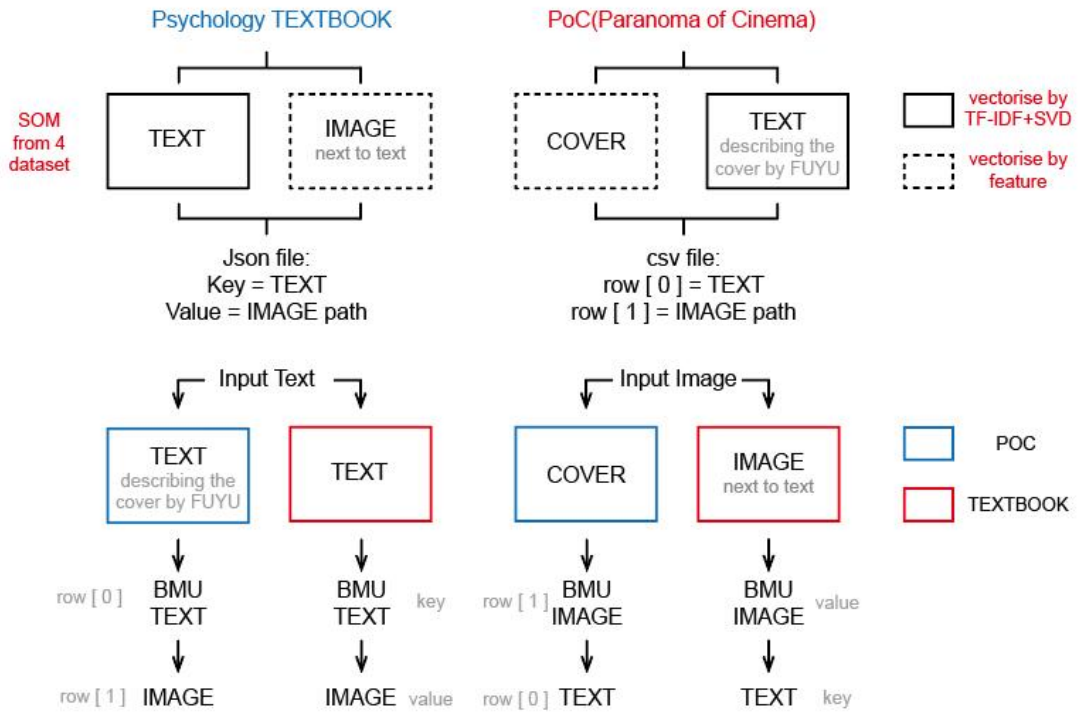
ONE DRIVE LINK:[RC11_231996457_SKILL]

Explanation and Outcomes the SOM final assignment

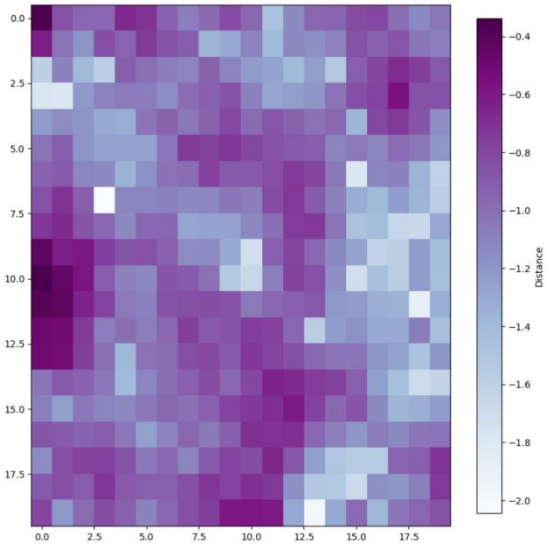
For the assignment, I mainly created a search engine for text and images, utilizing data from two sources, textbook and panorama of cinema. The process flow of making the search engine is as follows



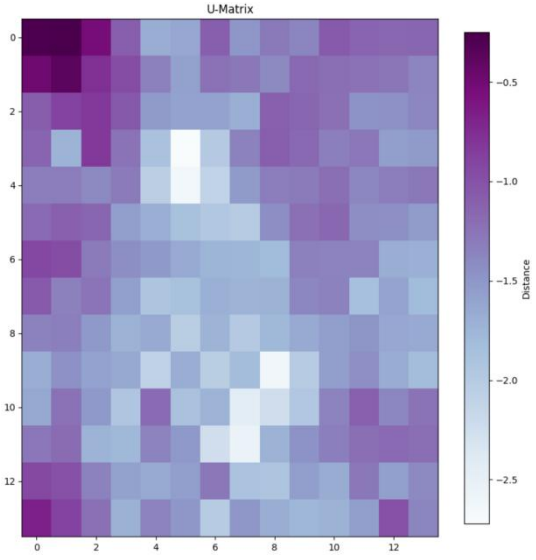
1. Four databases are formed through two sources, textbook and POC, which are textbook's text, the picture on the text page, POC's movie cover, and the text generated from the cover through FUYU. A total of two text map generated as well as two image library, text library with TF-IDF + SVD for vectorization, image library with feature vectorization, to generate four SOM.
2. TEXTBOOK text and image formed a JSON file, text is the key, image_path is the value. POC's cover and FUYU's text formed a csv file, text is the ROW[0], image_path is the ROW[1], this is done in order to get a later data can be found in addition to the way through the corresponding search a different type of data.
3. Now got four SOM, and two file used to correspond to, WE can start the search, when input text ,we can get BEST MATCHING text through the two text SOM.When the BMU text in POC FUYU SOM is find, we can find the corresponding line of the image_path in csv file to get the cover. When the BMU text in TEXTBOOK TEXT SOM is find, we can find the key to find the value, in the picture library is also the same principle. So we can get four results by inputing one element.

Specific explanation and outcomes:

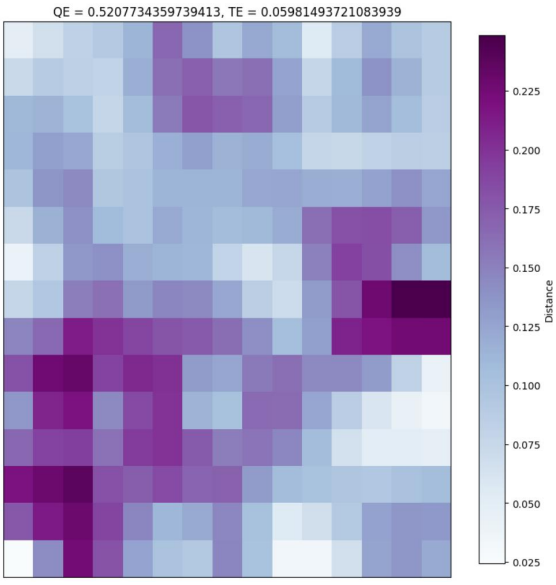
1. BUILD 4 SOM



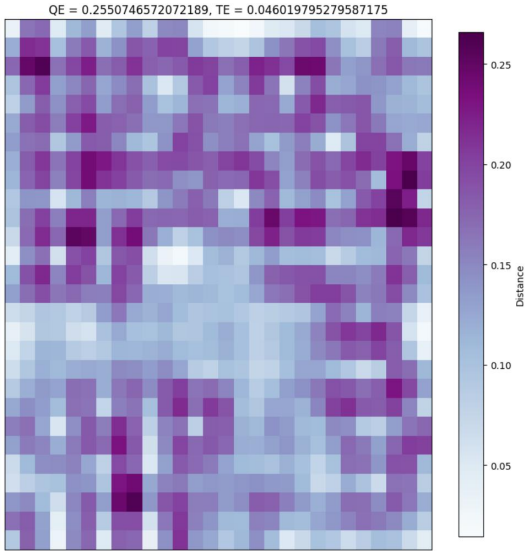
POC_COVER_IMAGE_SOM



TEXTBOOK_IMAGE_SOM



POC_FUYU_TEXT_SOM



TEXTBOOK_TEXT_SOM

1.1 BUILD TEXT SOM

Take POC_FUYU_TEXT_SOM as an example TEXTBOOK_TEXT_SOM has the same principle

https://github.com/UD-Skills-2023-24/github-codebook-RC11-23196457/blob/main/00%20SOM/POC_FUYU_TEXT_SOM.ipynb

1. IMPORT LIBRARY AND DEFINE THE FUNCTIONS

   imports all the libraries and defines all the functions needed. It is important to note here that the text of TEXT needs to be vectorized with the cosine distance, which overcomes the effects of length differences and data sparsity and focuses more on the content and subject matter of the text than on the form or other external factors.

```python
def cosine(a,b):
    return cosine_similarity([a], [b])[0][0]
```

```python
def find_BMUcos(SOM,x):
    somShape = SOM.shape
    simSOM = SOM.reshape((-1, len(x)))
    cos_sims = cosine_similarity([x], simSOM).reshape(somShape[:2])
    return np.unravel_index(np.argmax(cos_sims, axis=None), cos_sims.shape)
```

```python
def find_BMU_2(SOM,x):
    somShape = SOM.shape
    simSOM = SOM.reshape((-1, len(x)))
    cos_sims = cosine_similarity([x], simSOM).reshape(somShape[:2])
    return np.unravel_index(np.argpartition(cos_sims, -2, axis=None)[-2], cos_sims.shape)
```

```python
def find_closest_cos(data, v):
    cos_dist = cosine_similarity(data, [v])
    return np.argmax(cos_dist)
```

```python
def activate(train_data, SOM, p):
    #normalP = normalise(train_data, p)
    activatedSOM = np.array([[cosine_similarity(p, [c])[0][0] for c in r] for r in SOM])
    #normalisedActivatedSOM = normalise(activatedSOM, activatedSOM)
    activatedSOM = (activatedSOM -1)*(-1)
    return activatedSOM
```

Python

2. TEXT PROCESS

   For the text vectorization method I chose the TFIDF+SVD method, because in the testing process, the result of choosing the TF-IDF+SVD method is closer to the search sentence than the doc2vec vectorized text.

   When I search 'a man and a woman are talking to each other'

   The result of TF-IDF+SVD is 'There are a man and a woman in the image.'

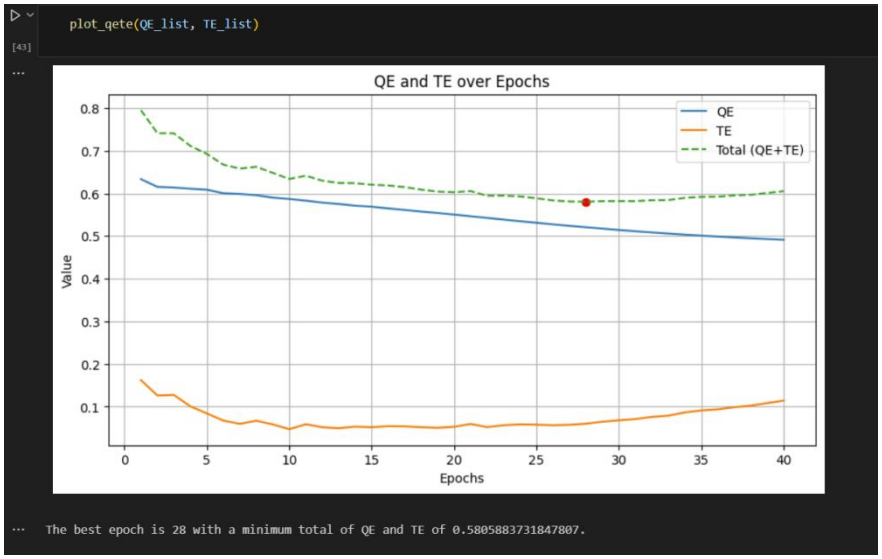
   The result of Doc2Vec is 'In the image, there is a baseball player wearing a baseball uniform with the number 12 on his shirt. He is standing next to a baseball bat, ready to hit the ball. The scene also features a baseball glove, which can be seen on the ground nearby.'

   Obviously easy to see that TF-IDF+SVD works better

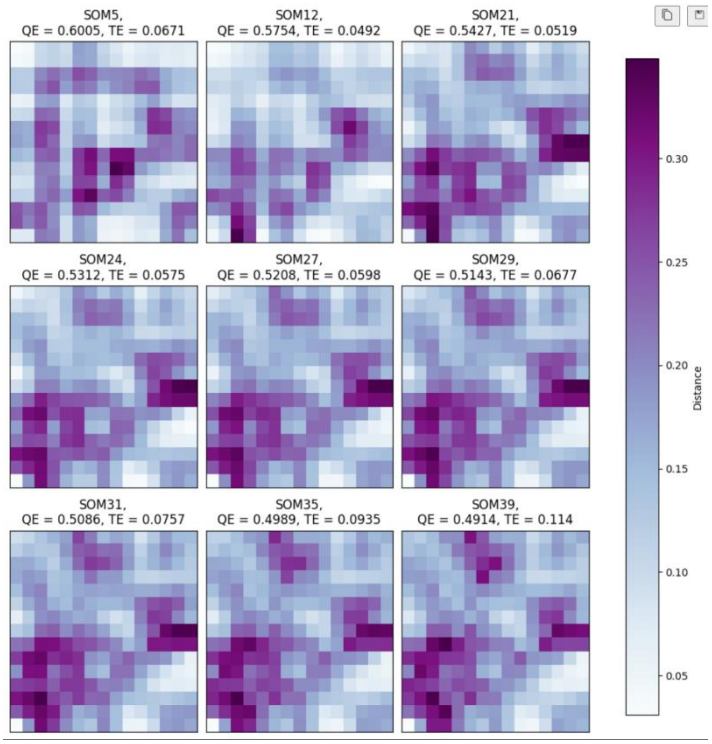
3. TRAIN SOM

```python
def plot_qete(qe_values, te_values):
    total_values = np.array(qe_values) + np.array(te_values)
    best_epoch = np.argmin(total_values) + 1

    epochs = len(qe_values)
    plt.figure(figsize=(10, 5))
    plt.plot(np.arange(1, epochs + 1), qe_values, label='QE')
    plt.plot(np.arange(1, epochs + 1), te_values, label='TE')
    plt.plot(np.arange(1, epochs + 1), total_values, label='Total (QE+TE)', linestyle='--')
    plt.scatter([best_epoch], [total_values[best_epoch-1]], color='red')
    plt.xlabel('Epochs')
    plt.ylabel('Value')
    plt.title('QE and TE over Epochs')
    plt.legend()
    plt.grid(True)
    plt.show()

    print(f"The best epoch is {best_epoch} with a minimum total of QE and TE of {total_values[best_epoch-1]}.")
```

In the SOM training process, PLOT_QETE function are added, saves lists containing qe value and te value, you can directly see the qe te with epoch changes, in addition, when the QE TE are relative minimum, that is, they add up to the smallest value, then the SOM is the best training, in the function at the same time also added the code to get the best SOM.



Here it shows that the best SOM is the 48, and according to the code order rule SOM_list[27] is the 28th, so SOM_list[27] is selected as the best SOM.




```
word_counts = [[0 for _ in range(n.shape[1])] for _ in range(n.shape[0])]

for i in range(n.shape[0]):
    for j in range(n.shape[1]):
        word_counts[i][j] = len(word_mapping[(i, j)])

for row in word_counts:
    print(row)
```

Python

```
[80, 17, 15, 40, 12, 42, 21, 45, 18, 22, 49, 9, 30, 8, 50]
[32, 32, 12, 30, 20, 16, 6, 21, 8, 5, 20, 7, 8, 8, 18]
[18, 8, 20, 35, 18, 14, 7, 18, 5, 10, 19, 2, 5, 7, 22]
[40, 13, 15, 24, 16, 15, 16, 24, 26, 9, 12, 10, 24, 6, 12]
[17, 5, 9, 12, 15, 20, 32, 9, 7, 5, 11, 8, 4, 2, 8]
[31, 15, 8, 36, 5, 14, 6, 4, 10, 6, 6, 7, 6, 5, 13]
[37, 17, 3, 9, 31, 4, 20, 11, 12, 19, 8, 6, 37, 3, 22]
[28, 19, 26, 1, 4, 17, 1, 10, 10, 12, 9, 0, 13, 3, 8]
[15, 7, 33, 20, 30, 7, 6, 7, 7, 13, 9, 25, 0, 0, 0]
[50, 7, 0, 2, 5, 0, 18, 10, 7, 10, 0, 1, 6, 8, 26]
[14, 19, 7, 31, 21, 0, 20, 10, 5, 4, 3, 10, 10, 4, 9]
[50, 10, 2, 5, 12, 0, 10, 6, 5, 3, 5, 9, 7, 6, 6]
[0, 4, 46, 4, 4, 23, 4, 4, 11, 4, 5, 0, 2, 1, 13]
[18, 3, 0, 25, 8, 5, 10, 1, 10, 2, 6, 0, 3, 0, 0]
[48, 25, 0, 35, 11, 13, 43, 0, 9, 15, 22, 7, 13, 0, 30]
```

```
data_dict = []
for i in range(len(som_poctextmodel)):
    row = []
    for j in range(len(som_poctextmodel[0])):
        row.append([])
    data_dict.append(row)
vectortextPairs=[]
for i in range(0,len(text_list)):
    vectortext={}
    vectortext['text']=text_list[i]
    vectortext['vector']=svd_matrix[i]
    vectortextPairs.append(vectortext)
for i in vectortextPairs:
    g,h = find_BMUcos(som_poctextmodel,i['vector'])
    data_dict[g][h].append(i)
```

```
with open('F:\\SOM_skill\\film\\POC_text_data_dict.pkl', 'wb') as f:
    pickle.dump(data_dict, f)
```

```
with open('F:\\SOM_skill\\film\\POC_text_data_dict.pkl', 'rb') as f:
    POC_text_data_dict = pickle.load(f)
```

Then save the TEXT contained after each unit, once again later search for BMU (beat matching unit) call.

1.2 BUILD IMAGE SOM

Take POC_COVER_IMAGE_SOM as an example TEXTBOOK_SOM_IMAGE_SOM has the same principle

https://github.com/UD-Skills-2023-24/github-codebook-RC11-23196457/blob/main/00%20SOM/POC_COVER_IMAGE_SOM.ipynb

1. IMPORT LIBRARY AND DEFINE THE FUNCTIONS

   imports all the libraries and defines all the functions needed. The important point to note here is that for images you need to normalize during vectorization, normalization ensures that each feature contributes equally to the processing. In the case of images, this means that individual pixels will not have different brightness or color intensities that affect the overall result.

```python
def normalise(train, p):
    min_d = np.min(train)
    max_d = np.max(train)
    normalised_p = (p-min_d)/(max_d - min_d)
    return normalised_p
```

   The Euclidean distance is also used in the computation, and when the data is normalized, the Euclidean distance can be used to compare different images more efficiently because it eliminates the effects of differences in the scale of the data.

```python
def euclidean(a, b):
    return np.linalg.norm(a-b)
```

```python
def activate(train_data, SOM, p):
    normalP = normalise(train_data, p)
    activatedSOM = np.array([[euclidean(normalP, c) for c in r] for r
    normalisedActivatedSOM = normalise(activatedSOM, activatedSOM)
    activatedSOM = (normalisedActivatedSOM -1)*(-1)
    return activatedSOM
```

2. IMAGE PROCESS

```python
def load_image(img_file, target_size=(224,224)):
    X = np.zeros((1, *target_size, 3))
    X[0, ] = np.asarray(tf.keras.preprocessing.image.load_img(
        img_file,
        target_size=target_size)
    )
    X = tf.keras.applications.mobilenet.preprocess_input(X)
    return X

def ensure_folder_exists(folder):
    if not os.path.exists(folder):
        os.makedirs(folder)
```

Python

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

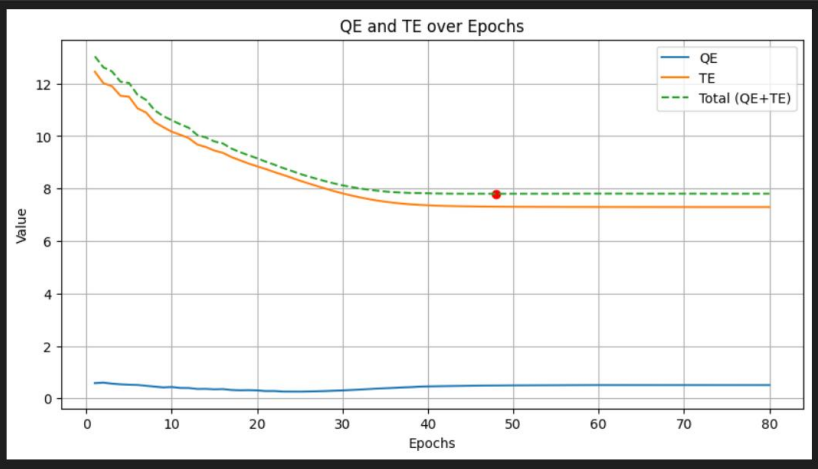
```python
model = tf.keras.applications.mobilenet.MobileNet(
# The 3 is the three dimensions of the input: r,g,b.
    input_shape=(224, 224, 3),
    include_top=False,
    pooling='avg'
)
```
Python

```python
features = []
for path in psychologyfiles:
    f = processImage(path, model)
    features.append(f)
```
Python

Vectorize the image by getting the image feature

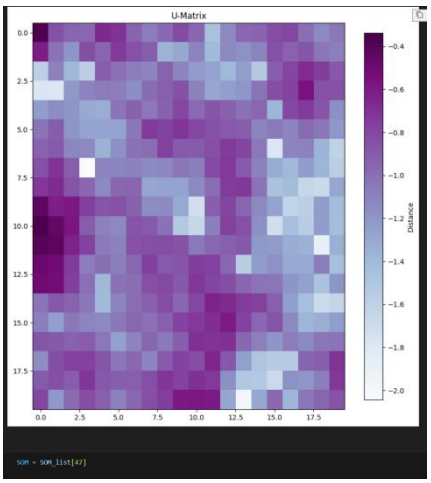
3. TRAIN SOM

```python
plot_qete(TE_list,QE_list)
```
[116]




```
The best epoch is 48 with a minimum total of QE and TE of 7.8020905990931775.
```

Here it shows that the best SOM is the 48, and according to the code order rule SOM_list[47] is the 48th, so SOM_list[47] is selected as the best SOM.



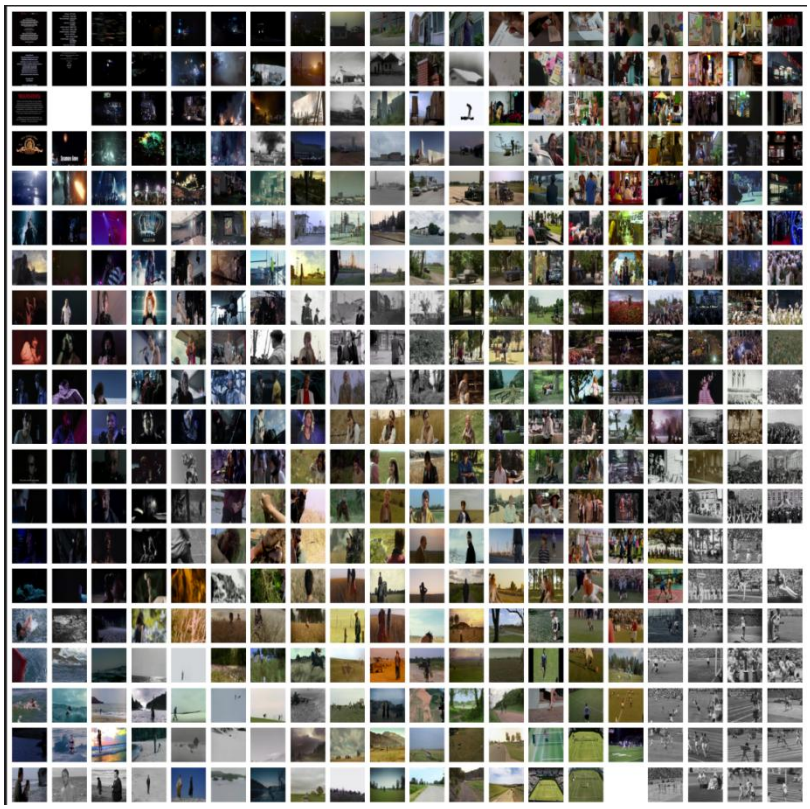
```
[[len(c) for c in r] for r in SOMimages]
```

```
[[31, 21, 1, 15, 12, 12, 23, 11, 16, 21, 18, 6, 18, 21, 10, 18, 16, 18, 9, 14],
 [8, 10, 14, 12, 12, 16, 5, 14, 10, 11, 11, 4, 12, 13, 14, 15, 13, 12, 9, 16],
 [19, 0, 10, 8, 15, 9, 9, 9, 8, 9, 9, 9, 10, 10, 7, 16, 10, 15, 11, 16],
 [4, 5, 10, 14, 14, 12, 8, 15, 6, 17, 5, 13, 9, 15, 11, 13, 9, 8, 14, 14],
 [11, 13, 16, 9, 10, 11, 6, 11, 15, 12, 11, 6, 11, 6, 5, 9, 15, 13, 10, 12],
 [11, 15, 8, 9, 14, 5, 16, 10, 10, 18, 5, 17, 11, 9, 5, 6, 5, 12, 6, 5],
 [13, 15, 11, 6, 9, 6, 7, 11, 11, 11, 7, 15, 8, 7, 6, 7, 11, 3, 7, 14],
 [9, 13, 13, 6, 13, 12, 13, 8, 3, 16, 5, 19, 13, 11, 8, 6, 8, 9, 13, 11],
 [20, 13, 15, 7, 7, 9, 6, 10, 13, 15, 9, 11, 11, 12, 7, 15, 5, 13, 7, 3],
 [20, 13, 15, 13, 5, 12, 13, 14, 6, 11, 14, 2, 12, 9, 7, 3, 11, 6, 1, 18],
 [20, 14, 11, 8, 10, 8, 14, 8, 6, 15, 12, 10, 9, 12, 7, 11, 5, 12, 10, 10],
 [17, 21, 14, 14, 8, 11, 10, 13, 16, 5, 4, 11, 7, 9, 11, 6, 9, 12, 6, 10],
 [17, 18, 13, 11, 11, 5, 12, 8, 11, 9, 13, 10, 7, 8, 6, 8, 12, 5, 1, 24],
 [20, 15, 17, 11, 10, 11, 8, 14, 6, 12, 10, 14, 10, 9, 6, 13, 7, 14, 5, 0],
 [18, 6, 13, 5, 11, 6, 10, 8, 11, 6, 11, 12, 17, 7, 11, 15, 7, 6, 7, 5],
 [4, 12, 5, 7, 11, 10, 10, 14, 11, 12, 8, 10, 13, 4, 12, 19, 3, 12, 8, 19],
 [16, 15, 15, 7, 10, 6, 9, 14, 6, 10, 10, 11, 13, 7, 6, 14, 6, 14, 10, 15],
 [13, 12, 17, 7, 11, 9, 4, 10, 13, 10, 8, 15, 8, 10, 7, 3, 19, 4, 15, 10],
 [12, 13, 16, 16, 11, 13, 11, 8, 11, 15, 9, 10, 6, 2, 9, 5, 4, 12, 7, 21],
 [20, 13, 11, 15, 14, 11, 13, 13, 18, 13, 18, 17, 12, 5, 23, 0, 13, 6, 18, 21]]
```

```
with open('F:\\SOM_skill\\film\\POCimage_SOMimages.pkl', 'wb') as f:
    pickle.dump(SOMimages, f)
```

```
with open('F:\\SOM_skill\\film\\POCimage_SOMimages.pkl', 'rb') as f:
    SOMimages = pickle.load(f)
```

Then the processed image corresponds to the trained SOM, attach thim, and save the image contained after each unit, once again later search for BMU (beat matching unit) call.

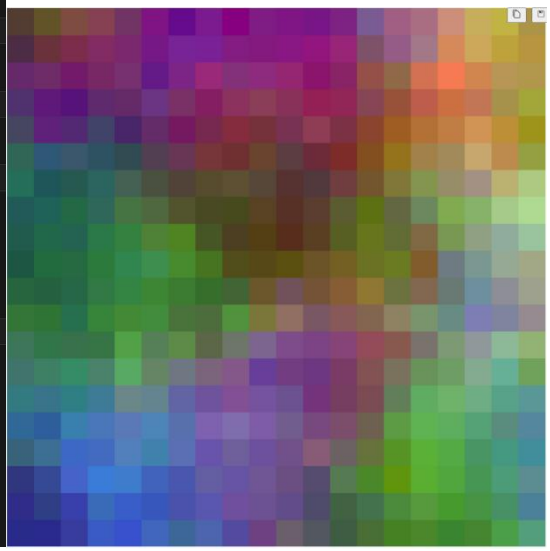


The picture of each neuron that most resembles the neuron next to it is displayed, so that an IMAGE MAP is obtained

4. FIT PCA

FIT PCA ON ALL CELLS

```
som_x, som_y, dimension = POC_image_SOM.shape
num_nodes = som_x * som_y
twod_som = POC_image_SOM.reshape(num_nodes, dimension)
pca = PCA(n_components=3)
PCAComponents = pca.fit_transform(twod_som)


normalized_colors = normalise(PCAComponents,PCAComponents)


rgb_colors = (normalized_colors * 255).astype(np.uint8)


som_colors = np.zeros((som_x, som_y, 3), dtype=np.uint8)
index = 0
for i in range(som_x):
    for j in range(som_y):
        som_colors[i, j] = rgb_colors[index]
        index += 1


fig, ax = plt.subplots(figsize=(som_x, som_y))
for i in range(som_x):
    for j in range(som_y):
        color = som_colors[i, j] / 255.0
        ax.add_patch(plt.Rectangle((j, i), 1, 1, color=color))

ax.set_xlim(0, som_y)
ax.set_ylim(0, som_x)
ax.set_aspect('equal')
ax.axis('off')
plt.show()
```

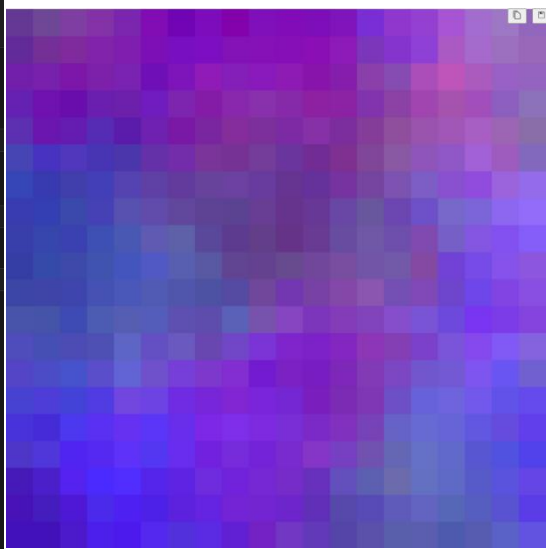


FIT PCA ON THE ENTIRE TRAINING SET

```
pca = PCA(n_components=3)
pca.fit(features)
```

```
PCA(n_components=3)
```

```
som_x, som_y, dimension = POC_image_SOM.shape
num_nodes = som_x * som_y
twod_som = POC_image_SOM.reshape(num_nodes, dimension)


transformed_weight = pca.transform(twod_som)
normalised = normalise(transformed_weight,transformed_weight)


rgb_color = (normalised * 255).astype(np.uint8)


som_pca_colors = np.zeros((som_x, som_y, 3), dtype=np.uint8)
index = 0
for i in range(som_x):
    for j in range(som_y):
        som_pca_colors[i, j] = rgb_color[index]
        index += 1
fig, ax = plt.subplots(figsize=(som_x, som_y))
for i in range(som_x):
    for j in range(som_y):
        color = som_pca_colors[i, j] / 255.0
        ax.add_patch(plt.Rectangle((j, i), 1, 1, color=color))

ax.set_xlim(0, som_y)
ax.set_ylim(0, som_x)
ax.set_aspect('equal')
ax.axis('off')
plt.show()
```



The reason for the difference in the results is that the first approach focuses on the internal weight structure of the SOM itself, while the second approach focuses on how the SOM represents and adapts to the structure of the entire training set.

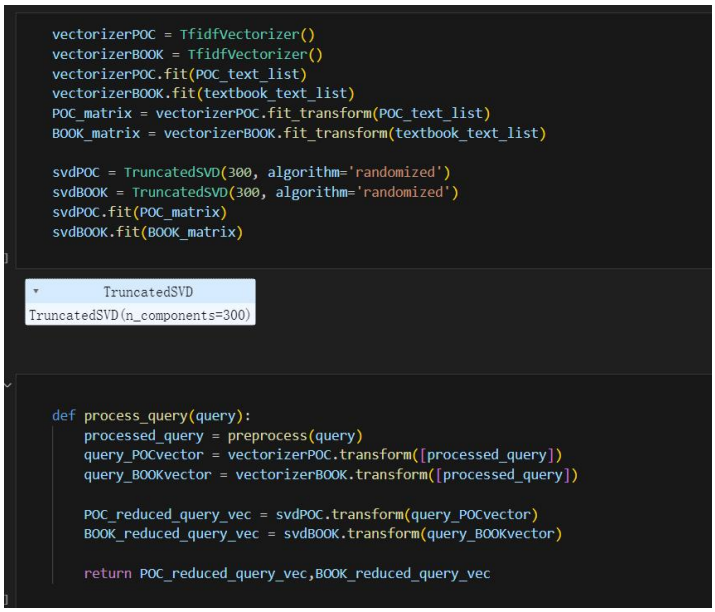
In addition, in the first approach, PCA is optimized based on the results of the SOM and may capture patterns and structures specific to the SOM. In the second approach, PCA reflects the main directions of variation in the original data, and the weights of the SOM are adapted to these global features.

2. BUILD SEARCH FUNCTION

2.1 BUILD TEXT SEARCH ENGINE

https://github.com/UD-Skills-2023-24/github-codebook-RC11-23196457/blob/main/00%20SOM/searchengine_TEXT.ipynb

The principle of search engine is to compare the vector data of query with the existing SOM data and activate the SOM to find the BEST MATCHING UNIT.The query has to be processed in the same vectorized way in order to get the similar content correctly and fairly.



```
vectorizerPOC = TfidfVectorizer()
vectorizerBOOK = TfidfVectorizer()
vectorizerPOC.fit(POC_text_list)
vectorizerBOOK.fit(textbook_text_list)
POC_matrix = vectorizerPOC.fit_transform(POC_text_list)
BOOK_matrix = vectorizerBOOK.fit_transform(textbook_text_list)

svdPOC = TruncatedSVD(300, algorithm='randomized')
svdBOOK = TruncatedSVD(300, algorithm='randomized')
svdPOC.fit(POC_matrix)
svdBOOK.fit(BOOK_matrix)
```

```
TruncatedSVD
TruncatedSVD(n_components=300)
```

```
def process_query(query):
    processed_query = preprocess(query)
    query_POCvector = vectorizerPOC.transform([processed_query])
    query_BOOKvector = vectorizerBOOK.transform([processed_query])

    POC_reduced_query_vec = svdPOC.transform(query_POCvector)
    BOOK_reduced_query_vec = svdBOOK.transform(query_BOOKvector)

    return POC_reduced_query_vec,BOOK_reduced_query_vec
```

Vectorize query with TF-IDF+SVD approach.

```python
def find_closet_media_resources(query):
    # Process the query
    POC_reduced_query_vec = process_query(query)[0]
    BOOK_reduced_query_vec = process_query(query)[1]

    # Find BMUs
    bmu_text_index = find_BMUcos(textbook_text_som_model, BOOK_reduced_query_vec[0])
    bmu_video_index = find_BMUcos(poc_fuyu_som_model, POC_reduced_query_vec[0])

    activatedSOM = activate(textbook_text_matrix, textbook_text_som_model, BOOK_reduced_query_vec[0])
    fig = plt.figure()
    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.BuPu, aspect='auto')
    plt.colorbar(im, shrink=0.95, label='Distance')
    plt.xticks([])
    plt.yticks([])
    plt.show()
    g, h = find_BMUcos(textbook_text_som_model, BOOK_reduced_query_vec[0])
    print((g, h))

    def find_most_similar(data_dict, bmu_index, query_vec):
        g, h = bmu_index
        max_similarity = -np.inf
        most_similar_item = None
        for item in data_dict[g][h]:
            item_vec = np.array(item['vector']).reshape(1, -1)
            similarity = cosine_similarity(item_vec, query_vec.reshape(1, -1))[0][0]
            if similarity > max_similarity:
                max_similarity = similarity
                most_similar_item = item
        return most_similar_item

    # Retrieve the most similar text and image in the textbook data
    most_similar_textbook = find_most_similar(textbook_text_data_dict, bmu_text_index, BOOK_reduced_query_vec)
    text_key = most_similar_textbook['text']
    image_path = textbook_file.get(text_key)


    # Retrieve the most similar video text in the POC data
    most_similar_video = find_most_similar(POC_text_data_dict, bmu_video_index, POC_reduced_query_vec)
    video_text = most_similar_video['text']
    image_path_for_video = None
    for entry in POC_all_list:
        if entry[0] == video_text:
            image_path_for_video = entry[1]
            break
```

```python
if image_path_for_video is None:
    image_path_for_video = "Default image path if text not found"

new_cover_path = image_path_for_video.replace('/content/drive/MyDrive/', 'F:\\SOM_skill\\film\\')

print(bmu_text_index)
print("Textbook_Image:")
display(Image(filename=image_path[0]))
print(text_key)



activatedSOM2 = activate(POCfuyu_text_matrix, poc_fuyu_som_model, POC_reduced_query_vec[0])
fig2 = plt.figure()
plt.figure(figsize=(10, 10))
im = plt.imshow(activatedSOM2, cmap=cm.BuPu, aspect='auto')
plt.colorbar(im, shrink=0.95, label='Distance')
plt.xticks([])
plt.yticks([])
plt.show()
g2, h2 = find_BMUcos(poc_fuyu_som_model, POC_reduced_query_vec[0])
print((g2, h2))

print(bmu_video_index)
print(video_text)
print("POC_Image:")
display(Image(filename=new_cover_path))

return text_key, image_path, video_text, new_cover_path
```

2.2 BUILD IMAGE SEARCH ENGINE

https://github.com/UD-Skills-2023-24/github-codebook-RC11-23196457/blob/main/00%20SOM/searchengine_IMAGE.ipynb

Images are vectorized in terms of features, so the query's image processing needs to be in terms of features as well.

```
def process_query(querypath):
    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'
)
    im = load_image(querypath)
    f = model.predict(im)[0]
    return f
```

```
def find_media_resources(querypath):
    # Process the query
    query_vec = process_query(querypath)

    # Find BMUs
    bmu_text_index = find_BMU(textbook_image_som_model, query_vec)
    bmu_video_index = find_BMU(poc_image_som_model, query_vec)

    g,h=find_BMU(textbook_image_som_model,query_vec)
    TEXTBOOK_closest_image_index=get_closest_image_textbook(g,h)
    if g >= len(TEXTBOOKimages) or h >= len(TEXTBOOKimages[g]):
        print("Error: Index g or h out of range.")
        return None
    image_path = TEXTBOOKimages[g][h][TEXTBOOK_closest_image_index]['image']
    if TEXTBOOK_closest_image_index >= len(TEXTBOOKimages[g][h]):
        print("Error: TEXT closest image index out of range.")
        return None
    matching_text = [key for key, value in textbook_file.items() if value == image_path]

    activatedSOM = activate_normalise(textbook_image_traindata, textbook_image_som_model, query_vec)
    fig = plt.figure()
    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.BuPu, aspect='auto')
    plt.colorbar(im, shrink=0.95, label='Distance')
    plt.xticks([])
    plt.yticks([])
    plt.show()

    display(Image(filename=querypath))
    print(bmu_text_index)
    print("Textbook Image:")
    print(image_path)
    display(Image(filename=image_path))
    print(matching_text)

    g2,h2=find_BMU(poc_image_som_model,query_vec)
    if g2 >= len(POCimages) or h2 >= len(POCimages[g]):
        print("Error: Index g or h out of range.")
        return None
    POC_closest_image_index=get_closest_image_POC(g2,h2)
    POC_image_path = POCimages[g2][h2][POC_closest_image_index]['image']
    if POC_closest_image_index >= len(POCimages[g2][h2]):
        print("Error: POC closest image index out of range.")
        return None
    final_POC_image_path = POC_image_path.replace('\\', '/').replace('F:/SOM_skill/film', '/content/drive/MyDrive')
```

```python
activatedSOM2 = activate_normalise(POC_image_traindata, poc_image_som_model, query_vec)
fig = plt.figure()
plt.figure(figsize=(10, 10))
im = plt.imshow(activatedSOM2, cmap=cm.BuPu, aspect='auto')
plt.colorbar(im, shrink=0.95, label='Distance')
plt.xticks([])
plt.yticks([])
plt.show()

print(bmu_video_index)
display(Image(filename=POC_image_path))
print(final_POC_image_path)
matching_value = None

for row in POC_all_list:
    if row[1] == final_POC_image_path:
        matching_value = row[0]
        break
if matching_value is not None:
    print(matching_value)
else:
    print("No matching value found.")

return image_path, matching_text, final_POC_image_path,matching_value
```

```python
from searchengineclass import ImageSOM
```

Python

```python
image_som = ImageSOM()
result = image_som.find_media_resources('F:\\SOM_skill\\film\\image_scrape\\arena_performance_dog\\frame-035512.jpg')
```

Python

Use the .py file to classify the search engine code and simplify the search steps.

Complexity Final assignment

Exercise One

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

```python
import numpy as np

def square_matrix_multiply(A, B):

    # get the the row and column of A
    n = A.shape[0]
    C = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]

    return C
```

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

The asymptotic time complexity of the above algorithm is O(n^3).

The reason are as follows.

First it will go through the first level of loop 'for i in range(n)',if A is an n x n matrix, this loop will iterate n times, then it will go to the second level of loop 'for j in range(n)', for n rows of A, it will iterate through the n columns of B, here it is n x n = n^2.Finally 'for k in range(n)', for C[i][j] The computation of the elements all need n times multiplication and (n-1)times addition, from the complexity considerations, it can be considered as n times of operations, for n x n elements in C all have to perform n times of operations, so the complexity is n x n x n = n^3

Suggestion:

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

```
A = np.array(np.random.rand(10, 10).tolist())
B = np.array(np.random.rand(10, 10).tolist())
C = np.array(np.random.rand(50, 50).tolist())
D = np.array(np.random.rand(50, 50).tolist())
```

```
start_time = time.time()
square_matrix_multiply(A, B)
custom_time = time.time() - start_time
print(custom_time, "seconds")
```

```
0.0009989738464355469 seconds
```

```
start_time = time.time()
square_matrix_multiply(C, D)
custom_time = time.time() - start_time
print(custom_time, "seconds")
```

```
0.09378194808959961 seconds
```

The larger the size of the matrix the more time it takes to process the algorithm.

3. Compare with built-in multiplication functions

```
start_time = time.time()
C@D
custom_time = time.time() - start_time
print(custom_time, "seconds")
```

```
0.0 seconds
```

```
start_time = time.time()
np.matmul(C, D)
custom_time = time.time() - start_time
print(custom_time, "seconds")
```

```
0.0 seconds
```

```
start_time = time.time()
C.dot(D)
custom_time = time.time() - start_time
print(custom_time, "seconds")
```

```
0.0 seconds
```

The process of built-in multiplication functions is more efficient.

3. Look at multiplying more than two matrices, or at other operations on matrices

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

```
A = np.array(np.random.rand(10, 10).tolist())
B = np.array(np.random.rand(10, 10).tolist())
C = np.array(np.random.rand(50, 50).tolist())
D = np.array(np.random.rand(50, 50).tolist())
```

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

```
result
```

```
array([[64.84075867, 89.70324947, 72.93678991, 80.57166739, 65.53088091,
        68.11134158, 88.74529554, 77.30743598, 88.1762775 , 52.65694485],
       [68.85430204, 94.92976334, 77.1037955 , 85.2877553 , 69.07655627,
        72.39465775, 93.95377091, 81.9315534 , 93.11136483, 55.76384274],
       [54.52479849, 75.85309296, 61.44298232, 67.72857147, 54.90049111,
        57.7235619 , 75.21775139, 65.08830524, 74.39606967, 44.07355428],
       [55.31865826, 76.31743956, 62.29912575, 68.68334306, 55.49589657,
        58.16401324, 76.07108291, 65.91478314, 75.46079072, 44.58335466],
       [44.10629742, 60.08415395, 49.12223869, 54.49080474, 43.91135092,
        45.94932901, 60.07886191, 52.18026011, 59.51756789, 35.39205038],
       [60.46786966, 84.08836369, 68.03194902, 75.21005999, 61.00089219,
```

Multiply multiple matrices

```
def transpose_matrix(matrix):
    rows = len(matrix)
    cols = len(matrix[0])

    transposed = [[0 for _ in range(rows)] for _ in range(cols)]

    for i in range(rows):
        for j in range(cols):
            transposed[j][i] = matrix[i][j]

    return transposed
```

```
A = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

transposed_A = transpose_matrix(A)

for row in transposed_A:
    print(row)
```

```
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
```

Transform the row and column of the matrice

Exercise Two

1. Describe and explain the algorithm. It should contain at least the following: recursiveness: How is it recursive? What is (the criteria for) the base case? How does the recursion step reduce to the base case?

The algorithm multiplies the square array by recursively calling itself.When n = 1, the size of the matrix is reduced to 1x1, the matrix contains only one element, the matrix multiplication at this point is a multiplication of a single number.

divide-and-conquer: How does this algorithm fit into the divide-andconquer approach? Explain each step of divide, conquer, and combine for this algorithm (as in slide 8 / pdf page 16 of the lecture slides).

In square_metric_multiplication_recursion algorithm,
for divide, it divides an n x n matrix into four n/2 x n/2 submatrices;
for conquer, the product of these submatrices is computed recursively, degenerating to single-number multiplication when n=1,
and for combine, the products of the resulting submatrices are combined back into the full n x n matrix.

2. Implement the recursive algorithm in Python. Reflect on which steps of the pseudocode were straightforward to implement and which hid a lot of complexity behind their language.

```python
def square_matrix_multiply_recursive(A, B):
    n = A.shape[0]
    C = np.zeros((n, n))

    if n == 1: #straightforward to implement
        C[0, 0] = A[0, 0] * B[0, 0]
    elif n > 1:
        # divide:hid a lot of complexity,The indexes and boundaries of the matrix
        #need to be handled correctly; incorrect handling can lead to logical errors,
        #ensuring that the partitioned submatrix correctly represents the corresponding
        #part of the original matrix
        mid = n // 2
        A11, A12 = A[:mid, :mid], A[:mid, mid:]
        A21, A22 = A[mid:, :mid], A[mid:, mid:]
        B11, B12 = B[:mid, :mid], B[:mid, mid:]
        B21, B22 = B[mid:, :mid], B[mid:, mid:]

        # conquer:Each recursive call requires the correct allocation and handling of submatrix multiplication
        C11 = square_matrix_multiply_recursive(A11, B11) + square_matrix_multiply_recursive(A12, B21)
        C12 = square_matrix_multiply_recursive(A11, B12) + square_matrix_multiply_recursive(A12, B22)
        C21 = square_matrix_multiply_recursive(A21, B11) + square_matrix_multiply_recursive(A22, B21)
        C22 = square_matrix_multiply_recursive(A21, B12) + square_matrix_multiply_recursive(A22, B22)

        # combine:In practice, it is necessary to precisely control the index and dimension of
        #the matrices, which is particularly easy to error in multi-dimensional matrices operations
        First_row = [C11[i] + C12[i] for i in range(len(C11))]
        Second_row = [C21[i] + C22[i] for i in range(len(C21))]

        return np.array(First_row + Second_row)
```

The green word are the explanation.

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

1. BASE SITUATION

When the matrix of n x n, n = 1, the matrix involves only the computation of the multiplication of one element from each of the two matrices, which has a complexity O(1).

2. DIVIDE

In the divide procedure, the input n x n matrix is divided into four n/2 x n/2 submatrices. The dividing matrix involves no computation and is also a constant time operation, so the total complexity is still O(1)

3. Conquer

The CONQUER step involves eight recursive calls dealing with the product of four subproblems, and each of the subproblems is of n/2 x n/2 in size, thus R(n) = 8T(n/2)

4. combine

This requires a series of addition operations, one for each element, thus:C(n) = O(n^2)

$T(n) = 8T(\frac{n}{2}) + O(n^2)$

$T(n) = aT(\frac{n}{b}) + f(n)$

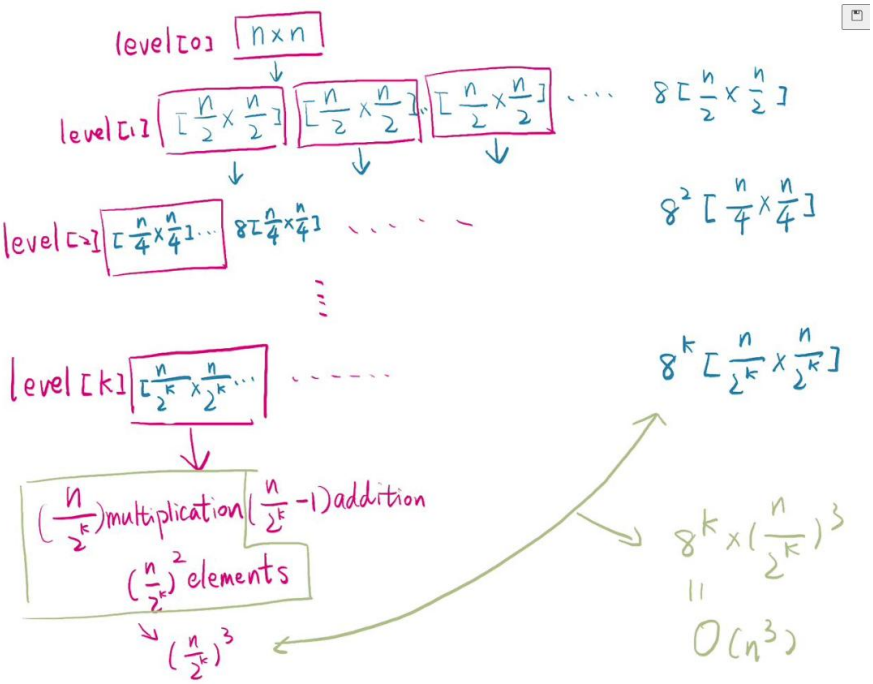
$n^{\log_b a} = n^{\log_2 8} = n^3$

$O(n^3)$

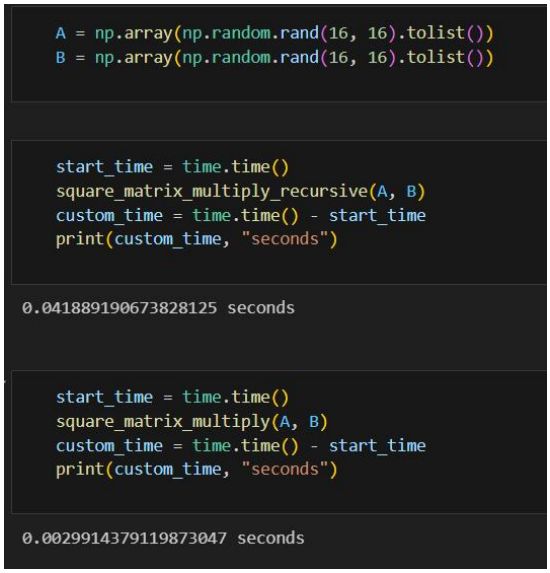
So base on the main theory

Suggestions:

1. Do a tree analysis



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



```
A = np.array(np.random.rand(16, 16).tolist())
B = np.array(np.random.rand(16, 16).tolist())
```

```
start_time = time.time()
square_matrix_multiply_recursive(A, B)
custom_time = time.time() - start_time
print(custom_time, "seconds")
```

```
0.041889190673828125 seconds
```

```
start_time = time.time()
square_matrix_multiply(A, B)
custom_time = time.time() - start_time
print(custom_time, "seconds")
```

```
0.0029914379119873047 seconds
```

The recursive algorithm takes more time to work.

Exercise Three

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

The aysmptotic complexity of addition/subtraction is O(n^2).
'The matrix_sub function performs a matrix subtraction operation.
Since only the elements in the corresponding positions need to be subtracted, its complexity is linear and proportional to the size of the matrix, which is n x n, so the complexity is O(n^2).
The aysmptotic complexity of multiplication is O(n^3),based on exercise two.

2. Do a complexity analysis of the Strassen algorithm.

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$$

a = 7：The algorithm reduces the number of recursive multiplications required at each level from the original eight to seven.
b = 2：Each call processes a submatrix half the size of the original matrix.
f(n) = O(n^2)：Compute ten new matrices s1,s2 .... Each is obtained by subtracting or adding n/2xn/2 submatrices, each of which has a total of n/2xn/2 = n^2/4 elements going into the operation, so the total complexity of the 10 matrices is 10xO(n^2/4) = O(n^2).Calculating Pi is similar to calculating Si, which ultimately simplifies to O(n^2)

so the whole complexity of the algorithm is $O(n^{\log_2 7})$ is more efficent than O(n^3)

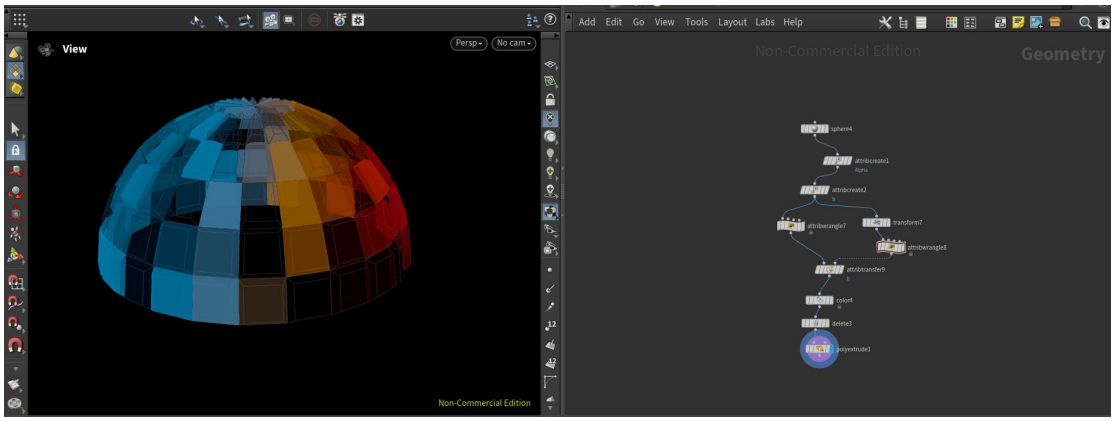
```python
def strassen_matrix_multiply(A, B):
    n = A.shape[0]
    if n == 1:
        return A @ B
    else:
        mid = n // 2
        A11 = A[:mid, :mid]
        A12 = A[:mid, mid:]
        A21 = A[mid:, :mid]
        A22 = A[mid:, mid:]
        B11 = B[:mid, :mid]
        B12 = B[:mid, mid:]
        B21 = B[mid:, :mid]
        B22 = B[mid:, mid:]

        S1 = B12 - B22
        S2 = A11 + A12
        S3 = A21 + A22
        S4 = B21 - B11
        S5 = A11 + A22
        S6 = B11 + B22
        S7 = A12 - A22
        S8 = B21 + B22
        S9 = A11 - A21
        S10 = B11 + B12

        P1 = strassen_matrix_multiply(A11, S1)
        P2 = strassen_matrix_multiply(S2, B22)
        P3 = strassen_matrix_multiply(S3, B11)
        P4 = strassen_matrix_multiply(A22, S4)
        P5 = strassen_matrix_multiply(S5, S6)
        P6 = strassen_matrix_multiply(S7, S8)
        P7 = strassen_matrix_multiply(S9, S10)

        C11 = P5 + P4 - P2 + P6
        C12 = P1 + P2
        C21 = P3 + P4
        C22 = P5 + P1 - P3 - P7



        top = np.hstack((C11, C12))
        bottom = np.hstack((C21, C22))
        return np.vstack((top, bottom))
```

```python
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = strassen_matrix_multiply(A, B)
print(C)
```

```
[[19 22]
 [43 50]]
```

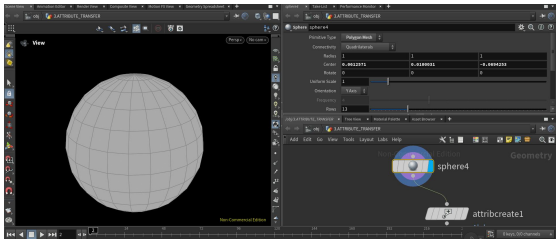
Description of houdini foundamental assignment

Example1:



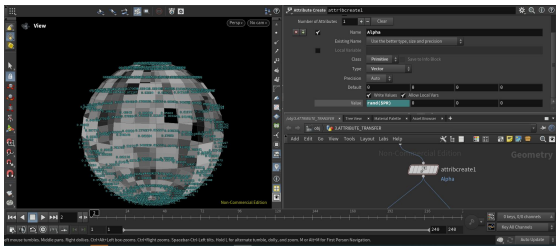
1:Sphere SOP

The Sphere is of primitive type:polygon mesh.One of the basic geometric data types that contains points,primitives of type polygon and vertices.



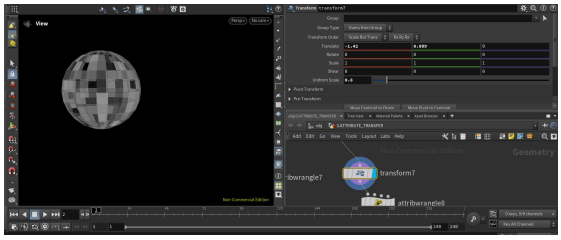
2:Attribute Create SOP

Create an ‘Attribute Create’ SOP, where we add a ”Alpha” attribute to the primitives. Then we set ‘rand($PR)’ to the value of this attribute and the alpha of each surfaces will be randomized.



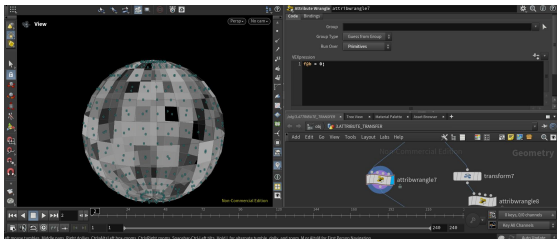
3:Transform SOP

We create an ‘Transform’ SOP to transform our original object. This time we change the scale of the original one and the distance between it.



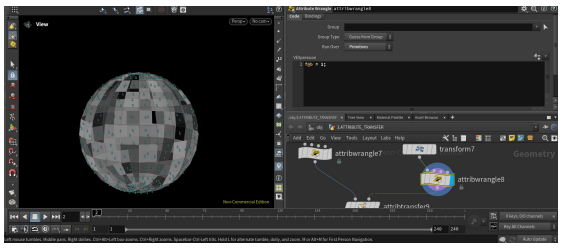
4:Attribute Wrangle SOP

We create a ‘Attribute Wrangle’ to set a ‘b’ attribute,and the ‘b’ set to 0 and it will run over the attributes.We use this this sop because later The primitive is given a different color by the change in its value due to the change in the relationship between the original object and the transformed object.



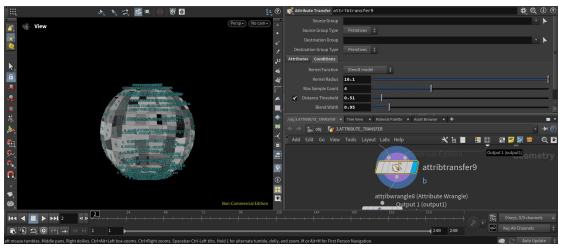
5:Attribute Wrangle SOP

All primitives are assigned the value 1 by the Attribute Wrangle SOP.



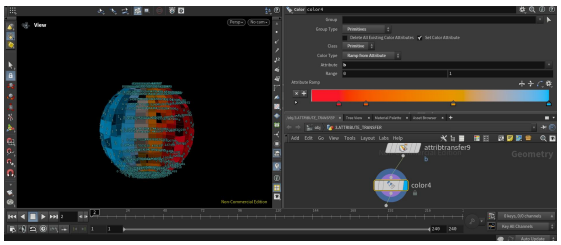
6:Attribute Transfer SOP

‘Attribute Transfer’SOP use the attribute ‘b’ to show the change between the original one and the transformed one.The value changed from 0 to 1 based on the attribute’b’.



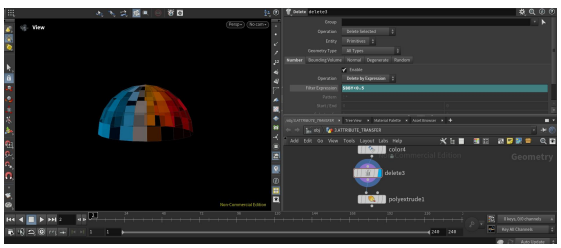
7:Color SOP

‘Color’SOP enable us to visualize changes in b-attribute values as colors.



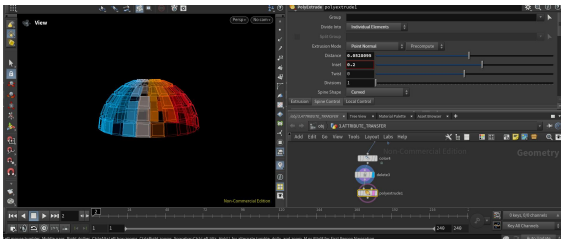
8:Delete SOP

We use ‘delete’ SOP to delete the Y-value is small than 0.5($BBY<0.5) to build a igloo-type architecture.

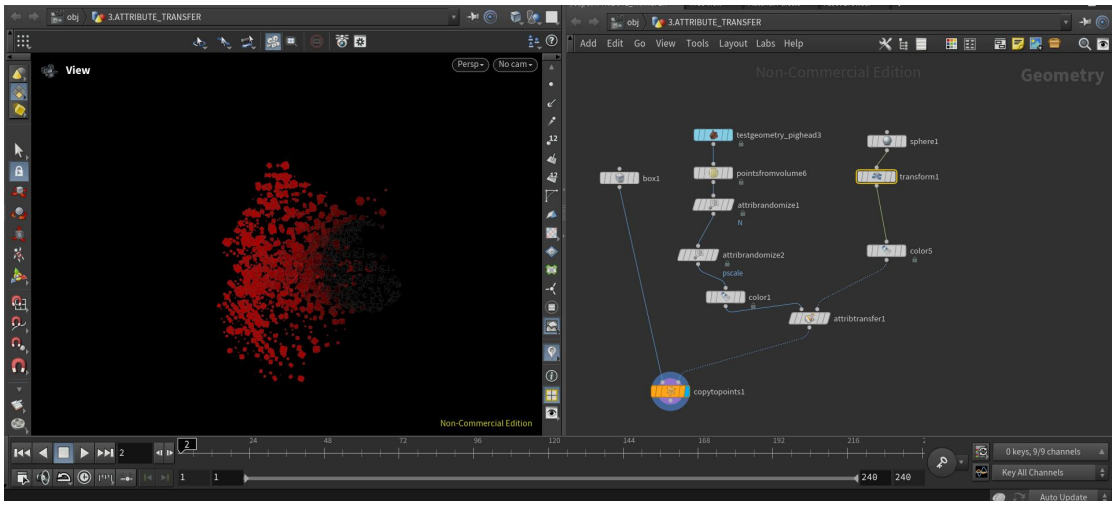


9:Polyextrude SOP

We use ‘polyextrude’ SOP to inset each primitve and give the primitive thickness.A kind of rainbow igloo done.



Example2:

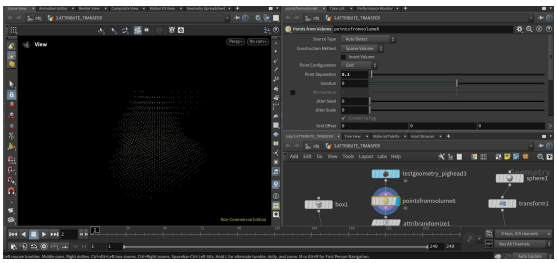


1:Pighead SOP

The pig head SOP is just the base of the object we're going to work on later, it can be anything like land, a tree, etc.

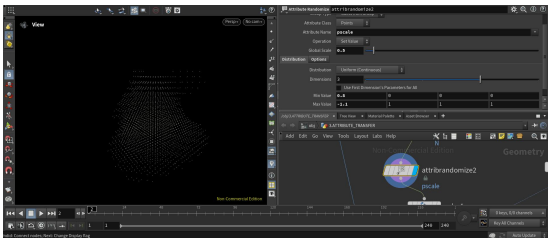
2:Points from volume SOP

Points from volume SOP converts volume to points and can modify the distance between points and isovalue.



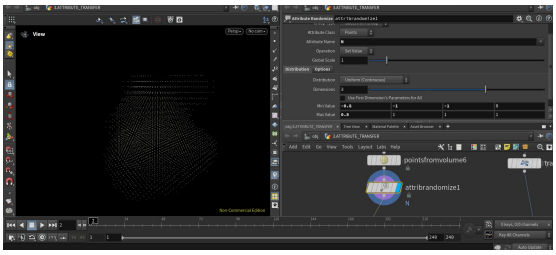
3:Attribute Randomize SOP

Creat a Attribute Randomize SOP where we set a attribute ‘pscale’.The pscale attribute controls the scale of points which means this time the scale of each points will be randomized.And the min scale and max scale can be set .



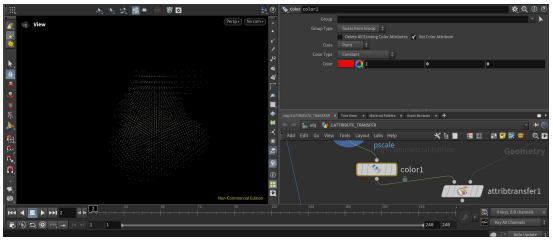
4:Attribute Randomize SOP

Creat a Attribute Randomize SOP where we set a attribute ‘N’.The N attribute controls the vector of points which means this time the vector of each points will be randomized.



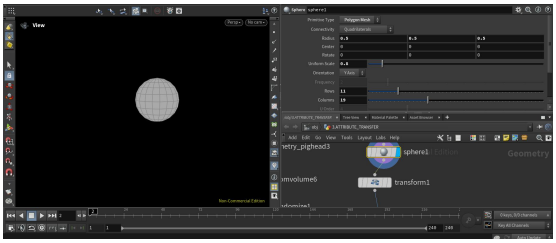
5:Color SOP

Set a color sop to make the pig head points red.



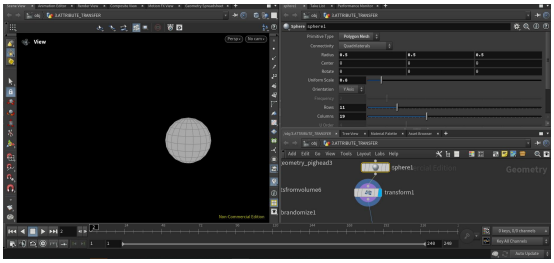
6:Sphere SOP

Set a sphere sop because later I want the pig head changes when it touches the sphere like an object hitting a face or the ground.



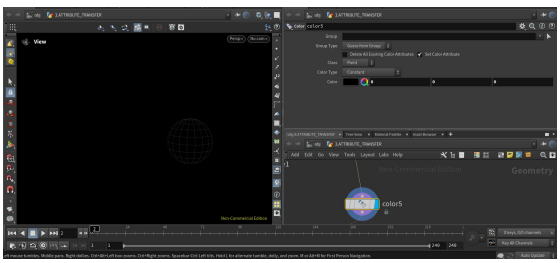
7:Transform SOP

Set a transform sop like a copy of the sphere, i can change the position or scale of the sphere through this SOP



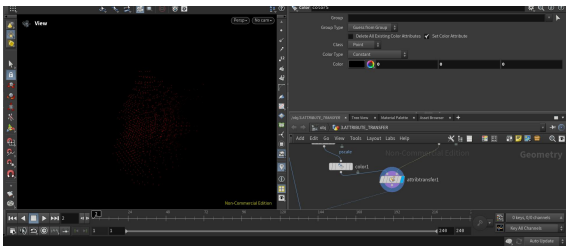
8:Color SOP

Set a color sop to make the sphere black.



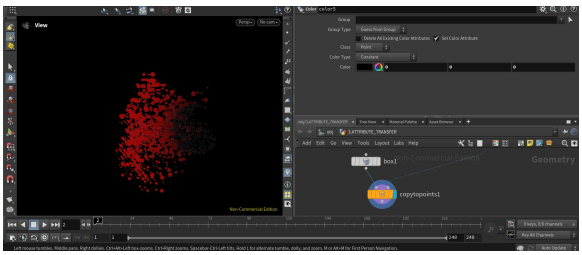
9:Attribute Transfer SOP

‘Attribute Transfer’SOP transfer the part of pig head influence by the sphere from red to black.

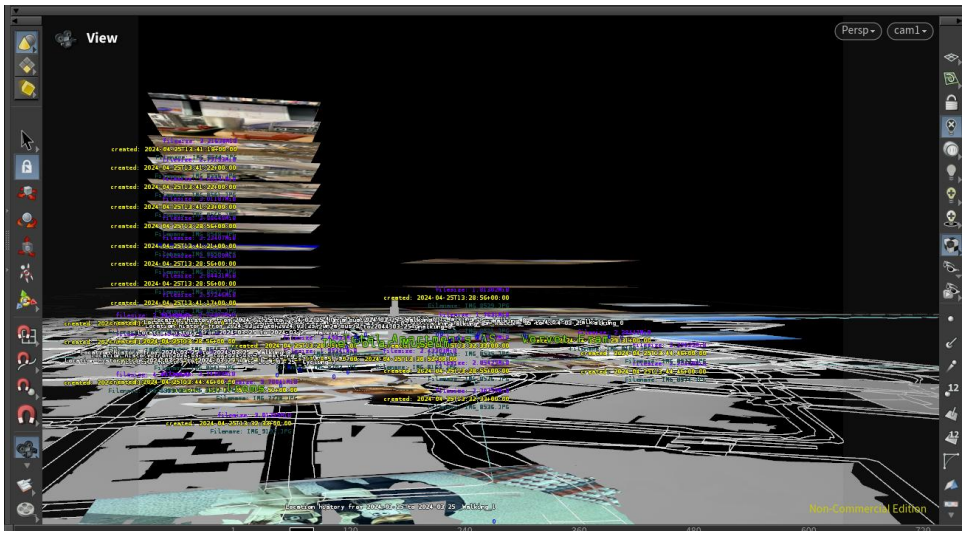


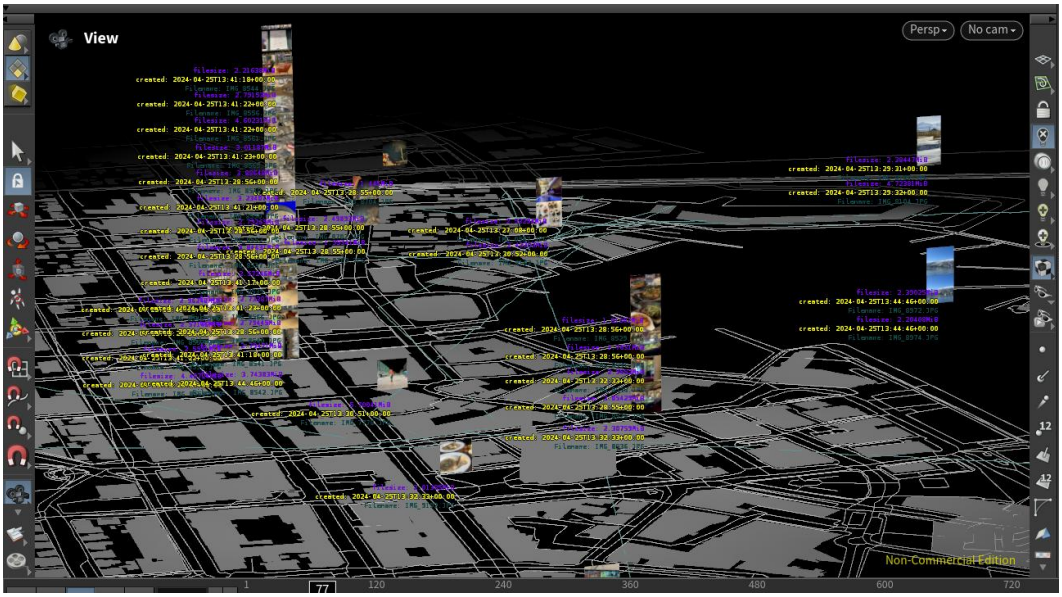
10:BOX SOP & Copy to points SOP

Set a box sop to a copy to points sop to visualise the the color,vector and position of the points by box-figure.



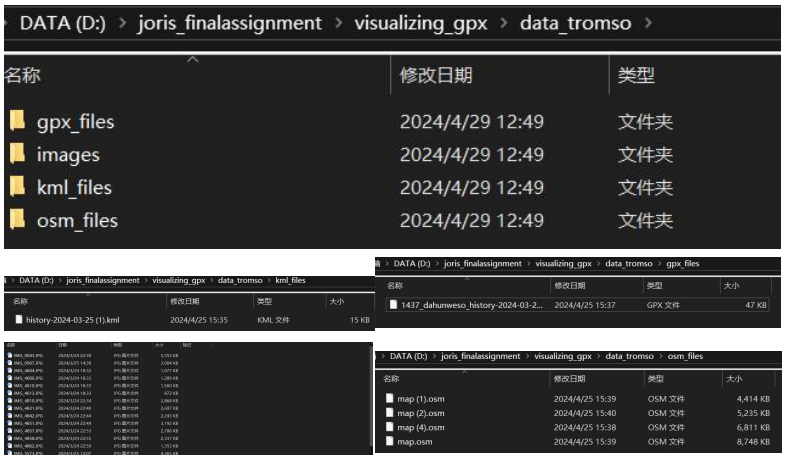
Description of Visualizing GPS and Image Metadata





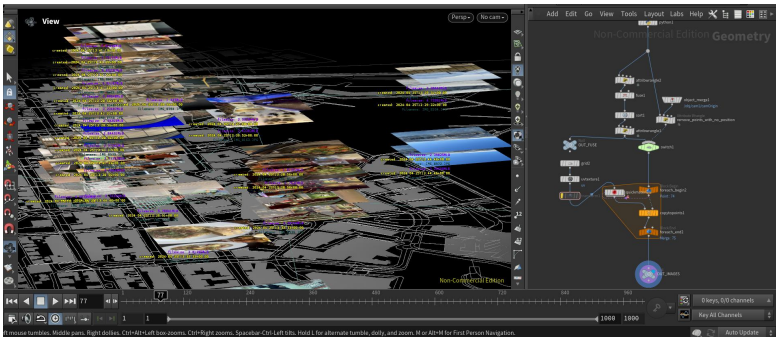
1:Load own data

Import photos and gps data from tromso, convert kml file to gpx form, enter osm for tromso play area, and then import the gpx for the tromso play area.



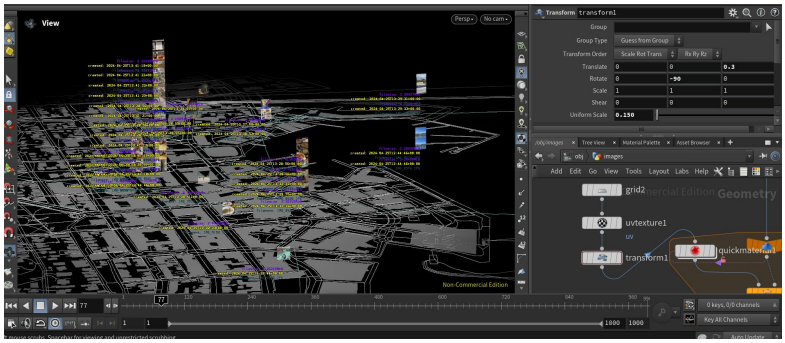
2:Depicting the travel

A.The A perspective which was taught at class,simply show the location of the photo in relation to the map as well as the time, name, size, and information about the photo itself.

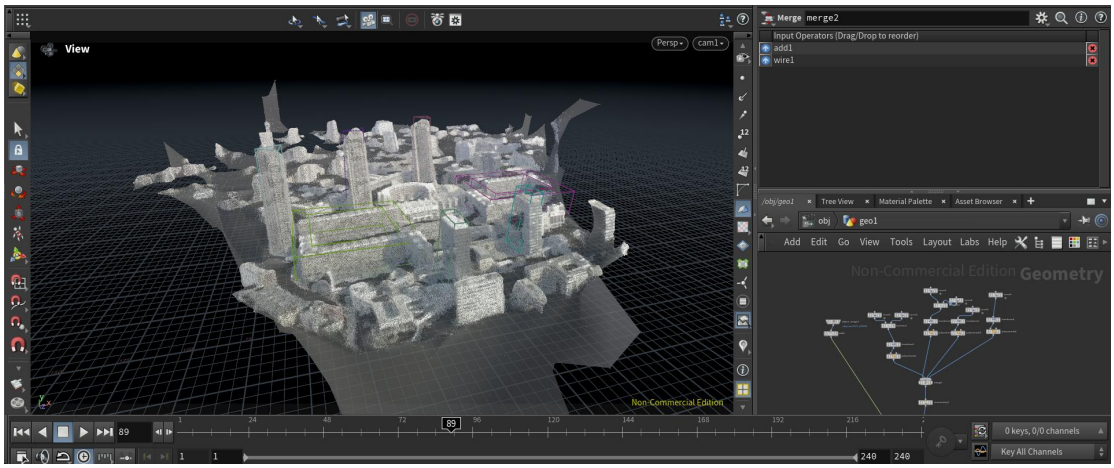


2:Depicting the travel

B.Creat a attribute transform SOP to change the rotation and scale of the photo,ans transfer from A perspective to B perspective.This perspective enales us to see the images as clearly as possible.

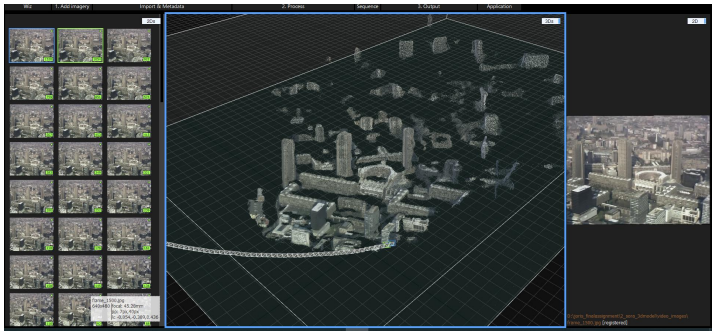


Description of video to 3D model



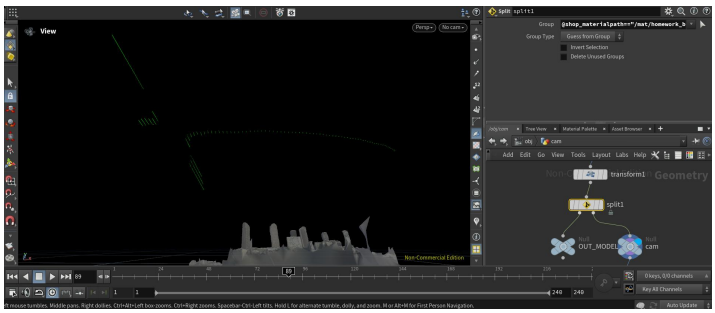
1:Generate a 3d model from a video

Download a barbican center and extract the frame by .py file given.Generate a model of the barbican center from sequential images in Realitycapture.



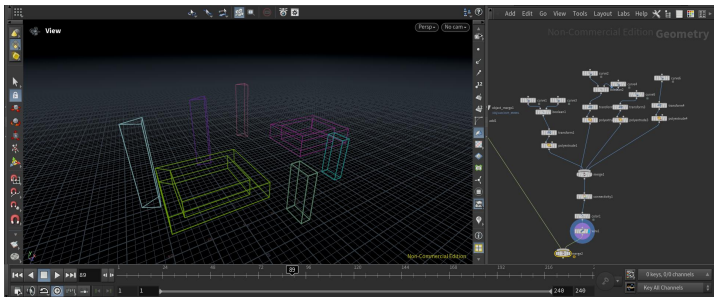
2:Reconstruct the camera path

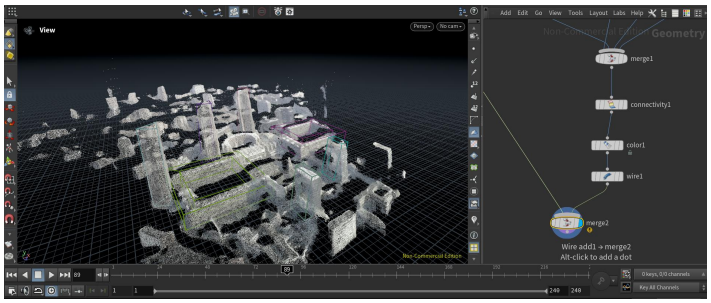
Split the model and camera path by copy the shop_materialpath value through a Split attribute SOP.



3:Add blocks to the barbican center.

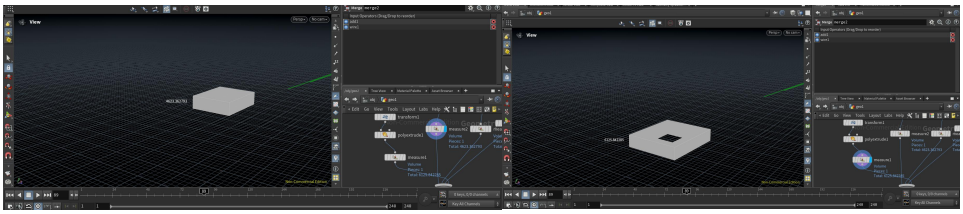
Add Curve SOPs and boolean SOPs to create abstract block of the barbican center .

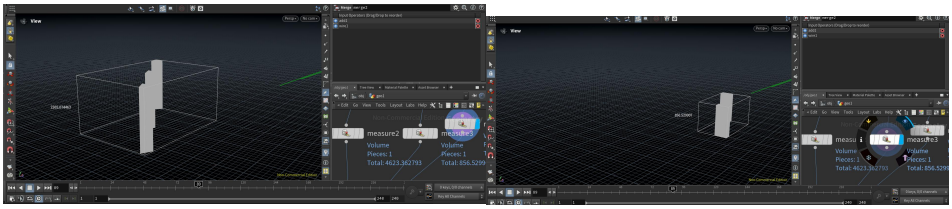




4:Add information to the subject

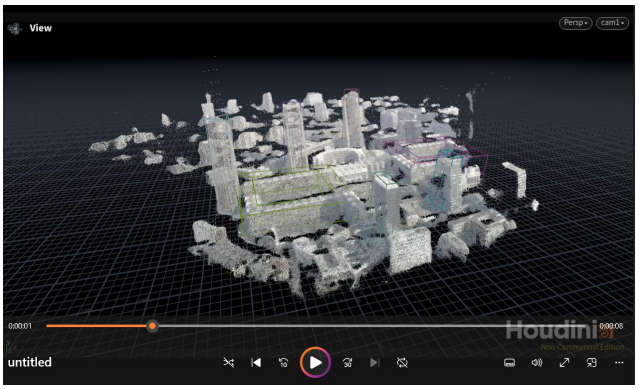
Add Measure SOPs to measure the total volume of the blocks and use tag to visualize on the scenview



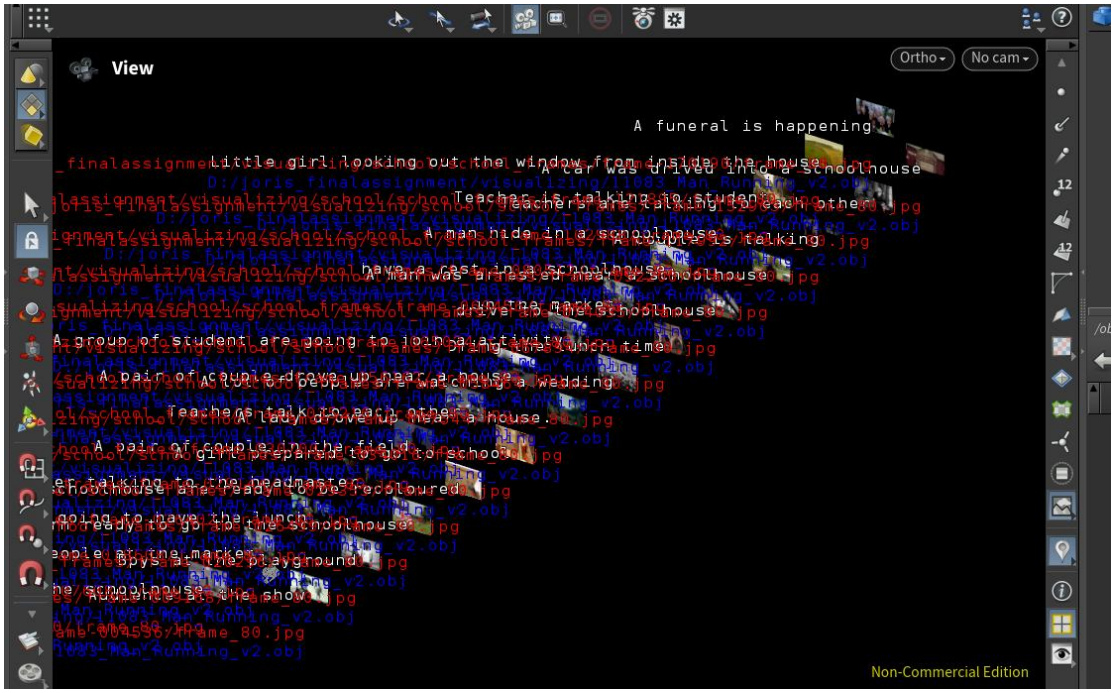


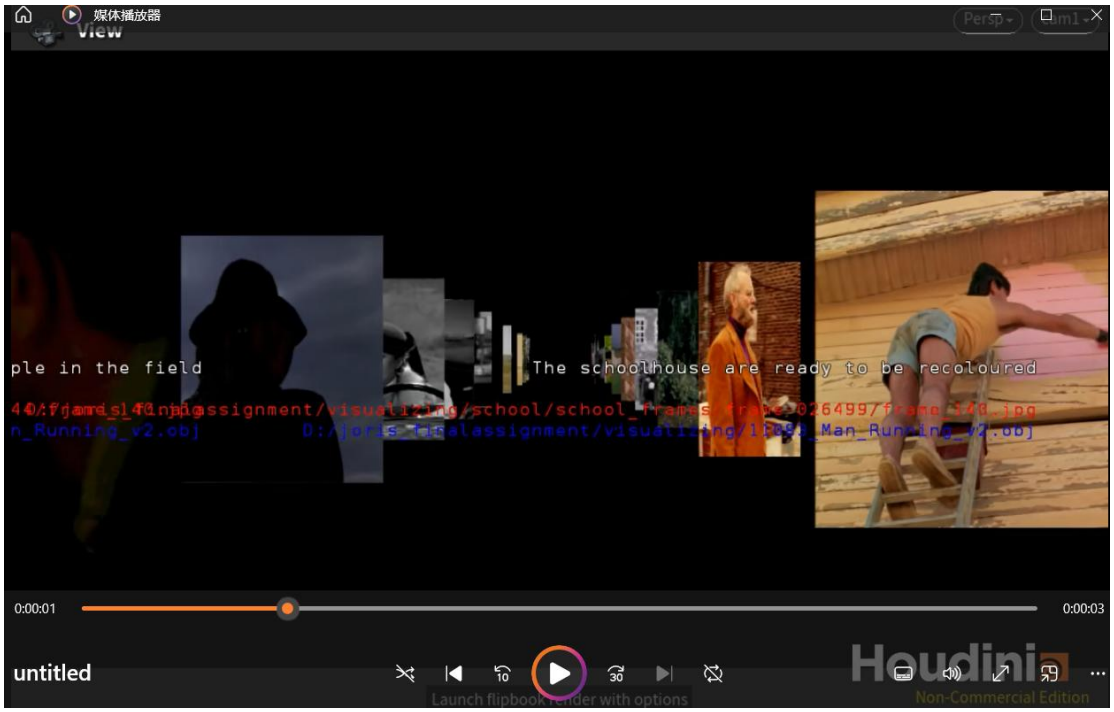
5:Visualize the model

Export the video through ffmpeg and render the frames of the model.



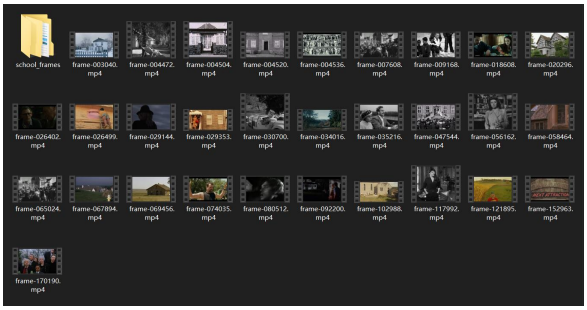
Description of Visualizing JSON in Houdini

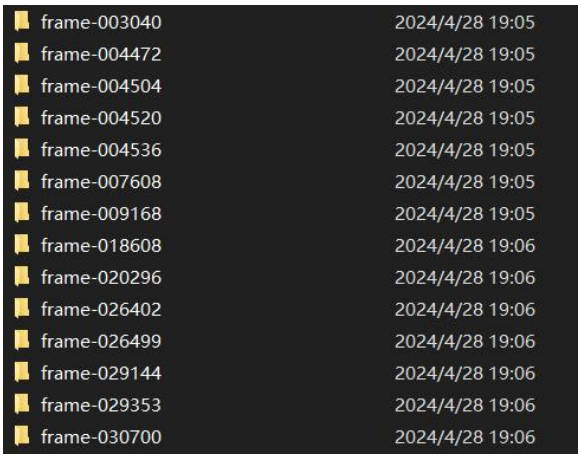




1:Download video based on my objects

Our design course project is about the relationship between knowledge and authority, and we need to extract the videos of the relevant authoritative institution. I scraped the school-related videos in the POC website and saved each video in a folder named after the video by dividing each video into ten frames.



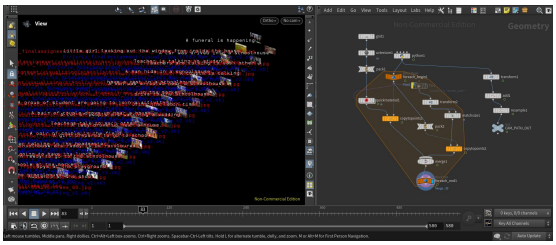


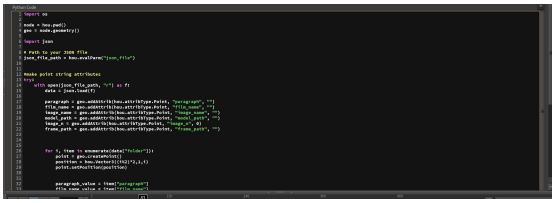
2:Create a JSON file

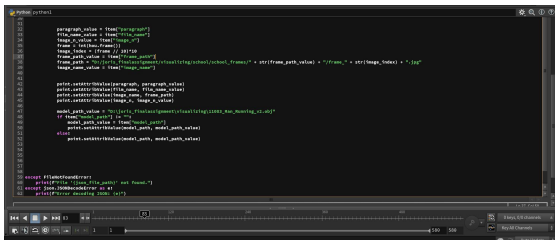
Create a JSON file based on the frames and list the paragraph,file_name,image_name,model_path,image_n,frame_path.

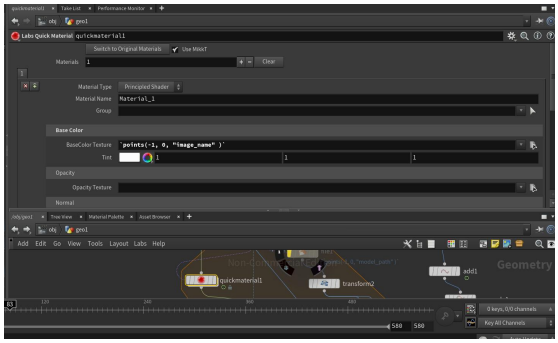
3:Visualising the process

The different tags are given the appropriate attributes via python code and assign the frame's image to the plane's material.



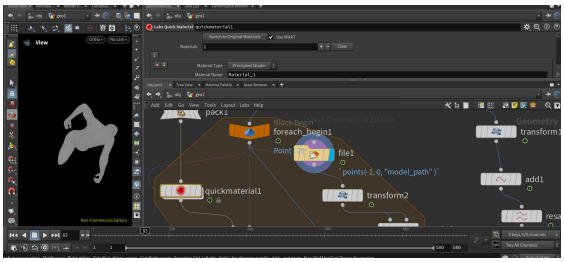


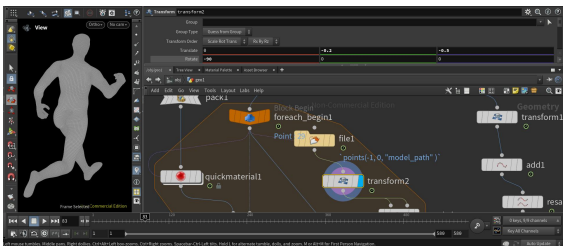




4:Load the model

Load a running man model and add a transform SOP to rotate model to ensure it in a right position.





5:Render the frame and video

Done.