Bartlett RC11 2023-2024

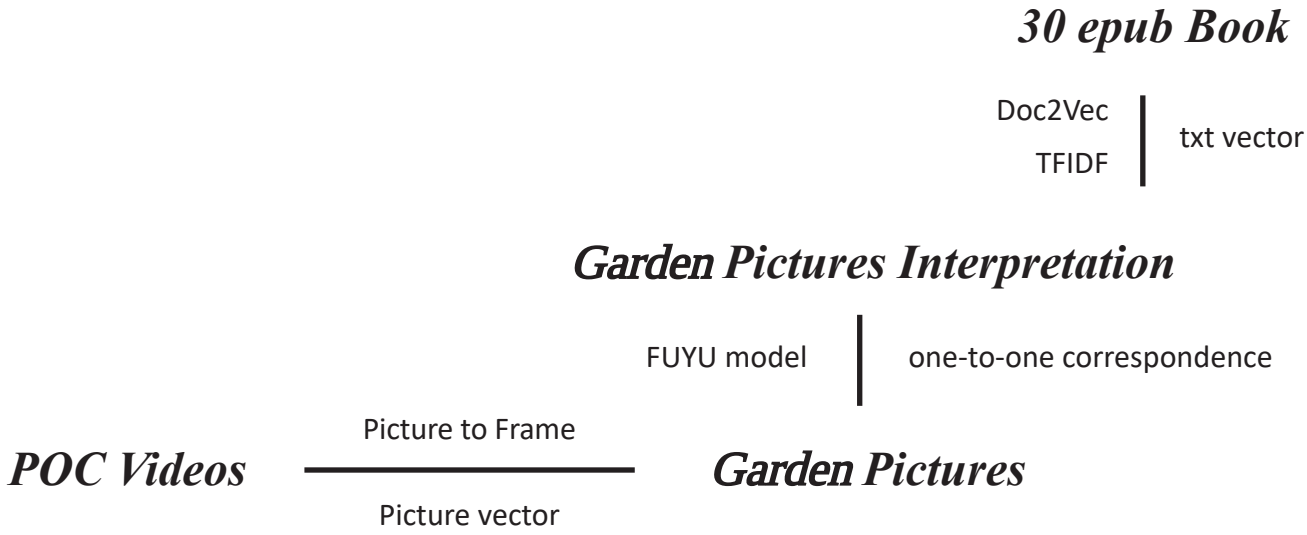
SN: 21093415

Contents

Explaination and Outcomes

VECTORIAL ENCODINGS OF
QUALITATIVE DOMAINS

For this assignment, I mainly completed the vectorisation and cross-searching of 30 book epubs about Landscape, 1000 images about gardens and 200 videos from poc. The specific vectorisation process is shown below.



onedrive link:
https://1drv.ms/f/c/860016f165a50123/Epibde-jxNZOjpxSpGu2ZTABmeh-6rYkr8jBChF6iZYJRw

Input from text:

1.Text input into 30 Books som to get a most related paragraph.

2.The paragraph as input into Garden 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 specific process:

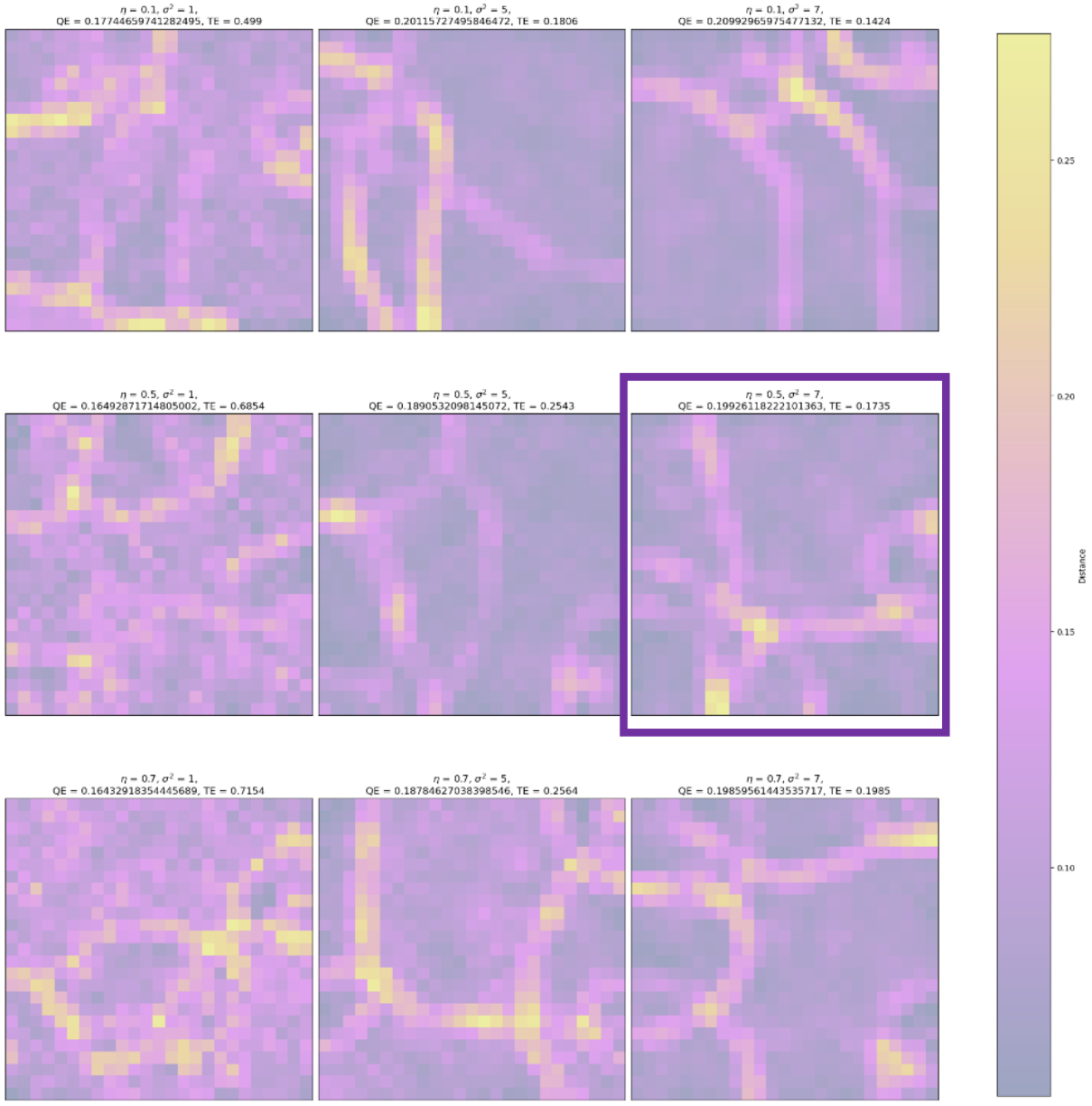
1. Text to text

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

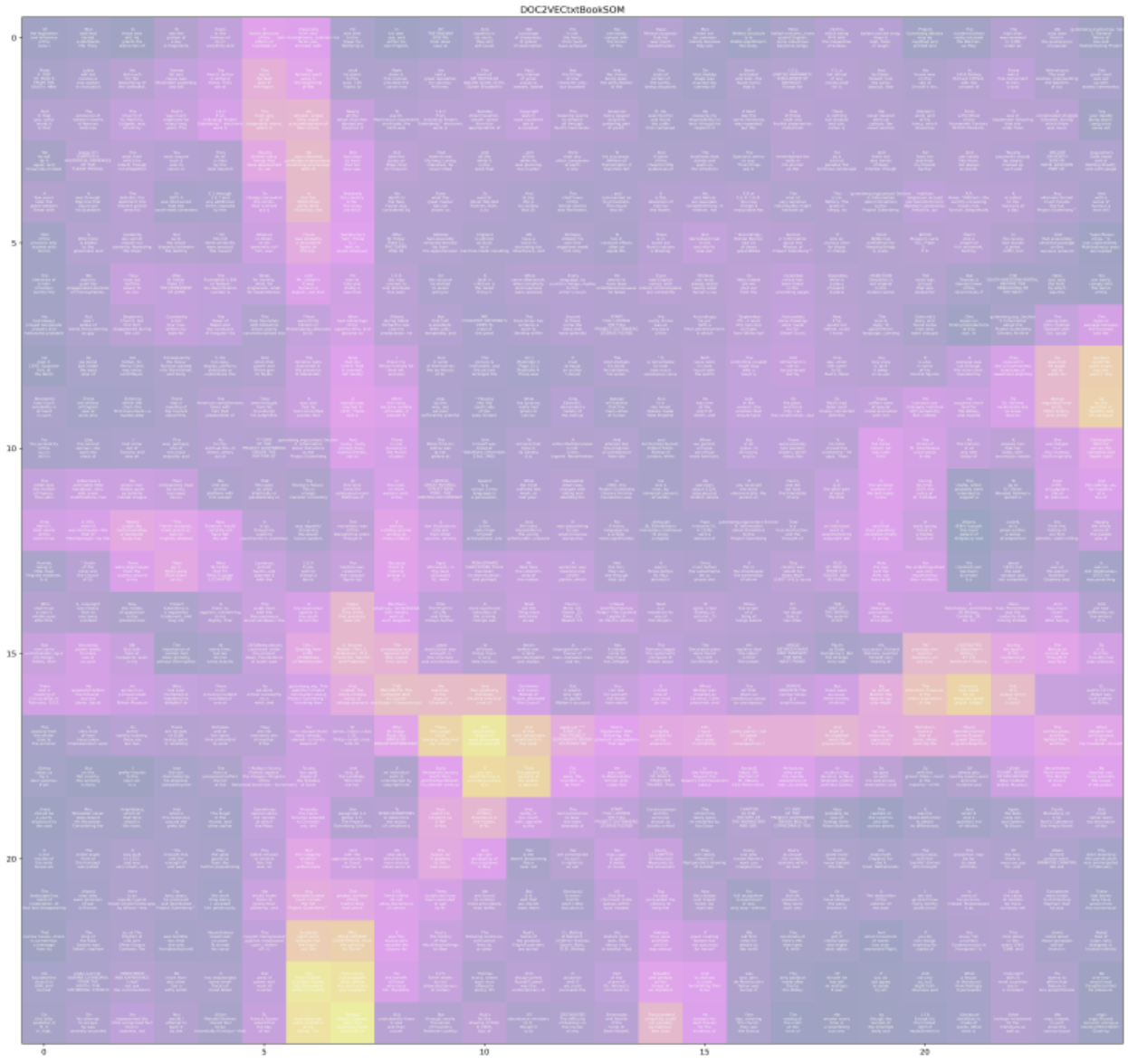


I chose one of the som with an index of 5, and the selection criterion was the one with the lowest sum of the two among the ones with the lower values of both QE and TE.

1.1.3 Link the sentence with som unit

```
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)
```

Paragraphs are read from the PARAGRAPHS1 list, and their respective vectors are retrieved from train_data1. Each paragraph is paired with its vector in a dictionary format and appended to the vectortextPairs list. The best matching units (BMUs) are identified using the find_BMU function and are then assigned to the appropriate slots in data_dict1.



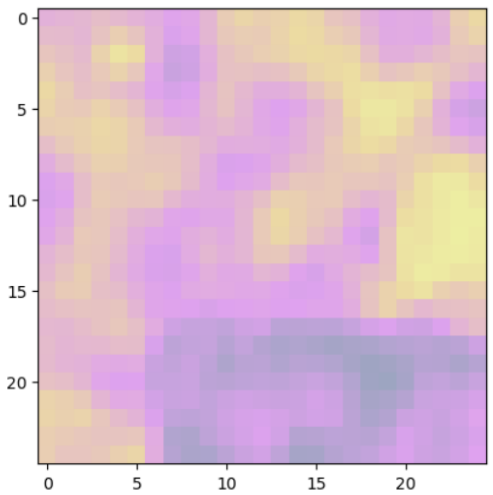
1.1.4 Create Searching Functions

I defined a search function that processes query words by transforming them into vectors through the Doc2Vec model. This function searches the text section for data most similar to a given query within a dataset organized by a SOM. It effectively identifies the most matching cell in the SOM and calculates the similarity of all text within that cell to the query vector using cosine similarity, eventually visualizing and returning the text passage with the highest similarity.

```
def search_TextSom1(SOM1, model1, data_dict1, query='buildings'):
    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
```

```
In [232]: search_TextSom1(SOM1,DOC2VEC_MODEL,data_dict1,'The moon is beautiful tonight ')
```

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




```
(12, 23)
```

```
Out[232]: [' Also in the patrician families of the field, the young people know what they are doing, and marry a neighbouring estate, or a co
vetable title, with some conception of the responsibilities they undertake.']
```

1.1.5 Train SOM with TFIDF

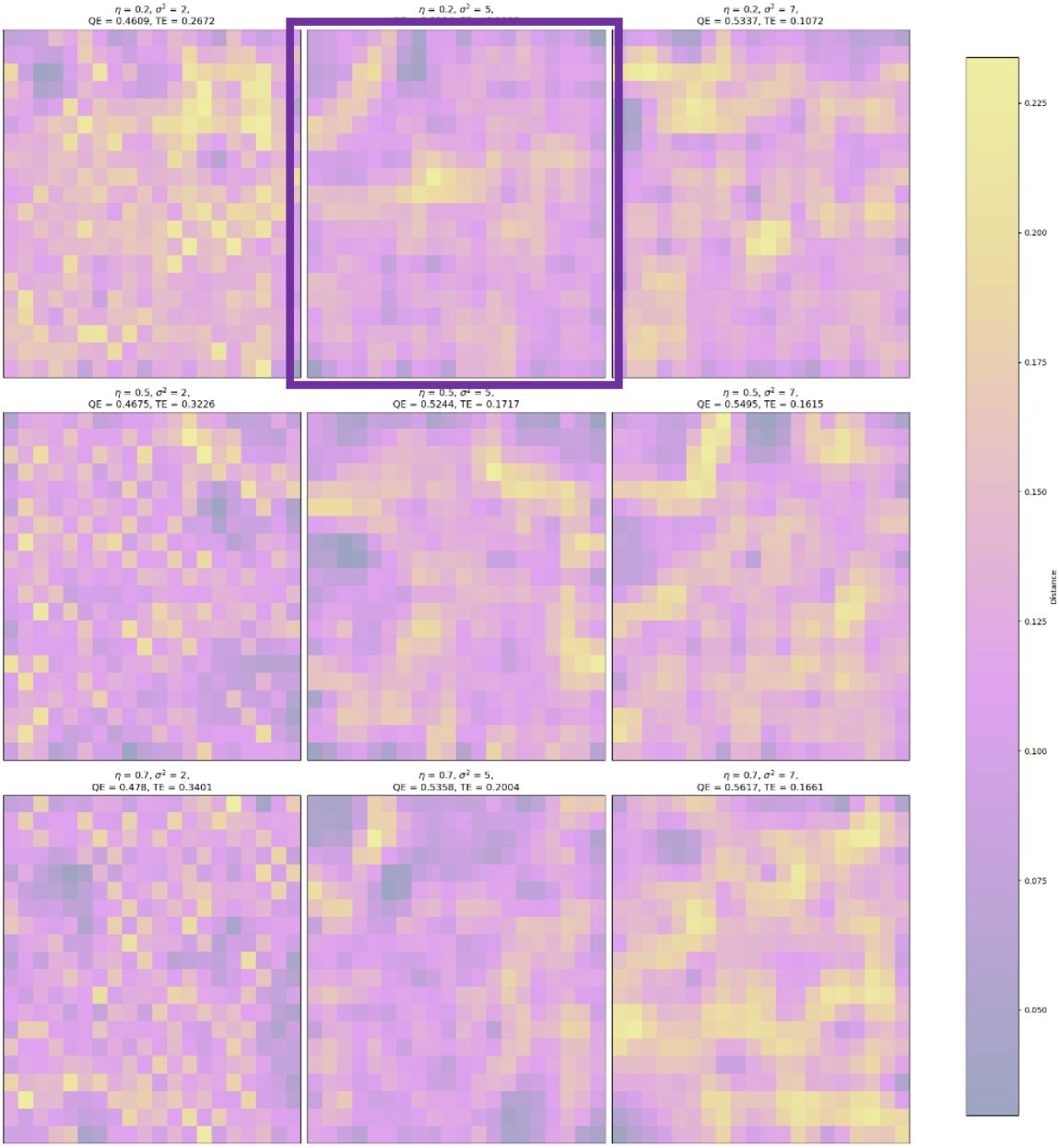
Vectorise text with tfidf, dimensionalise with SVD and train som

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
import scipy.sparse

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: 9999
Filtered SVD Matrix Shape: (9999, 250)
```

```
svd_u_matrix.shape
```

```
(9999, 250)
```



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

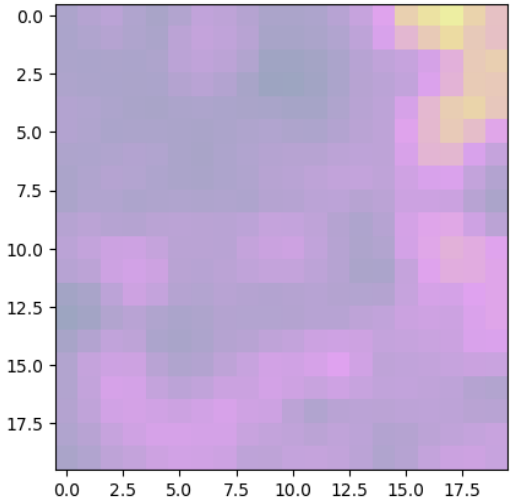
1.1.6 Create Searching Functions

Eliminating zero vectors because they contain no useful data, the text's TF-IDF representation is then transferred to a 2D SOM lattice. Subsequently, the questioning is vectorized using similarity principles to identify the activated SOM and the best matching unit (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='buildings'):

    result = []
    query_tfidf_vector = VECTORIZER_TFIDF.transform([query])
    query_svd_vector = svd.transform(query_tfidf_vector)
    activated_SOM = activatel(u_matrix_values, SOM2, query_svd_vector[0])

    fig = plt.figure()
    plt.imshow(activatel(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
```

In [248]:
```
search_TextSom2(SOM2, VECTORIZER_TFIDF,data_dict2,'The air inside the park is nice')
```



(0, 17)

Out[248]:
```
[{'paragraph': ' Peter; and also to give greater effect to the rich and gorgeous shrine of St. Etheldreda, said to have been of pure silver
adorned with jewels, which at that period stood near the altar and to her place of sepultre, indicated by a boss in the ceiling with her eff
igy on it.',
  'nr': 431,
  'bookID': 'pg20924-images-3'}]
```

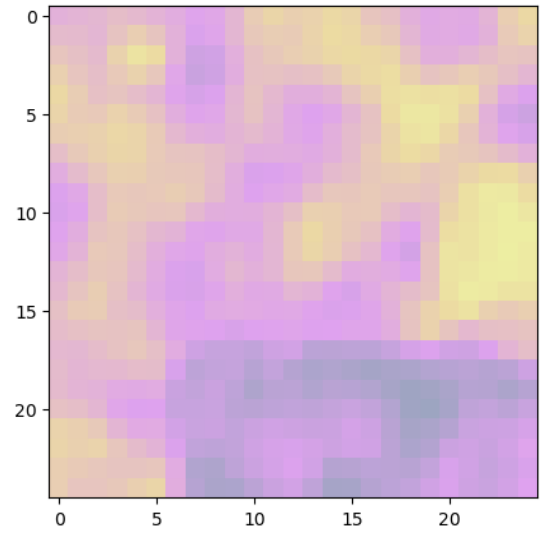
1.1.7 Doc2Vec VS TFIDF

As my project needed to respond to different group's perceptions of events and it was more important to consider semantic links rather than keywords, I chose doc2vec as a way of vectorising text.

Doc2Vec

```
search_TextSom1(SOM1,DOC2VEC_MODEL,data_dict1,'The moon is beautiful tonight ')
```

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



Advantages:
Doc2Vec effectively maps the semantic connections between words and documents, making it superior for computing semantic similarities.

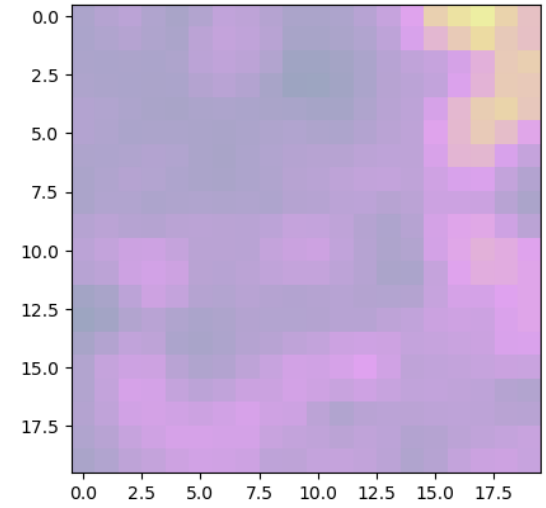
Disadvantages:
In contrast to TF-IDF, Doc2Vec models generally demand more computational power and time, and require substantial amounts of training data to achieve robust performance, rendering them less suitable for small datasets.

```
(12, 23)
```

```
[' Also in the patrician families of the field, the young people know what they are doing, and marry a neighbouring estate, or a covetable t
itle, with some conception of the responsibilities they undertake.']
```

TFIDF

```
search_TextSom2(SOM2, VECTORIZER_TFIDF,data_dict2,'The air inside the park is nice')
```



Advantages:
TF-IDF is straightforward and user-friendly, ideal for information retrieval and matching keywords in text, and offers quicker processing than Doc2Vec.

Disadvantages:
TF-IDF only accounts for the frequency of words and their document occurrence, neglecting the semantic connections between words, thus it may fail to reflect the semantic similarities among them.

```
(0, 17)
```

```
[{'paragraph': ' Peter; and also to give greater effect to the rich and gorgeous shrine of St. Etheldreda, said to have been of pure silver
adorned with jewels, which at that period stood near the altar and to her place of sepultre, indicated by a boss in the ceiling with her eff
igy on it.',
  'nr': 431,
  'bookID': 'pg20924-images-3'}]
```

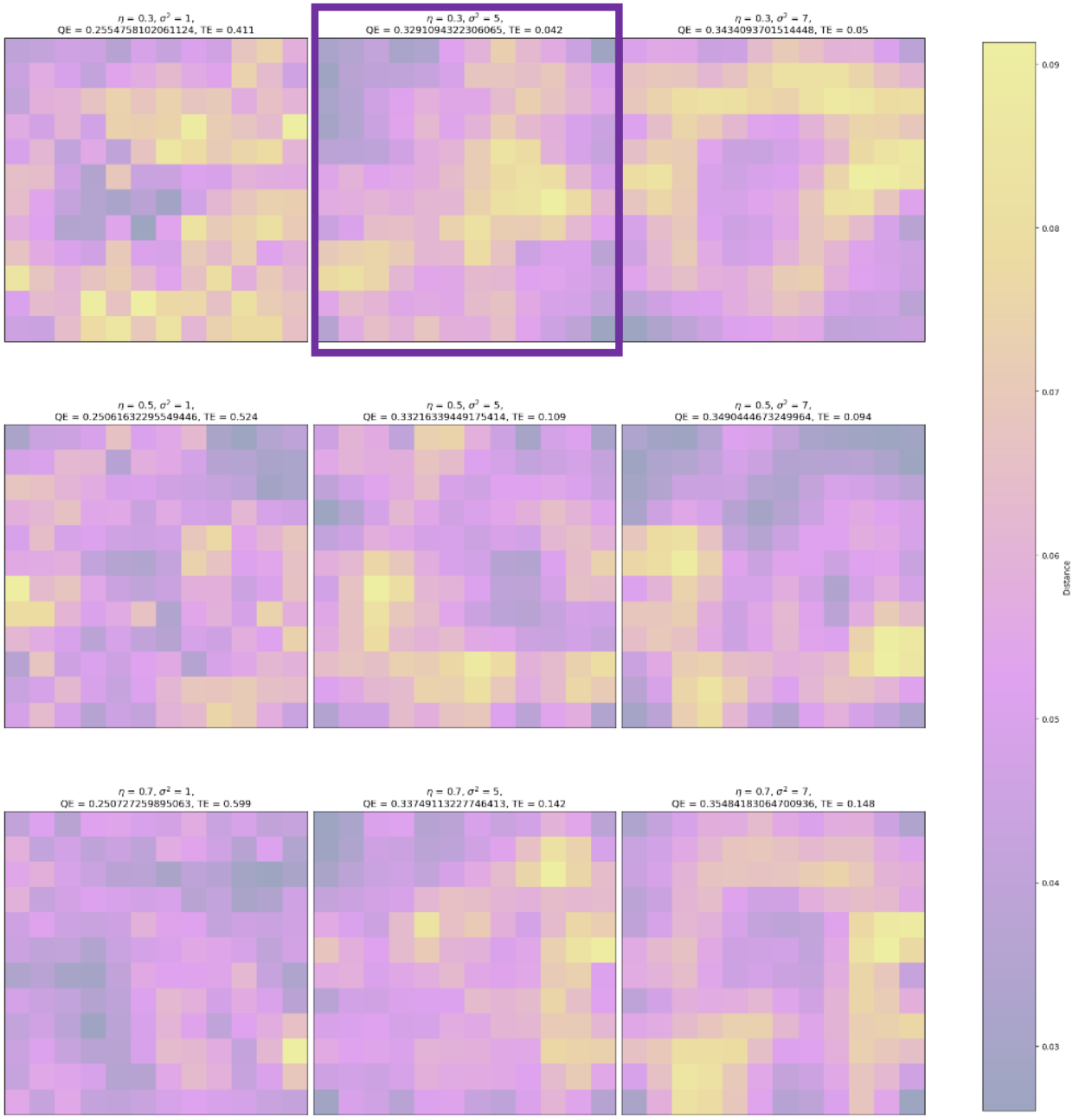
1.2 Text to image

1.2.1 Generate descriptive text for each image by fuyu model

```
In [259]: texts
```

```
s, some bushes, and trees. The tree is surrounded by bushes, while another tree is visible nearby. The bushes are placed throughout the g
arden, with some close to the tree and others further away. Additionally',
  'row_index': 8},
 {'nr': 8,
  'filename': 'image46.jpg',
  'content': 'The image features an outdoor area with a concrete wall, a bench, and trees. There is also a rock structure, possibly a roc
k wall, and a potted plant nearby.\n\nThere are several ferns and plants scattered throughout the area, adding greenery and life to the s
cene. Additionally, there are multiple potted plants, some placed close to the bench and others',
  'row_index': 9},
 {'nr': 9,
  'filename': 'image116.jpg',
  'content': 'The image features a bathroom with a black bathtub, a sink, and a potted plant. There is also a window in the bathroom.\n\n
There are multiple bottles in the bathroom, with one near the sink and another near the window. Additionally, there are four cups scatter
ed throughout the bathroom, with one close to the sink and three others near the window. A towel is hanging on the wall above the sink.',
  'row_index': 10},
 {'nr': 10,
  'filename': 'image84.jpg',
  'content': 'The image features a long indoor swimming pool surrounded by trees.\n',
  'row_index': 11},
```

1.2.2 Train description text SOM



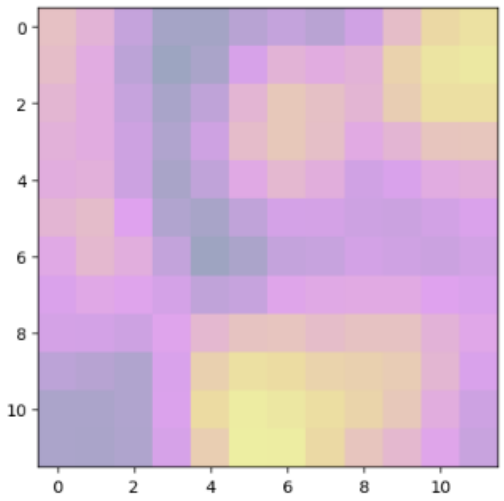
The som I chose is index 1.

1.2.3 Find the image according to the file description

Since the text SOM search part of the process is similar, it is omitted here and only the output results are shown.

```
result3 = search_TextSom3(SOM3, DOC2VEC_MODEL2, data_dict3, featureTextPairs2, query='She likes to go for walks')
```

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




```
(11, 5)
Most similar content: The image features a wooden park bench surrounded by flowers and bushes.
```

```
result3
```

```
[{'nr': 151,
  'filename': 'image51.jpg',
  'content': 'The image features a wooden park bench surrounded by flowers and bushes.\n'}]
```

```
filename3 = [item['filename'] for item in result3]
```

```
image_folder = r'D:\rc11\collection\images\garden'
image_path = os.path.join(image_folder, filename3[0])

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

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

```
Trying to open: D:\rc11\collection\images\garden\image51.jpg
```

1.3 Text to video

1.3.1 Extract Video frames

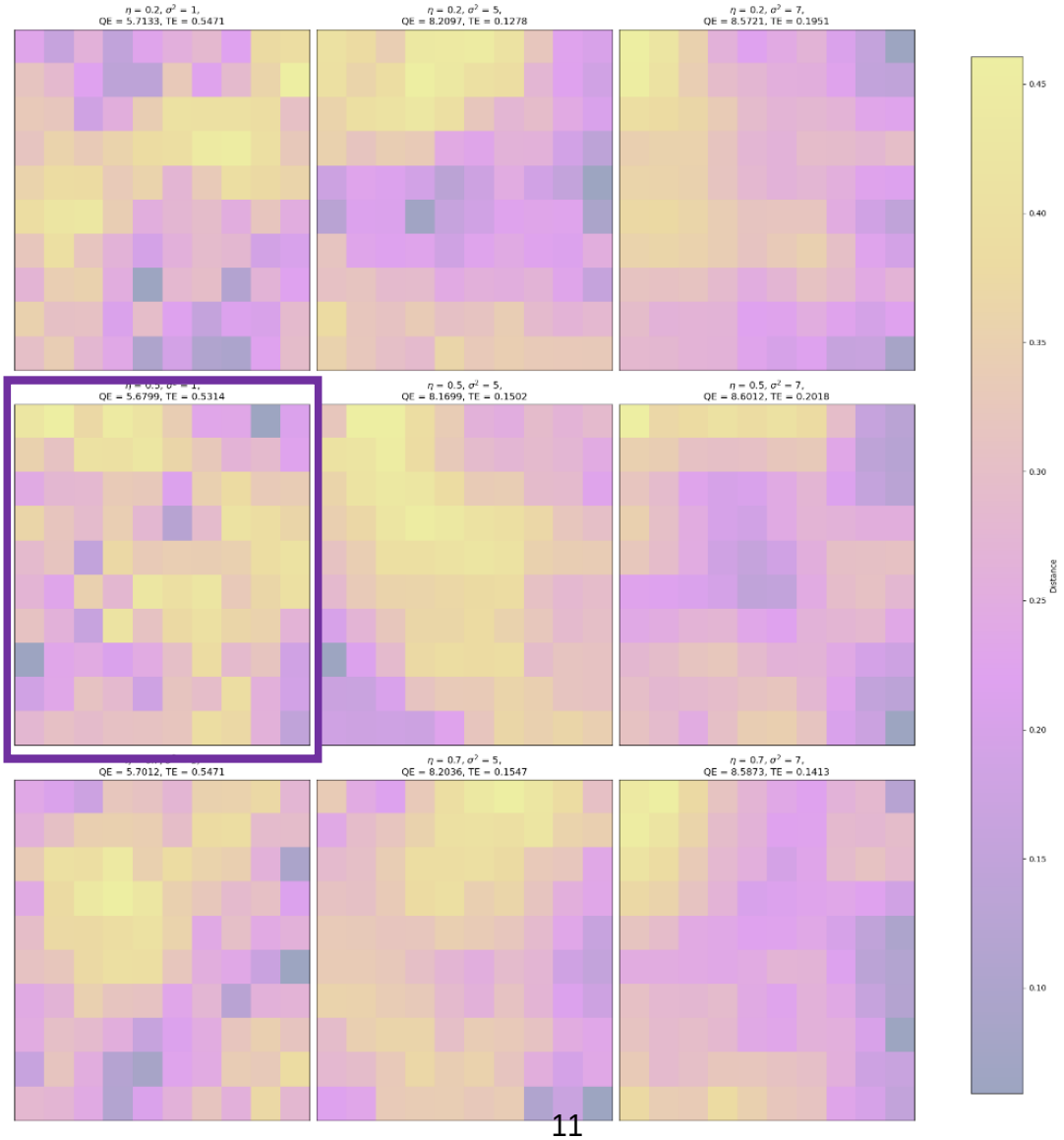
I chose to train the SOM by extracting a frame every ten seconds and vectorising the frame image.

```
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
```

```
film_features = process_all_films(r'D:\rc11\collection\video\ballroom', 10, model)
```

1.3.2 Train SOM

The som I chose is index 3.



1.4 search together

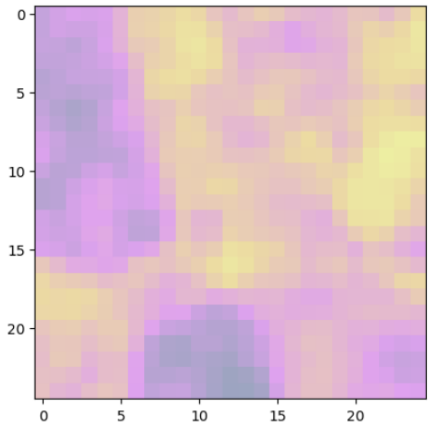
Because 10 examples are too many, only 1 is shown in the pdf, the rest of the results can be viewed in the source file.

```
query_texts = [
    "Libraries attract readers eager to dive into new novels and non-fiction.",
    "Gardening enthusiasts gather at community gardens to share tips and plants.",
    "Cooking classes draw food lovers looking to enhance their culinary skills.",
    "Yoga retreats offer participants a chance to relax and rejuvenate their spirits.",
    "Chess tournaments provide a platform for players to test their strategic thinking.",
    "Wildlife photography excursions capture nature's beauty and rare animal behaviors.",
    "Beach outings are popular for sunbathing and playing various water sports.",
    "Poetry readings provide a stage for poets to share their latest works with an audience.",
    "Amusement parks see families enjoying thrilling rides and games together.",
    "Craft fairs showcase handmade goods and artistic creations from local artisans."
]

image_folder = r'D:\rc11\collection\images\garden'
```

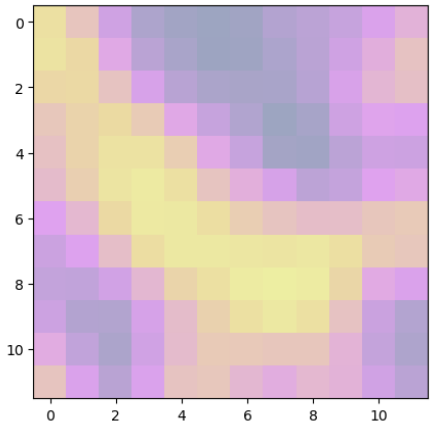
```
def process_query_and_video(query, SOM1, DOC2VEC_MODEL, data_dict1, SOM3, DOC2VEC_MODEL2, data_dict3, featureTextPairs2, image_folder, searc
    results = {}

    # 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"保存至: {output_path}")

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

Input: "Libraries attract readers eager to dive into new novels and non-fiction."

Output:

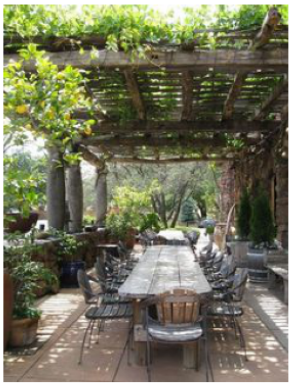



```
(9, 22)
[' 21. The first thing, therefore, you have to learn in landscape, is to outline; and therefore we must now know precisely what an outline
is, how it ought to be represented, and this it will be right to define in quite general terms applicable to all subjects.']

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

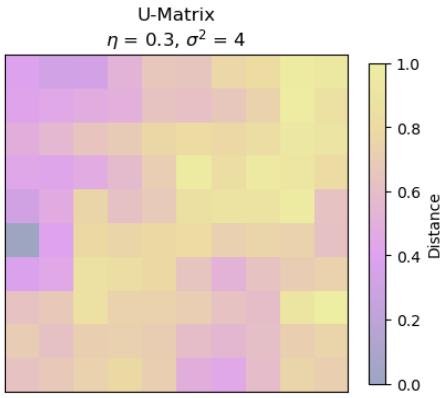


(8, 7)
Most similar content: The image features an outdoor dining area with a wooden dining table surrounded by chairs. There are several potted pl
ants scattered throughout the area, adding greenery and color to the scene.

The dining area has multiple chairs arranged around the table, some closer to the table and others further away. The chairs vary in size and
position, creating a comfortable



D:\rc11\collection\images\garden\image805.jpg
1/1 [==============================] - 0s 27ms/step
Query features shape: (1, 1024)
BMU location: (7, 9)



Moviepy - Building video output_videos\image805_trimmed_video.mp4.
MoviePy - Writing audio in image805_trimmed_videoTEMP_MPY_wvf_snd.mp3

MoviePy - Done.
Moviepy - Writing video output_videos\image805_trimmed_video.mp4

Moviepy - Done !
Moviepy - video ready output_videos\image805_trimmed_video.mp4
保存至: output_videos\image805_trimmed_video.mp4

2.0 Input from picture

2.1 Picture to picture

2.1.1 Process Garden Images

Load files from 1000 images folders, and preprocess them with image feature model

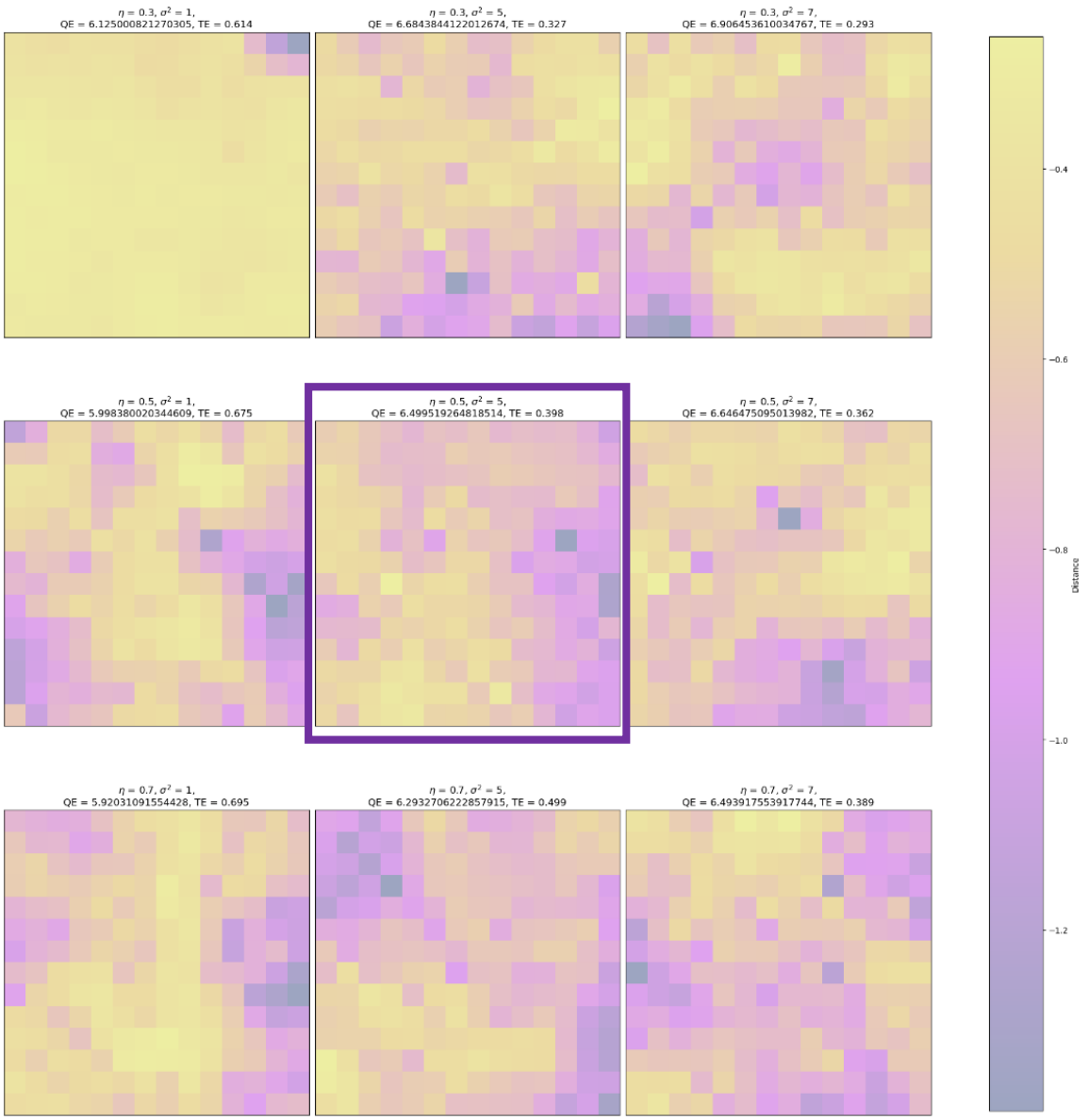
```
model = tf.keras.applications.mobilenet.MobileNet(
# The 3 is the three dimensions of the input: r, g, b.
  input_shape=(224, 224, 3),
  include_top=False,
  pooling='avg'
)
```

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

```
def extract_images_from_folders(main_folder_path):
    all_image_paths = []
    for root, dirs, files in os.walk(main_folder_path):
        for file in files:
            if file.lower().endswith(('.png', '.jpg', '.jpeg', '.gif')):
                file_path = os.path.join(root, file)
                all_image_paths.append(file_path)
    return all_image_paths
main_folder_path = r'D:\rc11\collection\images\garden'

all_image_paths = extract_images_from_folders(main_folder_path)
print("Total images found:", len(all_image_paths))
```

```
Total images found: 1000
```

2.1.2 Train Images SOM



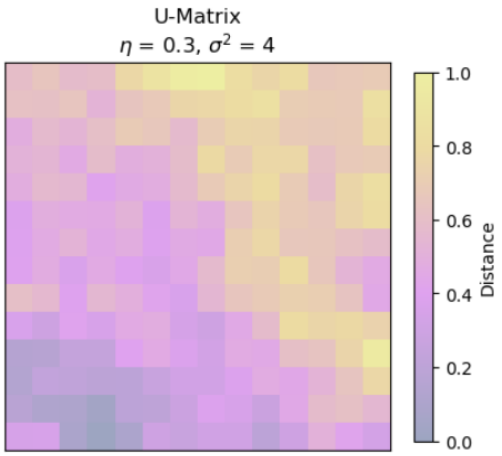
The som I chose is index 4.

2.1.3 Search

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

```
query = r'D:\rc11\collection\images\museum\52900762653_44466b92f6_z.jpg'
result5 = searchvideopicturesom5(query)
```

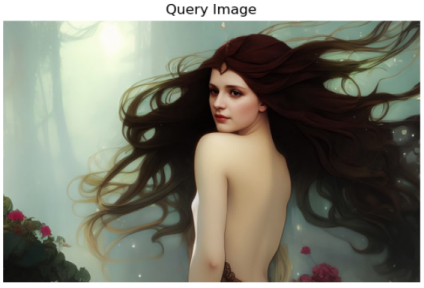
```
1/1 [==============================] - 0s 108ms/step
Query features shape: (1, 1024)
BMU location: (0, 7)
```




```
result5
```

```
[{'image': 'D:\\rc11\\collection\\images\\garden\\image691.jpg',
  'path': 'D:\\rc11\\collection\\images\\garden\\image691.jpg',
  'content': 'The image features a wooden dining table with a tea set on it, surrounded by chairs. The table is surrounded by several potted plants, creating a pleasant and inviting atmosphere. There are also vases and bowls on the table, adding to the aesthetic appeal of the scene. The setting appears to be a patio or outdoor area, perfect for enjoying a meal or'}]
```

```
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

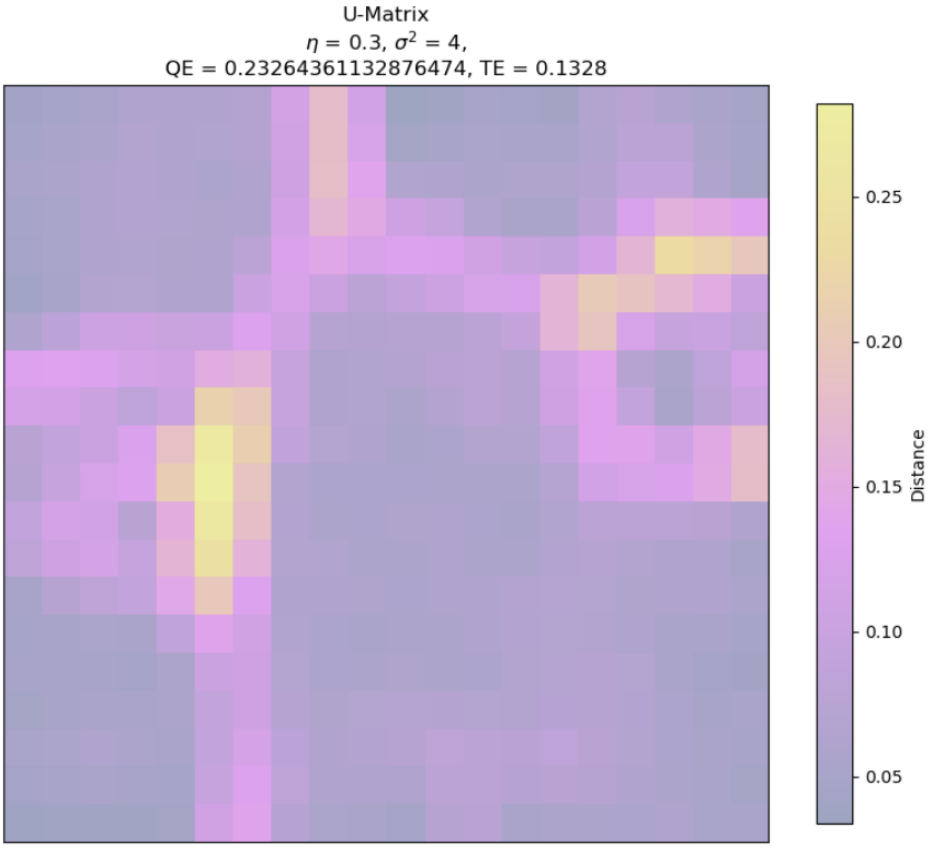
2.2.1 Import previously trained Text SOM

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

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

```
input_file_path = "PARAGRAPHS.pkl"
with open(input_file_path, "rb") as f:
    PARAGRAPHS = pickle.load(f)
```

```
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 employing the image description generated from the previous matching step as input to the text SOM, I transitioned from image to text.

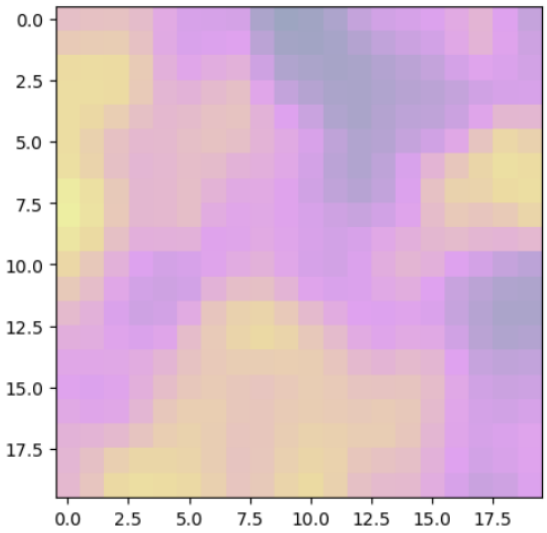
```
def search_TextSom1(SOM1, model, data_dict1, query='street'):
    result = []

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

    # Activate and display SOM
    fig = plt.figure()
    plt.figure(figsize=(5, 5))
    plt.imshow(activate1(train_data1, SOM1, query_vector), cmap=cmap)
    plt.show()

    # Find BMU location
    g, h = find_BMU1(SOM1, query_vector)
    print((g, h))

    # Check if BMU cell is empty
    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 []

    # Calculate similarity and find the most similar paragraph
    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]

    # Find the most similar paragraph
    if similarities:
        biggest = max(similarities, key=similarities.get)
        result.append(biggest)
    else:
        print("No similarities found.")

    return result
```

```
contents
```

```
['The image features a small white doorway surrounded by greenery and flowers.\n\nThe doorway is surrounded by bushes, trees, and flowers, creating a serene and peaceful atmosphere. The door is surrounded by greenery, making it an attractive and inviting entrance. The surrounding environment creates a calming ambiance, perfect for relaxation or a quiet walk']
```

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

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




```
(8, 0)
```

```
[' So it is always. Good crystals are friendly with almost all other good crystals, however little they chance to see of each other, or however opposite their habits may be; while wicked crystals quarrel with one another, though they may be exactly alike in habits, and see each other continually.']
```

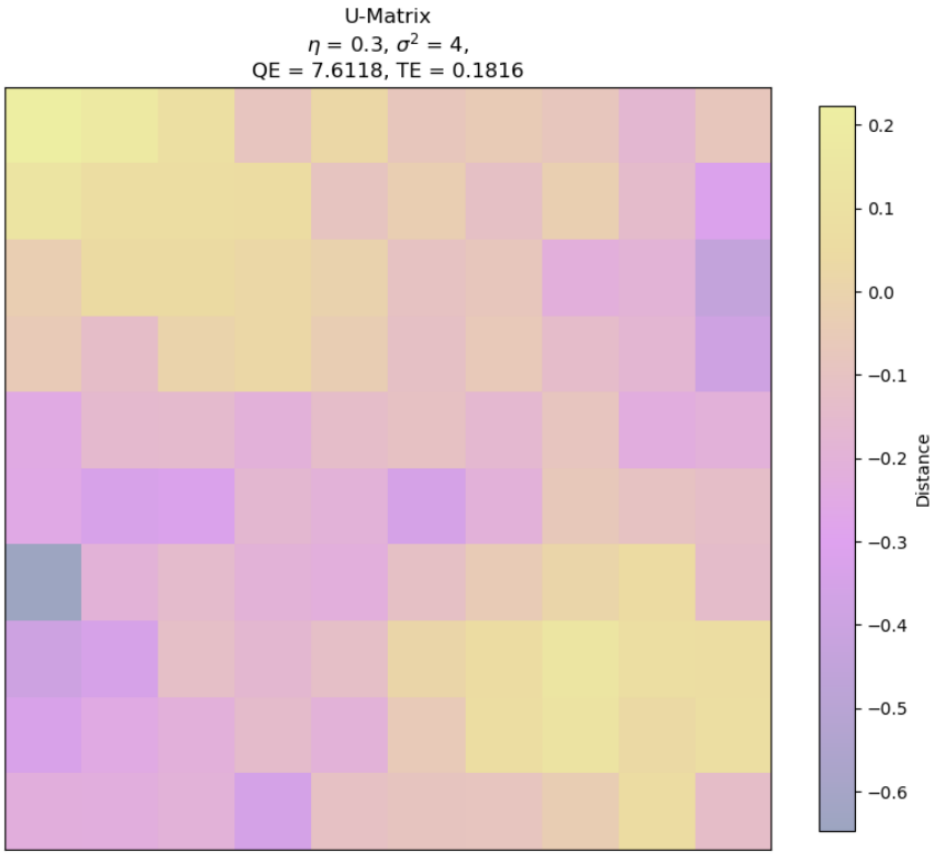
2.3 Picture to Video

2.3.1 Import previously trained Video SOM

```
with open('Video_featureSOM.pkl', 'rb') as f:
    SOM4= pickle.load(f)
```

```
import json
def load_from_json(filename):
    with open(filename, 'r') as f:
        data = json.load(f)
    return data
film_features = load_from_json('Database03_filmfeatures.json')
```

```
u_matrix_values = u_matrix(SOM4)
QE = round(calculateQE(SOM4, n_train_data4), 4)
TE = round(calculateTE(SOM4, n_train_data4), 4)

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()
```



2.3.1 Search

Summary of Processes

- **Data Preparation**: Load video features and metadata.
- **Query Processing**: Convert query (an image or a video frame) into a feature vector.
- **Feature Matching**: Use SOM to find the location in a trained map that best matches the query features.
- **Video Retrieval**: From the results of the SOM, identify relevant video content.
- **Video Processing**: Extract and possibly edit parts of the video that match the query criteria.
- **Output Presentation**: Show the results visually by plotting data and displaying video frames.

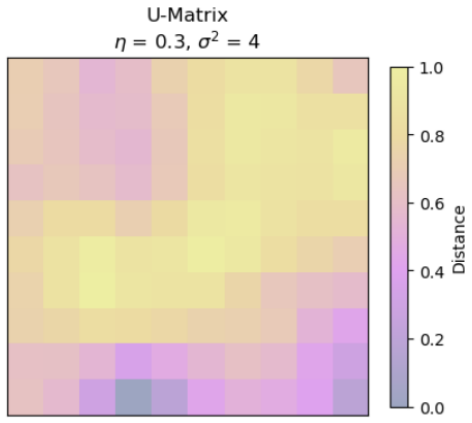
```
def searchvideopicturesom4(query):
    result = []
    query_features = []
    path = query
    q_f = processImage(path, model)
    query_features.append(q_f)

    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
```

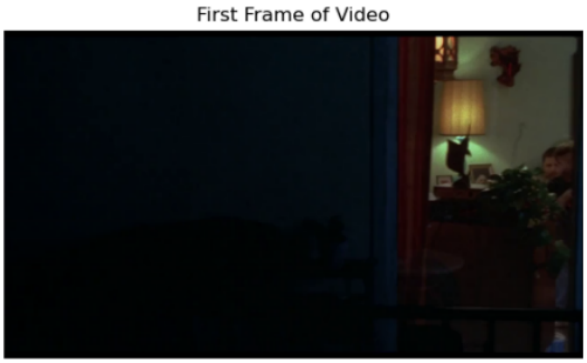
```
Path
```

```
['D:\\rc11\\collection\\images\\garden\\image723.jpg']
```

```
query = Path[0]
result4 = searchvideopicturesom4(query)
```

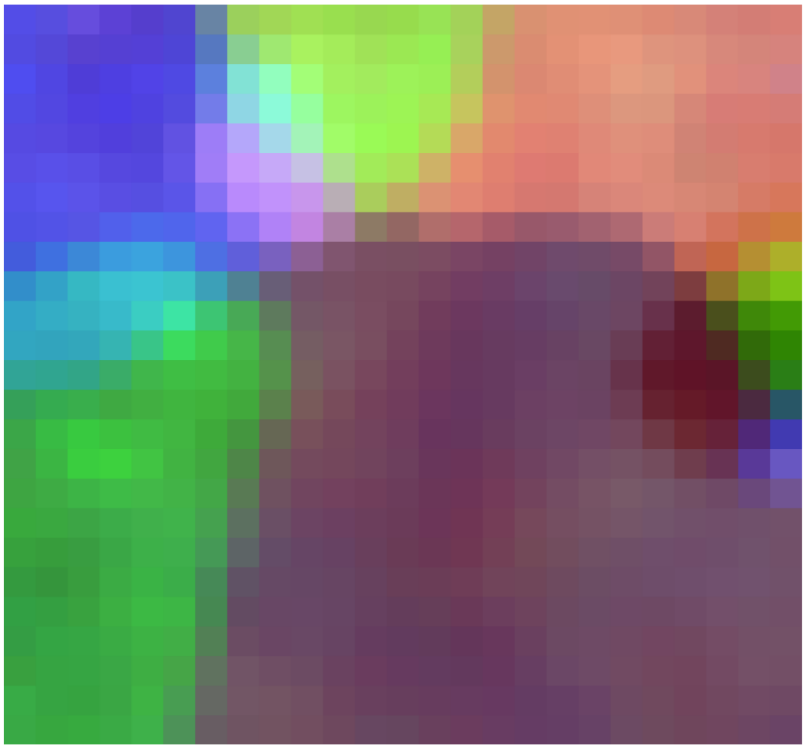
```
1/1 [==============================] - 0s 115ms/step
Query features shape: (1, 1024)
BMU location: (6, 2)
```





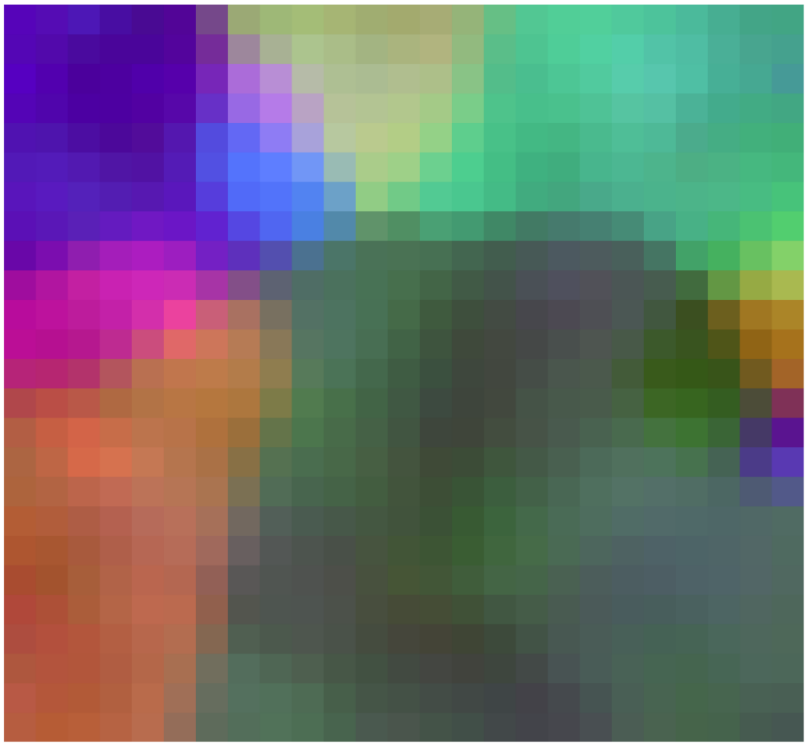
3.1 Fit PCA on all cells of the SOM

In the Doc2vec SOM, the vector of each cell is reduced to three dimensions using PCA, and these dimensions are assigned as RGB values to that particular cell. Ultimately, the SOM is displayed in this manner.



3.1 Fit PCA on the entire training set

In the Doc2vec SOM, the vector for each cell is compressed into three dimensions that span the entire database using PCA, with these three dimensions then designated as RGB colors for each specific cell. Upon rendering these back onto the SOM the visual result is presented as shown.



Complexity Assignment

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.

Code Implementation:

```python
import numpy as np

def square_matrix_multiply(A, B):
    """
    Multiplies two square matrices using a triple nested loop.

    Args:
    - A (list of list of int/float): The first matrix.
    - B (list of list of int/float): The second matrix.

    Returns:
    - C (list of list of int/float): The resulting product matrix.
    """
    n = len(A)
    C = [[0] * n for _ in range(n)]  # Initialize the resulting matrix
    for i in range(n):  # Iterate through the rows of A
        for j in range(n):  # Iterate through the columns of B
            for k in range(n):  # Perform the dot product
                C[i][j] += A[i][k] * B[k][j]
    return C

# Testing the function with example inputs
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]

result = square_matrix_multiply(A, B)

# Convert to numpy arrays for more readable printing
np_result = np.array(result)
print(f"Matrix A:\n{np.array(A)}")
print(f"Matrix B:\n{np.array(B)}")
print(f"Result of A * B:\n{np_result}")
```

```
Matrix A:
[[1 2]
 [3 4]]
Matrix B:
[[5 6]
 [7 8]]
Result of A * B:
[[19 22]
 [43 50]]
```

Explanation:

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

1. The outer loop iterates through the rows of matrix A (`i` index).
2. The second loop iterates through the columns of matrix B (`j` index).
3. The inner loop iterates over the shared dimension (`k` index), where matrix A's columns align with matrix B's rows.

The nested loops perform the dot product computation to accumulate the matrix multiplication results.

Code Implementation:

```
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 = 100  # 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 100x100 matrices: 0.1140 seconds
Time taken with NumPy for 100x100 matrices: 0.0010 seconds
Time taken to multiply three 100x100 matrices with nested lists: 0.2294 seconds
```

Explanation:

1. **Asymptotic Time Complexity Explanation:** The provided `square_matrix_multiply` function uses three nested loops, each iterating over the dimension `n` of the input square matrices `A` and `B`. Thus, the time complexity of this algorithm is $O(n^3)$. Each iteration computes the dot product of a row from matrix `A` and a column from matrix `B`, summing the result into the output matrix `C`.
2. **Implementation with Nested Lists and Matrix Multiplication Comparison:**
   - The implementation with nested lists is provided.
   - I'll compare the performance against NumPy's built-in matrix multiplication, which is optimized and typically much faster than a straightforward nested loop implementation in pure Python.
3. **Multiplying More Than Two Matrices:**
   - I will extend the implementation to handle the product of more than two matrices. This will involve a sequence of matrix multiplications, which can be less efficient without optimizations like matrix chain multiplication order optimization.

Exercise Two

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]
```

Explanation:

Recursiveness

1. **Base Case**: The recursion has a clear base case. When the matrix size `n` is reduced to 1, the function simply multiplies the two single elements of the matrices `A` and `B`. This is the simplest scenario, where no further division is possible, and the multiplication can be directly performed.
2. **Recursion Step**: The recursion step involves dividing the problem into smaller sub-problems, recursively solving these smaller problems, and then combining the results to form the solution to the original problem. Each recursive call reduces the matrix size by half (`n // 2`), progressively approaching the base case. This halving continues until the base case of a 1×1 matrix is reached.

Divide-and-Conquer

1. **Divide**: The matrix is divided into four sub-matrices of equal size. This is done by the `split_matrix` function, which partitions the input matrix `A` into top-left (`A11`), top-right (`A12`), bottom-left (`A21`), and bottom-right (`A22`) quarters, and similarly for matrix `B`. This division step breaks down the larger matrix multiplication problem into smaller parts.
2. **Conquer**: Once the matrices are divided, the algorithm conquers the smaller problems by recursively calling `square_matrix_multiply_recursive` on the smaller sub-matrices. This includes eight recursive calls to multiply these parts:
   - Multiply A_{11} with B_{11}
   - Multiply A_{12} with B_{21}
   - Multiply A_{11} with B_{12}
   - Multiply A_{12} with B_{22}
   - Multiply A_{21} with B_{11}
   - Multiply A_{22} with B_{21}
   - Multiply A_{21} with B_{12}
   - Multiply A_{22} with B_{22}

   These products are then added appropriately to form the entries of the resulting sub-matrices `C11`, `C12`, `C21`, and `C22`.
3. **Combine**: The final step of the divide-and-conquer approach is to combine the results of the conquered sub-problems into a single solution matrix `C`. The `combine_matrix` function accomplishes this by taking the four resulting sub-matrices and arranging them into their respective quadrants of the final combined matrix. The top half of the matrix `C` is formed by concatenating `C11` and `C12` side-by-side, and the bottom half by concatenating `C21` and `C22`, followed by stacking the top and bottom halves vertically.

Explanation of the Algorithm's Steps

- **Divide**: The original matrix multiplication problem is divided into smaller problems by splitting the matrices.
- **Conquer**: The algorithm recursively solves these smaller matrix multiplication problems. Each recursive call reduces the problem size further until the base case is reached.
- **Combine**: The results of the smaller problems are then combined to produce the final output matrix, effectively building the solution from the results of the recursively solved sub-problems.

This recursive approach efficiently applies the divide-and-conquer methodology but does so at the cost of increased recursive overhead and multiple additions, which makes it less practical for very large matrices without optimizations such as Strassen's algorithm.

Exercise Three

Reflecting on Matrix Operations: Addition/Subtraction vs. Multiplication

Matrix Addition and Subtraction:

- **Complexity**: These operations are $O(n^2)$ for an $n \times n$ matrix. Each element of the matrix undergoes a single operation (addition or subtraction), directly corresponding to the total number of elements in the matrix, which is n^2.
- **Operations**: In matrix addition (or subtraction), element-wise addition (or subtraction) is performed across the two matrices. This linear approach to each element makes the operation less complex and faster to compute compared to multiplication.

Matrix Multiplication:

- **Complexity**: The naive approach to matrix multiplication has a complexity of $O(n^3)$ for an $n \times n$ matrix. This is because each element of the resulting matrix is computed by taking the dot product of the corresponding row and column, which involves n multiplications and $n - 1$ additions for each of the n^2 elements.
- **Operations**: Multiplication combines both addition and multiplication and integrates results across rows and columns, unlike addition or subtraction, which are straightforward and isolated to corresponding elements.

Complexity Analysis of the Strassen Algorithm

The Strassen algorithm is a divide-and-conquer approach to matrix multiplication that optimizes the number of recursive multiplications.

Standard Recursive Multiplication:

- Typically, 8 multiplications are required for each recursive step (each quarter of the resulting matrix requires 2 multiplications).

Strassen's Algorithm:

- Strassen's method reduces the number of recursive multiplications to 7 by introducing extra additions and subtractions (which are cheaper, as explained). This is achieved by calculating 7 intermediate matrices (M1 through M7) using strategic combinations of additions and subtractions before and after the multiplications.

Complexity Formula Change:

- **Standard Recursive:** $T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$ for the work to split, multiply, and combine matrices.
- **Strassen's Algorithm:** $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$ where $O(n^2)$ accounts for the additional and subtractive operations required to setup and combine the results from the 7 multiplications.

Optimization Impact:

- This change in the recursion formula affects the overall complexity. While naive recursive multiplication has a complexity of $O(n^3)$, Strassen's algorithm improves this to approximately $O(n^{\log_2 7}) \approx O(n^{2.81})$. This improvement is beneficial for large matrices.

Implementation Considerations

Transitioning from standard recursive multiplication to Strassen's algorithm involves repurposing the recursive structure to handle fewer multiplicative calls but includes more additive and subtractive operations. These changes are critical in large-scale computational scenarios where the cubic growth of the naive approach becomes infeasible.

Thus, Strassen's algorithm is a practical enhancement over the simple divide-and-conquer approach by effectively balancing the computational costs between more frequent but cheaper additions/subtractions and less frequent but costlier multiplications. This balanced approach significantly reduces the operational overhead for large matrix multiplications.

Code Implementation:

```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)]


def strassen_matrix_multiply(A, B):
    n = len(A)
    if n == 1:  # Base case: when matrices become 1x1
        return [[A[0][0] * B[0][0]]]
    else:
        # Splitting and recombining matrices
        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

# Testing the function with an example
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]

result = strassen_matrix_multiply(A, B)

# Printing the result
print("Result of Strassen's matrix multiplication:")
for row in result:
    print(row)
```

```
Result of Strassen's matrix multiplication:
[19, 22]
[43, 6]
```

Explanation:

`matrix_sub` Function

This function performs element-wise subtraction between two matrices `A` and `B`, assuming both have the same dimensions $n \times n$. It iterates through each matrix element, calculating the difference $A[i][j] - B[i][j]$ for each corresponding element, resulting in a new matrix of the same size.

`strassen_matrix_multiply` Function

This function is the core of the Strassen algorithm and uses several strategies to efficiently multiply two square matrices `A` and `B`.

Base Case

The recursion terminates when the matrix size reduces to 1 (i.e., $n = 1$), where it simply multiplies the single elements together. This base case is critical for stopping the recursion and starting the process of combining results.

Divide

The function splits each matrix into four sub-matrices using the `split_matrix` function. This step is essential for breaking down the large matrix multiplication problem into smaller, more manageable sub-problems.

Conquer: Strassen's Seven Multiplications

Strassen's algorithm significantly optimizes the multiplication process by only requiring seven multiplications of the sub-matrices, as opposed to eight, as seen in traditional divide-and-conquer approaches:

- **M1 to M7** are the seven products, calculated using strategic combinations of addition and subtraction among the sub-matrices. These multiplications effectively use the properties of matrix operations to reduce the overall computational load.

Combine

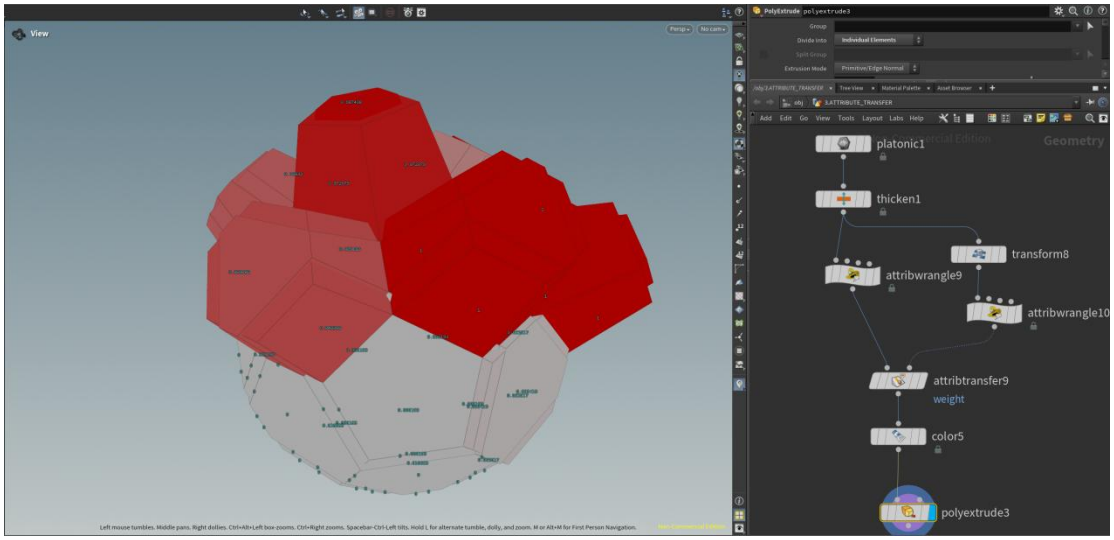
The results from the seven multiplicative operations are combined in a specific manner to form the final matrix `C`:

- **Formulation of C11, C12, C21, and C22**: These involve intricate additions and subtractions of the products `M1` through `M7` to assemble the correct entries for each quadrant of the result matrix.
- **Recomposition**: Using the `combine_matrix` function, these four sub-matrices are then pieced together to form the complete matrix `C`. This step is where the divided matrix parts are consolidated back into a full matrix.

Houdini Assignment

1 Houdini Fundamentals

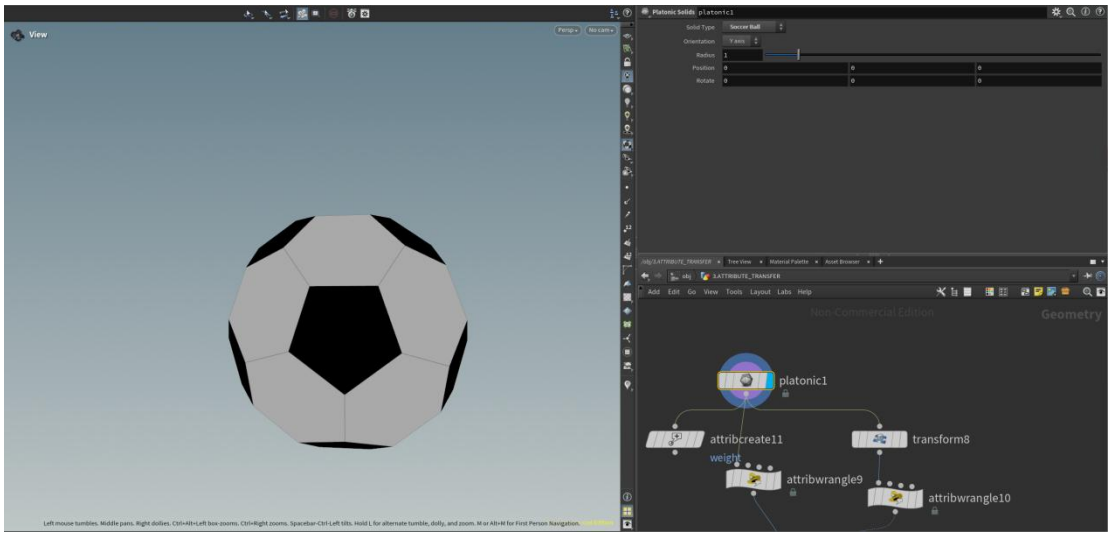
Example one:



1: Platonic Solids SOP

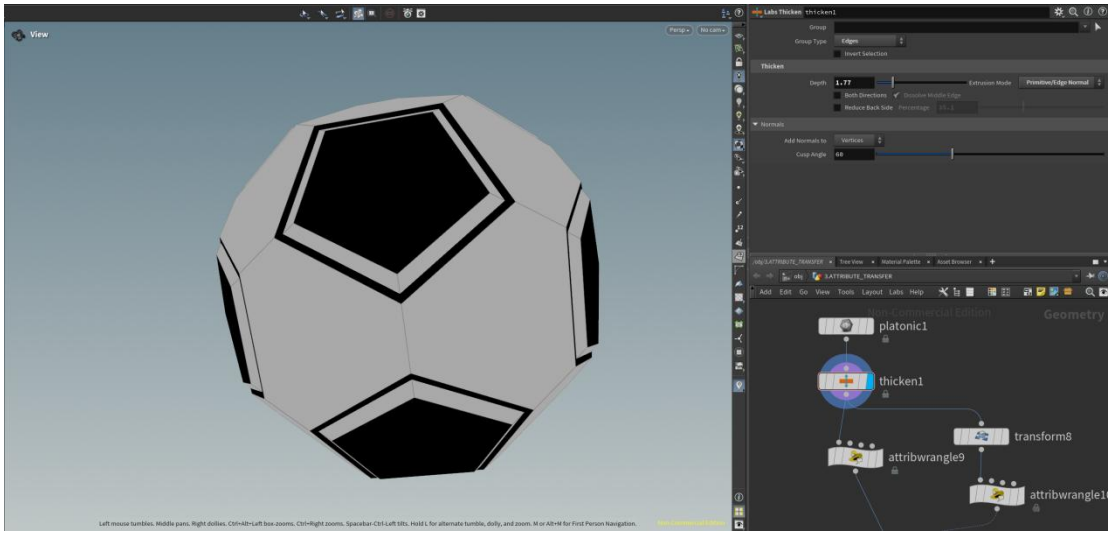
The Platonic Solids SOP generates platonic solids of different types. Platonic solids are polyhedrons which are convex and have all the vertices and faces of the same type. There are only five such objects, which form the first five choices of this operation.

This node can create seven different polyhedral forms such as Utah Teapot and Tetrahedron, I chose to create a Soccer Ball.



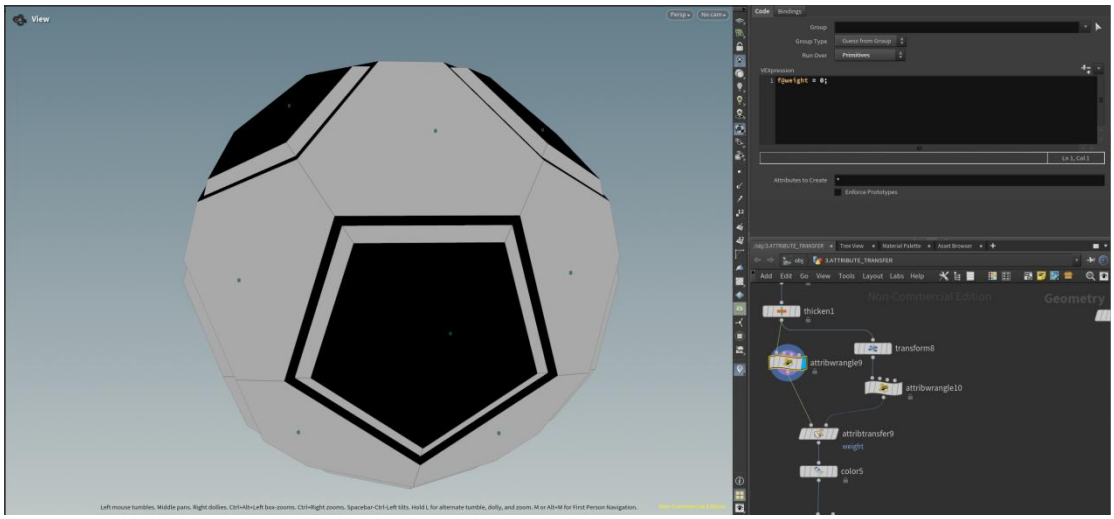
2: Labs Thicken SOP

I create a Labs Thicken node to be used as an auxiliary node that will extrude the mesh along its mean normal. We can adjust the parameter values from 0-10 to extrude the depth of the object.



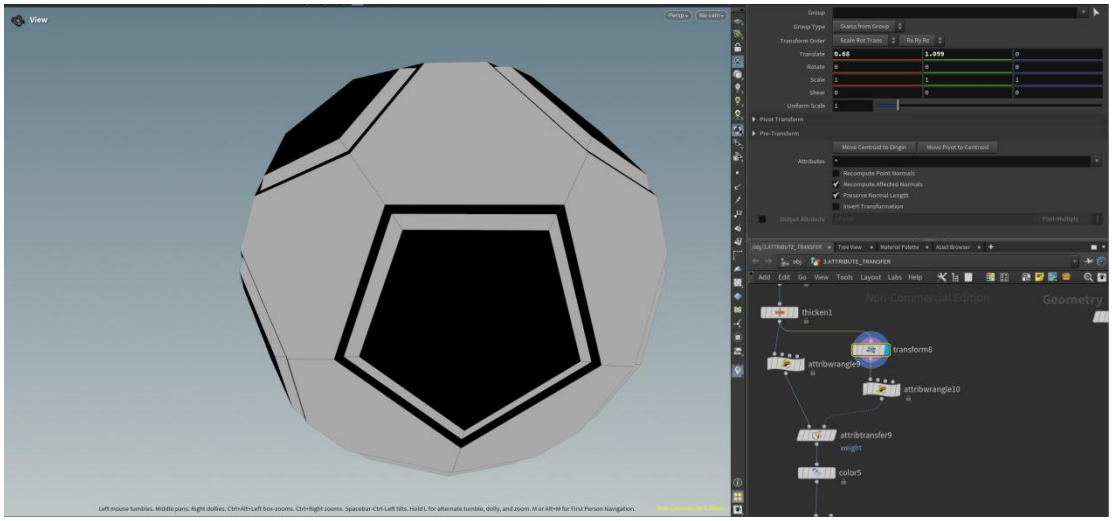
3: Attribute Wrangle SOP

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.



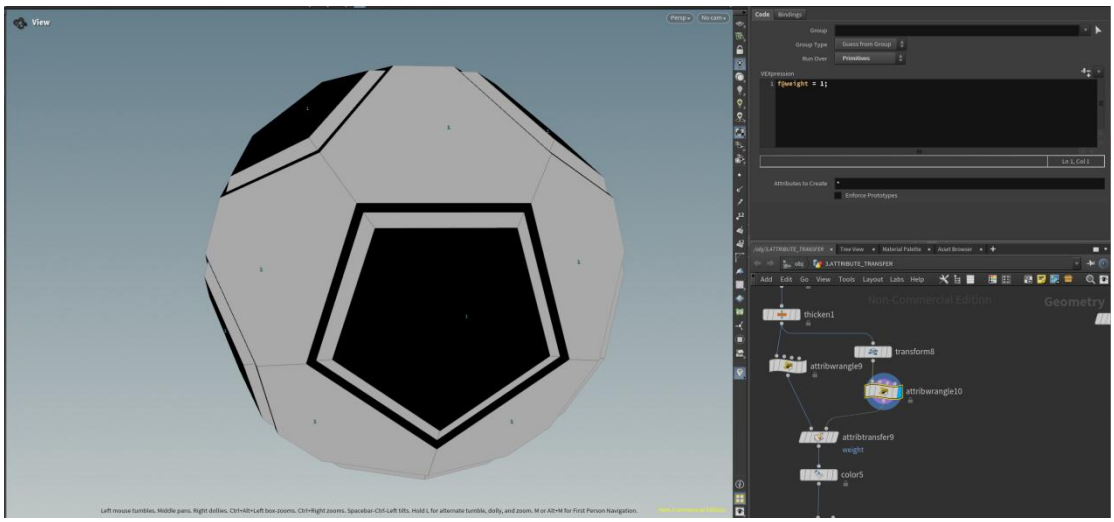
4: Transform SOP

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.



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.



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.

I made the following settings:

Kernel Radius (10): This value specifies the radius around each source element within which the transfer affects the destination elements. A larger radius results in attributes affecting a wider area of the destination geometry.

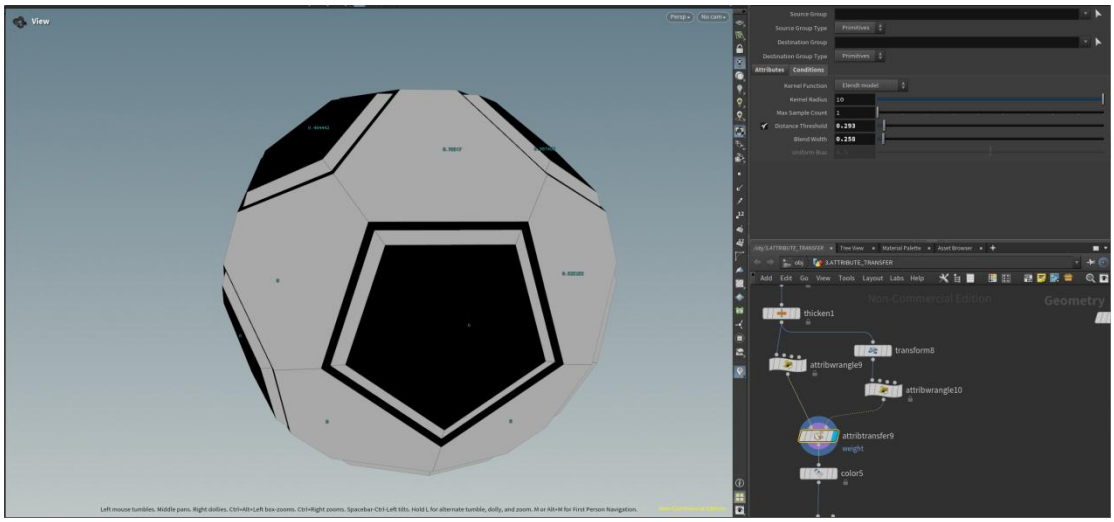
Max Sample Count (1): This controls the number of samples used in the attribute transfer calculation per destination element. Setting it to 1 means each destination element will consider the closest source element only, which might result in a more "hard-edged" transfer as opposed to a smoother blend that would come from considering multiple samples.

Distance Threshold (0.293): This parameter likely limits the maximum distance at which the

attribute transfer can occur. If the destination element is farther than this distance from any source element, the transfer may not affect it.

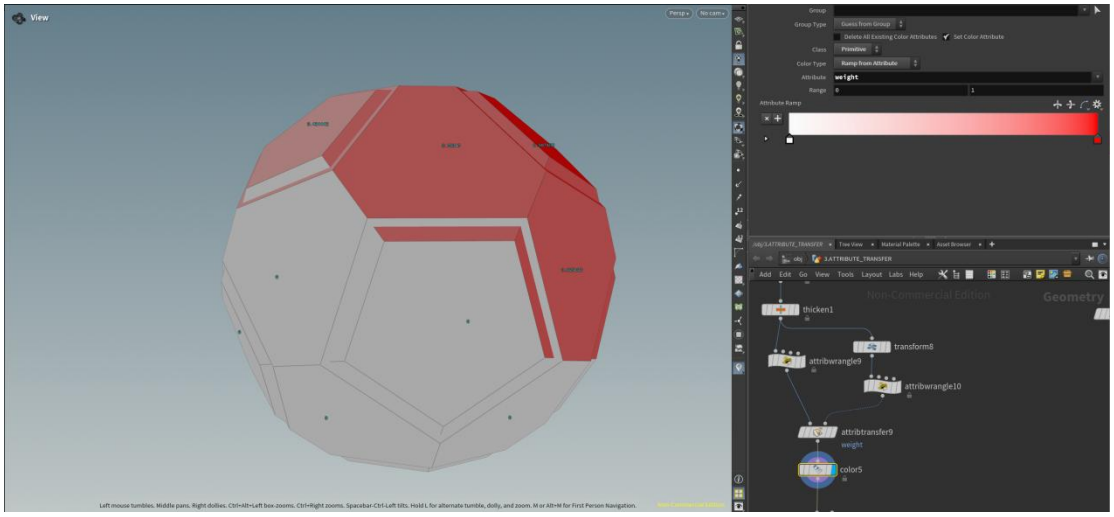
Blend Width (0.258): This determines the smoothness of the transition of the transferred attribute. A higher blend width results in a smoother blend, softening the boundaries where attributes change.

Uniform Bias (0.5): This setting might control the bias in the blending calculation between source and destination, affecting how the attribute values are averaged or weighted during the transfer.



7: Color SOP

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



8: PolyExtrude SOP

Using the “weight” value in d”distance scale”, I use the weight value as a multiplier of the extrusion value.

I made the following settings:

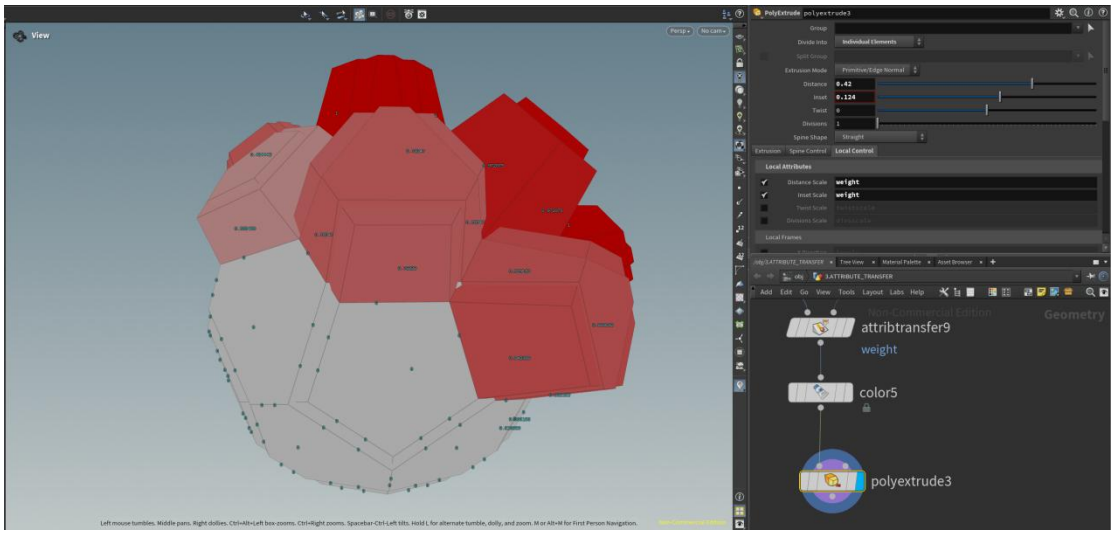
Distance (0.42): This setting controls how far the polygons are extruded from their original

surface. A distance of 0.42 units means each selected polygon face will be pushed outwards by this amount.

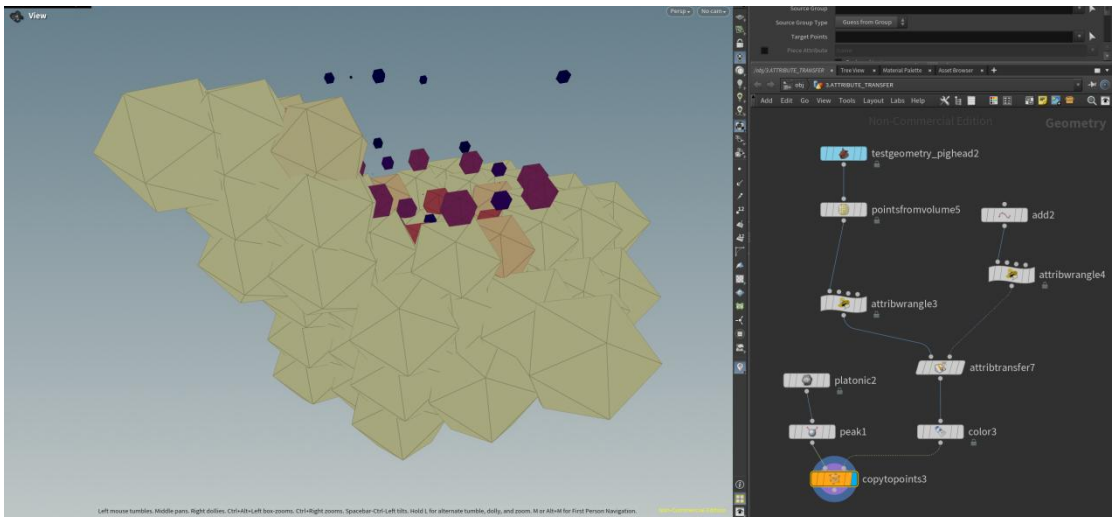
Inset (0.124): The inset value dictates how much the extruded faces shrink inward from their original edges as they are extruded. An inset of 0.124 units will reduce the size of each extruded face, creating a kind of frame around the extruded area, which is typical for adding detail or a layered effect.

Twist (0): This parameter would allow the extrusion to rotate around the normal axis by a specified degree.

Divisions (1): This controls the number of segments in the extrusion. Setting it to 1 means there is only one segment, thus no subdivision occurs along the extrusion path. Increasing this value would create more geometry along the extrusion, useful for more complex bending or twisting effects.



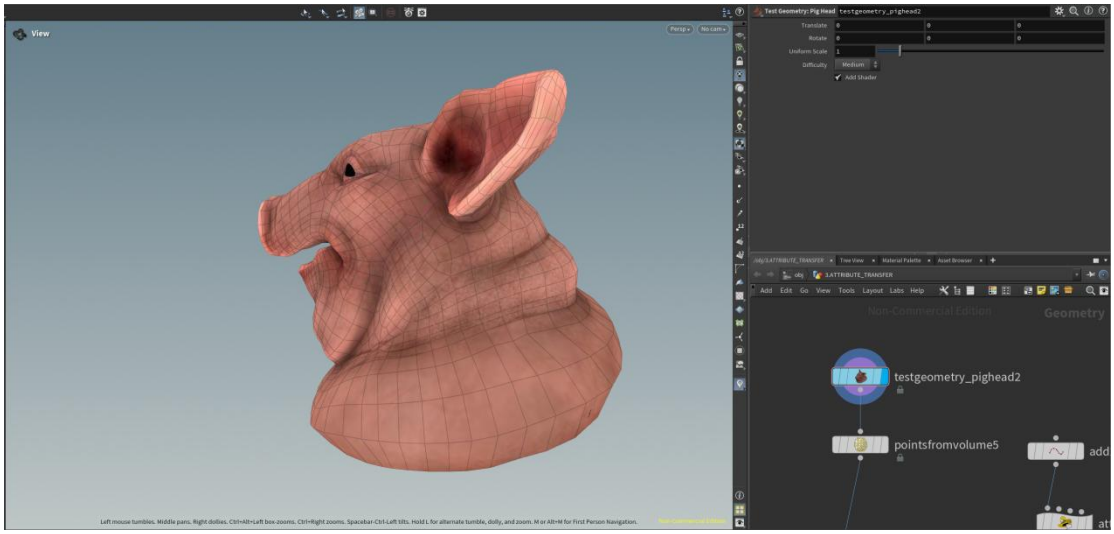
Example two:



1: testgeometry_pighead

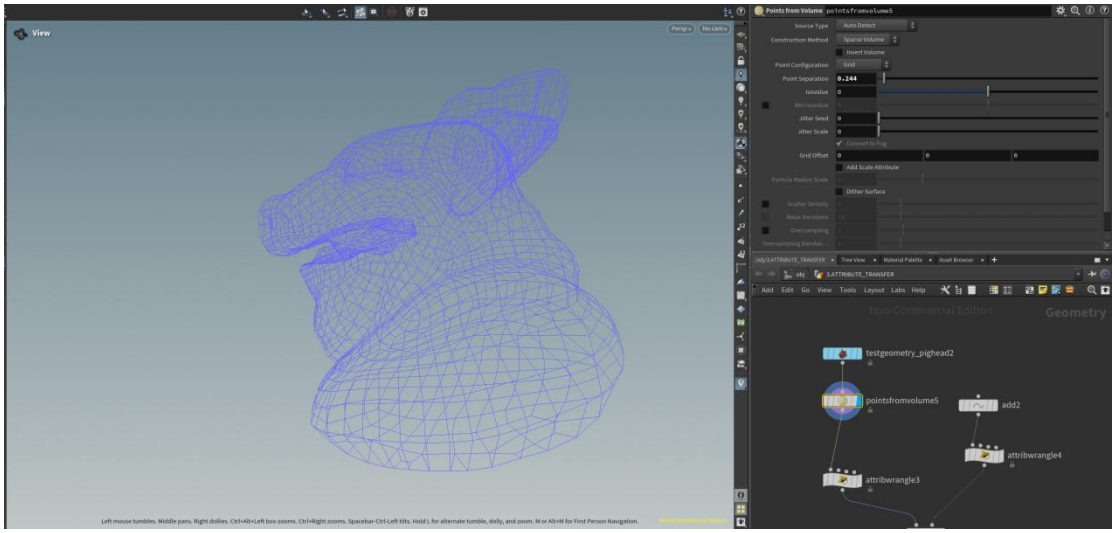
I create a pig head, which can be used as test geometry. The “testgeometry_pighead” node is a

commonly used node for testing and learning purposes as it generates a simple pig head model.



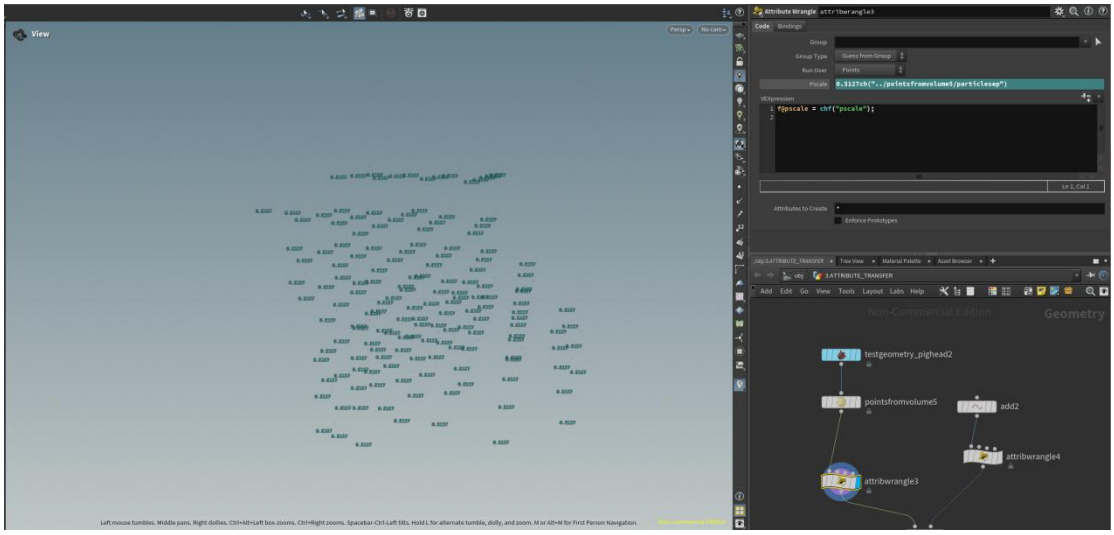
2: pointsfromvolume

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.



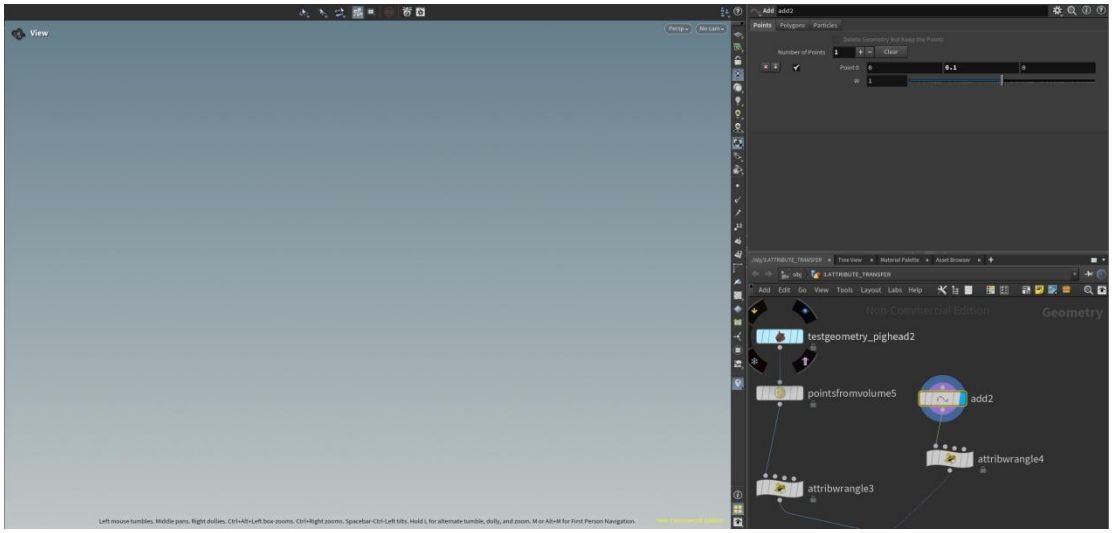
3: attribwrangle

This Attribute Wrangle node is primarily set up to manipulate attributes on points using VEX code. The node's code section contains one line of VEX expression:float pscale = chf("pscale"); This code retrieves a floating-point parameter named pscale from the node's parameter interface and assigns its value to an attribute also named pscale. In the node interface, the pscale value is controlled by a slider, currently set to 0.3127. This means that all processed points will receive this pscale value as their attribute, which is typically used to control the scale or size of the points, or other scale-related effects.



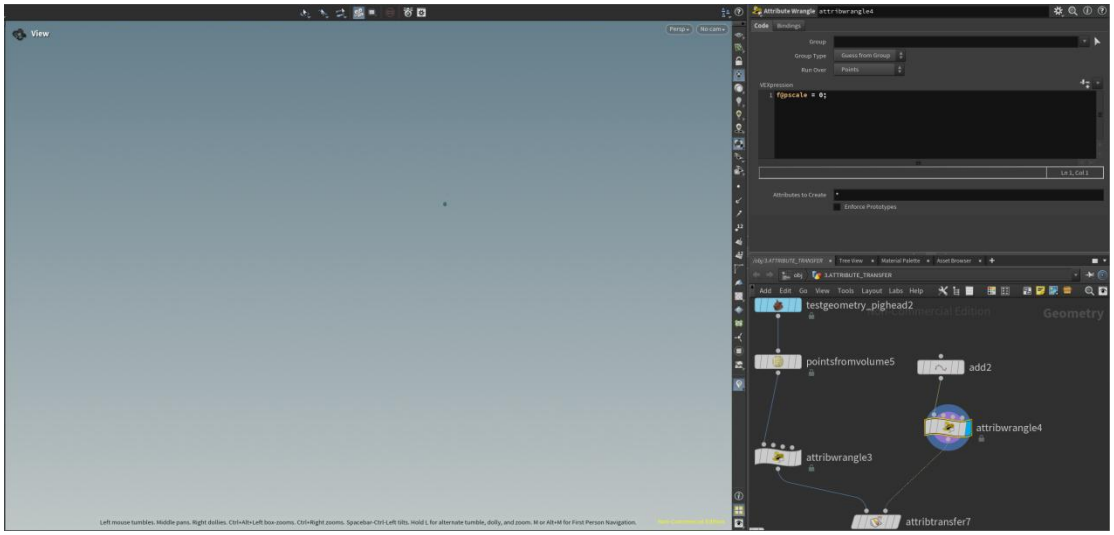
4: add

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.



5: attribwrangle

In the Attribute Wrangle node setup, I did a simple attribute manipulation based on the VEX programming language. This line of code: f@pscale = 0 sets the “pscale” attribute of all processed points to 0. In Houdini, the pscale attribute is commonly used to control the size or scale of points, such as in particle systems or when rendering instances. Setting it to 0 typically means that these points will not be visually represented, or their effect is negated.



6: attribtransfer

The “Attribute Transfer” node setting shows how to transfer attributes between geometries. Kernel Function: Set to “Eldendt model” which is a weight model used to determine how attributes are blended.

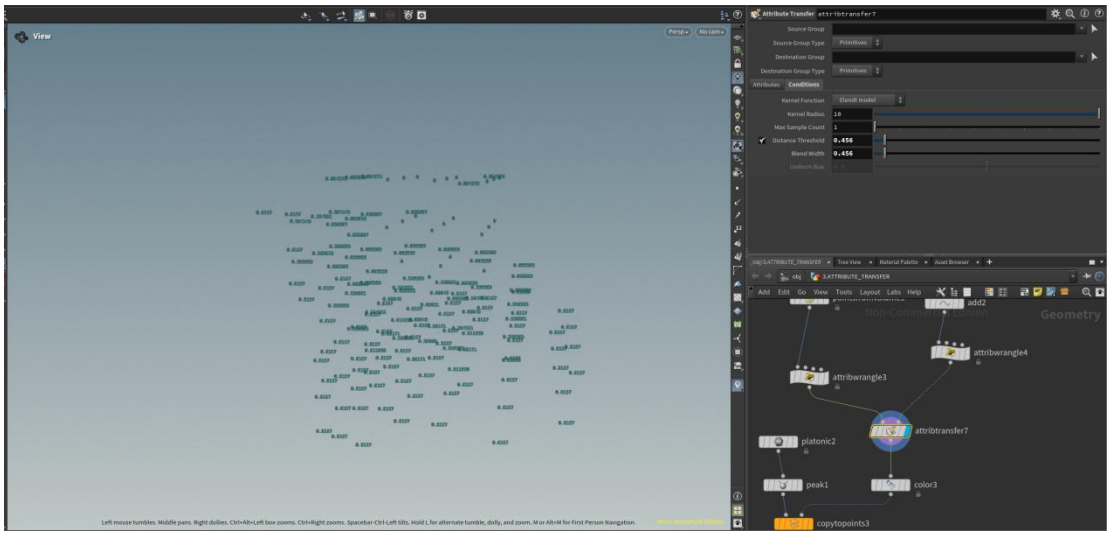
Kernel Radius: Set to 10, which dictates the proximity considered during attribute transfer.

Max Sample Count: Set to 1, indicating that each destination primitive can receive attributes from at most one source primitive.

Distance Threshold: Set to 0.456, determining the maximum distance threshold for attribute transfer.

Blend Width: Also set to 0.456, used to determine the width of the attribute value blending.

Uniform Bias: Set to 0.5, used to adjust the bias in blending attributes.

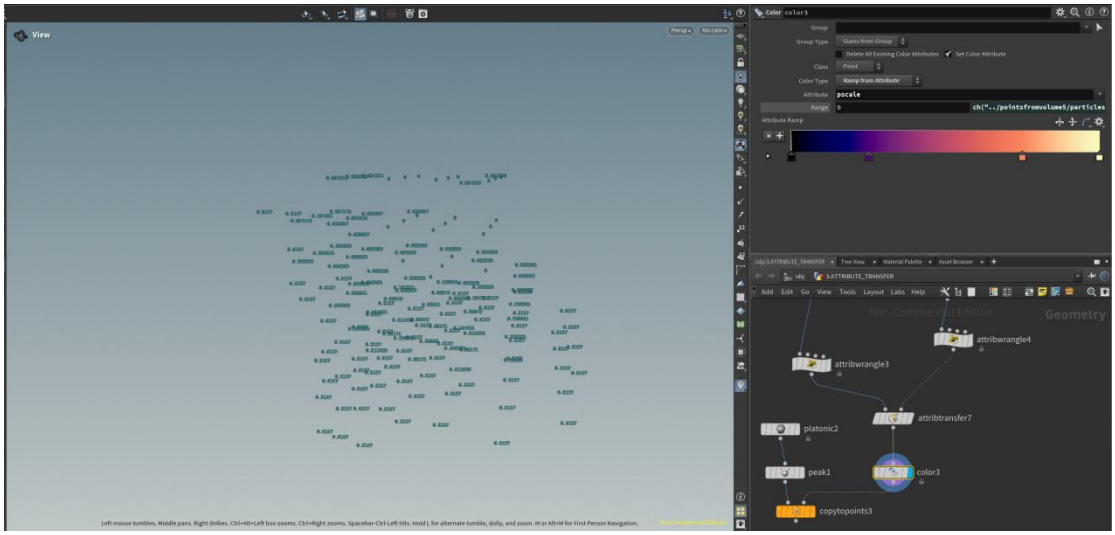


7: color

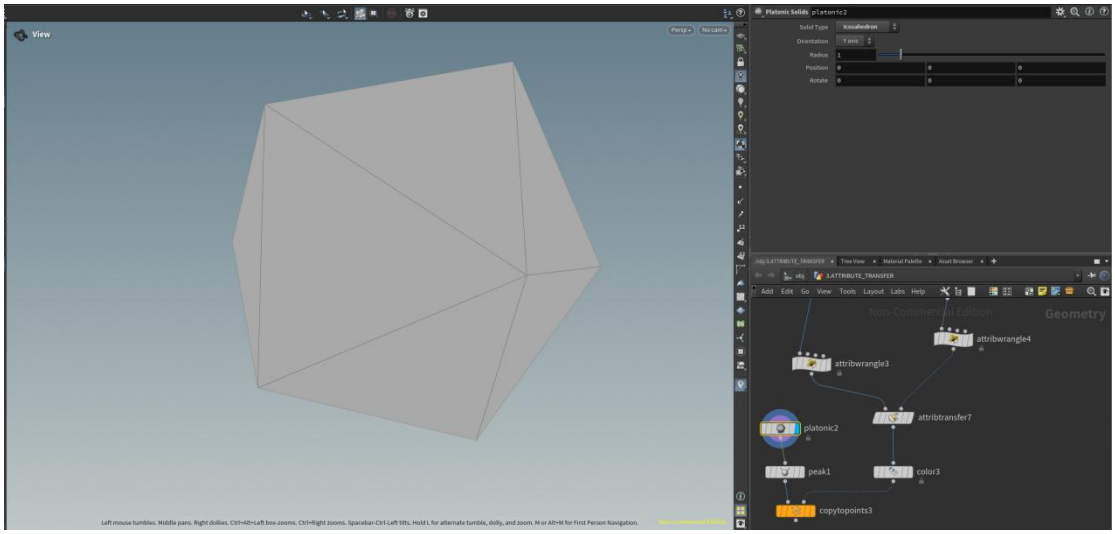
This “Color” node is used to color geometry based on the pscale attribute of points. The settings are as follows:

Group Type: Chosen “Guess from Group,” which automatically infers the geometry group being processed.

Set Color Attribute: This option is checked, indicating that the newly calculated color will be set as the color attribute of the points.
Class: Set to “Point” meaning the color attribute will apply to individual points.
Color Type: Selected “Ramp from Attribute” which means colors are determined based on attribute values. The color is mapped through a color ramp based on the value of the pscale attribute.
Attribute: Specifies pscale as the attribute used for generating colors.
Attribute Ramp: Displays a color ramp, where the colors of points can be adjusted based on the value of pscale. The colors range from purple to red to yellow, allowing visual differentiation of points with different pscale values.

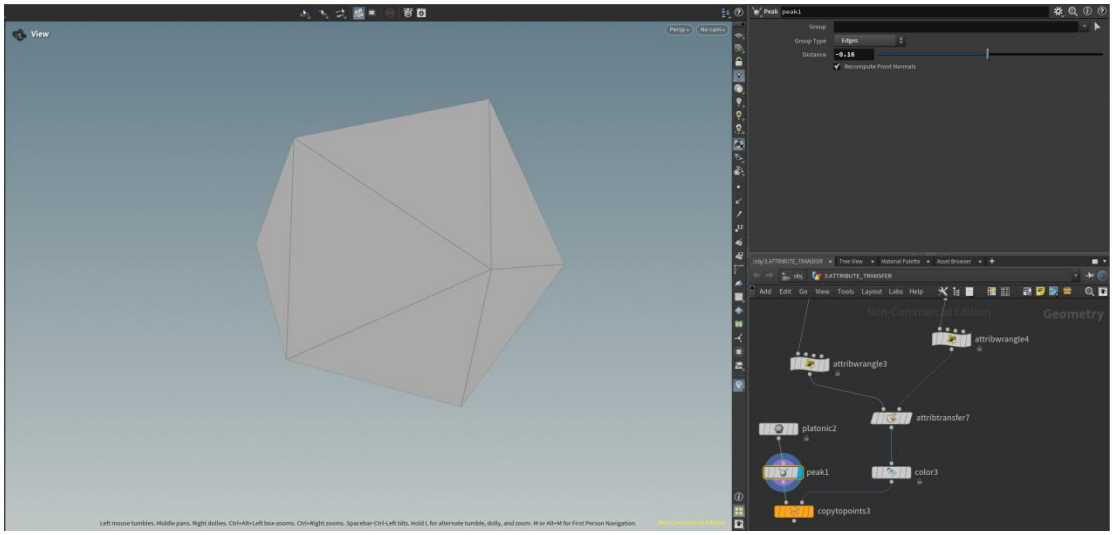


8: platonic
The “Platonic Solids” node is used to generate various Platonic solids, I chose to create a “Icosahedron" which is a complex polyhedron consisting of 20 equilateral triangular faces.



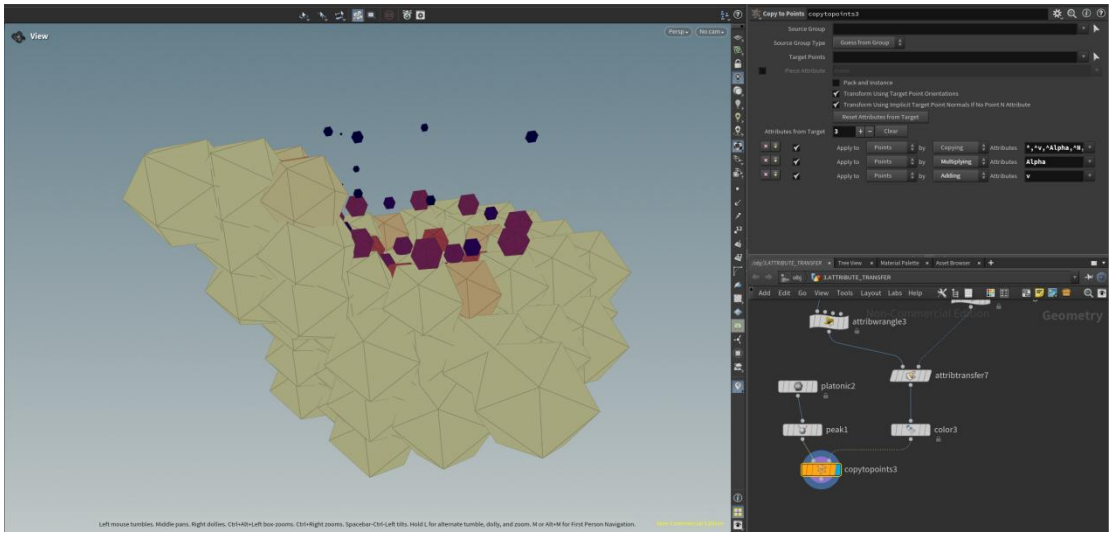
9: peak
The “Peak” node is used to adjust the edges or vertices of the geometry to change the shape by

moving them. I set Distance to -0.16, which means that the selected edges will move inward by 0.16 units along their normal.



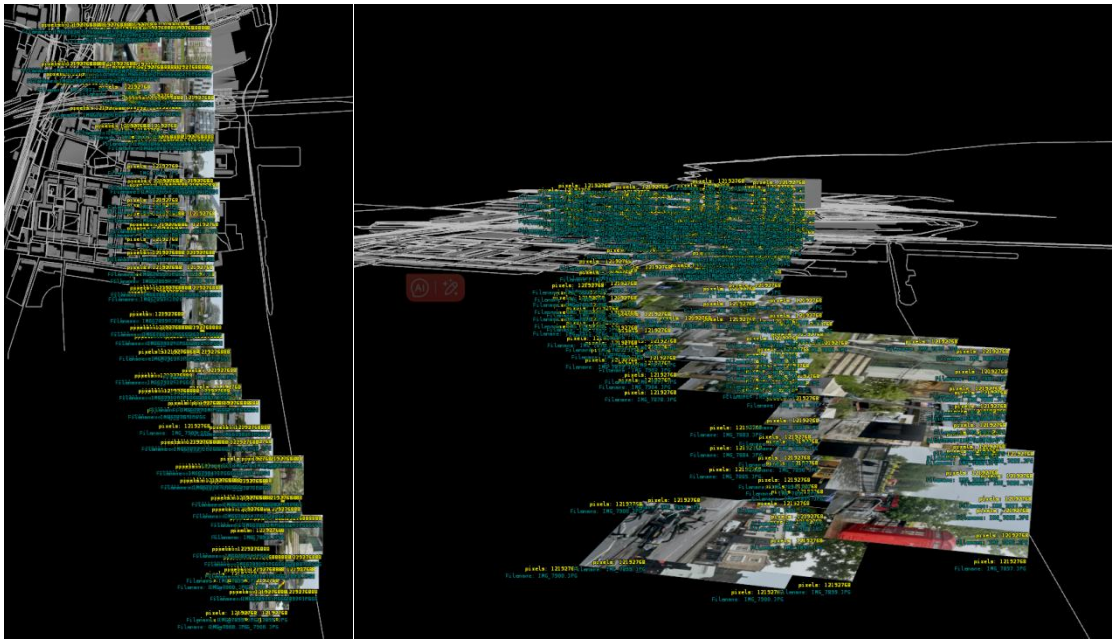
10: copytopoints

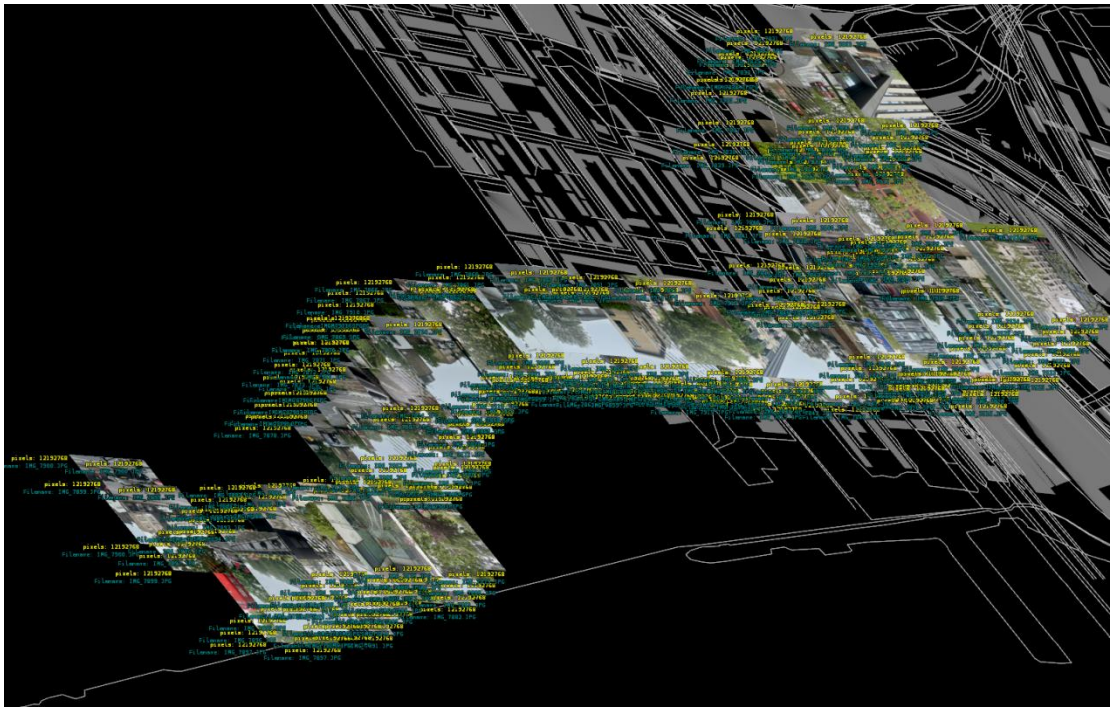
The “Copy to Points” node setting is used to copy the source geometry to the target points while allowing properties to be carried and modified while copying.

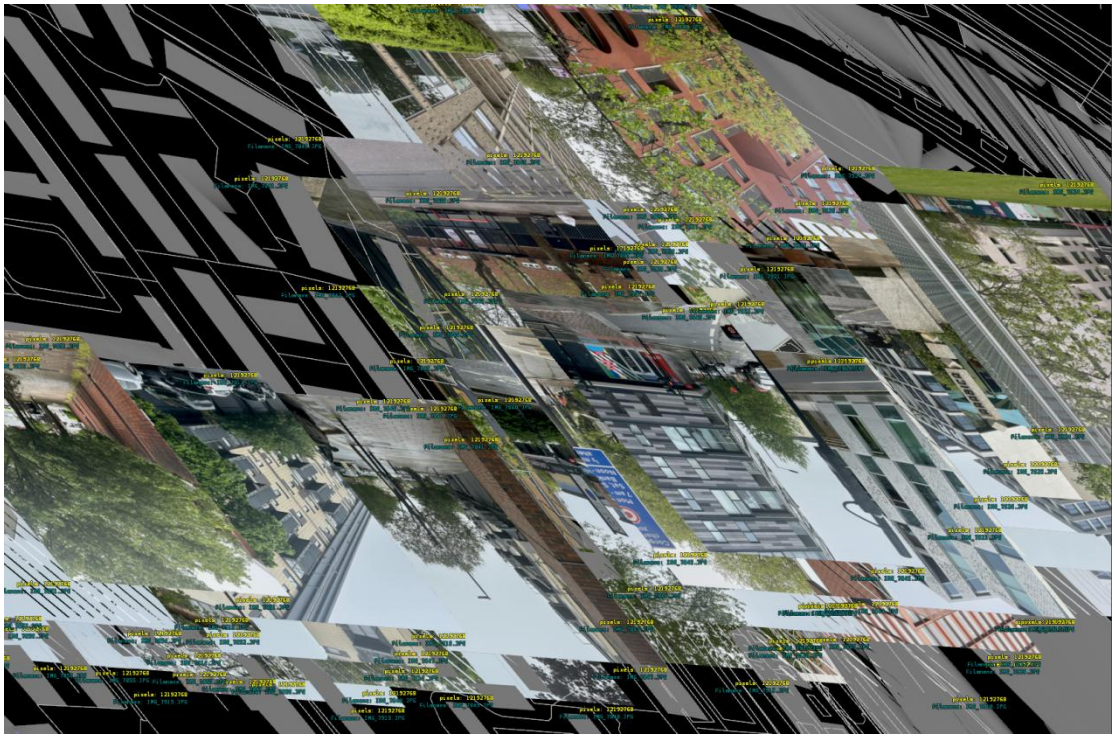


2 Visualizing GPS and Image Metadata

Firstly, I exported the kml file from google map, then converted the kml file to gpx file and imported it into houdini. Then I downloaded the osm file from Openstreetmap, and finally imported the pictures with location information from my mobile phone into houdini, generating a series of pictures depicting my travel track.

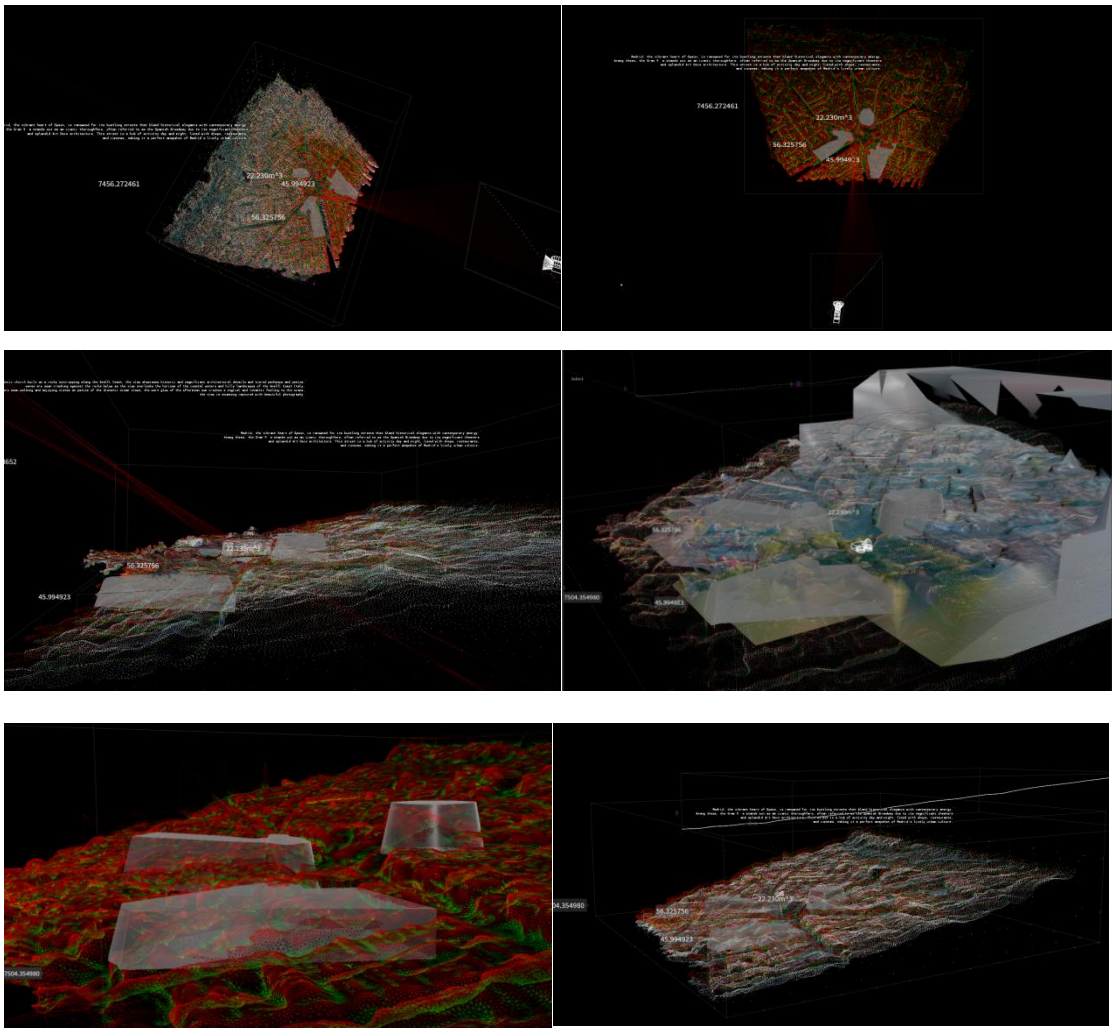






3 video to 3D Model

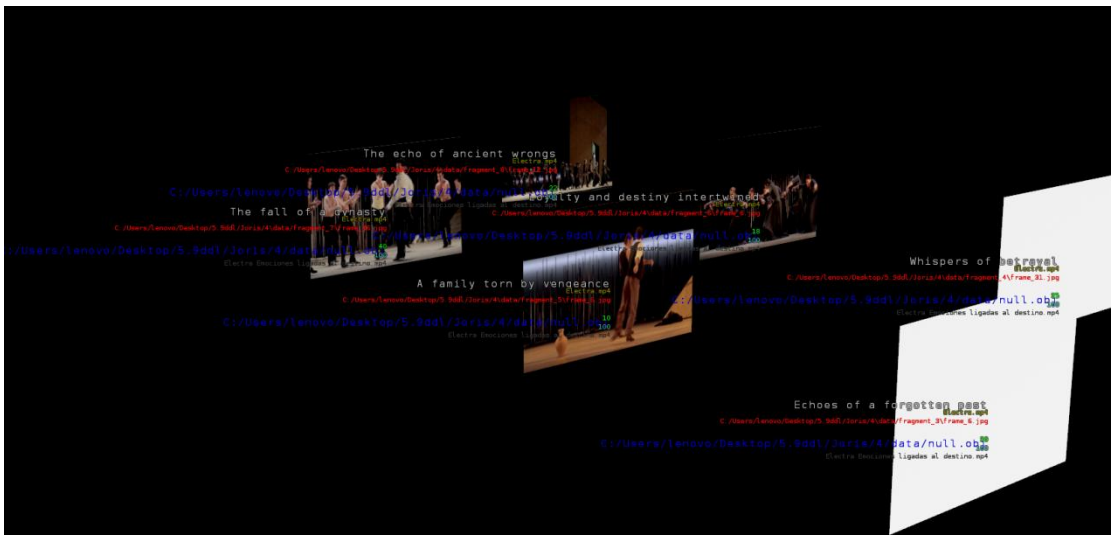
The video was first broken down into a series of image frames, which were then used to generate a 3D model using RealityCapture. The 3D model was then imported into Houdini and processed, with the camera paths reconstructed and textures and details added to make the model more lifelike. Then added volume speculation, curvature, an additional 3D model to the subject. Finally visualised the model as a collection of images.

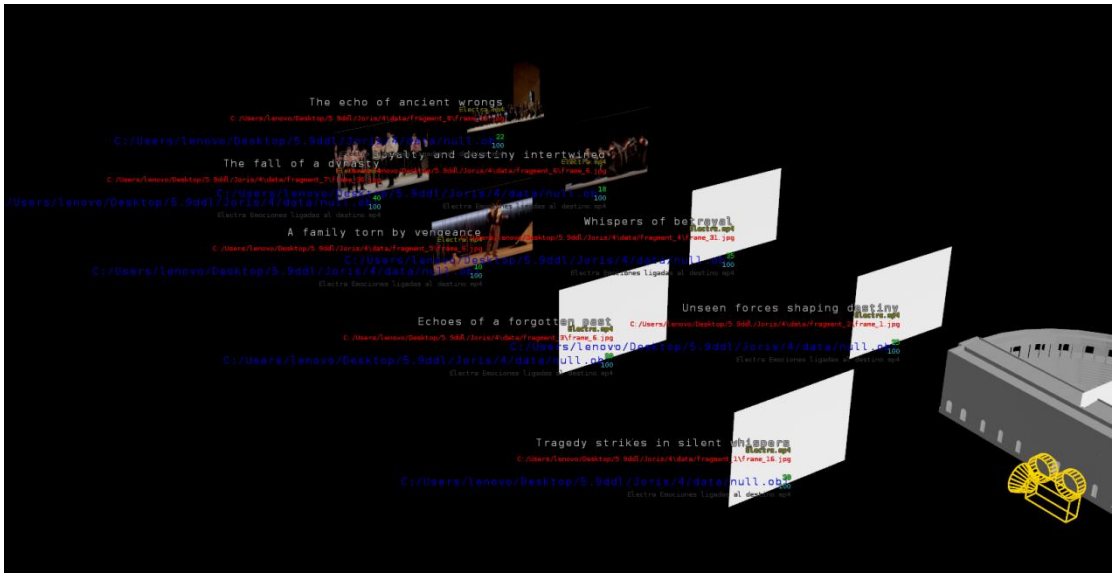


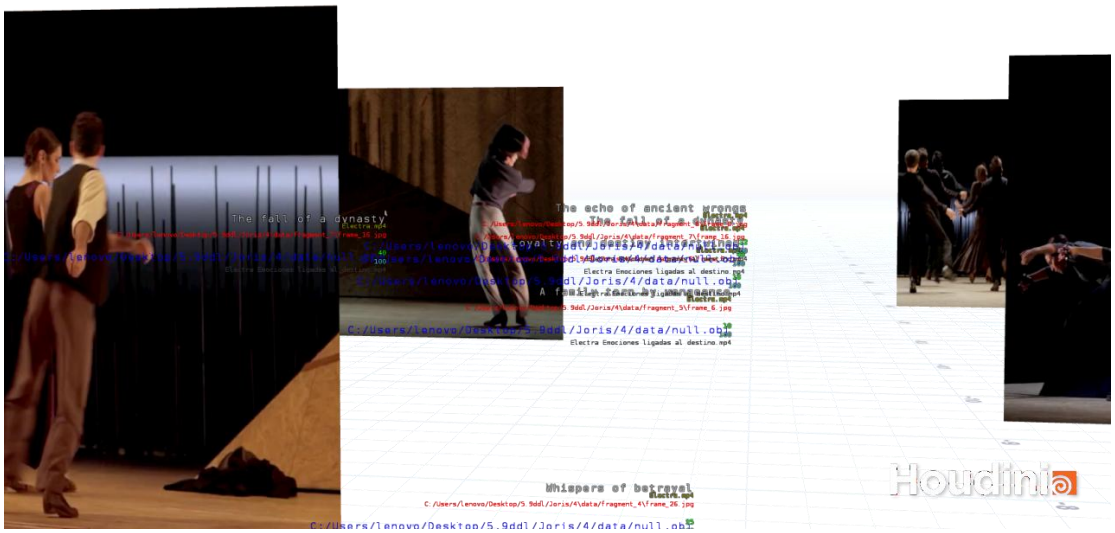
4 Visualizing JSON in Houdini

Firstly I extracted the theatre video associated with my project into a series of image frames and imported them into houdini. A Python script was then used in Houdini to read a JSON file, where the JSON file included the frame fragment information. Then we downloaded a model of the theatre from the sketchfab website, imported the model into blender and converted it to obj format and imported it into houdini. Finally set up the camera path.


```
  "paragraph": "A family torn by vengeance",
  "filmID": "Electra.mp4",
  "image name": "frame",
  "model path": "",
  "image n": 10,
  "time": 100,
  "film path": "Electra Emociones ligadas al destino.mp4"
},
{
  "paragraph": "Loyalty and destiny intertwined",
  "filmID": "Electra.mp4",
  "image name": "frame",
  "model path": "",
  "image n": 18,
  "time": 100,
  "film path": "Electra Emociones ligadas al destino.mp4"
},
{
  "paragraph": "The fall of a dynasty",
  "filmID": "Electra.mp4",
  "image name": "frame",
  "model path": "",
  "image n": 40,
  "time": 100,
  "film path": "Electra Emociones ligadas al destino.mp4"
},
{
  "paragraph": "The echo of ancient wrongs",
  "filmID": "Electra.mp4",
  "image name": "frame",
  "model path": "",
  "image n": 22,
  "time": 100.
```





onedrive link:
https://1drv.ms/f/c/860016f165a50123/EpWao2ofV21HjzUyeRMIAH0B2hy12LBDuGFkH78AscSlZw?e=hcNsTI