allowed to be nugatory.']
<Figure size 640x480 with 0 Axes>
```




```
(5, 7)
Most similar content: In the image, there are multiple objects depicting a group of people. The focal point is a painting of a group of men playing inst
ruments, including trumpets, trombones, and trombone. The painting is painted on a canvas and depicts a lively scene with dancing people. Additionally,
there are multiple people present in the painting
```




```
Moviepy - Building video output_videos\ReinaSofia_1045_trimmed_video.mp4.
MoviePy - Writing audio in ReinaSofia_1045_trimmed_videoTEMP_MPY_wvf_snd.mp3

MoviePy - Done.
Moviepy - Writing video output_videos\ReinaSofia_1045_trimmed_video.mp4

Moviepy - Done !
Moviepy - video ready output_videos\ReinaSofia_1045_trimmed_video.mp4
saved to: output_videos\ReinaSofia_1045_trimmed_video.mp4
```

first frame of video

2.0 Input from picture

2.1 Picture to picture

2.1.1 Process Museum Images

```
model = 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'
)
```

Use TensorFlow's MobileNet model to extract features from a series of images and combine those features with information such as image descriptions

```
featureImagePairs6[0]
```

```
{'filename': 'ReinaSofia_0.jpg',
 'path': 'D:\\YeqinHUANG_Collection\\ucl\\Term3\\Portfolio\\week3\\museumReinaSofia\\ReinaSofia_0.jpg',
 'content': 'In the image, there are various objects, including abstract art, a painting, and a piece of furniture. The artwork is painted on a canvas a
nd features a gray and yellow color scheme. The painting is surrounded by various objects, including chairs and a table. The chairs are placed around th
e painting, with one chair close to the painting and another chair further away',
 'file_index': 0,
 'feature': array([0.18213311, 1.8348762 , 0.        , ..., 0.        , 0.41940853,
        0.        ], dtype=float32)}
```

2.1.2 Train Images SOM

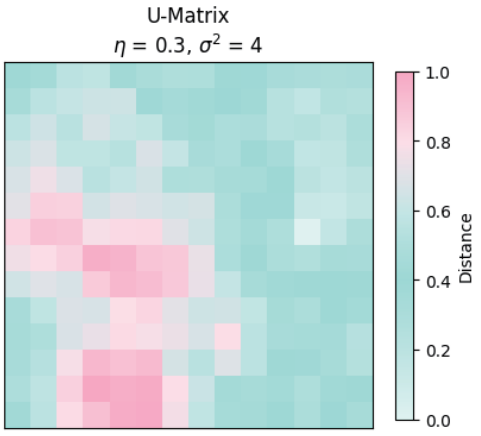
```
generate_SOMimages_grid(SOMimages)
```

2.1.3 Search

By matching the image, we can get the relevant description of the image to be used for image-to-text matching.

[63]:
```
query = r'D:\YeqinHUANG_Collection\ucl\Term3\skill\python\database02_photo\museumReinaSofia\Buque en reparación (Ship Under Repair).jpg'
result5 = searchvideopicturesom5(query)
```

```
1/1 [==============================] - 0s 489ms/step
Query features shape: (1, 1024)
BMU location: (12, 3)
```




```
result5
```

```
[{'image': 'D:\\YeqinHUANG_Collection\\ucl\\Term3\\Portfolio\\week3\\museumReinaSofia\\ReinaSofia_346.jpg',
  'path': 'D:\\YeqinHUANG_Collection\\ucl\\Term3\\Portfolio\\week3\\museumReinaSofia\\ReinaSofia_346.jpg',
  'content': 'The image features a cityscape with buildings and spires, surrounded by mountains.\n\nIn the scene, there are multiple buildings, includin
g a large city with spires, a castle, and several other smaller buildings. Additionally, there are spires in various locations throughout the picture. T
he mountainous landscape surrounding the city is covered in clouds, creating a pictures'}]
```

```
import matplotlib.pyplot as plt
from PIL import Image

def show_query_and_results(query_image_path, result_image_paths):
    image_paths = [query_image_path] + [item['image'] for item in result_image_paths]
    titles = ["Query Image"] + [f"Result {i+1}" for i in range(len(result_image_paths))]

    # 确定布局，按行数显示
    n_images = len(image_paths)
    fig, axes = plt.subplots(1, n_images, figsize=(5 * n_images, 5))

    if n_images == 1:
        axes = [axes]

    for ax, path, title in zip(axes, image_paths, titles):
        try:
            # 使用 PIL 加载图像
            image = Image.open(path)
            ax.imshow(image)
            ax.axis('off')
            ax.set_title(title)
        except Exception as e:
            ax.set_title(f"Error: {e}")
            ax.axis('off')

    plt.tight_layout()
    plt.show()

show_query_and_results(query, result5)
```

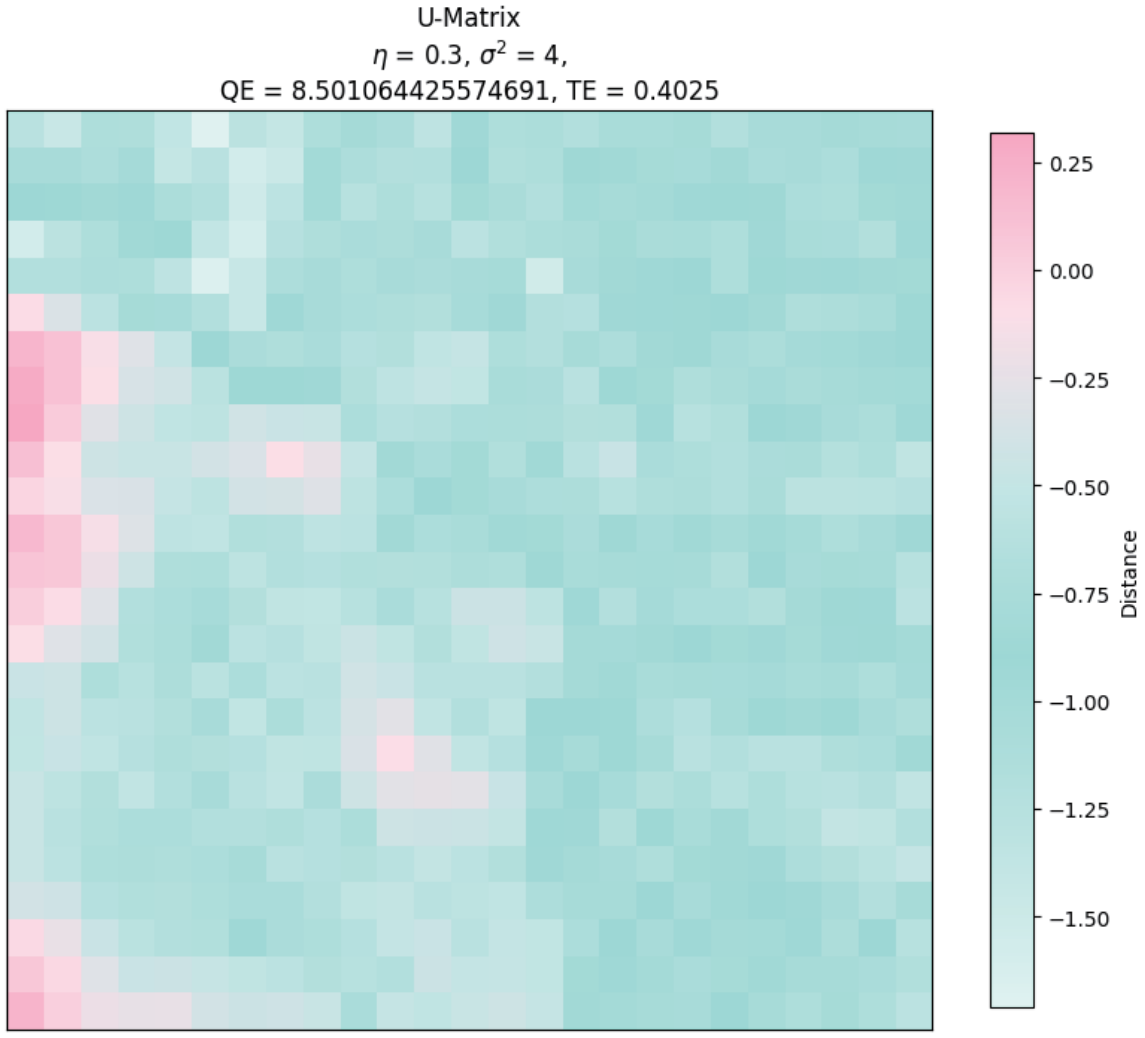
2.2 Picture to text

2.2.1 Import previously trained Text SOM

[68]:
```
with open('Doc2Vec_txtBookSOM.pkl', 'rb') as f:
    SOM1 = pickle.load(f)
```

[70]:
```
input_file_path = "train_data1.pkl"
with open(input_file_path, "rb") as f:
    train_data1 = pickle.load(f)
```

[72]:
```
u_matrix_values = u_matrix(SOM1)
QE = round(calculateQE(SOM1, train_data1), 250)
TE = round(calculateTE(SOM1, train_data1), 250)

plt.figure(figsize=(10, 8))
im = plt.imshow(u_matrix_values, cmap=cmap, 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()
```



2.2.2 Search

By using the image description obtained from the matching in the previous step as input for the text SOM, I achieved the goal of going from image to text.

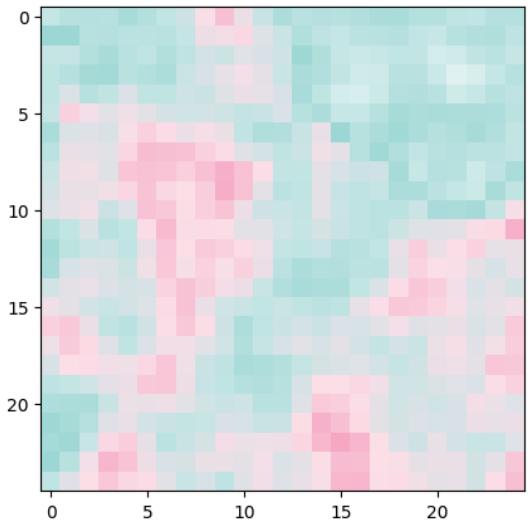
```
def search_TextSom1(SOM1, model, data_dict1, query='street'):
    result = []

    # 处理查询
    query = [query]
    preprocessed_query = preprocess2(query)
    query_vector = DOC2VEC_MODEL.infer_vector(preprocessed_query[0])

    # 激活并显示 SOM
    fig = plt.figure()
    plt.figure(figsize=(5, 5))
    plt.imshow(activate1(train_data1, SOM1, query_vector), cmap=cmap)
    plt.show()

    # 查找 BMU 位置
    g, h = find_BMU1(SOM1, query_vector)
    print((g, h))

    # 检查 BMU 单元是否为空
    if g >= len(data_dict1) or h >= len(data_dict1[g]) or not data_dict1[g][h]:
        print("No data found for this BMU.")
        return []

    # 计算相似度并找到最相似的段落
    similarities = {}
    for i in data_dict1[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]])[0][0]

    # 找到最相似的段落
    if similarities:
        biggest = max(similarities, key=similarities.get)
        result.append(biggest)
    else:
        print("No similarities found.")

    return result
```

```
contents
```

```
['The image features a cityscape with buildings and spires, surrounded by mountains.\n\nIn the scene, there are multiple buildings, including a large ci
ty with spires, a castle, and several other smaller buildings. Additionally, there are spires in various locations throughout the picture. The mountaino
us landscape surrounding the city is covered in clouds, creating a pictures']
```

```
search_TextSom1(SOM1,DOC2VEC_MODEL,data_dict1,contents[0])
```

```
<Figure size 640x480 with 0 Axes>
```




```
(22, 15)
```

```
['[140] This provision, intended for the protection of the youthful and incapable, was retained in the practice of the Inquisition, because it was neces
sary to render valid the various compulsory acts of the accused in the successive steps of his trial, but in order that it might not by any chance be of
value to him, and to preserve the secrecy of the Holy Office, the custom was adopted of appointing the advocate or preferably the gaoler, or messenger,
or some other underling of the tribunal to serve as curador.']
```

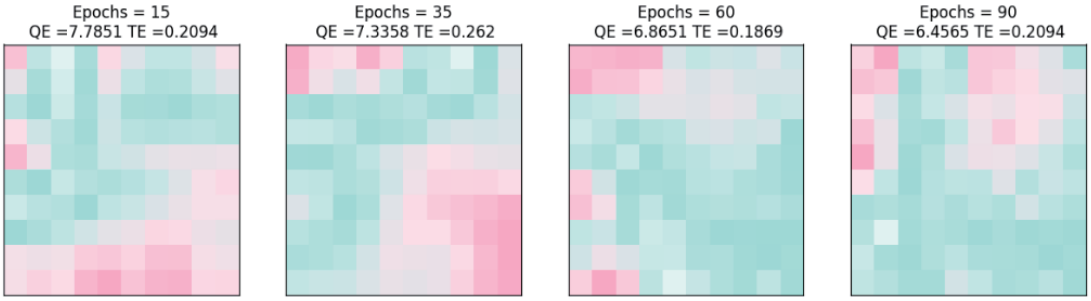
2.3 Picture to Video

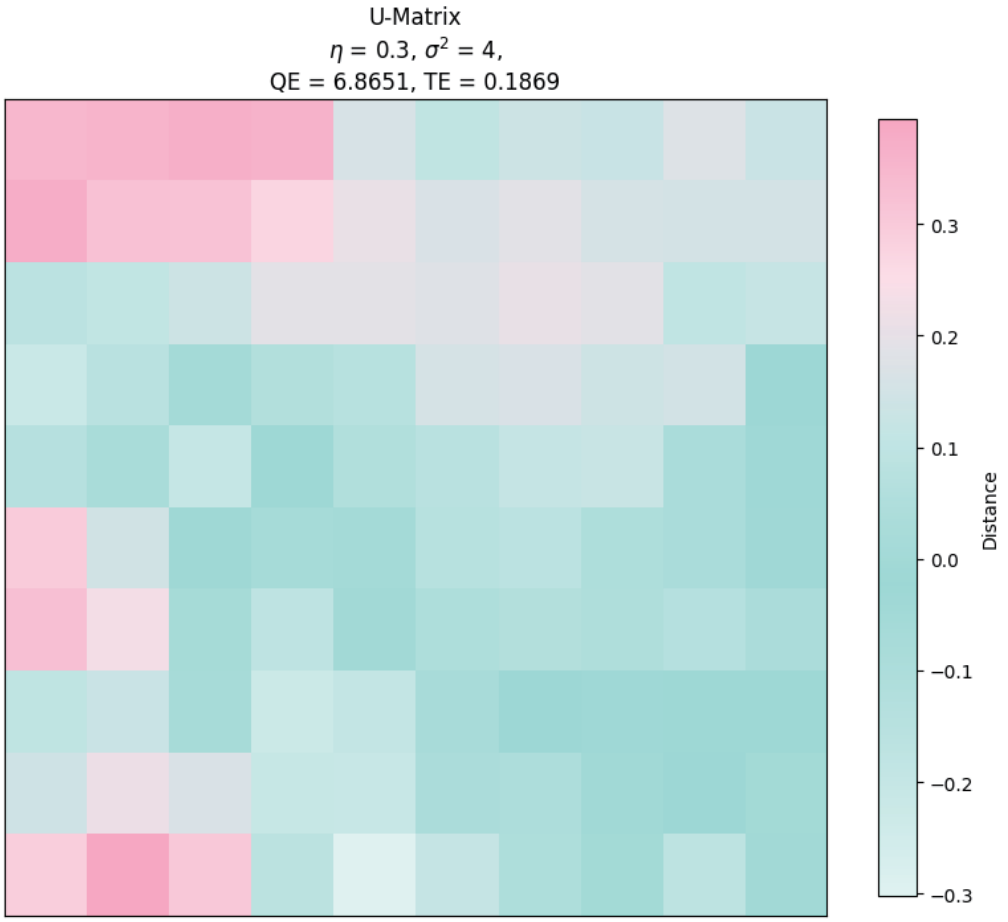
2.3.1 retrain Video SOM

Since it was previously found that the matching video did not change much, the SOM was re-trained with frame pictures as a check, but it was found to be due to too few video frames, which is explained here

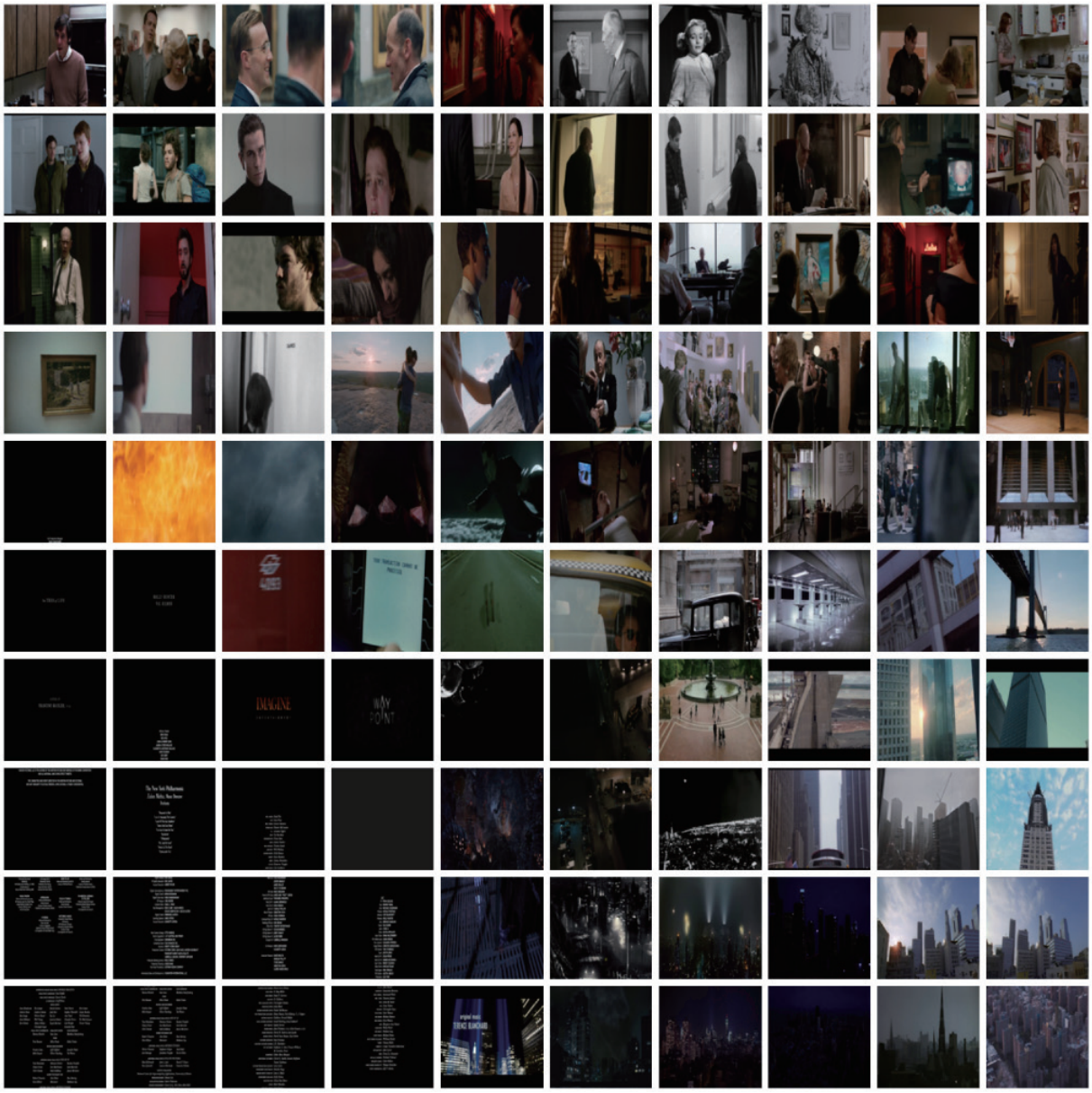
```
from matplotlib.colors import LinearSegmentedColormap
total_epochs = 0
SOMS7 = []
vector_size = 1024
fig, ax = plt.subplots(
    nrows=1, ncols=4, figsize=(15, 3.5),
    subplot_kw=dict(xticks=[], yticks=[]))

for epochs, i in zip([15, 20, 25, 30], range(0,4)):
    print(i)
    total_epochs += epochs
    rand = np.random.RandomState(0)
    SOM7 = rand.uniform(0,1,(m,n,vector_size))
    SOM7 = train_SOM(SOM7, n_train_data7, learn_rate = .5, radius_sq = 7, epochs = epochs)
    SOMS7.append(SOM7)
    QE = round(calculateQE(SOM7, n_train_data7), 4)
    TE = round(calculateTE(SOM7, n_train_data7), 4)

    ax[i].imshow(u_matrix(SOM7),cmap=cmap,aspect='auto')
    ax[i].title.set_text('Epochs = ' + str(total_epochs) + '\n QE =' + str(QE) + ' TE =' + str(TE))
```

```
0
1
2
3
```





```
generate_SOMimages_grid(SOMimages2)
```



2.3.2 Search

```
Path
```

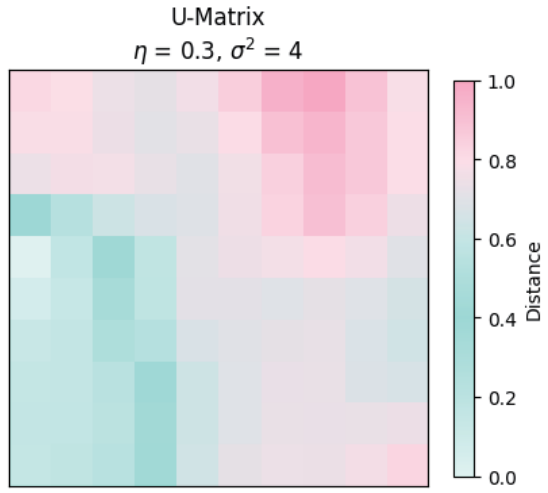
```
['D:\\YeqinHUANG_Collection\\ucl\\Term3\\Portfolio\\week3\\museumReinaSofia\\ReinaSofia_943.jpg']
```

```
path
```

```
'video_96_15s.jpg'
```

```
query = Path[0]
result7 = searchvideopicturesom7(query)
```

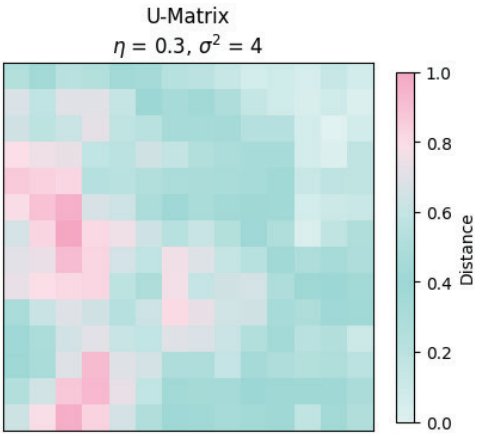
```
1/1 [==============================] - 0s 28ms/step
Query features shape: (1, 1024)
BMU location: (0, 7)
```

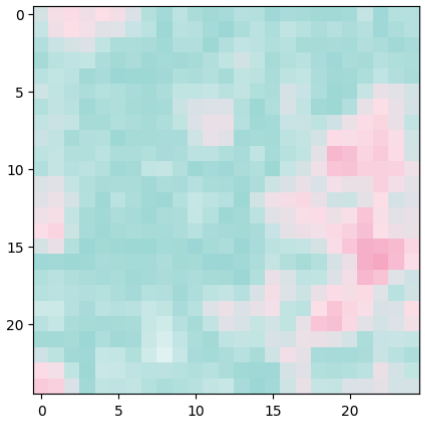



```
def show_query_and_results(query_image_path, result_image_paths):
    image_paths = [query_image_path] + result_image_paths
    titles = ["Query Image"] + [f"Result {i+1}" for i in range(len(result_image_paths))]

    n_images = len(image_paths)
    fig, axes = plt.subplots(1, n_images, figsize=(5 * n_images, 5))

    if n_images == 1:
        axes = [axes]

    for ax, path, title in zip(axes, image_paths, titles):
        try:
            image = img_reshape(path)
            ax.imshow(image)
            ax.axis('off')
            ax.set_title(title)
        except Exception as e:
            ax.set_title(f"Error: {e}")
            ax.axis('off')

    plt.tight_layout()
    plt.show()

show_query_and_results(query, result7)
```

2.4 search together

Asked by ChatGPT randomly given, unabridged to test the results of the programme, because 10 examples too much in the pdf only show one here, the remaining results can be viewed in the github-Python_Skillwork_search from picture file.





(16, 22)

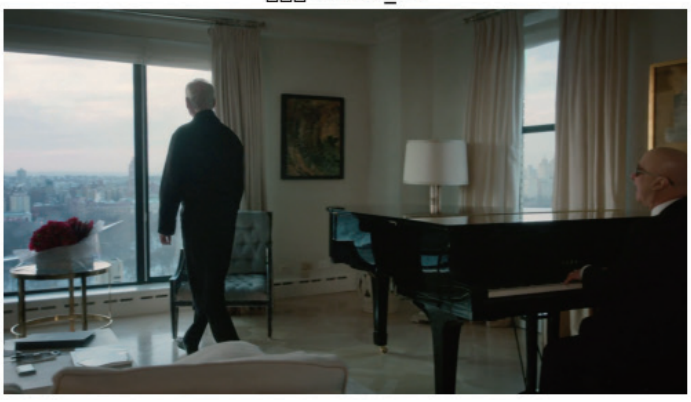
Query Image

Result 1

```
['The image features a painting of a man riding a horse and a crowd of people standing around. The painting captures the scene with a man fal
ling off his horse, seemingly in the middle of a protest or demonstration.\n\nIn addition to the man and horse, there are multiple people pre
sent in the scene, some standing closer to the horse while others are farther away. The crowd is dispersed throughout the']
<Figure size 640x480 with 0 Axes>
```

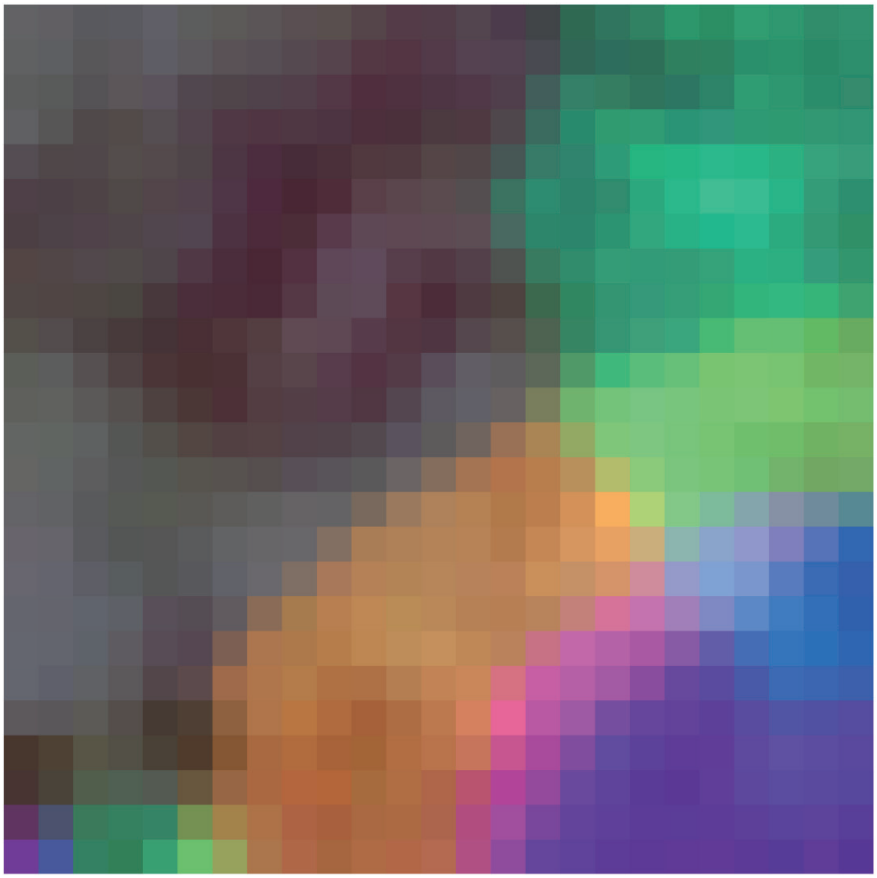
```
[' In the centre is a huge cross; the flagstones of the court were all up, and the bones from many disturbed graves were being thrown into a
pit. The beautiful cloisters proper are filled with modern opaque glass—"Muy frio" answered the verger to my question, "Por que?"—and no doub
t it is in the winter months.']
```

□□□: video_31

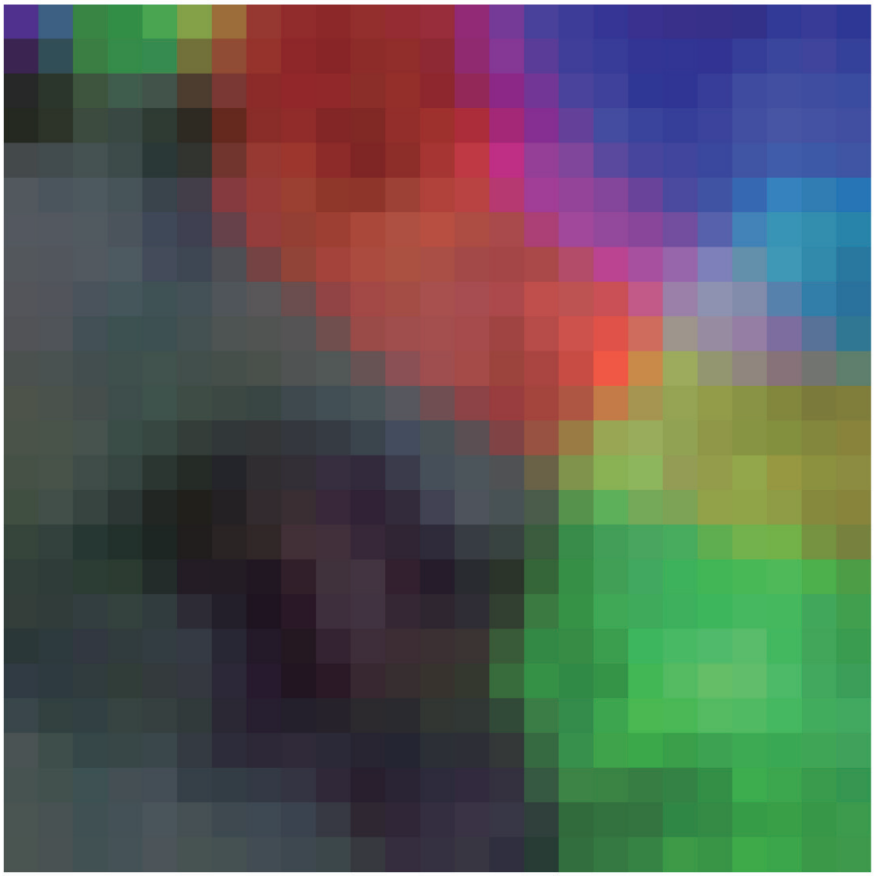


```
Moviepy - Building video D:\YeqinHUANG_Collection\ucl\Term3\skill\python\output_videos\video_31_10s_to_15s_extracted.mp4.
MoviePy - Writing audio in video_31_10s_to_15s_extractedTEMP_MPY_wvf_snd.mp3

MoviePy - Done.
Moviepy - Writing video D:\YeqinHUANG_Collection\ucl\Term3\skill\python\output_videos\video_31_10s_to_15s_extracted.mp4
```

3.1 Fit PCA on all cells of the SOM



3.1 Fit PCA on the entire training set



Fitting PCA to all SOM cells mainly analyses the main features of each local cluster, whereas fitting PCA to the whole training set analyses the main direction of variation in the overall data, and thus the two will be different.

02 Houdini:

OneDrive Link

Assignment 01

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvyh4_ucl_ac_uk/EvY42PAag8BFjvgJCosGQSMB2eIr24AkUiFCtNeLDJ5Z4g?e=dRcM0h

Assignment 02

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvyh4_ucl_ac_uk/EjJQnO81AJJEsrPa3B5UnicBVY8v-aFw0-UxvTOmX-5FcQ?e=r2j8pd

Assignment 03

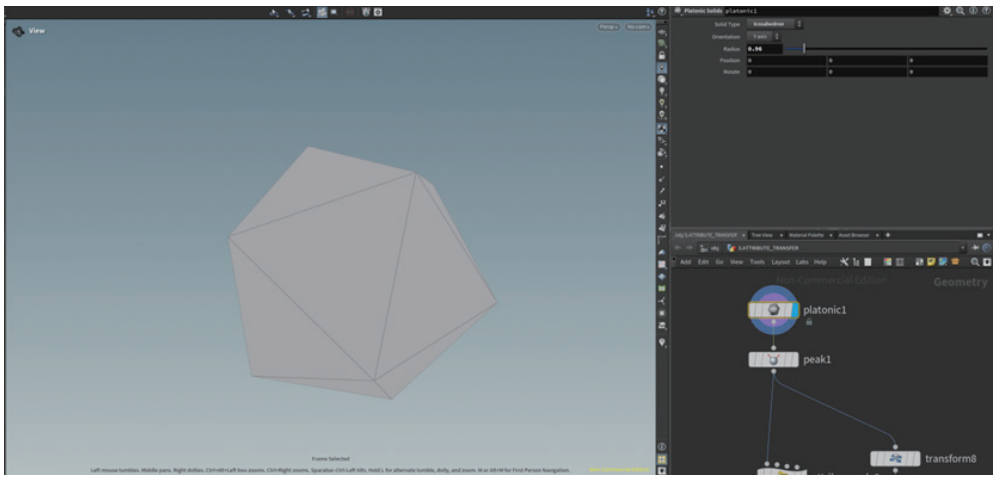
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvyh4_ucl_ac_uk/Emtq1NOdCDRKnjVgApPNXxQBXjJKaau9at7iL34lKPAOsQ?e=gifXuj

Assignment 04

https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvyh4_ucl_ac_uk/EvkkKVk6rxVNjN81P5W28VgBfAKtDrFwtVBg-xDjpSjzVQ?e=5WcNsS

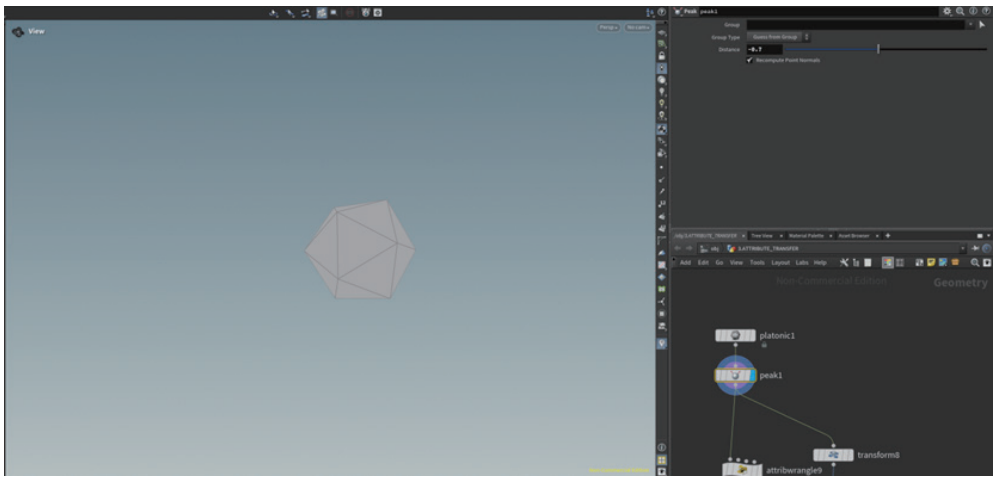
1.1.1 Platonic Solids SOP

The Platonic Solids SOP produces various types of platonic solids. Platonic solids are polyhedrons which are convex and have all the vertices and faces of the same type. There are only five such objects, which form the first five choices of this operation. This node can create seven different polyhedral forms such as Utah Teapot and Tetrahedron, I chose to create a Icosahedron.



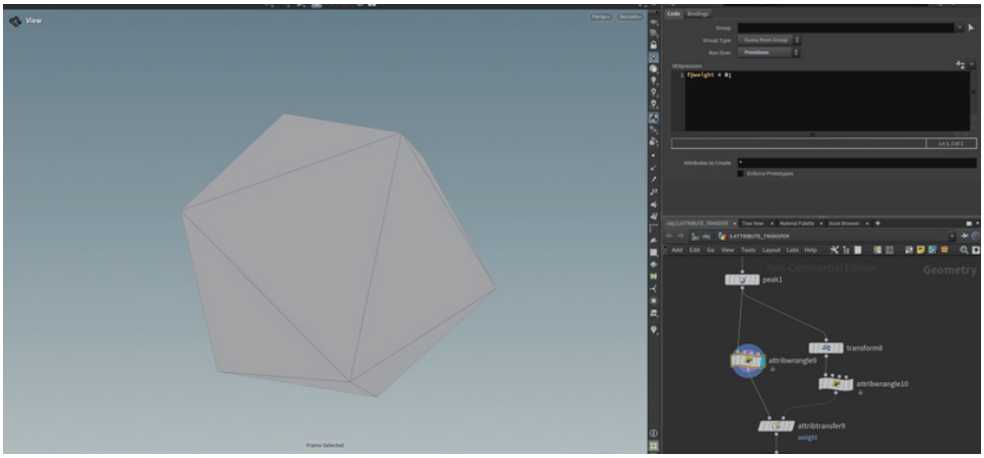
1.1.2 Peak SOP

This node modifies the platonic solid by scaling along its surface normals. The settings in the right-hand panel show the "Distance" value set to -0.7, meaning it contracts or inflates the geometry inwards.



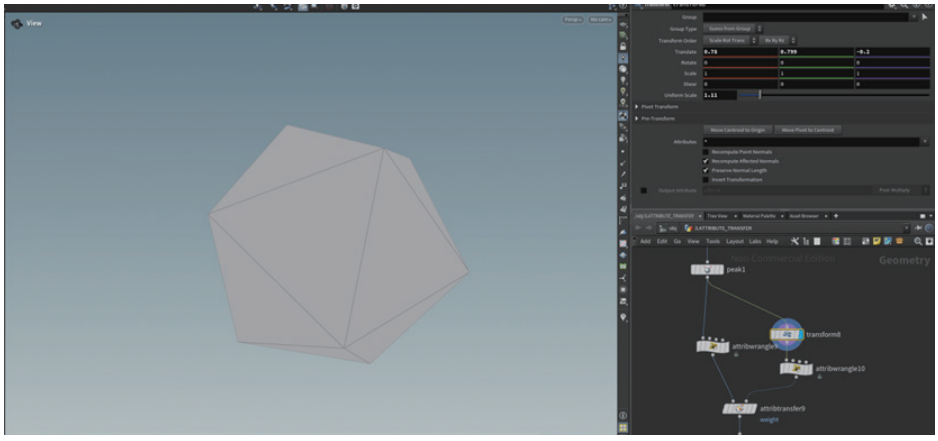
1.1.3 Attribute Wrangle SOP

I create an “Attribute Wrangle” SOP, where I make a “weight” attribute. The attribute is of type float as described by the “f@” declaration. The attribute is set to run over primitives. This is because later in the node setup, I will use a Polyextrude node where we will need a primitive attribute to drive the extrusion value. The value is set to 0 (which will be interpreted as 0.0 because of the float declaration) because later, I can interpolate between 0 and 1.



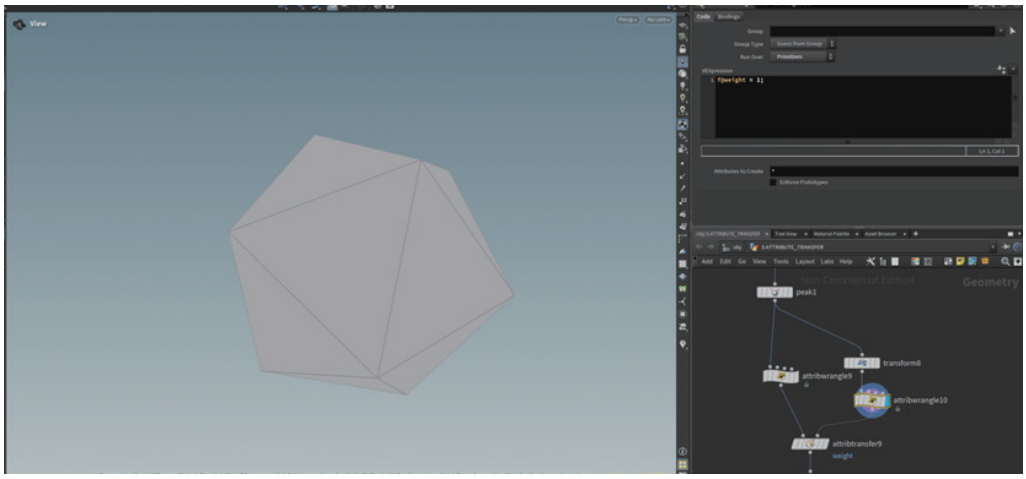
1.1.4 Transform SOP

Then, I lay down a “Transform” SOP. This will be the attractor; it is just a duplicate of the original sphere. The distance between this duplicate (attractor) and the original sphere will dictate the final extrusion value.



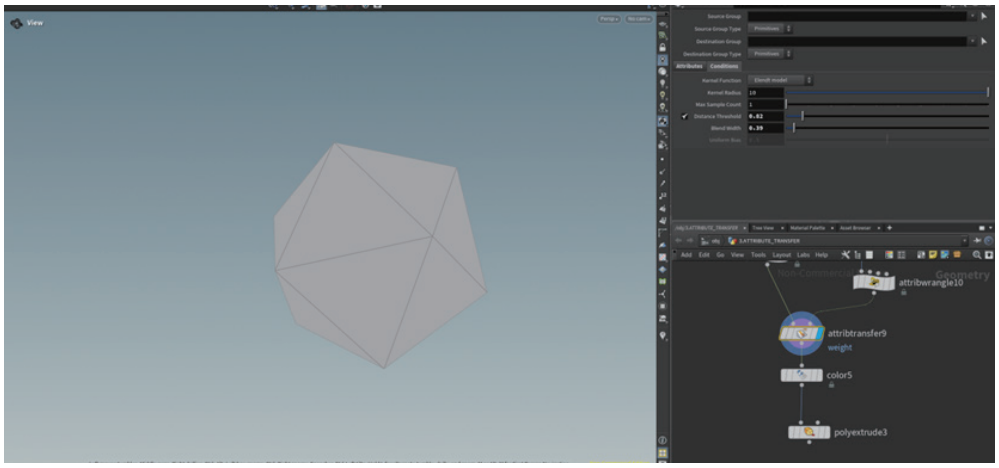
1.1.5 Attribute Wrangle SOP

The same as the previous Wrangle SOP, only with the opposite value. The reason I interpolate between 0 and 1 is because it is easier to work with normalized values than arbitrary values.



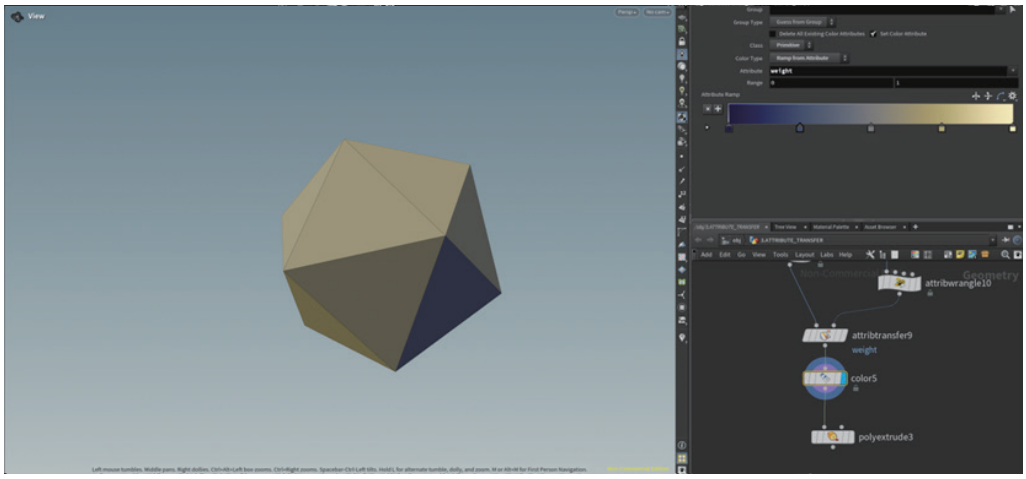
1.1.6 Attribute Transfer SOP

The “Attribute Transfer” SOP uses the “weight” primitive attribute to write out the distance value between the original sphere and the attractor. By setting the original “weight” attribute as float, I can have value interpolations between 0 and 1. If I had set the weight attributes to integers, I could not interpolate; the output values would be binary (0 or 1).



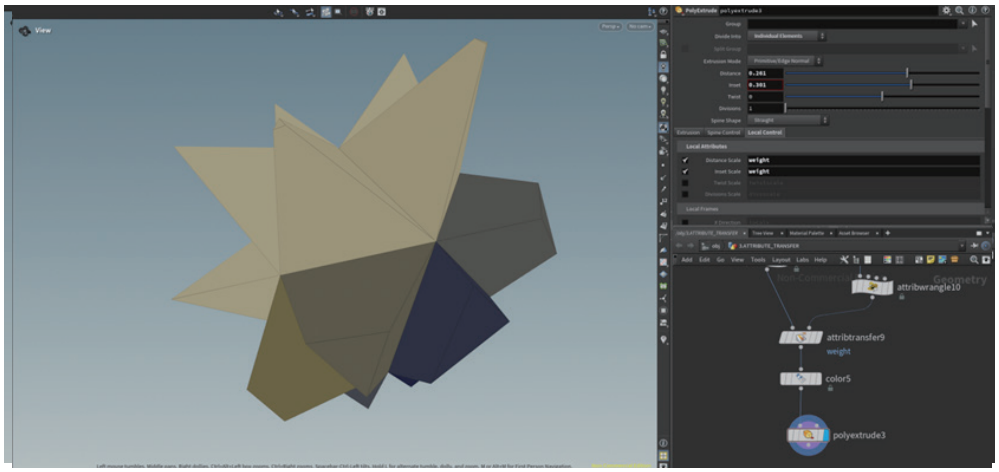
1.1.7 Color SOP

A visualisation of the “weight” primitive attribute according to the “viridis” colorscheme. Because I normalized my values between 0 and 1, it is more stable if I would ever change distances, or geometry inputs.



1.1.8 PolyExtrude SOP

Using the “weight” value in d”distance scale”, I use the weight value as a multiplier of the extrusion value. If I set the distance value to 0, the extrusion is 0 (0 x weight). If I set the distance value to 1, I get the full weight value as extrusion (1 x weight)
Distance: This parameter is set to 0.261, which defines the extrusion depth outward or inward.
Inset: This value is 0.301, which creates an inward offset (inset) or outward expansion (outset) of the extruded faces.

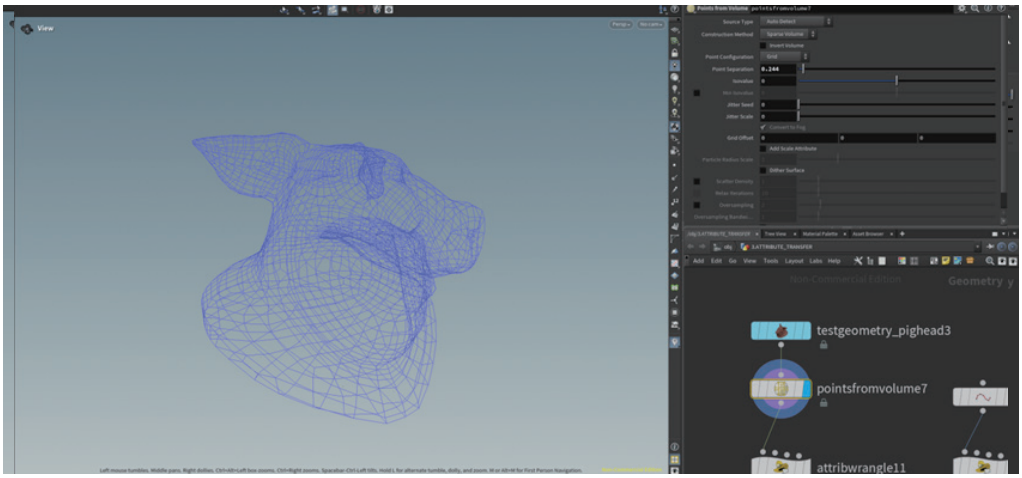


1.2.1 testgeometry_pighead

I create a pig head, which can be used as test geometry. In Houdini, the "testgeometry_pighead" node is one of several built-in test geometry nodes provided by the software. These nodes generate standardized 3D models that are commonly used for testing, learning, and experimenting with different workflows and techniques.

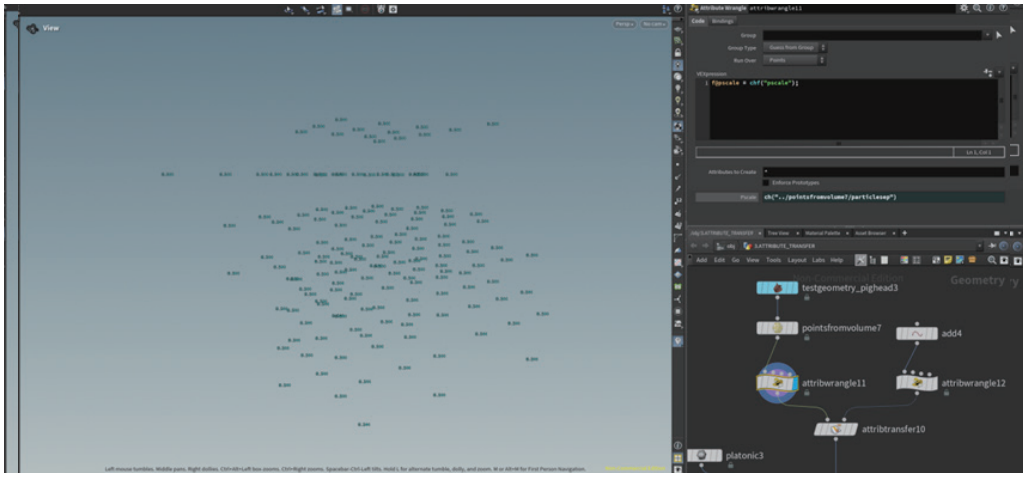
1.2.2 pointsfromvolume

This operator is used to generate a regular set of points that fill a given volume. This node is particularly useful for extracting information from complex 3D volume data and is commonly used in simulation, rendering, and geometry generation. I adjusted the point separation, and increasing this value will generate fewer total points.



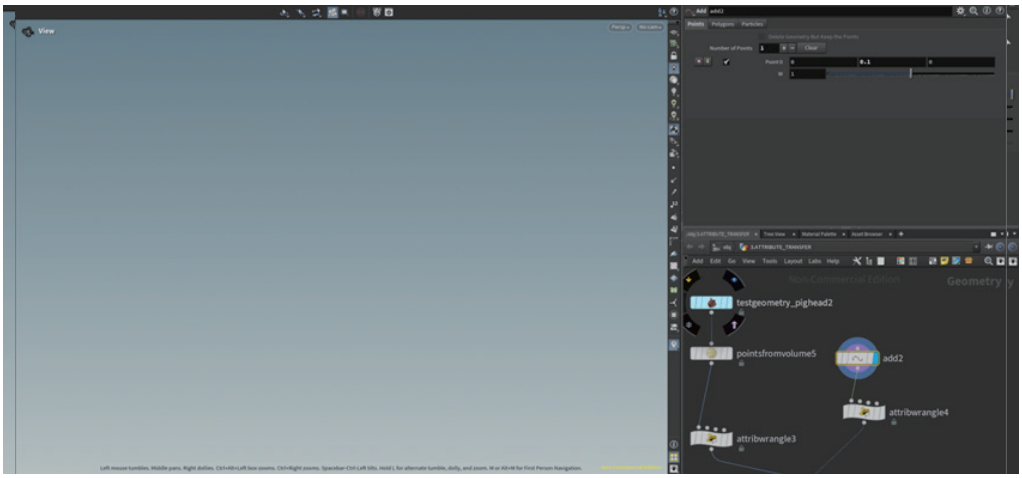
1.2.3 testgeometry_pighead

This Attribute Wrangle node is primarily set up to manipulate attributes on points using VEX code. The VEXpression code defines a simple formula for scaling attributes. The expression f@pscale = chf("pscale"); is used to assign a value to the pscale attribute, which controls point scaling in Houdini. The chf function retrieves the value from a parameter slider named "pscale" (likely a float slider). The value is sourced from ../pointsfromvolume7/particlesep, which references a parameter called particlesep in the pointsfromvolume7 node.



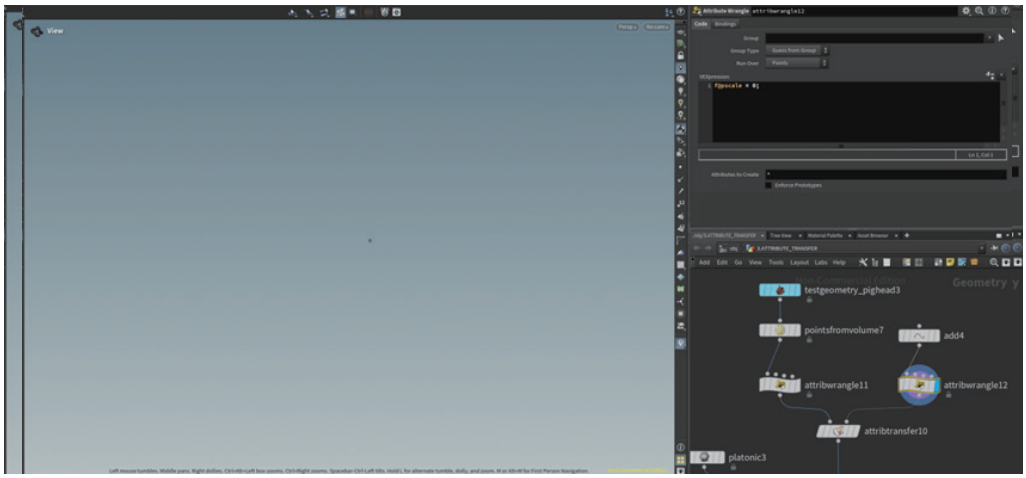
1.2.4 add

I create an “add” node, the primary function is to create points. I can specify the number of points, their positions, and other attributes through the interface. Point 0: This section allows specific settings for an individual point. The position is set to (0, 0.1, 0), indicating the point’s coordinates in 3D space. The W value is set to 1, which is typically used in a homogeneous coordinate system.



1.2.5 attribwrangle

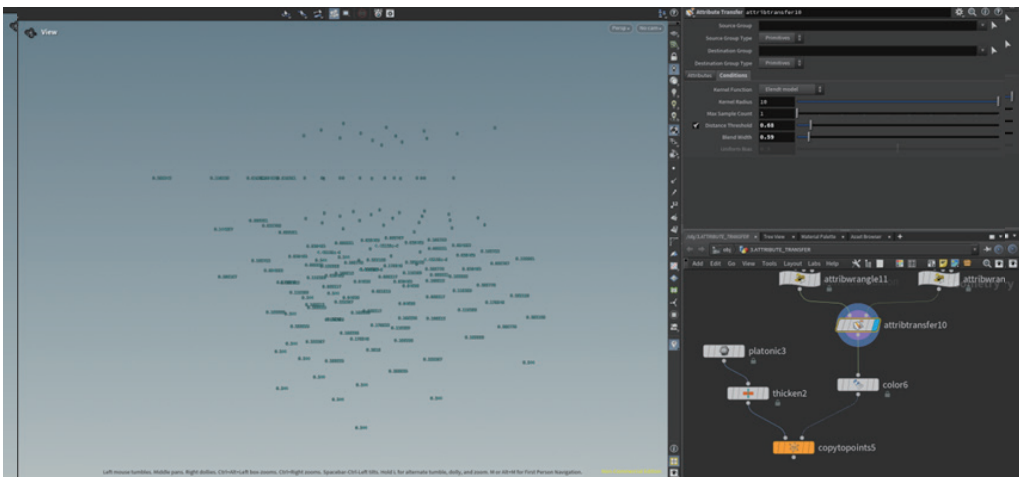
In the Attribute Wrangle node setup, I did a simple attribute manipulation based on the VEX programming language. The VEX code reads f@pscale = 0;. This assigns the value 0 to the pscale attribute for each point. The prefix f@ indicates that the attribute pscale is a float type. As pscale determines the point scale in Houdini, setting it to zero effectively scales down all points to invisible sizes. The attribwrangle node is connected to a series of upstream nodes, including pointsfromvolume, which generates points inside the volume of the pig head.



1.2.6 attribtransfer

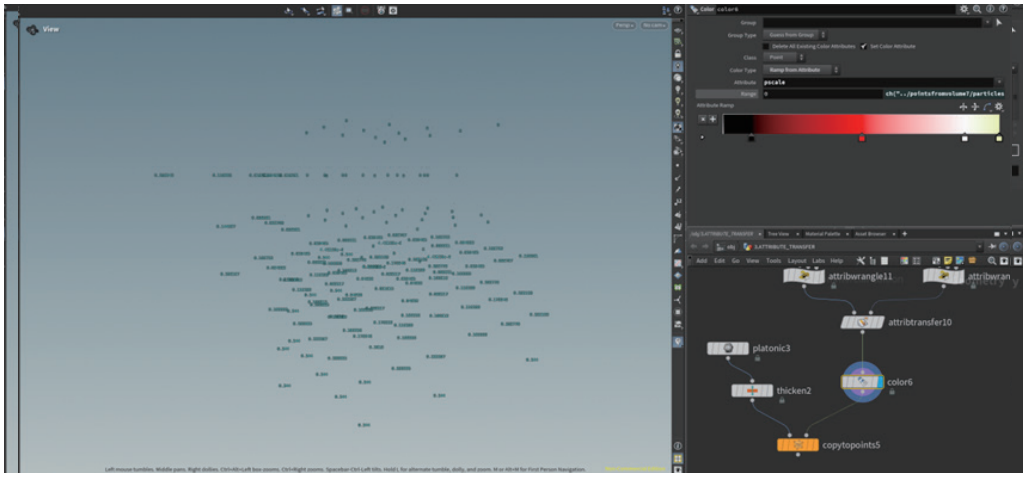
This node transfers specific attributes from one geometry to another based on proximity. It's typically used to transfer color, normals, or custom attributes between geometries.

Kernel Function: The blending model used to transfer the attributes is set to "Blend Model."
Kernel Radius: The search radius used to identify nearby source geometry primitives is set to 10.
Max Sample Count: Limits the number of nearby primitives considered for each transfer, currently set to 1.
Distance Threshold: Only transfers attributes if the source geometry is within this distance to the target. The threshold is 0.68.
Blend Width: Controls the blending between multiple attributes; set to 0.59.



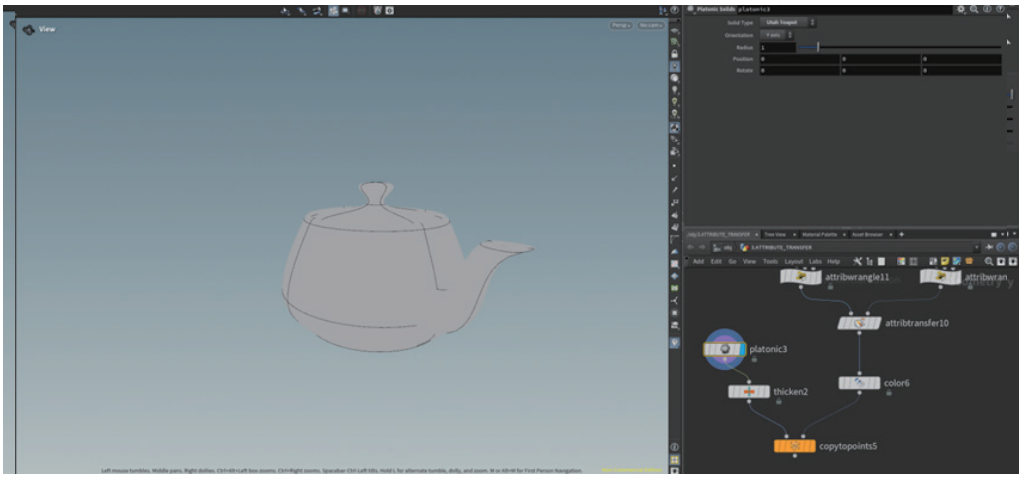
1.2.7 color

This “Color” node assigns a gradient of colors to the points based on their scale, allowing visualization of the attribute's values across the geometry. Colour type I chose “Ramp from Attribute”: The color type is set to use a color ramp based on an attribute, meaning the gradient of colors will change according to the values of a specific attribute.



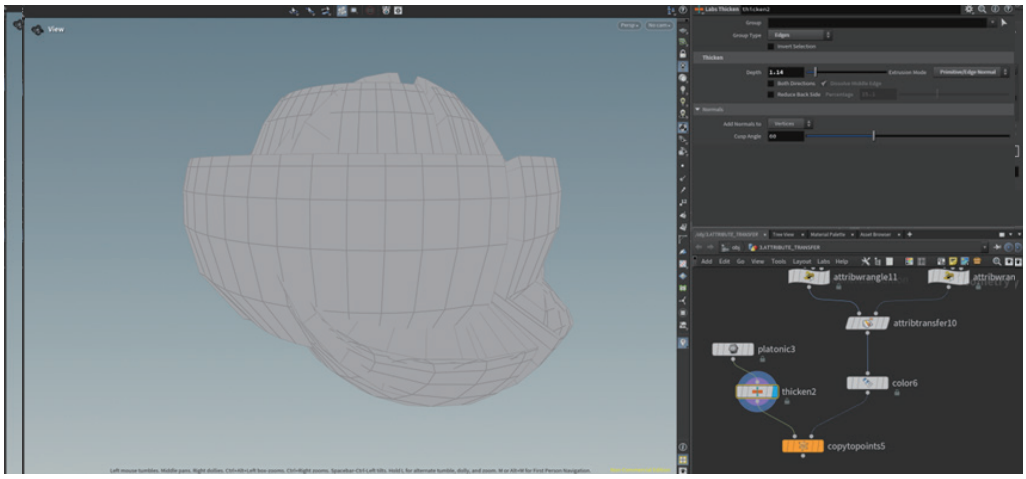
1.2.8 platonic

The “Platonic Solids” node is used to generate various Platonic solids, I chose to create a “Utah Teapot", and I set the radius to 1 to define the overall dimensions of the teapot model.



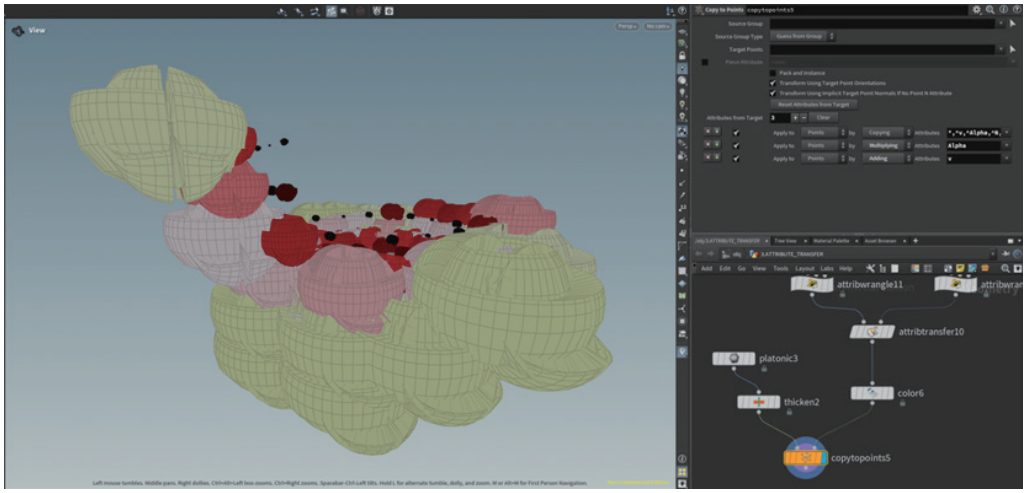
1.2.9 thicken

the Thicken node adds depth to a geometry based on the defined parameters. I set the depth parameter to 1.14 to control the thickness of the geometry. The higher the value, the thicker the surface extrusion. The node receives input from the platonic node, which generates a Utah Teapot model.

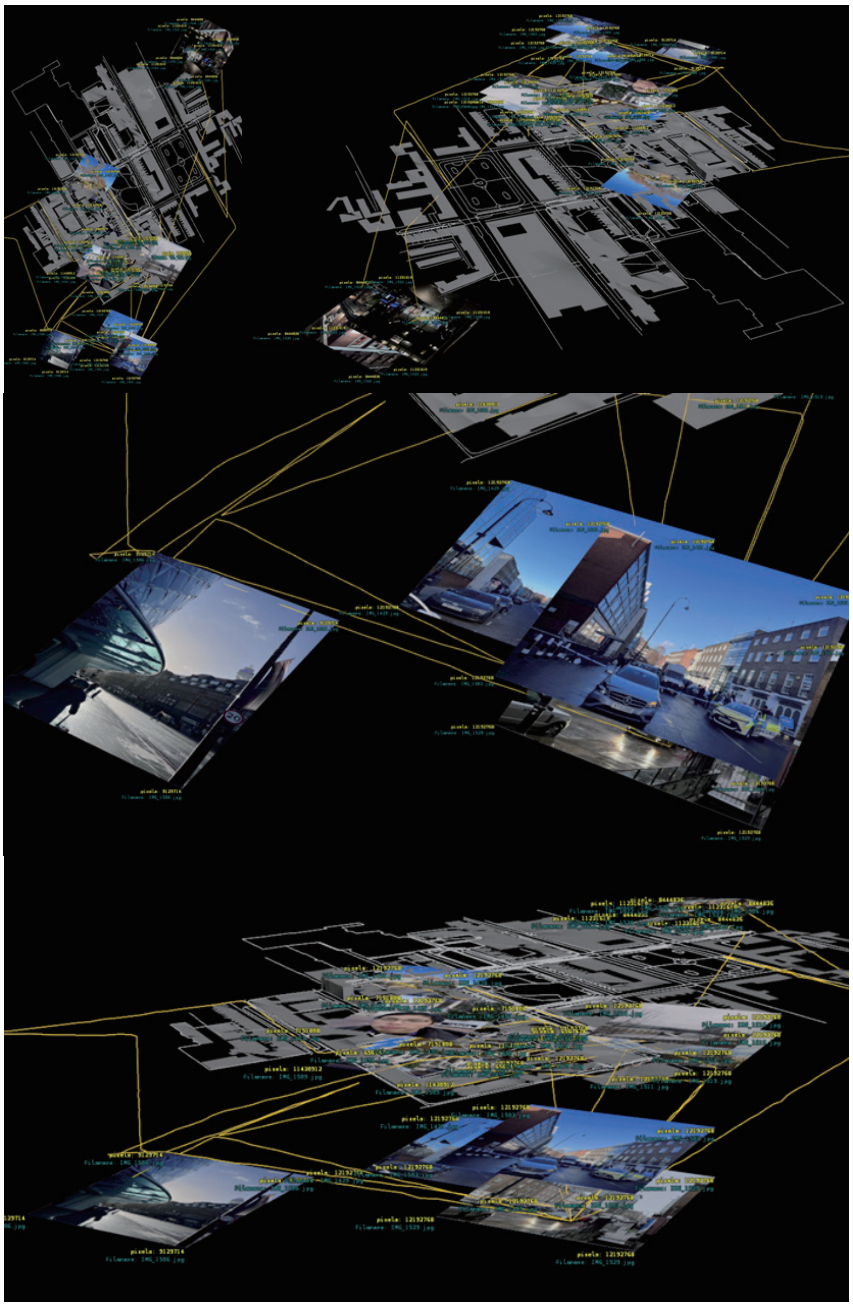


1.2.10 copytopoints

The “Copy to Points” node setting is used to copy the source geometry to the target points while allowing properties to be carried and modified while copying. This node takes the thickened, colored teapot geometry and copies it to points generated and modified by the above nodes.



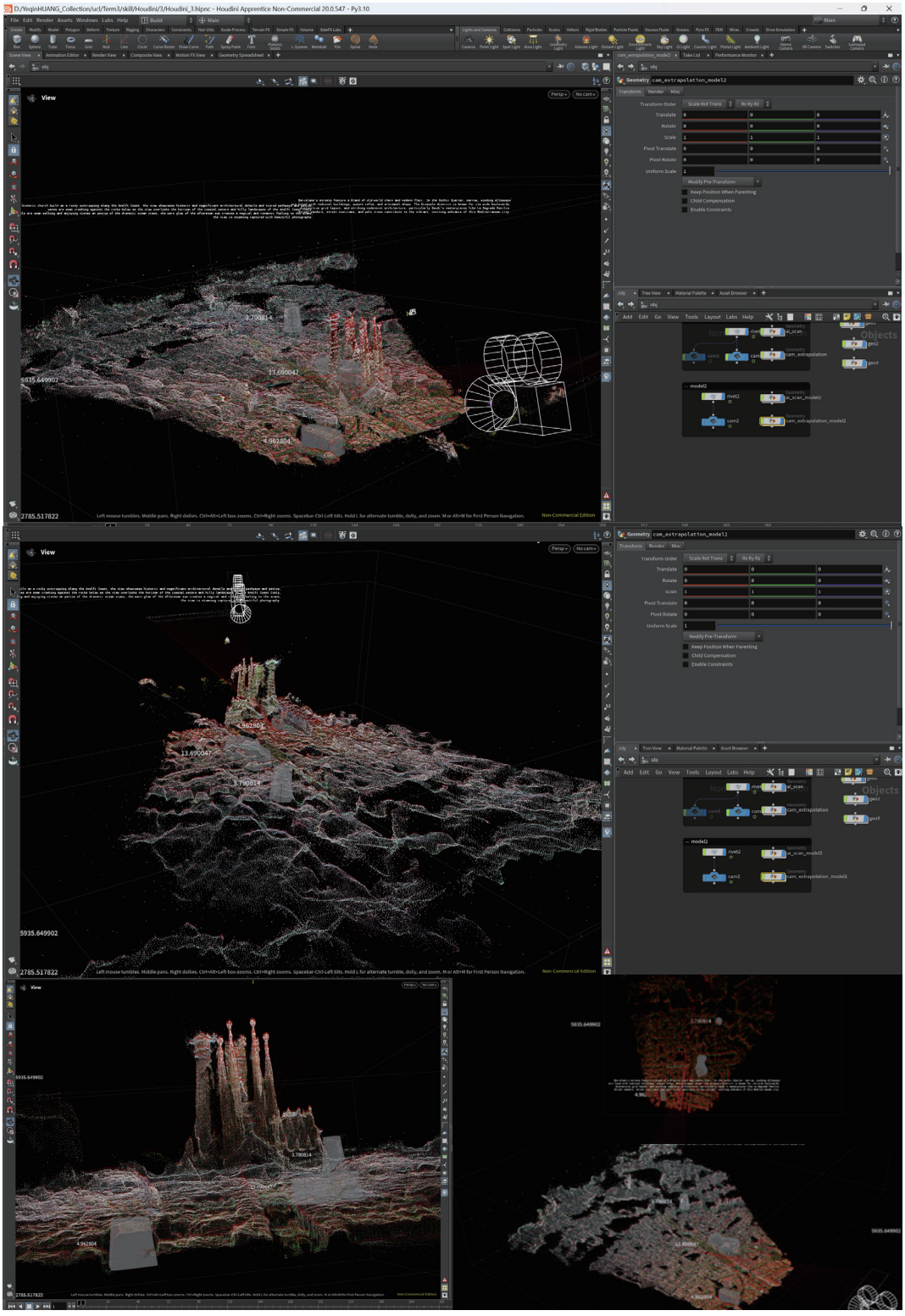
2 Visualizing GPS and Image Metadata



First, I exported a KML file from Google Maps, then converted the KML file to a GPX file and imported it into Houdini. Next, I downloaded an OSM file from OpenStreetMap. Finally, I imported pictures with location information from my mobile phone into Houdini, creating a series of images depicting my travel route.

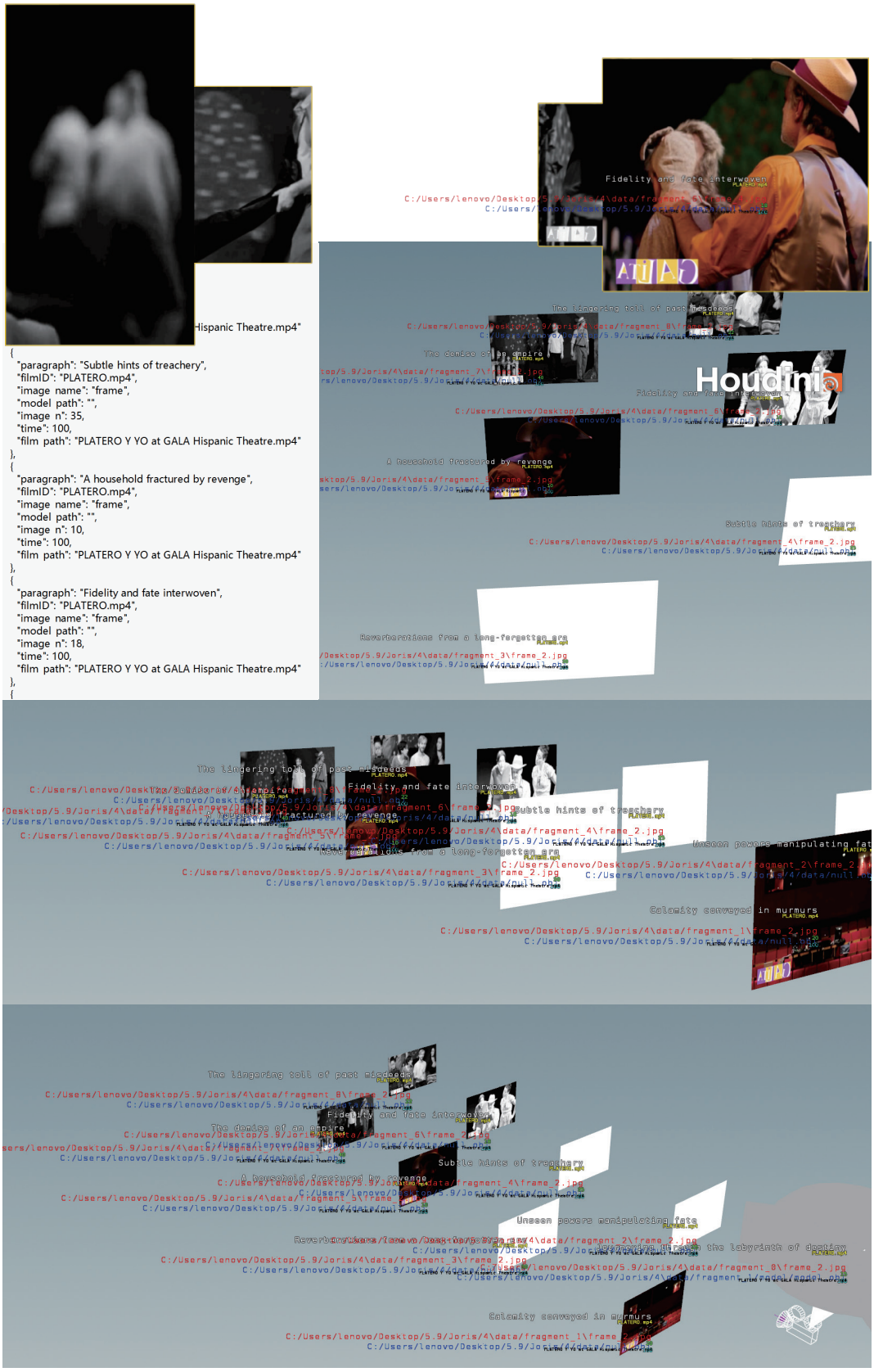
3 video to 3D Model

The video was initially decomposed into a series of image frames, which were then utilized to create a 3D model through RealityCapture. This 3D model was subsequently brought into Houdini for processing, where the camera paths were recreated and textures and details were applied to enhance realism. Volume speculation, curvature, and an extra 3D model were added to the subject. Ultimately, the model was visualized as a set of images.



4 Visualizing JSON in Houdini

I first constructed a json file containing the Spanish film, the lines given to each frame, a randomly assigned image_n, and relevant time, film_path, etc. information. I then import the film frames into houdini and match the json information to each frame to visualise the json file in houdini.



02 Introduction Computability & Complexity

Exercise One

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.

Explanation:

The code implements a classical matrix multiplication algorithm for two $n \times n$ square matrices A and B, resulting in a product matrix C which will also be $n \times n$. The function `square_matrix_multiply` multiplies the two matrices using a triple nested loop structure:

1. **Outer Loop**:
   - The outer loop iterates through the rows of matrix A (indexed by `i`). This loop sets the context for which row of matrix A will be used in the calculations for multiplying elements with matrix B's columns.
2. **Middle Loop**:
   - The second loop iterates through the columns of matrix B (indexed by `j`). For each column in B, this loop helps accumulate the product of elements from the selected row of A and the corresponding elements from this column of B.
3. **Inner Loop**:
   - The innermost loop iterates over the shared dimension (indexed by `k`), where the columns of matrix A align with the rows of matrix B. The products of corresponding elements from the current row of A and the current column of B are computed and accumulated to form a single element of the resulting matrix C.

Code Implementation:

```python
import numpy as np

def square_matrix_multiply(A, B):
    n = len(A)
    C = [[0] * n for _ 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

# testing the function with example
A = np.array([[1, 2], [3, 4]])
B = np.array([[2, 0], [1, 2]])
C = square_matrix_multiply(A, B)
print("Result of A * B:")
print(np.array(C))
```

```
Result of A * B:
[[ 4  4]
 [10  8]]
```

Explanation:

1. **Explanation of Asymptotic Time Complexity:** The `square_matrix_multiply` method employs three nested loops that iterate over the dimension `n` of the square matrices `A` and `B` provided as input. Consequently, this algorithm exhibits a time complexity of $O(n^3)$. During each cycle, the function calculates the dot product between a row in matrix `A` and a column in matrix `B`, accumulating the results in the output matrix `C`.

2. **Implementation Using Nested Lists and Comparison with Matrix Multiplication:**
   - An implementation utilizing nested lists is offered.
   - I will evaluate this implementation against the matrix multiplication provided by NumPy, which is typically much quicker due to optimizations over the basic nested loop approach used in standard Python.
3. **Multiplication of More Than Two Matrices:**
   - I plan to adapt the implementation to manage the multiplication of more than two matrices. This process will entail performing multiple matrix multiplications, which can be suboptimal without techniques such as optimizing the order of matrix chain multiplication.

Code Implementation:

```python
import time

def multiply_matrices_chain(matrices):
    """ Multiplies a chain of matrices using the simple method. """
    result = matrices[0]
    for matrix in matrices[1:]:
        result = square_matrix_multiply(result, matrix)
    return result

def compare_matrix_multiplications(n):
    A = np.random.rand(n, n).tolist()
    B = np.random.rand(n, n).tolist()

    # Measure nested list multiplication
    start_time = time.time()
    C1 = square_matrix_multiply(A, B)
    nested_list_time = time.time() - start_time

    # Measure numpy multiplication
    A_np, B_np = np.array(A), np.array(B)
    start_time = time.time()
    C2 = np.dot(A_np, B_np)
    numpy_time = time.time() - start_time

    return nested_list_time, numpy_time

# Example usage
n = 200  # Change this value to compare different matrix sizes
times = compare_matrix_multiplications(n)
print(f"Time taken with nested lists for {n}x{n} matrices: {times[0]:.4f} seconds")
print(f"Time taken with NumPy for {n}x{n} matrices: {times[1]:.4f} seconds")

# Multiplying more than two matrices
matrices = [np.random.rand(n, n).tolist() for _ in range(3)]
start_time = time.time()
result_matrix = multiply_matrices_chain(matrices)
chain_multiply_time = time.time() - start_time
print(f"Time taken to multiply three {n}x{n} matrices with nested lists: {chain_multiply_time:.4f} seconds")
```

```
Time taken with nested lists for 200x200 matrices: 0.8795 seconds
Time taken with NumPy for 200x200 matrices: 0.0040 seconds
Time taken to multiply three 200x200 matrices with nested lists: 1.7740 seconds
```

Exercise Two

Explanation:

Recursiveness

- **Base Case**: The recursive function `square_matrix_multiply_recursive` defines its base case when the matrix size `n` is reduced to 1. At this point, the multiplication is straightforward—just the product of two single elements, `A[0][0] * B[0][0]`. This case acts as the termination point for the recursion, preventing infinite recursive calls.
- **Recursive Reduction**: Each recursive step splits the matrices `A` and `B` into four sub-matrices each, effectively reducing the problem size by half in each dimension (from `n` to `n/2`). The recursive calls then multiply these smaller matrices, which continue to be split until they reach the base case size of 1×1. This halving continues logarithmically with respect to the dimension of the matrix, which means the depth of the recursive tree is $\log n$.

Divide-and-Conquer Approach

- **Divide**: The function `split_matrix` divides a given matrix into four equally sized sub-matrices $A_{11}, A_{12}, A_{21}, A_{22}$ and $B_{11}, B_{12}, B_{21}, B_{22}$. This step is crucial as it breaks down the larger matrix multiplication problem into smaller, more manageable parts. This division is performed regardless of the size of the matrix until the base case is reached.
- **Conquer**: After the division, the function `square_matrix_multiply_recursive` conquers the problem by recursively solving eight smaller matrix multiplication problems. These are the multiplications of the sub-matrices derived from the original matrices A and B:
  - $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$
  - $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$
  - $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$
  - $C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$

  Each of these products is itself a matrix multiplication, which is why the recursion is necessary. The results from these smaller multiplications are then combined in the next step.
- **Combine**: The function `combine_matrix` takes the results of the recursive multiplications (i.e., the matrices $C_{11}, C_{12}, C_{21}, C_{22}$) and combines them back into a single matrix C. The top part of C is constructed by concatenating C_{11} and C_{12} side by side, and similarly for the bottom part with C_{21} and C_{22}. This step reconstructs the multiplied matrix from its constituent parts, completing the divide-and-conquer process.

Overview

This recursive approach illustrates a classic example of the divide-and-conquer strategy applied to matrix multiplication. By breaking the problem into smaller sub-problems, solving each independently, and combining the results, the algorithm efficiently manages the complexity of matrix multiplication, albeit with a high computational cost due to the recursive calls. This method, while conceptually elegant, is generally slower in practice compared to optimized iterative approaches, especially for larger matrices, unless further optimizations (like Strassen's algorithm) are applied to reduce the number of recursive multiplications needed.

Code Implementation:

```
def split_matrix(A, n):
    """ Split the matrix into four sub-matrices """
    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

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

def combine_matrix(A, B, C, D, n):
    """ Combine four sub-matrices into one matrix after recursion """
    top = [A[i] + B[i] for i in range(n)]
    bottom = [C[i] + D[i] for i in range(n)]
    return top + bottom

def square_matrix_multiply_recursive(A, B):
    """ Recursive square matrix multiplication using the divide-and-conquer approach """
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]  # Base case: single element multiplication
    else:
        mid = n // 2
        A11, A12, A21, A22 = split_matrix(A, mid)
        B11, B12, B21, B22 = split_matrix(B, mid)

        # Recursive calls for sub-problems, contributing to the recursion tree
        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))

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

```
def square_matrix_multiply(A, B):
    """ Non-recursive square matrix multiplication """
    n = len(A)
    C = [[0] * n for _ 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

# Test case
A = [[1, 2, 3, 4],
     [5, 6, 7, 8],
     [9, 10, 11, 12],
     [13, 14, 15, 16]]
B = [[16, 15, 14, 13],
     [12, 11, 10, 9],
     [8, 7, 6, 5],
     [4, 3, 2, 1]]
result_recursive = square_matrix_multiply_recursive(A, B)
result_non_recursive = square_matrix_multiply(A, B)

print("Result of Recursive A * B:")
for row in result_recursive:
    print(row)

print("\nResult of Non-Recursive A * B:")
for row in result_non_recursive:
    print(row)
```

```
Result of Recursive A * B:
[80, 70, 60, 50]
[240, 214, 188, 162]
[400, 358, 316, 274]
[560, 502, 444, 386]

Result of Non-Recursive A * B:
[80, 70, 60, 50]
[240, 214, 188, 162]
[400, 358, 316, 274]
[560, 502, 444, 386]
```

Exercise Three

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.

Analysis of Matrix Operations: Addition/Subtraction Versus Multiplication

Matrix Addition and Subtraction:

- **Complexity**: The operations of adding or subtracting matrices are performed in $O(n^2)$ time for matrices of size $n \times n$. This is due to the direct operation on each of the matrix's n^2 elements, where each undergoes one operation—either addition or subtraction.
- **Operations**: During addition or subtraction, operations are applied directly to corresponding elements of two matrices. This direct element-to-element operation simplifies the process compared to multiplication, leading to faster computation times.

Matrix Multiplication:

- **Complexity**: Traditional matrix multiplication involves a complexity of $O(n^3)$, where each element of the resulting matrix is determined by a dot product—a sequence involving n multiplications and $n - 1$ additions, applied across all n^2 elements.
- **Operations**: Unlike addition or subtraction, multiplication integrates information across entire rows and columns, combining multiplication and addition operations. This integration makes it more computationally intensive.

Complexity Analysis of the Strassen Algorithm

The Strassen algorithm represents an advanced divide-and-conquer strategy in matrix multiplication that reduces the recursive multiplication count.

Standard Recursive Multiplication:

- In the typical recursive approach, 8 multiplications are required at each recursive depth to combine quarters of the matrix, addressing each section of the resulting matrix twice.

Strassen's Algorithm:

- By contrast, the Strassen algorithm reduces these multiplications to 7 by strategically increasing the number of additions and subtractions, which are computationally cheaper. It achieves this through the computation of seven intermediate products (M1 to M7), formulated by adding and subtracting matrix sections before and after their multiplication.

Changes in Complexity Formula:

- **Standard Recursive Method**: The complexity is expressed as $T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$, considering the split, multiply, and combine stages.
- **Strassen's Approach**: It alters the complexity to $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$, with the $O(n^2)$ representing the additional operations for setting up and combining the intermediate results.

Impact of Optimization:

- Modifying the recursion equation thus adjusts the overall complexity. Strassen's algorithm lowers the complexity to about $O(n^{\log_2 7}) \approx O(n^{2.81})$, which provides a noticeable advantage when handling large matrices.

Implementation Implications

The transition from a standard recursive multiplication to the Strassen method involves a shift in focus from purely multiplicative efforts to a balanced mix of more frequent yet less expensive operations (additions and subtractions) and fewer multiplications. This adjustment is crucial in scenarios dealing with large matrices where the cubic growth of standard multiplication becomes unsustainable.

Therefore, Strassen's algorithm offers a significant refinement over straightforward divide-and-conquer techniques, balancing computational expenses effectively and reducing the overall operational load in large-scale matrix multiplications.

Code Implementation:

In [14]:
```python
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)]

# Define the Strassen algorithm matrix multiplication function
def strassen_matrix_multiply(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        # Splitting and recombining matrices like in Exercise 2
        mid = n // 2
        A11, A12, A21, A22 = split_matrix(A, mid)
        B11, B12, B21, B22 = split_matrix(B, mid)

        # Strassen's multiplication operations
        M1 = strassen_matrix_multiply(matrix_add(A11, A22), matrix_add(B11, B22))
        M2 = strassen_matrix_multiply(matrix_add(A21, A22), B11)
        M3 = strassen_matrix_multiply(A11, matrix_sub(B12, B22))
        M4 = strassen_matrix_multiply(A22, matrix_sub(B21, B11))
        M5 = strassen_matrix_multiply(matrix_add(A11, A12), B22)
        M6 = strassen_matrix_multiply(matrix_sub(A21, A11), matrix_add(B11, B12))
        M7 = strassen_matrix_multiply(matrix_sub(A12, A22), matrix_add(B21, B22))

        # Combining results
        C11 = matrix_add(M1, M4)
        C11 = matrix_sub(C11, M5)
        C11 = matrix_add(C11, M7)
        C12 = matrix_add(M3, M5)
        C21 = matrix_add(M2, M4)
        C22 = matrix_add(M1, M3)
        C22 = matrix_sub(C22, M2)
        C22 = matrix_sub(C22, M6)

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

# Test case for the Strassen algorithm
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
result = strassen_matrix_multiply(A, B)
print("Result of Strassen's Matrix Multiplication:")
for row in result:
    print(row)
```

```
Result of Strassen's Matrix Multiplication:
[19, 22]
[43, 6]
```

Explanation:

Function: `matrix_sub(A, B)`

This function computes the element-wise subtraction of two matrices, A and B, which must be of the same dimensions. It:

- Takes two matrices, A and B, as inputs.
- Determines the size of the matrices ($n \times n$).
- Returns a new matrix where each element (i, j) is the result of $A[i][j] - B[i][j]$.

Function: `strassen_matrix_multiply(A, B)`

This function multiplies two square matrices A and B using the Strassen algorithm, which is a divide-and-conquer method. It:

1. **Base Case**: If the matrix size n is 1 (i.e., a 1×1 matrix), it multiplies the single elements of A and B directly. This is the simplest case where no further splitting is needed.
2. **Recursive Case**:
   - **Splitting Matrices**: The function first splits each matrix into four sub-matrices. This is done at the midpoint:
     - $A11, A12, A21, A22$ are the sub-matrices of A.
     - $B11, B12, B21, B22$ are the sub-matrices of B.
   - **Strassen's 7 Multiplications**:
     - $M1$ through $M7$ are the seven products needed for Strassen's algorithm, calculated using specific combinations of additions and subtractions of the sub-matrices:
       - $M1 = A11 \times (B11 + B22)$
       - $M2 = (A21 + A22) \times B11$
       - $M3 = A11 \times (B12 - B22)$
       - $M4 = A22 \times (B21 - B11)$
       - $M5 = (A11 + A12) \times B22$
       - $M6 = (A21 - A11) \times (B11 + B12)$
       - $M7 = (A12 - A22) \times (B21 + B22)$
   - **Combining Results**: The results from these seven multiplications are combined to form the final matrix C using additional matrix additions and subtractions:
     - $C11 = M1 + M4 - M5 + M7$
     - $C12 = M3 + M5$
     - $C21 = M2 + M4$
     - $C22 = M1 + M3 - M2 - M6$
   - The final result C is assembled from these four sub-matrices $C11, C12, C21$, and $C22$.
3. **Return**: The function finally returns the resultant matrix C, which is the product of A and B.