COMPUTING SKILLS ASSIGNMENTS

Tutor: Julian Besems **P2 - P18**

Tutor: Joris Putteneers **P19-P36**

Tutor: Ceel Pierik **P37-P44**

RC11

Student Number: 23103502

GitHub Link: ***https://github.com/UD-Skills-2023-24/23103502.git***

SKILLS CLASS RC11 - 23/24

VECTORIAL ENCODINGS OF QUALITATIVE DOMAINS

Tutor: Julian Besems

EXPLANATION AND OUTCOMES

Student Number: 23103502

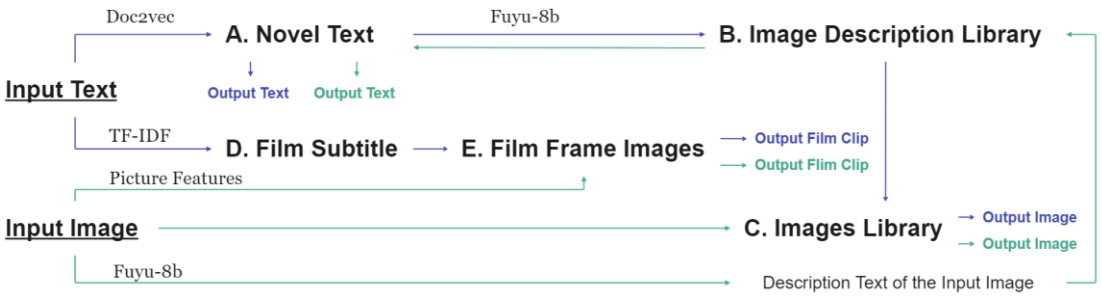
SEARCH ENGINE:

Text Input:

1. Text query activate novel text SOM to get most similar novel paragraph.
2. Search for novel paragraphs within the image Fuyu text SOM to locate the index of the image and its description.
3. Search for novel paragraphs in movie subtitles SOM to identify the most similar subtitles and display that movie clip.

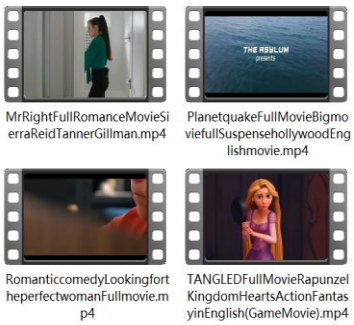
Image Input:

1. Image input activates image SOM to find the most similar image and its Fuyu description.
2. Search within movie frames SOM for the matching image to find the most similar frame and display that movie clip.
3. Search for the description text of the matched image in novel text SOM to find the most similar novel paragraph.



DATASETS:

(***OneDrive Link:*** *Film: Film Film_Frames: Film_Frames Image: Image Text: Text*)
I created a text dataset of 15 novels, a picture dataset of 2,000 characters, and a video dataset of 4 movies.

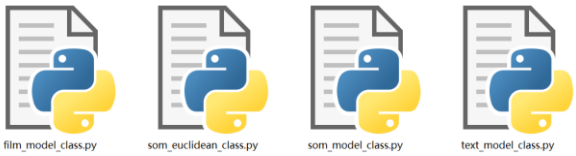


[Flim Dataset] *[Text Dataset]* *[Image Dataset]*

CODE FILES:

(***GitHub Link:*** ***https://github.com/UD-Skills-2023-24/23103502.git***)
I have written two main notebook files for executing code:
one (*23103502_CompleteVersion.jpynb*) for training various datasets and saving pickle files, and another (*23103502_ConciseVersion_SearchEngine.ipynb*) for running a streamlined search engine code that directly uses all the pickle files.
Additionally, there are four ".py" files for function calls.

[Main Notebooks]



[.py files]

EXPLANATION:

PART Ⅰ: 23103502_CompleteVersion.jpynb —— Build SOM Datasets

The following content can be found under corresponding headings and code in the notebook.

A. Text Matching Text SOM

A.1 Read the Epub

Load the novel text from the path and use TextModel import from the text_model_class.py to vectorize it. This allows for selecting different modes of vectorization.

```
NOVEL_DIR=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Text'
NOVEL_FILES = []
NOVEL_FILES.extend(glob.glob(os.path.join(NOVEL_DIR, '*.epub')))
```

```
novel_model = TextModel(NOVEL_FILES,vectorization='lsa', dimension = 300, min_df=3)
```

A.2 Vectorize Text Dataset

I selected TF-IDF for text vectorization and applied SVD for rearranging vectors and reducing dimension, which in my code is labeled as 'lsa'.

I choose Doc2Vec because it generates low-dimensional dense vectors that capture the semantic information of documents and understands the semantic relationships between words. In contrast, TF-IDF produces high-dimensional sparse vectors that represent only unique words and fail to capture semantic relationships.

Below, I will demonstrate a straightforward example of these three vectorization techniques using the same query.

```
query='Life is like a box of chocolates. You never know what you are gonna get.'
```

Text Vectorization Mode: TF-IDF

```
tfidf_train_data=tfidf_novel_model.vector_matrix

tfidf_result=tfidf_novel_model.search(query, n=3)
tfidf_result
```

```
[{'paragraph': '" He leaned confidentially across the table. "If you ever want a box cracked, look up the Lizard." "Meaning?" asked Jimmy. "Me, bo, I\'m the Lizard." "Box cracked?" repeated Jimmy. "An ice-box or a hot box?" His visitor grinned.',
  'nr': 94,
  'ID': 'burroughs-efficiency-expert',
  'type': 'epub'},
 {'paragraph': ' Get back the gold, and shoot down the robbers like dogs. They can't get away, you know. They're somewhere on this Island, and I mean to find them." "There's the ship." "What of it?" "If they get aboard and sail away we'll be in a bad box.',
  'nr': 702,
  'ID': 'baum-sam-steele-s-adventures-on-land-and-sea',
  'type': 'epub'},
 {'paragraph': ' I don\'t know whether you get me or not." "I get you," replied the Lizard, "and while you may never wear diamonds, you\'ll get more pleasure out of life than I ever will, provided you don\'t starve to death too soon.',
  'nr': 274,
  'ID': 'burroughs-efficiency-expert',
  'type': 'epub'}]
```

Text Vectorization Mode: lsa

```
lsa_train_data=lsa_novel_model.vector_matrix

lsa_result=lsa_novel_model.search(query, n=3)
lsa_result
```

```
[{'paragraph': ' I don\'t know whether you get me or not." "I get you," replied the Lizard, "and while you may never wear diamonds, you\'ll get more pleasure out of life than I ever will, provided you don\'t starve to death too soon.',
  'nr': 274,
  'ID': 'burroughs-efficiency-expert',
  'type': 'epub'},
 {'paragraph': 'Chapter VIII (Extracts from the diary of Sir Eustace Pedler, M.P.) It is an extraordinary thing that I never seem to get any peace. I am a man who likes a quiet life. I like my Club, my rubber of Bridge, a well-cooked meal, a sound wine.',
  'nr': 280,
  'ID': 'christie-man-in-the-brown-suit',
  'type': 'epub'},
 {'paragraph': ' "I would like to know what way it is with you in that shape of a beast," she said; "and I would like to know what will happen me after I get the sway over Connaught." "Indeed it is a tormented beast I am," he said, "and it is in many shapes I have been.',
  'nr': 1670,
  'ID': 'gregory-cuchulain-of-muirthemne',
  'type': 'epub'}]
```

Text Vectorization Mode: Doc2Vec

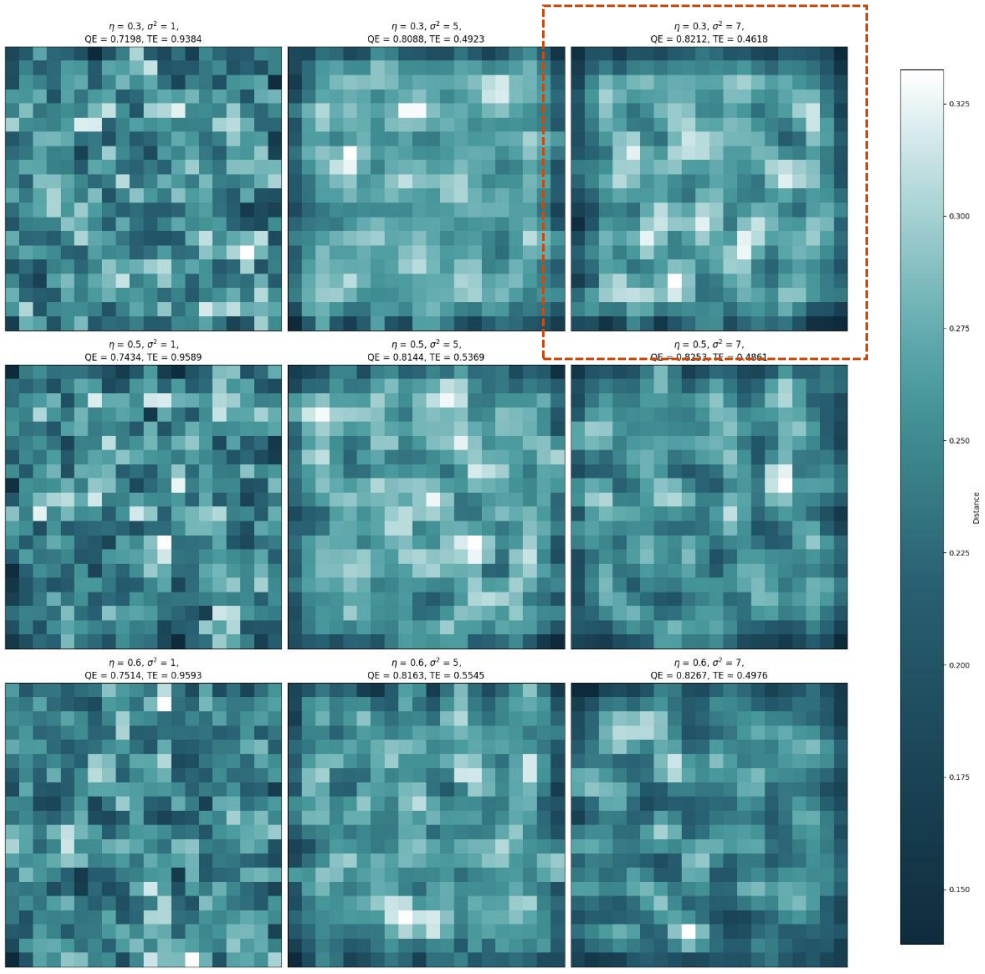
```
doc2vec_train_data=doc2vec_novel_model.vector_matrix

doc2vec_result=doc2vec_novel_model.search(query, n=3)
doc2vec_result
```

```
[{'paragraph': '"  "I'm sorry for interrupting you."  "That's all right," I said.  "I couldn't sleep. I th
ought a wash would do me good."  It sounded rather as though it were a thing I never had as a general rule.
 "I'm so sorry, miss," said the stewardess again.',
  'nr': 414,
  'ID': 'christie-man-in-the-brown-suit',
  'type': 'epub'},
 {'paragraph': '.',
  'nr': 588,
  'ID': 'burroughs-efficiency-expert',
  'type': 'epub'},
 {'paragraph': ' But how did he square Sir Eustace? It looks as though he had some hold over him."  "Or over Pa
gett," I suggested in spite of myself.  "You don't seem to like Pagett, Anne. Sir Eustace says he's a most ca
pable and hard-working young man.',
  'nr': 638,
  'ID': 'christie-man-in-the-brown-suit',
  'type': 'epub'}]
```

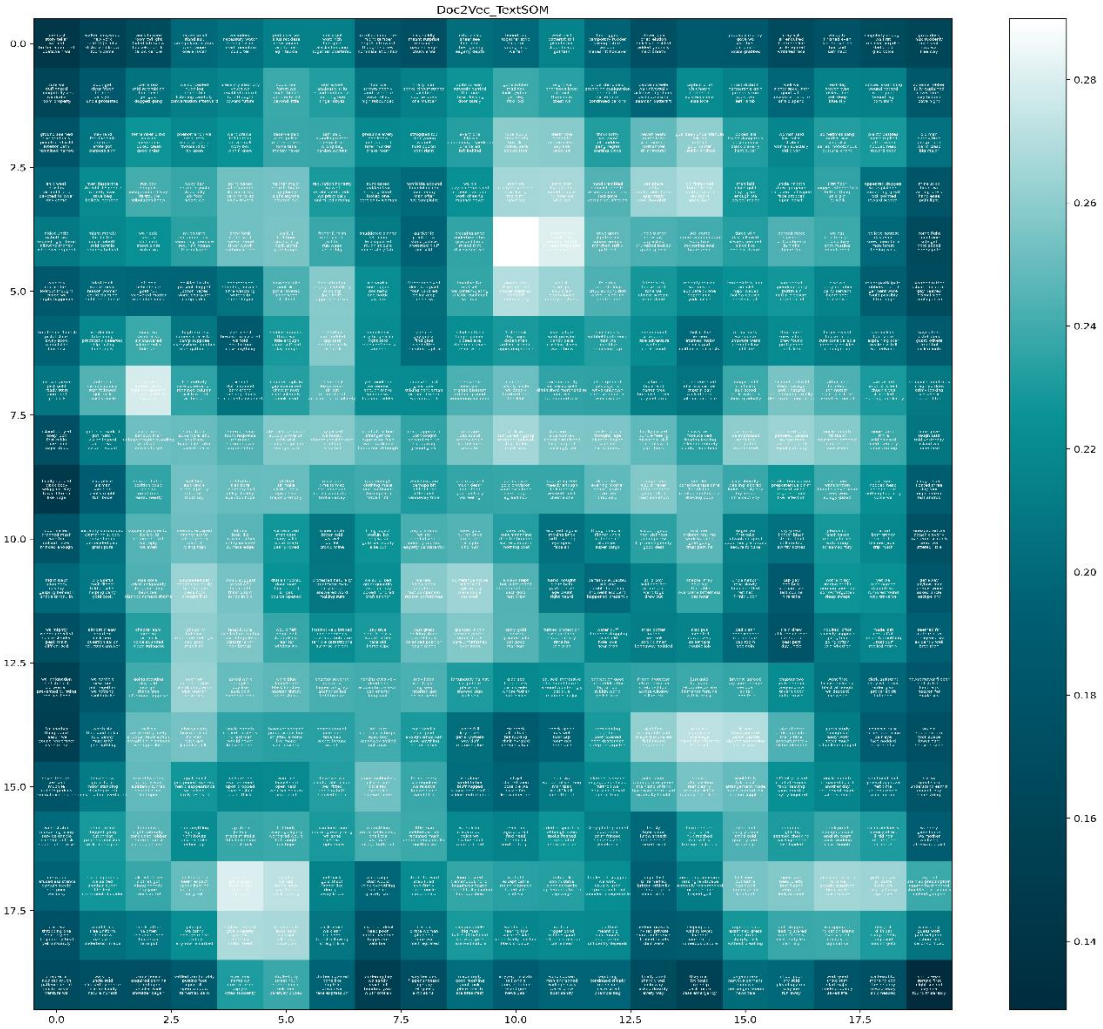
Each time I vectorize the data, I can select the method of vectorization directly in the code. I primarily developed three vectorization methods: TF-IDF, lsa (which combines TF-IDF with SVD), and doc2vec.

A.3 Build Text SOM

Build a SOM with Doc2vec with 300 dimensions.



Here I chose the SOM with the index 2 since it has a relatively lower QE as well as the lower TE. And the selected SOM U-Matrix with text data mapping on it, is shown below.



A.4 Searching BMU from Query

Then connect the paragraphs to their Best Matching Units and compile a data_dict to keep track of this information.

```
def get_TextAndVector_dict(som, original_paragraph_list, train_data):
    data_dict=[]
    for i in range(len(som)):
        row = []
        for j in range(len(som[0])):
            row.append([])
        data_dict.append(row)

    vectortextPairs=[]
    for i in range(0, len(original_paragraph_list)):
        vectortext={}
        vectortext['text']=original_paragraph_list[i]
        vectortext['vector']=train_data[i]
        vectortextPairs.append(vectortext)
    for i in vectortextPairs:
        g, h =SOMlib.find_BMU1(som, i['vector'])
        data_dict[g][h].append(i)

    return data_dict
```

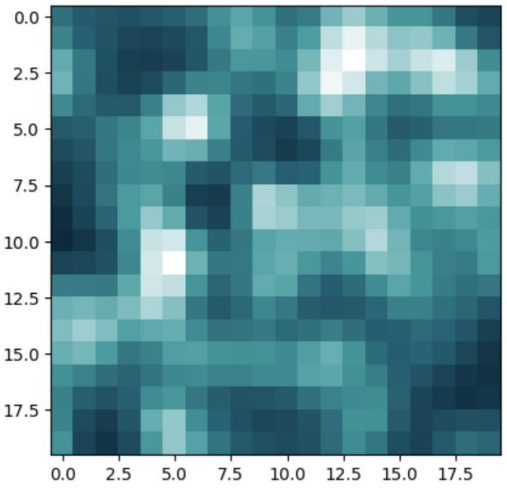
```
data_dict=get_TextAndVector_dict(som, paragraphs_list, doc2vec_train_data)
```

I created a search function to process the text query through the SOM, returning the result from the BMU that is closest in terms of the cosine distance of that lattice vector and printing it out. Simultaneously, it will display the activated SOM.

```
def search_TextSom(som, model,data_dict,query='street',vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(train_data,som, model.vectorize(query)), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query))
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    elif vectorization =='lsa':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(train_data,som, model.vectorize(query)[0]), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query)[0])
        print('The best matching unit(bmu) is: {}'.format((g, h)))

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

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

```
query='Life is like a box of chocolates. You never know what you are gonna get.'
print('The matching query is: '+ query)

train_data=doc2vec_train_data
search_TextSom(som,doc2vec_novel_model,data_dict,query,vectorization ='doc2vec')
```

```
The matching query is: Life is like a box of chocolates. You never know what you are gonna get.
```



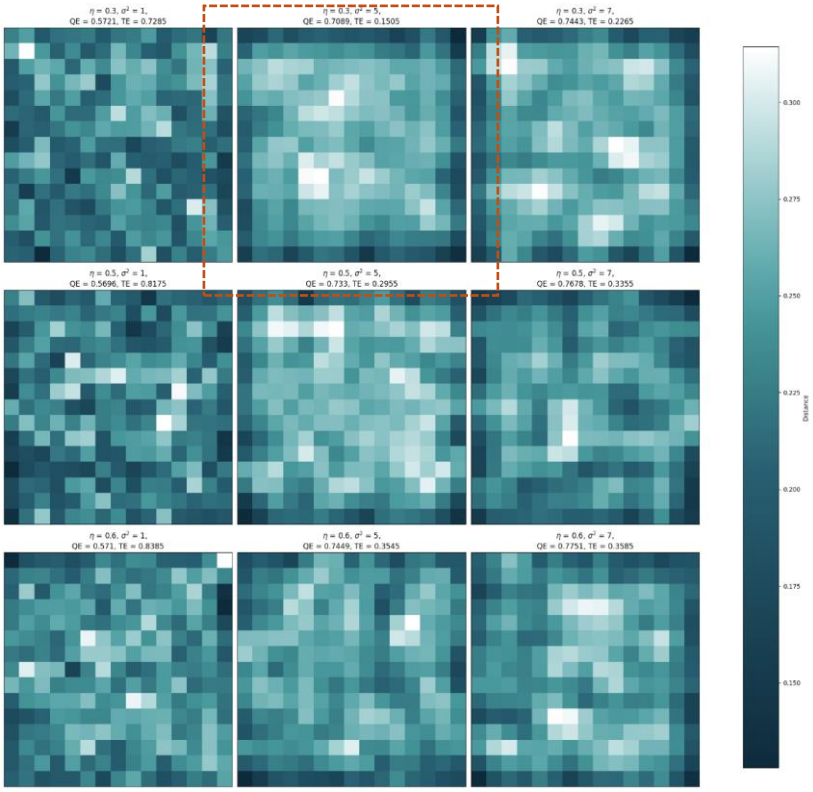

```
The best matching unit(bmu) is: (2, 13)
```

```
[{'paragraph': ' And they held the place, and fought Conchubar outside; and for a long time they did not let any
one go past them. And Gerg stood outside the door, and a hewing and cutting was aimed at him on every side, and
five men of the Fomor fell by him, and Imrinn the Druid, along with them, and he cut his head off and brought it
to the door with him.',
  'nr': '1020',
  'ID': 'gregory-cuchulain-of-muirthemne',
  'type': 'epub'}]
```

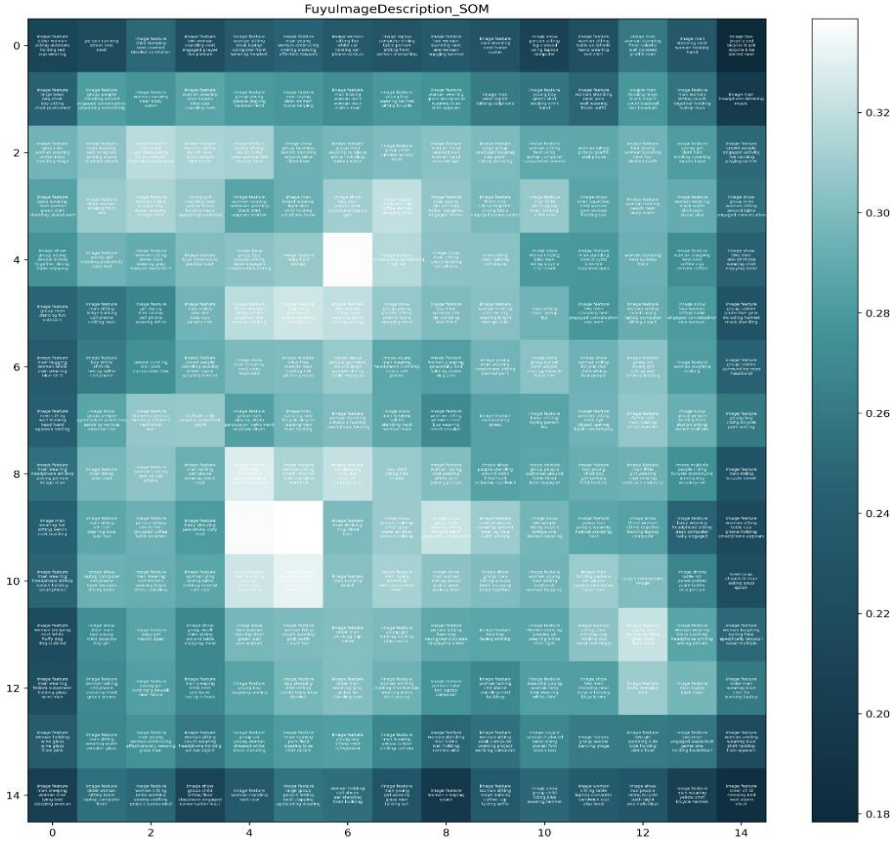
B. Text Matching Image

B.1 Build Fuyu Text SOM

Loading Image_to_Text.csv and the specific steps that follow are like those above, so I show you SOM training results directly.



The reason of SOM selection is similar with the above which is based on QE and TE value.



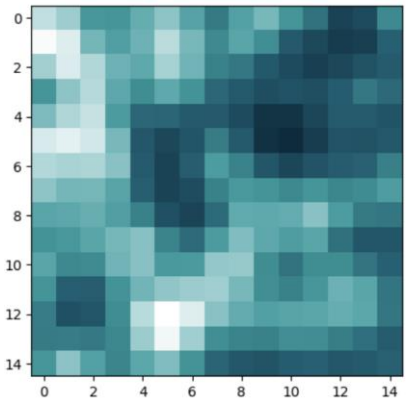
B.2 Searching the description by query

The function can search the input text query for SOM, returning the description and index of the best matched image. It displays the activated SOM and prints the matched image.

```
def search_ImageTextSom(som, model,data_dict,image_folder,query='street',vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(ImageText_train_data,som, model.vectorize(query)), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query))
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    elif vectorization =='lsa':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(ImageText_train_data,som, model.vectorize(query)[0]), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query)[0])
        print('The best matching unit(bmu) is: {}'.format((g, h)))

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

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

    # print the image
    if result[0] is not None:
        image_name=result[0]['filename']
        image_path = os.path.join(image_folder, image_name)
        img = Image.open(image_path)
        display(img)

    return result
```

```
ImageLibrary_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Image'

query=query_list[10]
print('The matching query is: '+ query)

search_ImageTextSom(ImageText_som,ImageText_model,FuyuText_data_dict,ImageLibrary_folder,query,vectorization ='doc2vec')
```

```
The matching query is: The colors nature displays at sunset are truly breathtaking.
```




```
The best matching unit(bmu) is: (12, 5)
```

```
[{'filename': '(699).jpg',
  'text': 'The image features a man walking down a street while talking on his cell phone.'}]
```

C. Image Matching Image SOM

C.1 Loading Images

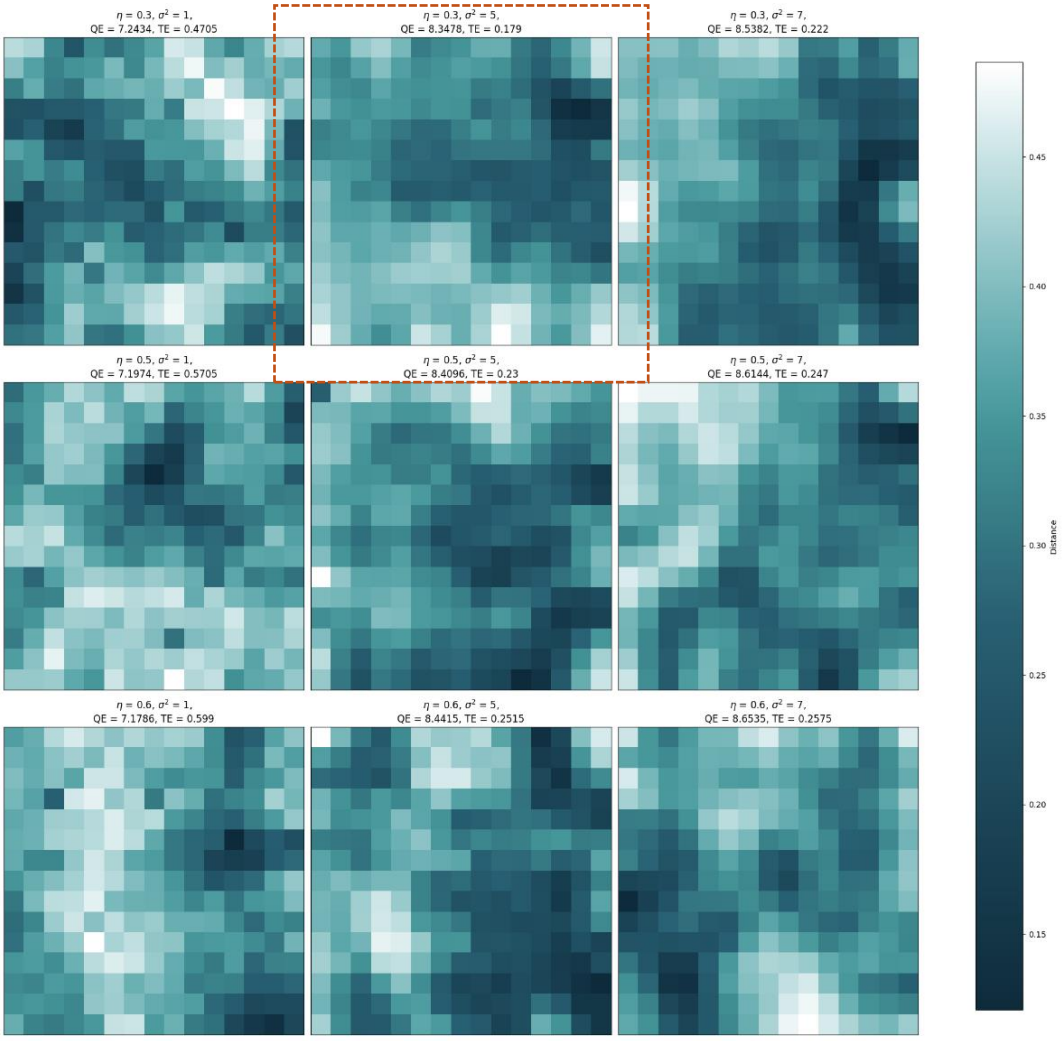
Read images from the Image Folder within Datasets and process them using an image feature extraction model.

```
ImageModel = 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'
)
```

```
Image_path=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Image'
ImageFiles = os.listdir(Image_path)
```

C.2 Build Image SOM

Create a SOM with feature with 1024 dimension and normalize the training data before training. For the 1024-dimension vector matrix, the Euclidean distance calculation will better than cosine, so I import the Euclidean distance related function from the *SOM_euclidean_class.py*.



I chose the SOM with the index1.

The image mapping is shown below:

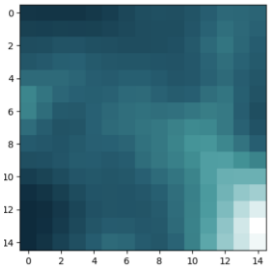
C.3 Searching the Image by input image

The Euclidean distance is used to determine the picture that most closely corresponds to the unit by measuring the distance between the unit picture vector and the unit itself. Additionally, the best matching picture is identified by locating the BMU of the new picture.

```
def search_ImageSom(path,model,som):
    result=[]
    query_features=[]

    #print query image
    img = Image.open(path)
    display(img)

    q_f = processImage(path, model)
    query_features.append(q_f)
    fig = plt.figure()
    plt.imshow(e_SOMlib.activate(n_train_data, som, query_features), cmap=cmap)
    plt.show()

    g,h=e_SOMlib.find_BMU(som,query_features)
    print('The best matching unit(bmu) is: {}'.format((g, h)))
    closest_image_index=e_SOMlib.get_closest_image(data_dict,som,g,h)
    result.append(image_data_dict[g][h][closest_image_index]['image'])

    #print result image
    if result[0] is not None:
        image_name=result[0]
        image_path = os.path.join(image_folder, image_name)
        matched_img = Image.open(image_path)
        display(matched_img)

    return result
```

```
path=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Image\(222).jpg'
image_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Image'

data_dict=image_data_dict
model=ImageModel
som=image_som

search_ImageSom(path,model,som)
```

```
1/1 [==============================] - 0s 86ms/step
```




```
The best matching unit(bmu) is: (13, 14)
```

```
['(1163).jpg']
```

D. Text Matching Film Clips

D.1 Load film subtitle database

Process the film subtitle from '.srt' subtitle file to text list and preprocessed it.

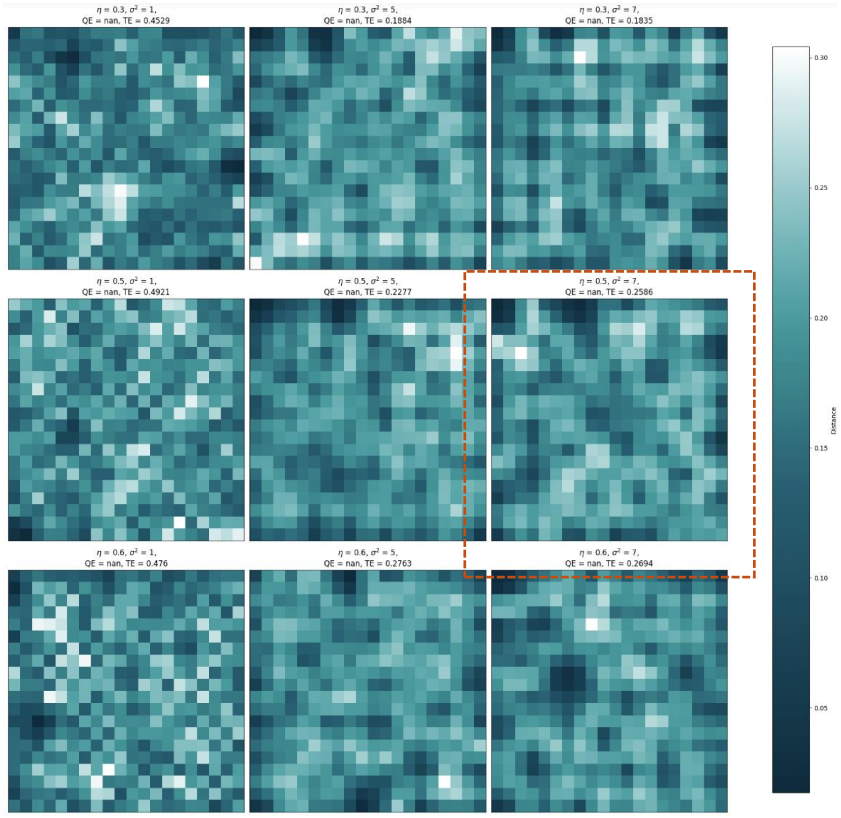
```
import Code.film_model_class as Filmlib
from Code.film_model_class import FilmSubtitleModel
```

```
film_path = r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film'
Filmlib.remove_spaces_in_filenames(film_path)
```

```
all_subtitles = Filmlib.process_srt_files(film_path)
merged_subtitles = Filmlib.merge_subtitles(all_subtitles)

merged_subtitles
```

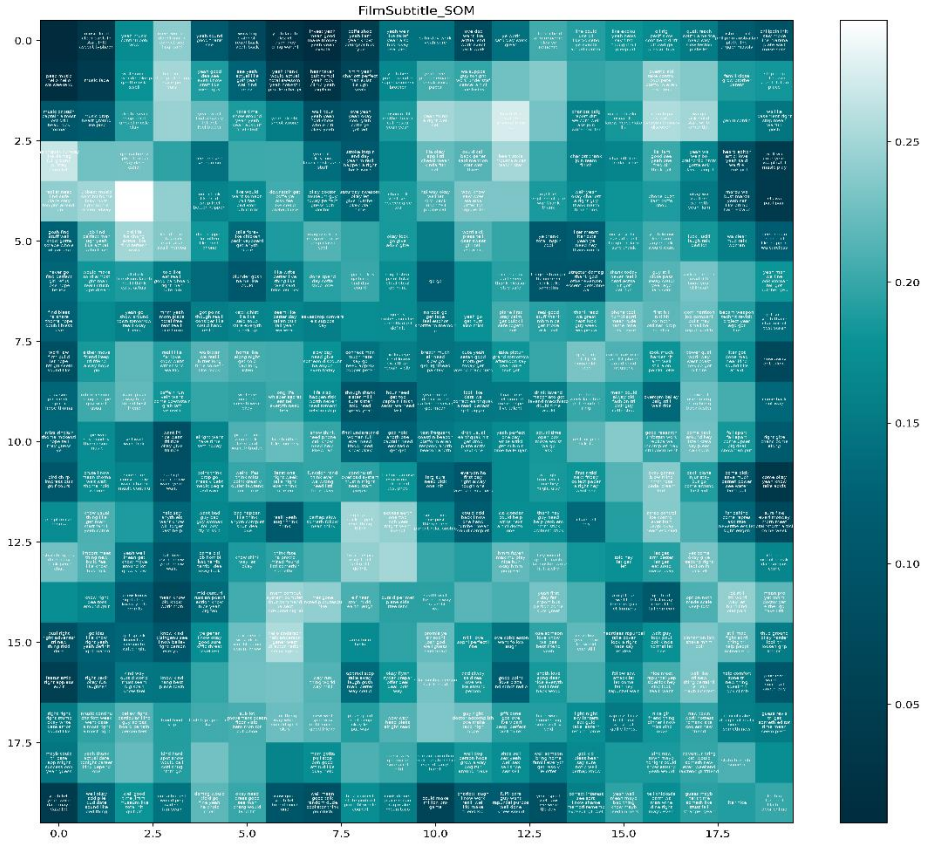
```
    'end_time': '00:00:32,698',
    'text': "It's gonna be a breeze! ♪ UPBEAT MUSIC PLAYS ♪ [ DEEP EXHALE ]",
    'file_name': 'MrRightFullRomanceMovieSierraReidTannerGillman'},
   {'index': 5,
    'start_time': '00:00:34,200',
    'end_time': '00:01:12,655',
    'text': '♪ UPBEAT MUSIC CONTINUES ♪ ♪ Home, leaving in the morning light ♪ ♪ Home, dreaming you can always
find ♪',
    'file_name': 'MrRightFullRomanceMovieSierraReidTannerGillman'},
   {'index': 6,
    'start_time': '00:01:14,240',
    'end_time': '00:01:38,389',
    'text': '♪ Gone, do you even want to try? ♪ ♪ Gone, falling for another guy ♪ ♪ ♪',
    'file_name': 'MrRightFullRomanceMovieSierraReidTannerGillman'},
   {'index': 7,
    'start_time': '00:01:40,100',
    'end_time': '00:02:11,047',
    'text': '♪ Na na na na na na ♪ ♪ ♪ ♪ MUSIC FADES ♪',
    'file_name': 'MrRightFullRomanceMovieSierraReidTannerGillman'},
   {'index': 8,
```

D.2 Build Subtitle SOM



And I chose the SOM with the index 5.

And the selected SOM U-Matrix with data mapping on it, is shown below.



D.3 Searching the Subtitle by query

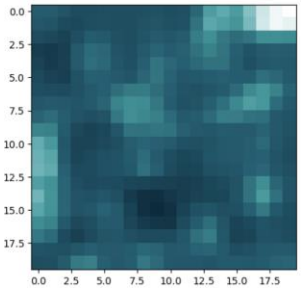
The basic searching idea of this function is like the previous text search.

```
def search_SubtitleSom(som, model,data_dict,film_folder,query='street',vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(subtitles_train_data,som, model.vectorize(query)), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query))
        print('The best matching unit(bmu) is: {}'.format((g, h)))

    elif vectorization =='lsa':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(subtitles_train_data,som, model.vectorize(query)[0]), cmap=cmap)
        plt.show()
        g, h = SOMlib.find_BMU1(som, model.vectorize(query)[0])
        print('The best matching unit(bmu) is: {}'.format((g, h)))

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

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

    # Display the Film:
    if result[0] is not None:
        matched_film_path=Filmlib.check_film_exist(result,film_folder)
        Filmlib.display_matched_filmclip(result, matched_film_path,query)

    return result
```

After matching the most similar subtitles, it will return the start time and end time of the subtitles according to the index and use the ffmpeg command to display the film clips.

```
query='May the Force be with you.'
print('The matching query is: '+ query)

film_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film'
search_SubtitleSom(subtitle_som,film_model,subtitle_data_dict,film_folder,query,vectorization ='lsa')
```

```
The matching query is: May the Force be with you.
```




```
The best matching unit(bmu) is: (0, 19)
PlanetquakeFullMovieBigmoviefullSuspensehollywoodEnglishmovie
Found: C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film\PlanetquakeFullMovieBigmoviefullSuspensehollywoodEnglishmovie.mp4
```

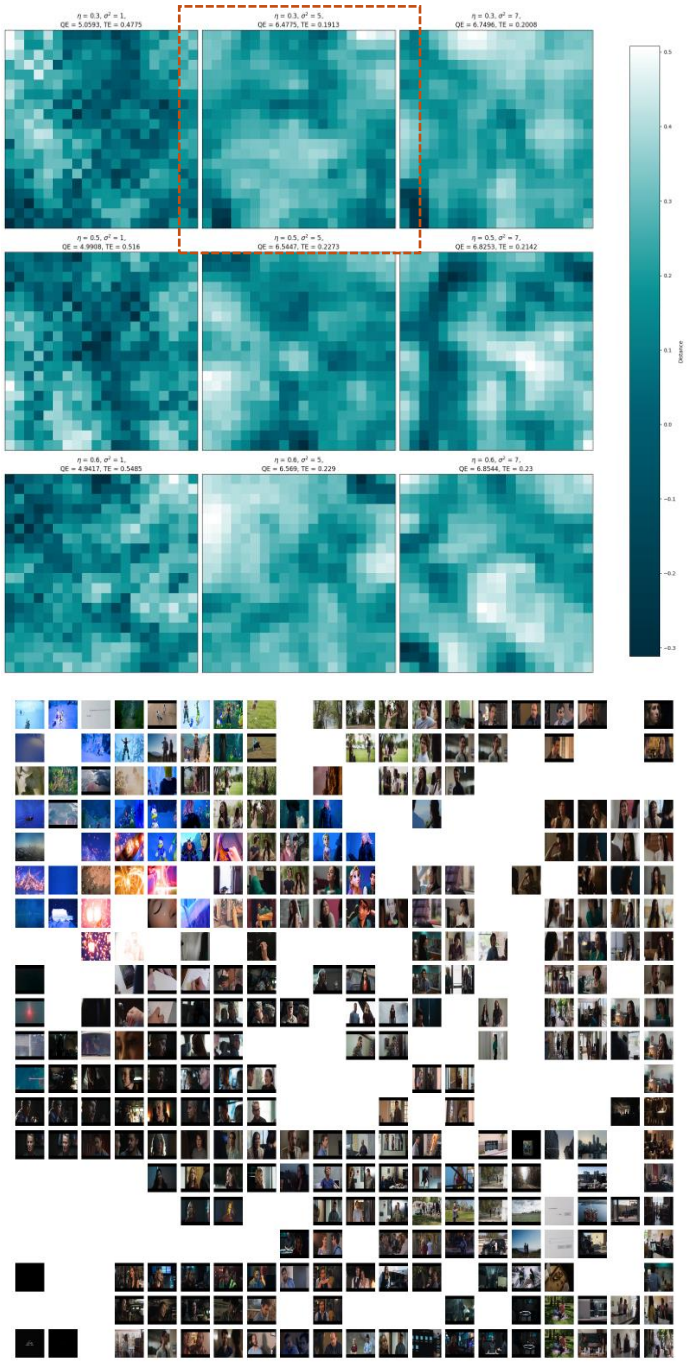

```
[{'index': 1180,
  'start_time': '01:13:11,560',
  'end_time': '01:13:16,440',
  'text': 'freeze and Harden the magma along the plate line and stop the movement from transferring from one plate to another',
  'file_name': 'PlanetquakeFullMovieBigmoviefullSuspensehollywoodEnglishmovie'}]
```

E. Image Matching Film Frame

E.1 Loading Film Frames

The implementation logic is the same as before and will not be repeated here. The data set I'm using here is the number of frames of the movie, and I get 1000 frames per movie.

E.2 Build film frames SOM



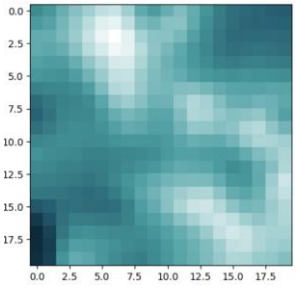
E.3 Searching the Film by Input Image

The idea is like the search for film subtitles above. However, the distinction lies in how, upon identifying the most similar film frame, a 2-second segment surrounding that frame is extracted based on its timestamp to produce the matched clip.

```python
def search_FramesSom(path,model,som,all_frames_info):
    result=[]
    query_features=[]

    #print query image
    img = Image.open(path)
    display(img)

    q_f = processImage(path, model)
    query_features.append(q_f)
    fig = plt.figure()
    plt.imshow(e_SOMlib.activate(n_train_data, som, query_features), cmap=cmap)
    plt.show()

    g,h=e_SOMlib.find_BMU(som,query_features)
    print('The best matching unit(bmu) is: {}'.format((g, h)))
    closest_image_index=e_SOMlib.get_closest_image(data_dict,som,g,h)
    result.append(data_dict[g][h][closest_image_index]['image'])

    #print result image
    if result[0] is not None:
        image_name=result[0]
        image_path = os.path.join(frames_folder, image_name)
        matched_img = Image.open(image_path)
        display(matched_img)
        print(result)

        # display the clip
        film=getFilmAroundFrame(film_path,all_frames_info,result,2)
        film_name=all_frames_info[find_idx(all_frames_info,result)]['movie_file_name']

    return film_name
```

```python
path=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Image\(542).jpg'
frames_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film_Frames'
film_path=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film'

data_dict=frames_data_dict
model=FramesModel
som=frame_som

search_FramesSom(path,model,som,all_frames_info)
```

1/1 [==============================] - 0s 55ms/step



The best matching unit(bmu) is: (2, 6)

['MrRightFullRomanceMovieSierraReidTannerGillman_frame_916.png']

'MrRightFullRomanceMovieSierraReidTannerGillman.mp4'

PART II : 23103502_ConciseVersion_SearchEngine.jpynb

To make the **Search Engine** code look more concise, this part directly uses code written with the pickle file trained from the datasets in part Ⅰ.

The following content can be found under corresponding headings and code in the notebook.

1. Create Input Function

The two functions here mainly play the role of concatenation, that is, input text and picture, can get the remaining different modes of data.

Text Input

```
def search_from_inputText(query):
    print('The matching query is: '+ query)
    text_result=search_TextSom(text_som,doc2vec_novel_model,doc2vec_data_dict,query,vectorization ='doc2vec')
    print('\n'+'The best matching text result is: '+str(text_result)+'\n')

    ImageLibrary_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Image'
    image_result=search_ImageTextSom(ImageText_som,ImageText_model,FuyuText_data_dict,ImageLibrary_folder,query,vectorization ='doc2vec')
    print('\n'+'The best matching image result is: '+str(image_result)+'\n')

    film_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film'
    film_result=search_SubtitleSom(subtitle_som,film_model,subtitle_data_dict,film_folder,query,vectorization ='lsa')
    print('\n'+'The best matching film clip result is: '+str(film_result))
```

Image Input

```
def search_from_inputImage(img_path):
    #print query image
    print('The input image is: ')
    img = Image.open(img_path)
    display(img)

    image_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Image'
    image_result=search_ImageSom1(img_path,ImageModel,image_som,image_folder)
    print('\n'+'The best matching image result is: '+str(image_result)+'\n')

    frames_folder=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film_Frames'
    film_path=r'C:\Users\12179\Desktop\Julian_SkillClass_Assignment\Dataset\Film'
    film_result=search_FramesSom1(img_path,FramesModel,frame_som,all_frames_info,frames_folder,film_path)
    print('\n'+'The best matching film clip result is: '+str(film_result)+'\n')

    query= get_image_text(ImageText_list,image_result)
    text_result=search_TextSom(text_som,doc2vec_novel_model,doc2vec_data_dict,query,vectorization ='doc2vec')
    print('\n'+'The best matching text result is: '+str(text_result))
```

Video Input

This code can also use video as input, the difference is that the logic is to convert the video into a frame image and then search, essentially the same as using an image as input, so I only demonstrated two methods.

2. Search Engine Examples

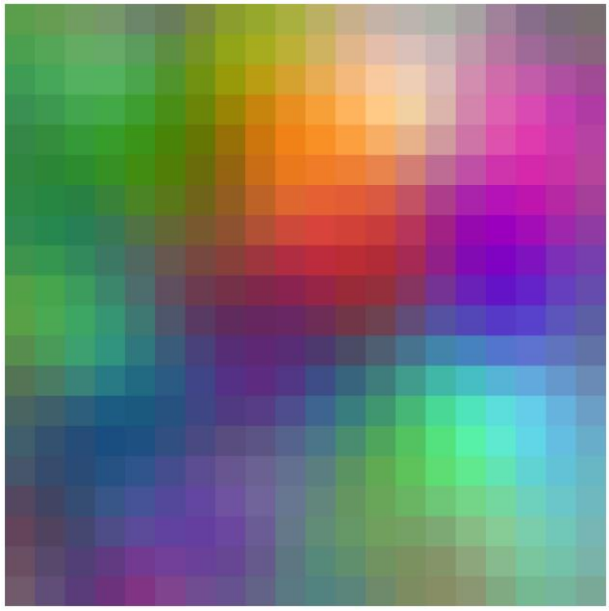
I use text and images as input in my notebook and provide 10 samples for each input.

PARTIII: Fit PCA

The following content can be found under corresponding headings and code in the notebook.

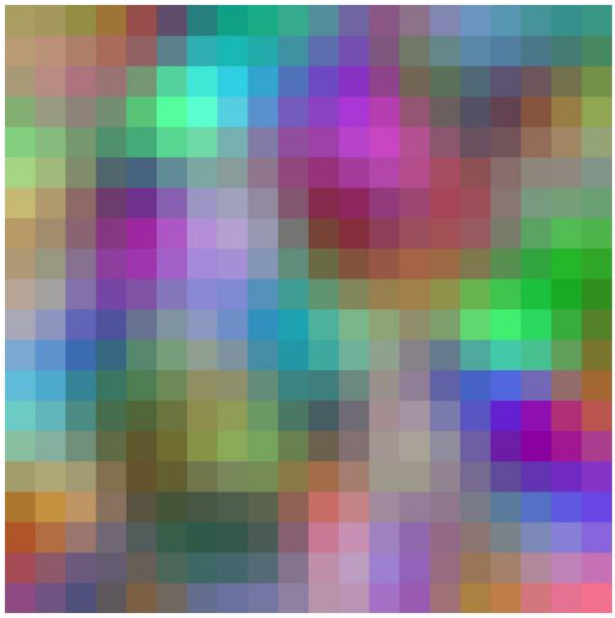
1. Fit PCA on all cells

Each cell's vector in the Doc2vec SOM is reduced to three dimensions through PCA, with these three dimensions assigned as RGB values representing colors to that cell. The displayed result appears as follows:



2. Fit PCA on the entire training

The vector for each cell in the Doc2vec SOM is compressed into three dimensions across the entire database using PCA, and these dimensions are allocated to each specific cell as RGB values representing colors. Upon rendering these back to the SOM, the outcome is displayed as follows:



SKILLS CLASS RC11 - 23/24

HOUDINI ASSIGNMENT

Tutor: Joris Putteneers

Student Number: 23103502

GitHub Link: https://github.com/UD-Skills-2023-24/23103502.git

1 Houdini Fundamentals

Blooming Flower

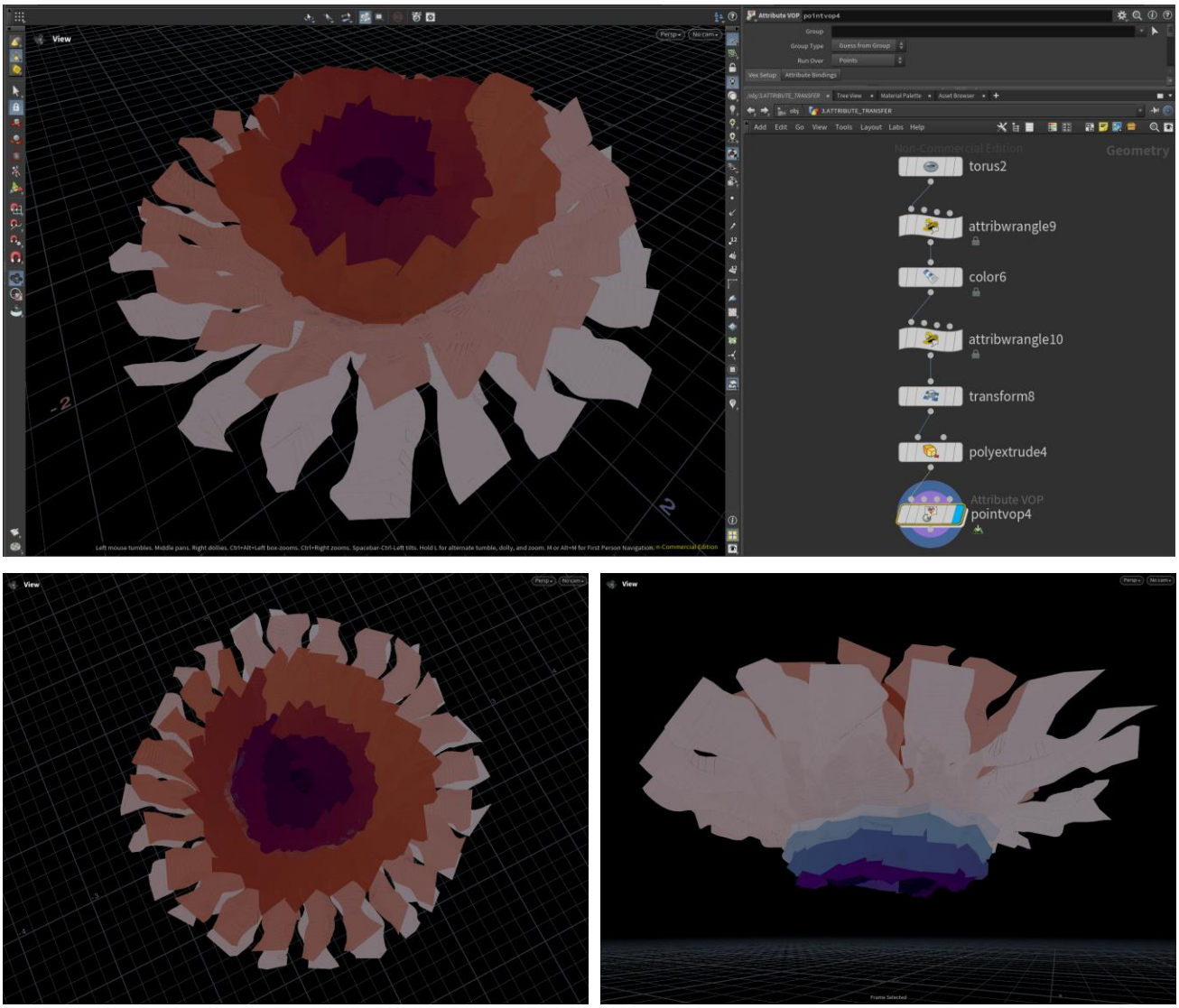
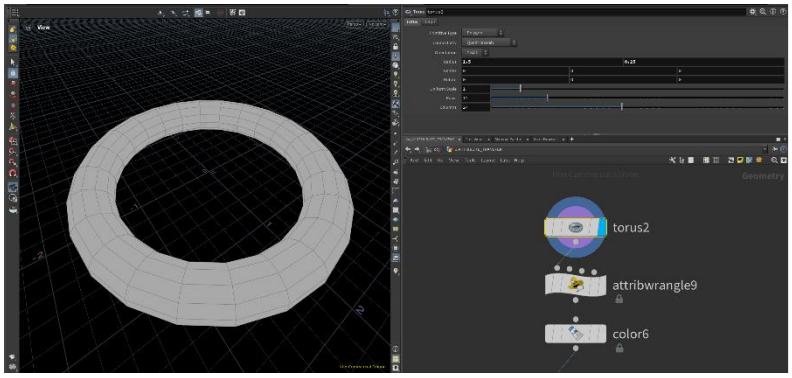


Figure 1: Houdini Fundamentals: Setup 1 Overview

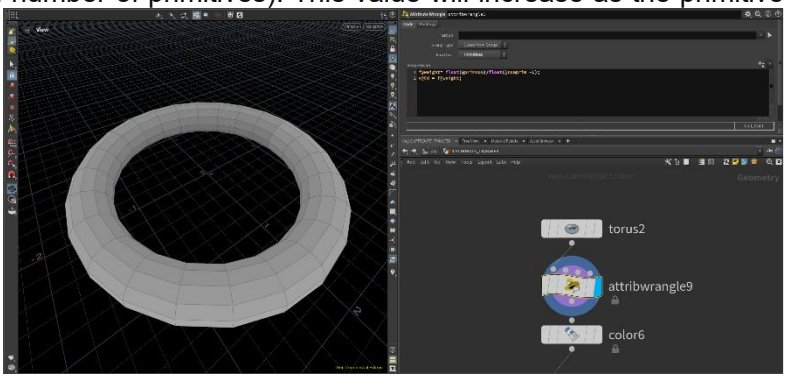
1: Torus SOP

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



2: Attribute Wrangle SOP

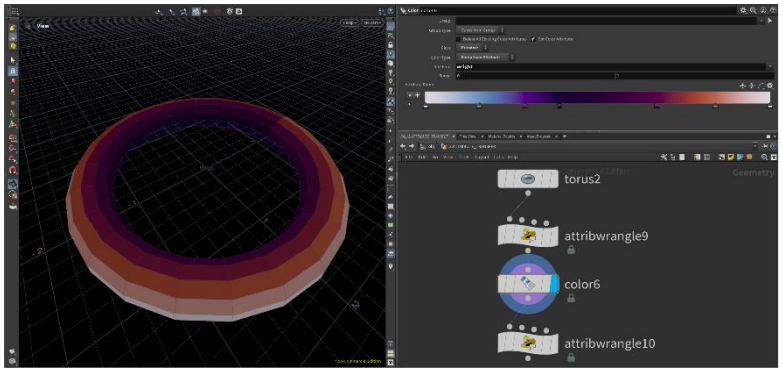
I create an ‘Attribute Wrangle’ SOP, where I make a ‘weight’ attribute. The attribute is of type float as described by the ‘f@’ declaration. ‘@primnum’ represents the index number of the current primitive (like a polygon face). ‘@numprim’ represents the total number of primitives in the object. This code is calculating a ratio by dividing the current primitive's index by the total number of primitives minus one (since the last primitive's index is one less than the total number of primitives). This value will increase as the primitive index increases, ranging from 0 to almost 1 (excluding 1, because indices start at 0).



‘@cd’ is a three-element vector used to store color information (RGB). It is being assigned the value of ‘f@weight’, which aims to create a gradient effect based on the primitive index.

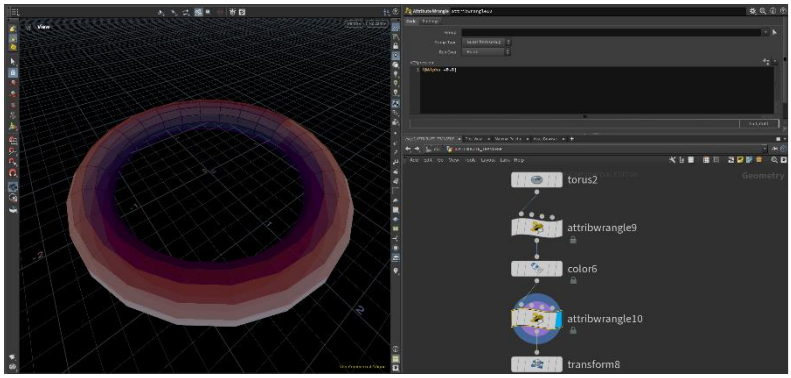
3: color SOP

The color6 node is set to use the ‘Ramp from Attribute’ mode for generating color. This mode drive color based on the value of a specific attribute. A visualisation of the ‘weight’ primitive attribute according to the ‘twilight’ color scheme. The ‘weight’ value near ‘0 ‘ display the color at the left end of the ramp (purple), while values near ‘1’ would display the color at the right end of the ramp (orange).



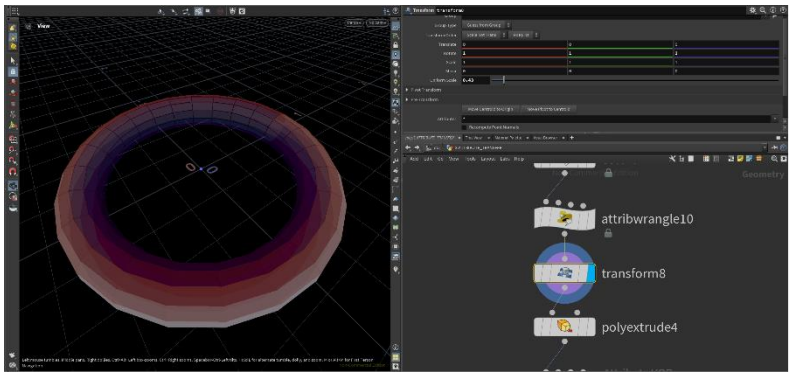
4: Attribute Wrangle SOP

I create another ‘Attribute Wrangle’ SOP to make this geometry a little more transparent. This line of code defines a new floating-point attribute named ‘Alpha’ with a value of 0.8. ‘Alpha’ is commonly used to represent opacity, where ‘1.0’ is fully opaque and ‘0.0’ is fully transparent. Thus, a value of ‘0.8’ for this attribute implies the geometry is partially transparent, but mostly opaque.



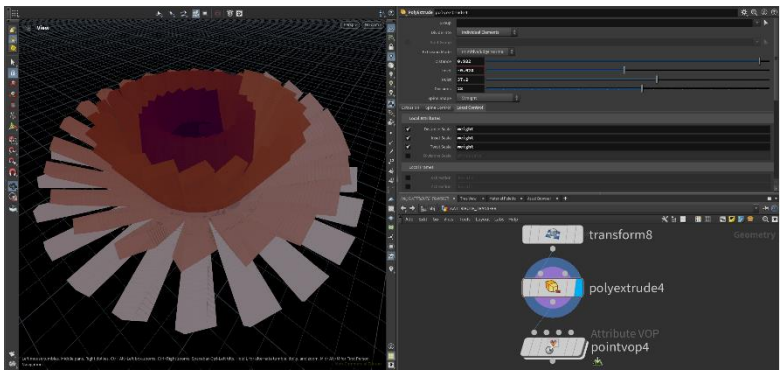
5: Transform SOP

Then, I lay down a ‘Transform’ SOP. After I did the rest, I tried to debug the ‘Translate,’ ‘Rotate,’ and ‘Scale’ parameters and found that the geometry improved when the parameters were left unchanged. At the same time, when I adjusted the ‘Uniform Scale’ to 0.43, which indicates that the geometry is uniformly scaled to 43% of its original size in all directions, the heart of the ‘Blooming Flower’ looked better.



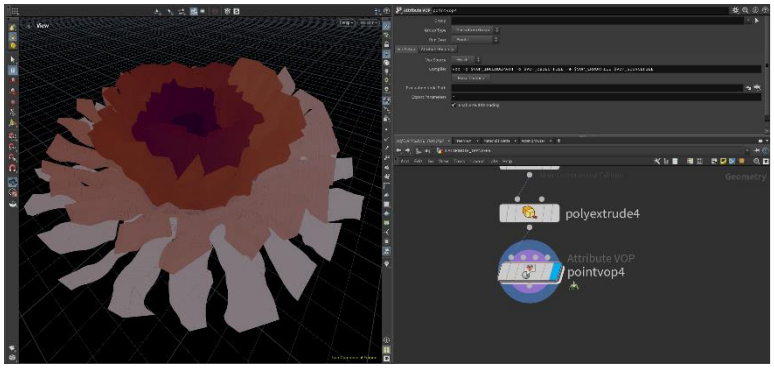
6: Poly Extrude SOP

Then I did the ‘polyextrude’ operation. This node is configured to treat each face as an ‘Individual Element,’ allowing each one to be extruded separately rather than as a cohesive group. The extrusion is performed with a ‘Distance’ of 0.932 units along the normal direction, an ‘Inset’ of -0.024 to scale each face inward, and a ‘Twist’ of 37.2 degrees to rotate each face around its normal as it extrudes. To add complexity and smoothness, the operation includes ‘Divisions’ of 28, creating a subdivided extrusion. Local attributes are also finely controlled by the 'weight' attribute, which scales the distance, inset, and twist of the extrusion, enabling a varied and detailed transformation of the geometry.

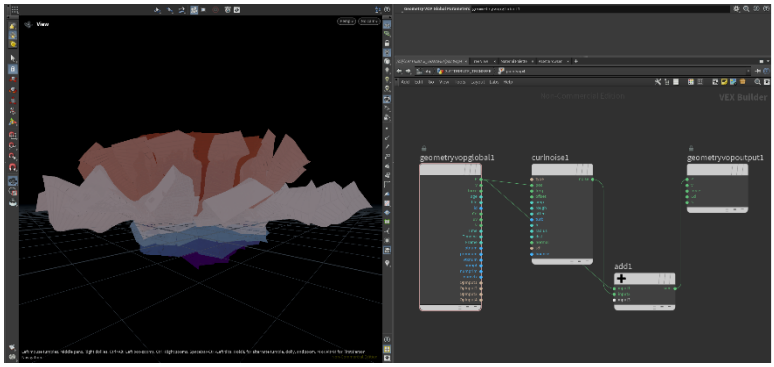


7: Point Vop SOP

To make this geometry as soft as a flower, I lay down a Point Vop SOP. The settings of the pointvop 4 indicate it operates on points. This means any operations conducted within this node will affect each point of the geometry.



In the VEX Builder, curlnoise1 is typically used to generate positions or forces with curl noise, a type of noise that can create effects resembling vortices or other natural phenomena. In this network, the output of the curlnoise1 node is connected to an add node, suggesting that the noise values are being used to alter the positions of the points, creating more complex geometric shapes.



The parameters of curlnoise1 allow for adjustments of the noise type, position, offset, amplitude, roughness, etc. These parameters collectively affect the characteristics of the generated noise and the resulting effect. The geometryvopoutput1 node indicates that all modifications will be written back to the point attributes of the geometry. Such a setup is used to create special deformation effects.

Meteor Shower

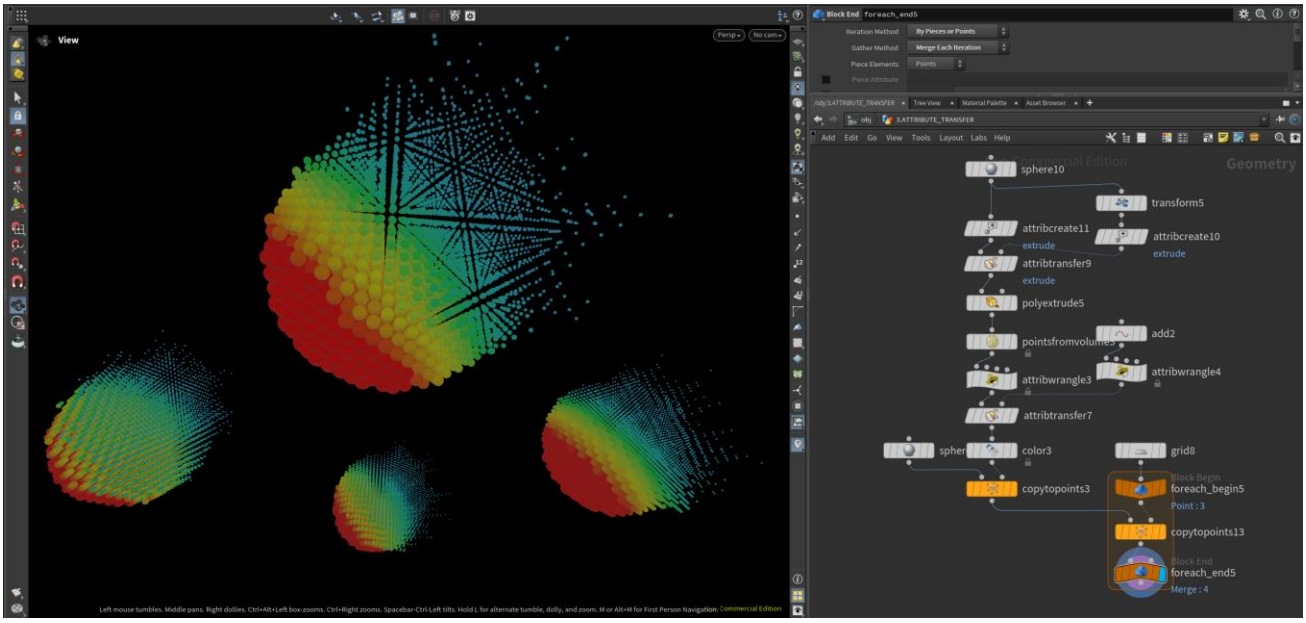
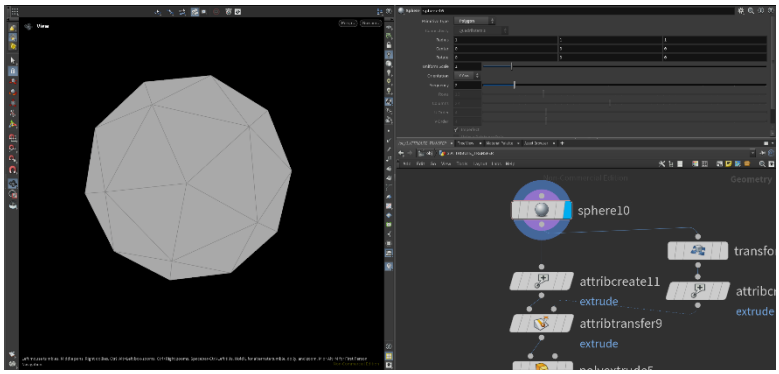


Figure 2: Houdini Fundamentals: Setup 2 Overview

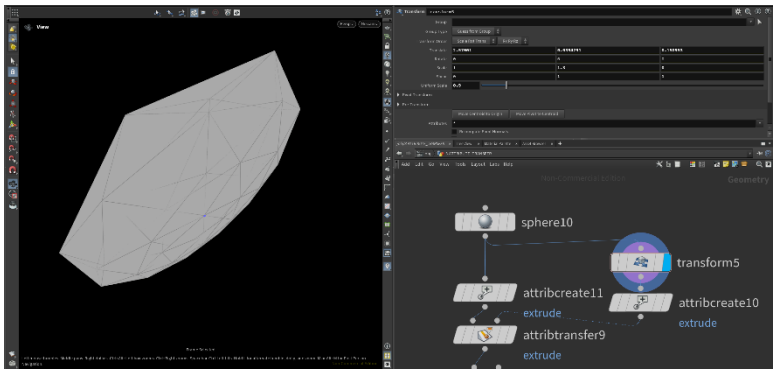
1: Sphere SOP

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



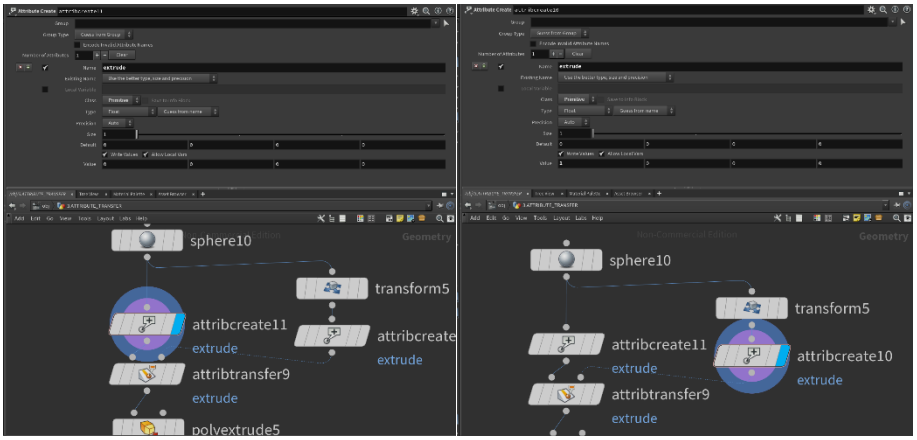
2: Transform SOP

The ‘transform5’ node is being used to adjust the position and shape of the geometry. This step is to make the geometry more like a meteor.



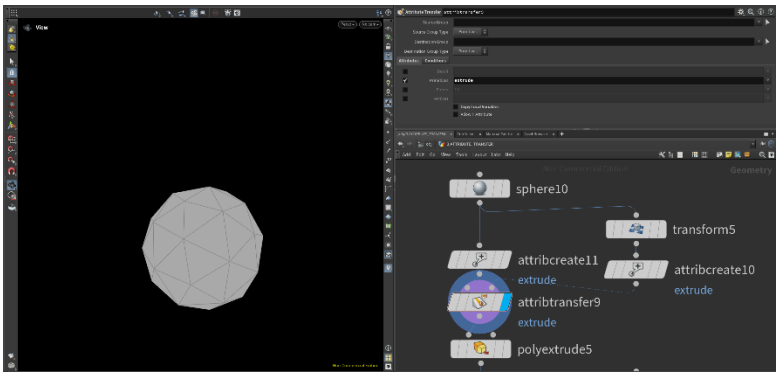
3: Attribute Create SOP

I create an ‘attribcreate11’ node with a primitive attribute named ‘extrude’, whose class is of type float and whose value is set to ‘0’. Similarly, I also create the ‘attribcreate10’ node, whose value is set to ‘1’.



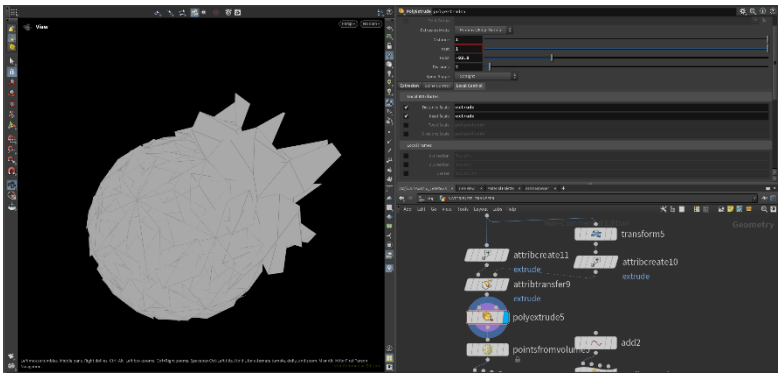
4: Attribute Transfer SOP

The ‘attribtransfer9’ node is used to pass an attribute named ‘extrude’ created by the ‘attribcreate10’ and ‘attribcreate11’ nodes. The transfer of this property is used for subsequent extrusion, deformation, etc., so that they change according to ‘0’ to ‘1’.



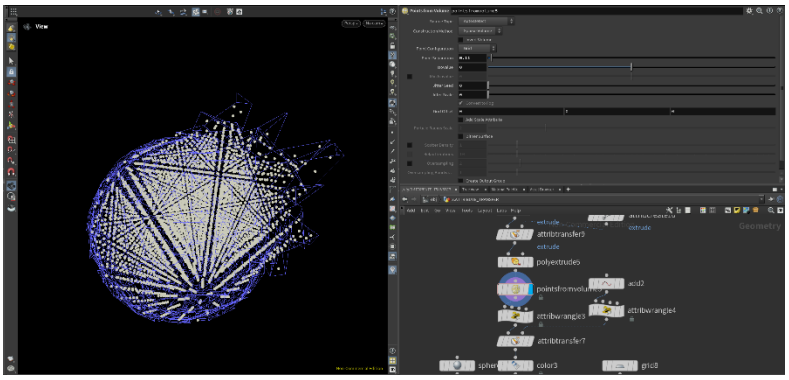
5: Ploy Extrude SOP

In this screenshot, I am using the ‘polyextrude5’ node to perform an extrusion operation on a geometry. I have set the extrusion distance to ‘1’, and the inset to ‘1’, which creates a narrower banding effect on the edges of each face. Additionally, I have applied a twist of ‘-93.8’ degrees, which causes each extruded face to rotate around its normal, creating a complex twisting effect. I also set two divisions, which smooths out the extruded section and allows the twist to have a gradient effect along the height of the extrusion.



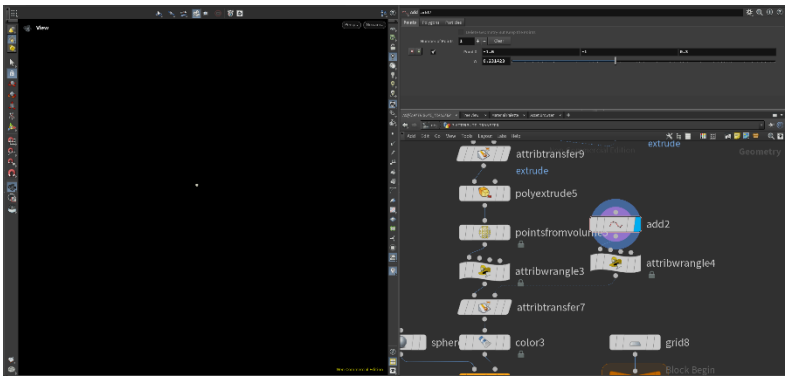
6: Points from Volume SOP

Then, I use the ‘pointsfromvolume5’ node to generate points from a volume. I've chosen the Grid as the point configuration method and set the Point Separation to 0.11 to determine the density of points. This process creates a series of evenly distributed points that fill the volume of the original geometry.



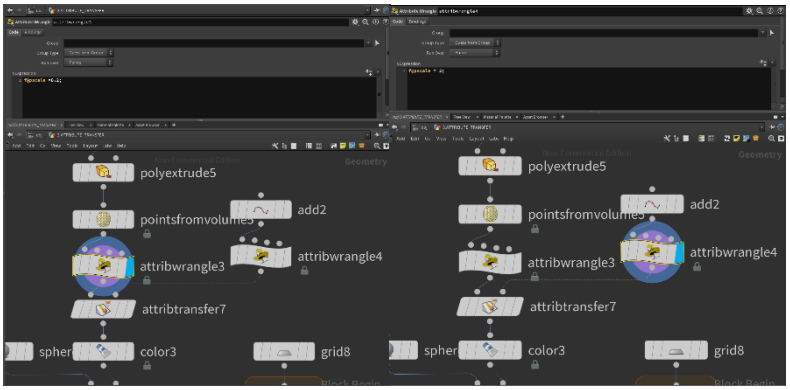
7: Add SOP

In this step, I used the ‘add2’ node to create the individual points. I set up a specific location to give it a similar role to the light source.



8: Attribute Wrangle SOP

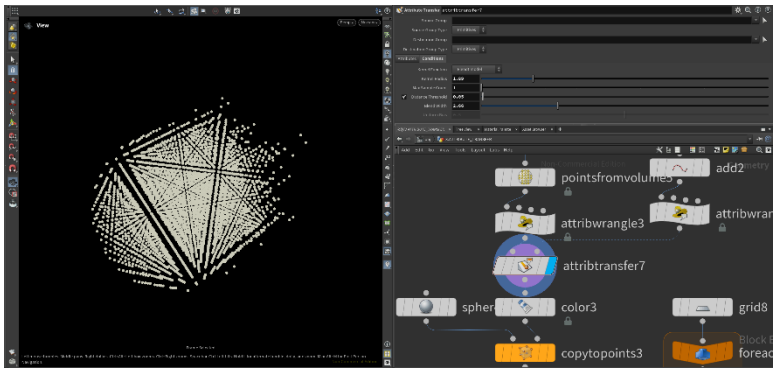
In these two steps, I first set a point scale attribute ‘pscale’ using the ‘attribwrangle3’ node, where I assigned a smaller value to ‘pscale’, affecting the size of the points.
Then, in the ‘attribwrangle4’ node, I increased the ‘pscale’ value to ‘2’, thereby enlarging the points. These



actions are in preparation for visual effects where the size of the points is crucial to the outcome.

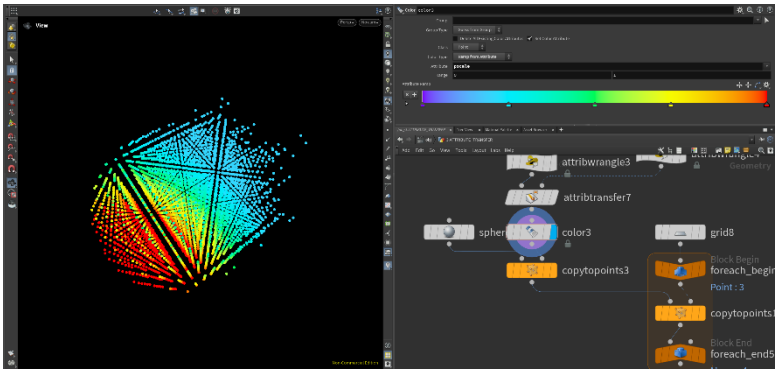
9: Attribute Transfer SOP

In this step, I am using the ‘attribtransfer7’ node. My goal is to transfer certain attributes to the point cloud, affecting their appearance. I've selected a blend model and associated parameters, which will determine how the attributes are blended and transferred.



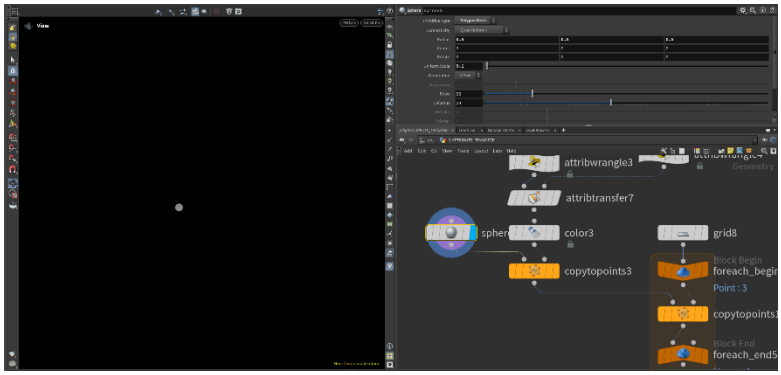
10: Color SOP

Then, I've applied color to the points in my Houdini scene using the ‘color3’ node. I mapped the ‘pscale’ attribute to a color ramp, which gave each point a color based on its size. Because of the ‘add2’ node, the results in the view show a color gradient from one end of the color spectrum to the other, indicating a change in the ‘pscale’ value of the point. And I also chose a color close to the falling meteor.



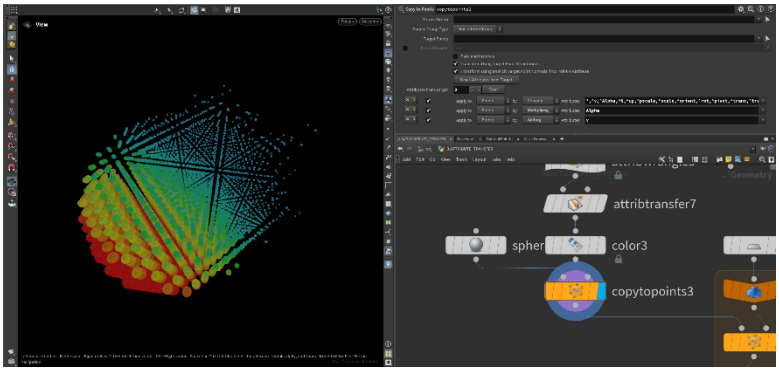
11: Sphere SOP

In this step, I'm using the ‘sphere3’ node to create a small, scaled-down sphere geometry. I’ve adjusted the Uniform Scale to 0.1, because I need a smaller element for further operations.



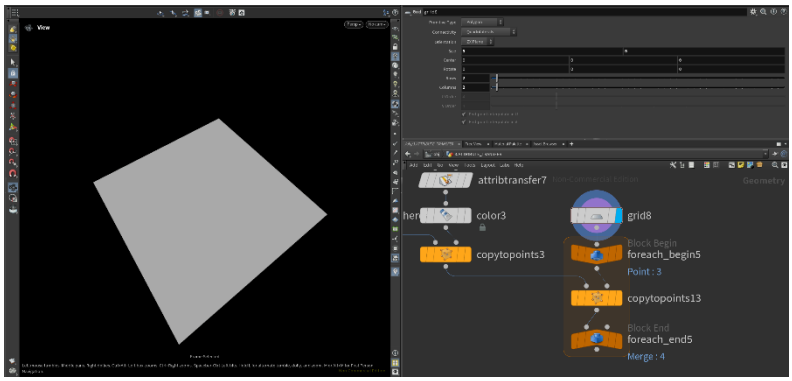
12: Copy to Points SOP

I'm using the ‘copytopoints3’ node to instance geometry onto points. By matching attributes ‘pscale’from points to the instanced geometry, I'm creating a variation in size, which is visible as a gradient of different sized geometries in the viewport.



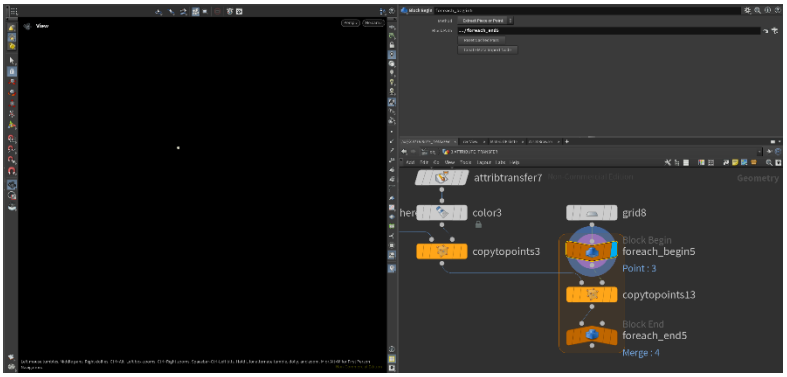
13: Grid SOP

Here, I'm working with the ‘grid8’ node in Houdini to create a simple grid. I've set the grid size and defined the number of rows and columns, which gives me a foundation for more complex operations.



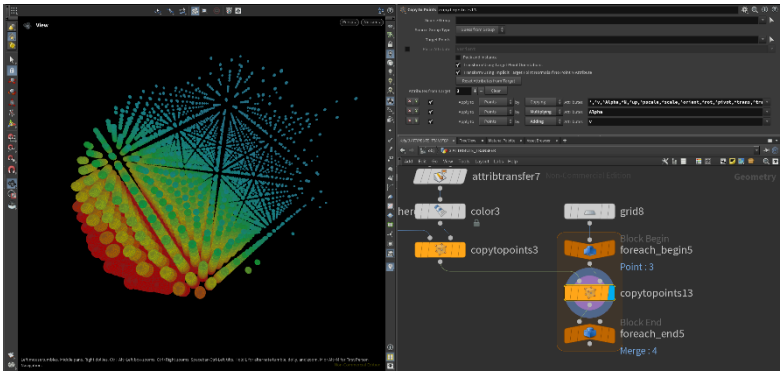
14: For Each Begin SOP

I'm setting up a foreach loop with the ‘foreach_begin5’ and ‘foreach_end5’ nodes. The loop will execute the network of nodes inside it for each iteration, which is a powerful way to apply operations to multiple items in a controlled and procedural manner.



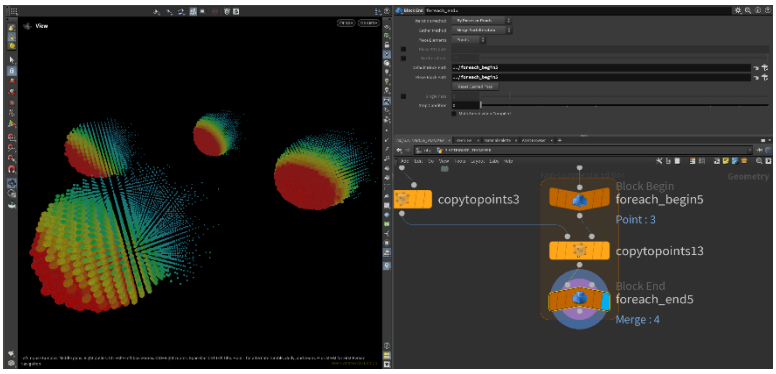
15: Copy to Points SOP

In this step, I’m using the ‘copytopoints13’ node to copy geometry onto points. This allows me to instance an object at each point position.



16: For Each End SOP

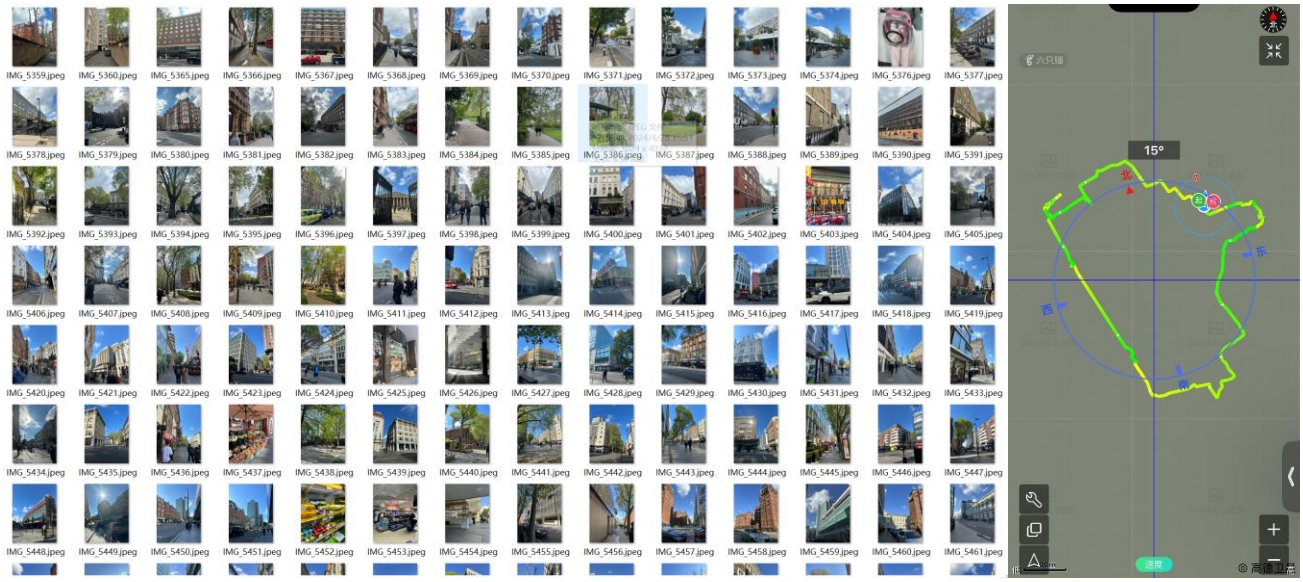
And that's when a meteor shower is formed.



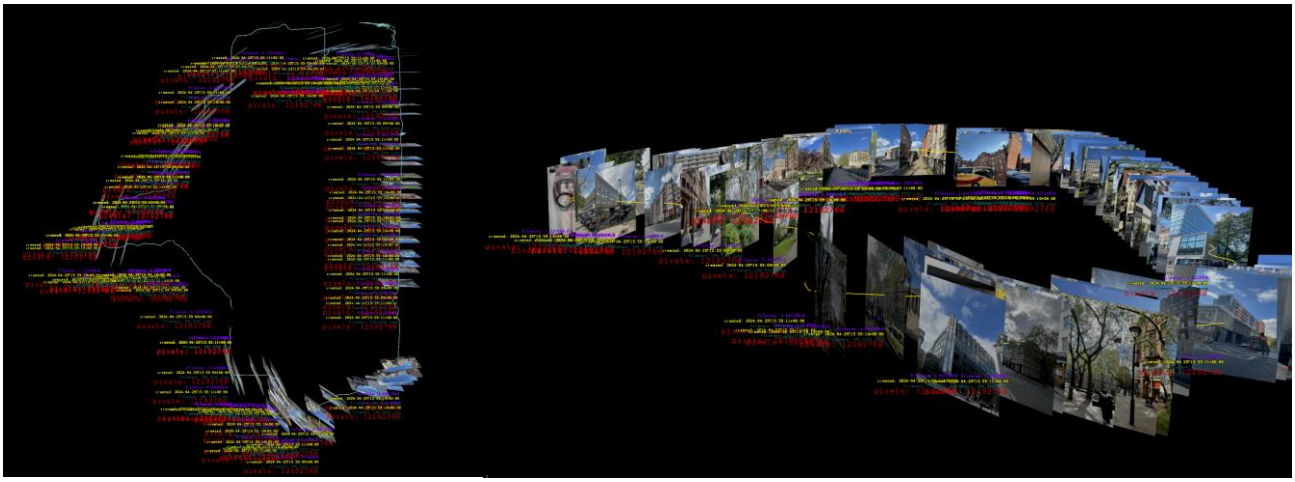
2 Visualizing GPS and Image Metadata

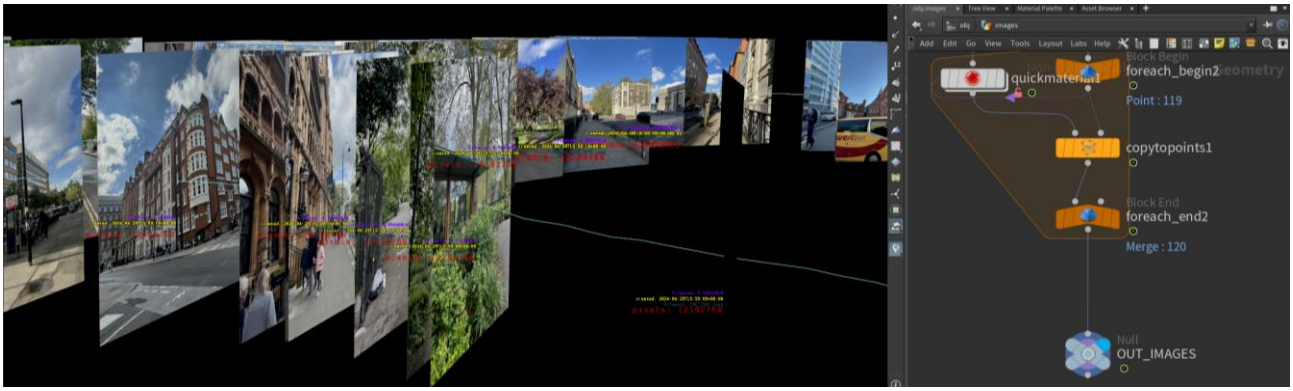
• Load in your own images and GPS data.

When I walked around London, I took 126 photos along the way and recorded my movements. These are screenshots of 126 photos I imported into Houdini. Based on the Timeline in my Google Map cannot be displayed, I found a Chinese mobile app called Six Feet, that can record travel routes, generate data, and generate GPX files.



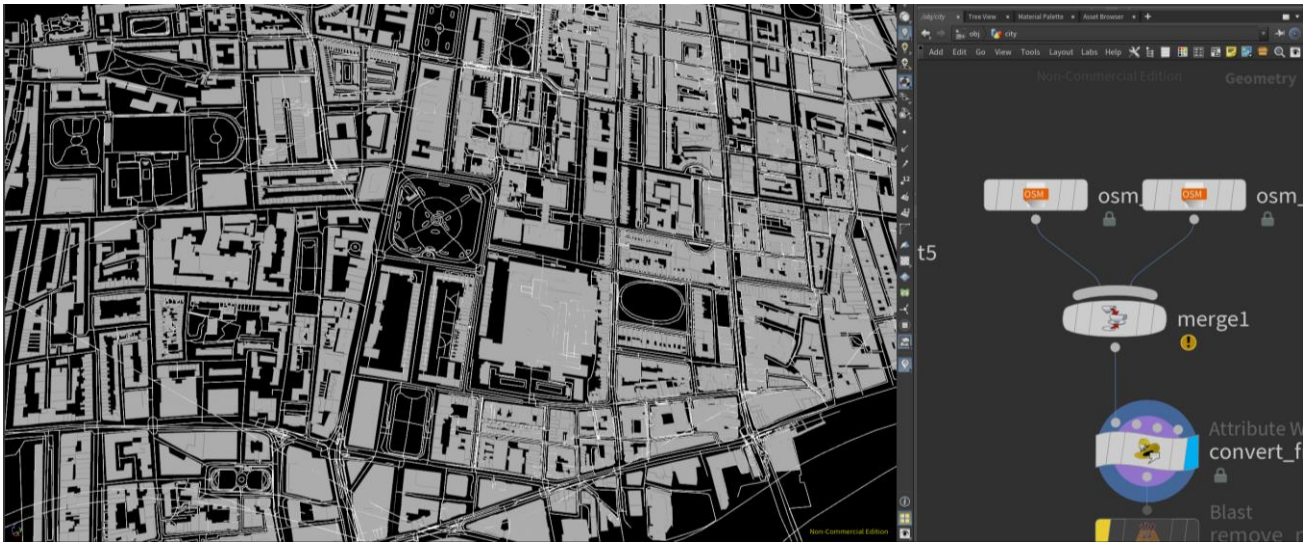
After that, I imported the photos and gpx data into houdini and visualized the image information.



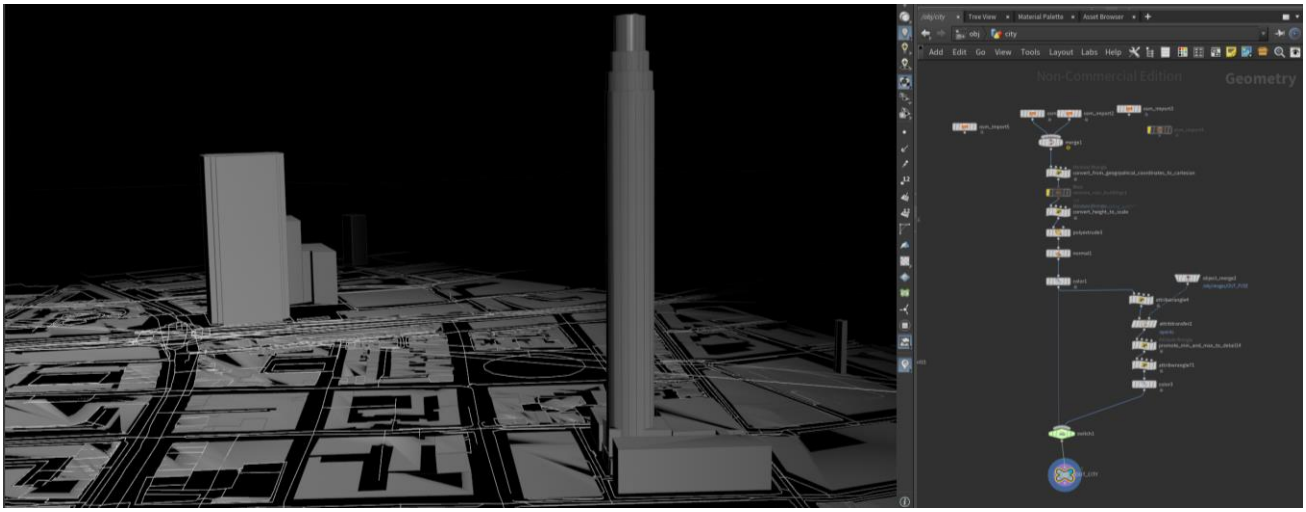


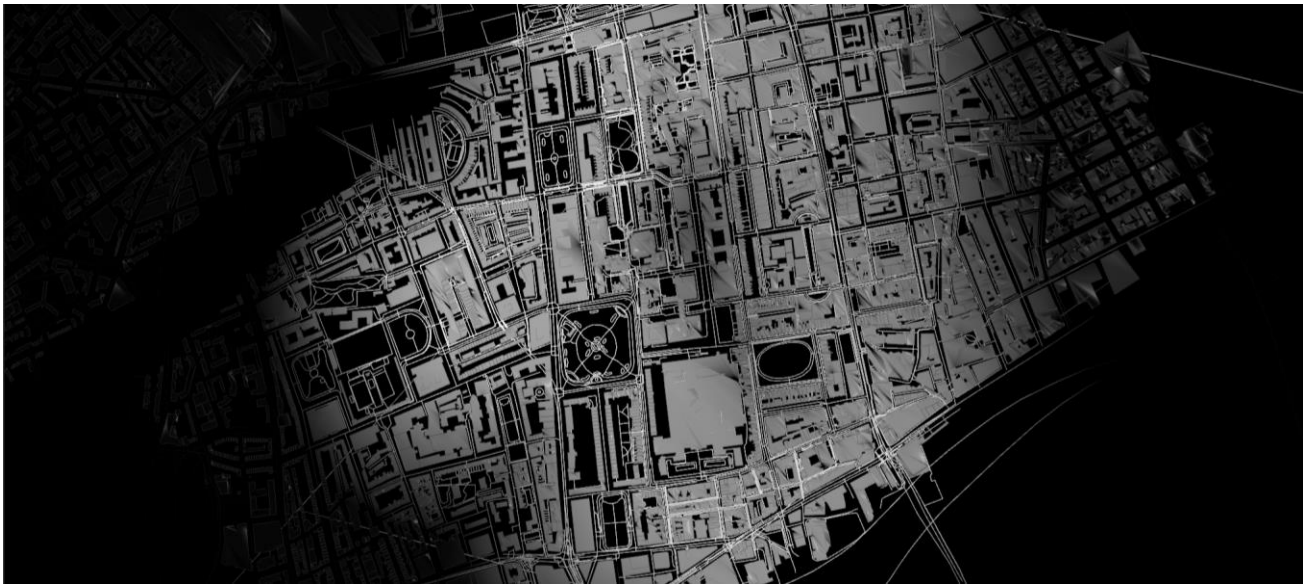
• Generate maps and buildings

I used OpenStreetMap to export the location's geographic information and import it into Houdini.



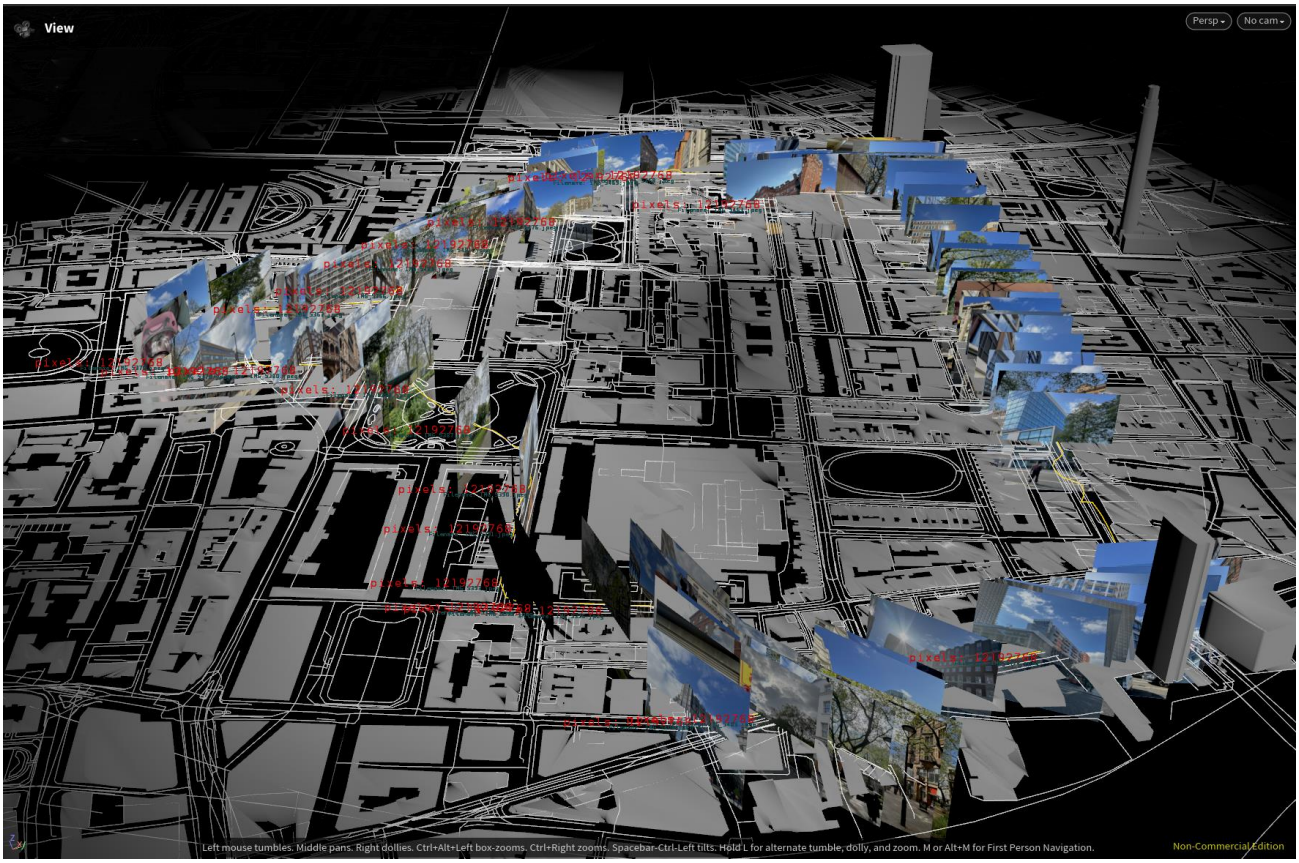
Then, using a collection of nodes, I've constructed the city's geometry, which add realism to the scene. At the same time, I also used a series of nodes to blur the map without photos and tracks.

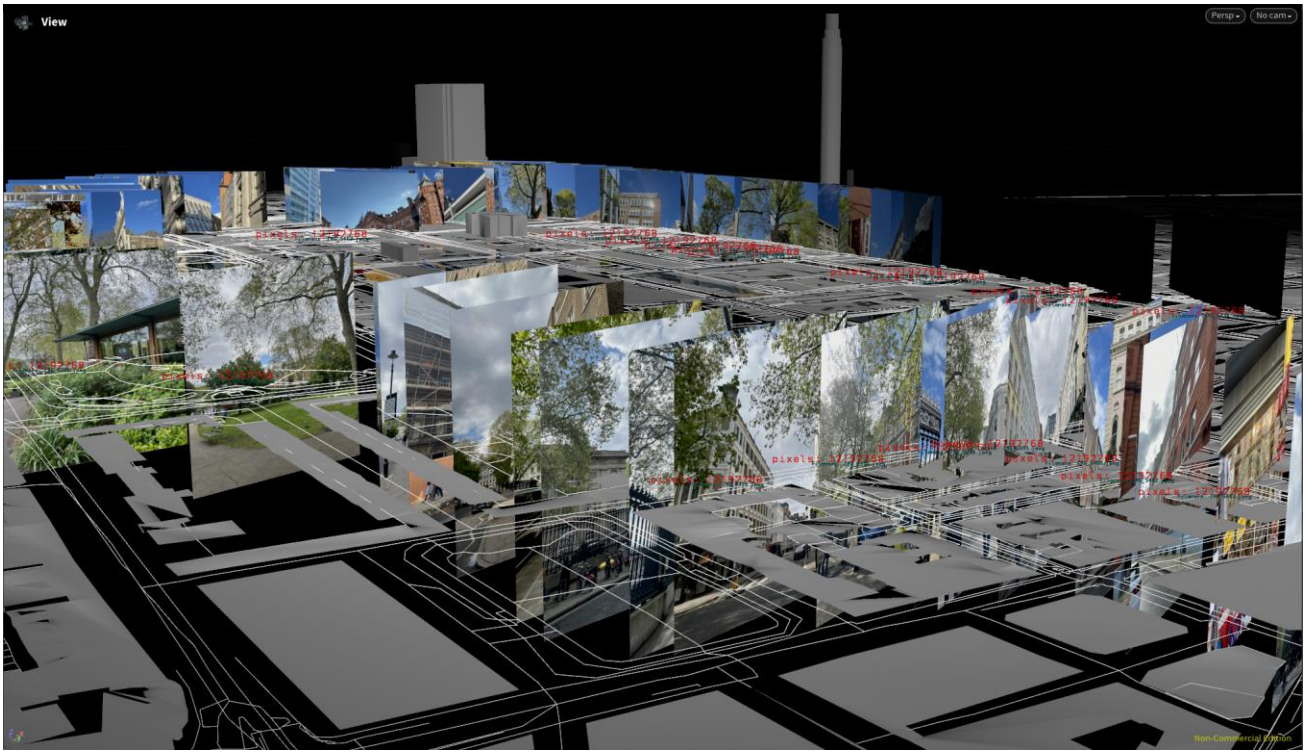




• Generate a series of images depicting your travels.

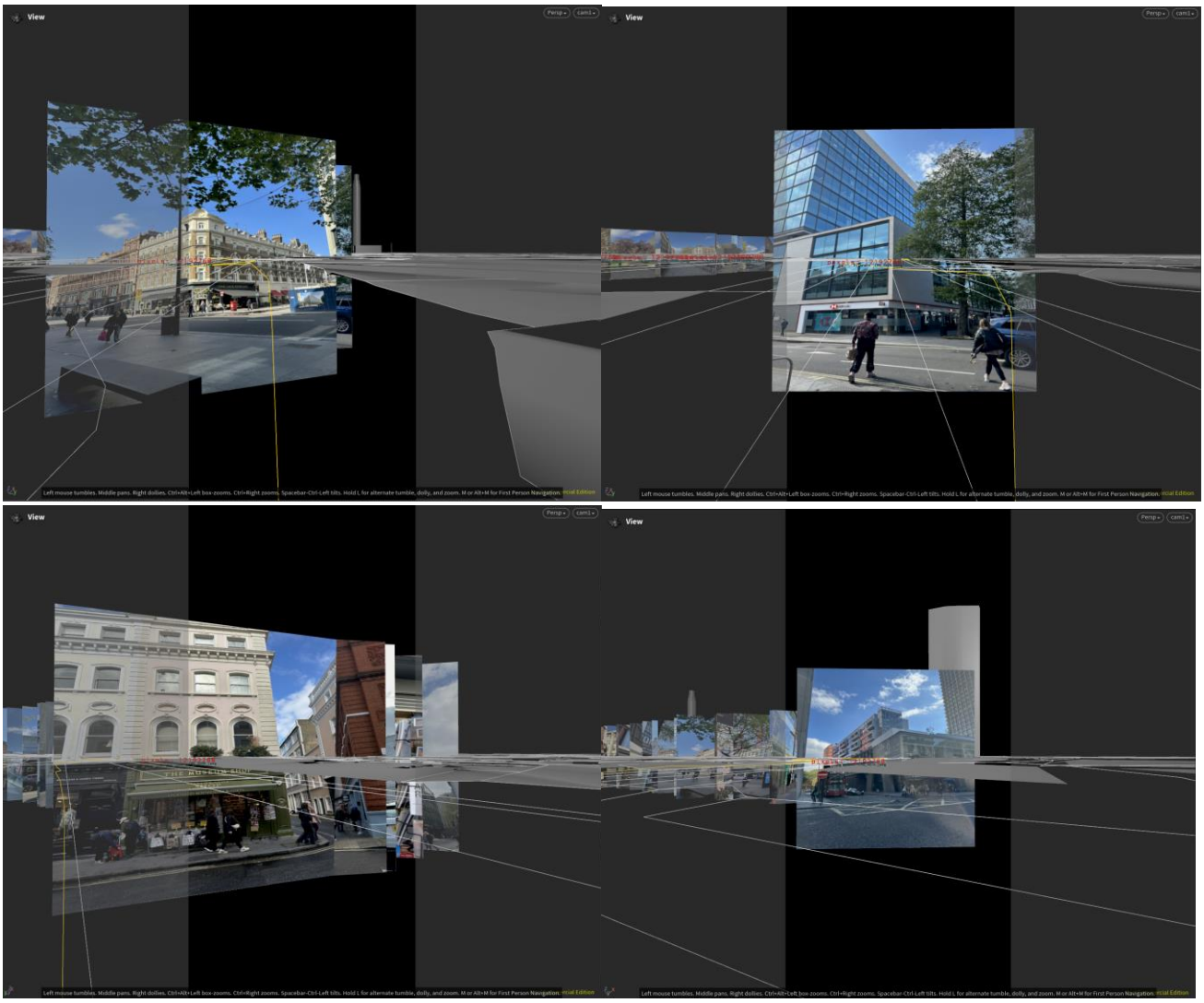
This is the result after I imported photos, GPX, OSM into Houdini.





The 'cam1' node indicates that I've established a camera path to navigate through the 3D environment to render an animation or assess the spatial arrangement from different vantage points.

Here are some examples.



Here are some other ways to view the project.

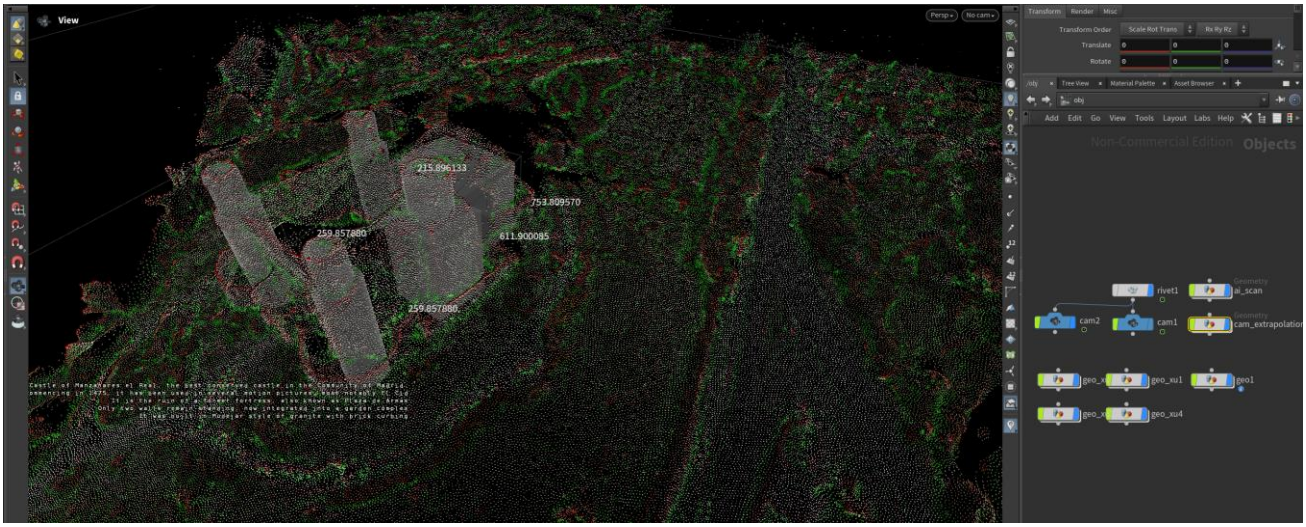
3 Video to 3D Model

• Generate one or more photogrammetry models from a video/ livestream/ web scraped/Google Images/YouTube. Take footage related to your projects.

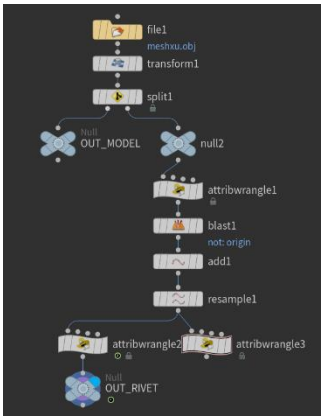
First, I chose the Manzanares el Real Castle, located near Madrid, Spain, which is one of the sites in our project, and scraped this video from YouTube.

Then, I used code to convert the video into a collection of frame-by-frame images, and these video frames were imported into RealityCapture to generate a reconstructed model.

After rebuilding the 3D model with RealityCapture and exporting the mesh, load the model into Houdini.

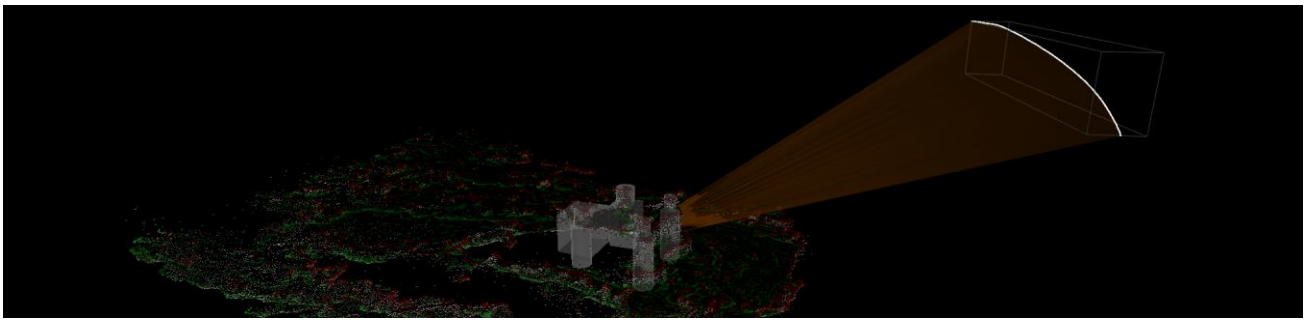


• Reconstruct your camera path.



The script in ‘attribwrangle1’ node modified the trajectory of points in my 3D scene by interpolating their positions and normal to create a smoother animation path. Additionally, I employed the 'add1' node to refine point groups along their trajectory and the 'resample1' node to ensure even segments and uniform spacing along the curve.

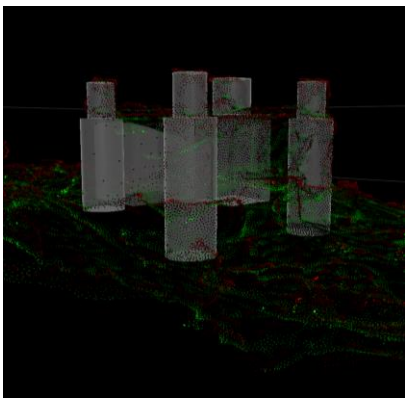
The next step involves creating an ‘attriwrangle’ node and using the code depicted in the diagram to determine the center point of each camera's projection surface, as well as to generate the respective lines for the camera orientation towards the scene.

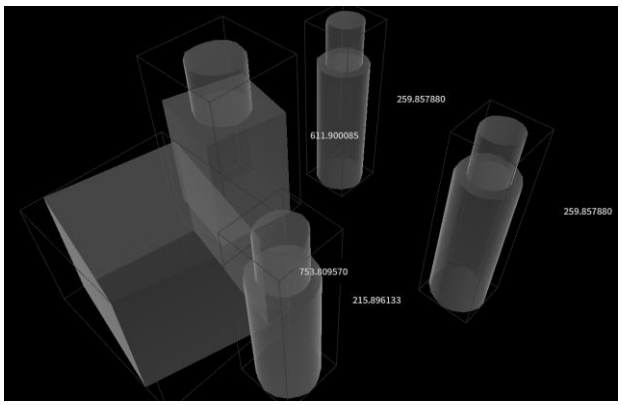


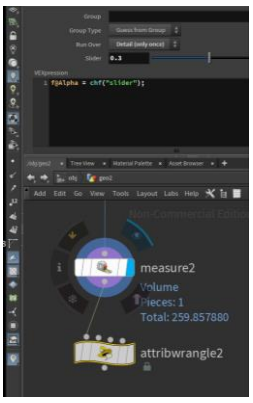
• Find a way of adding information to the subject.

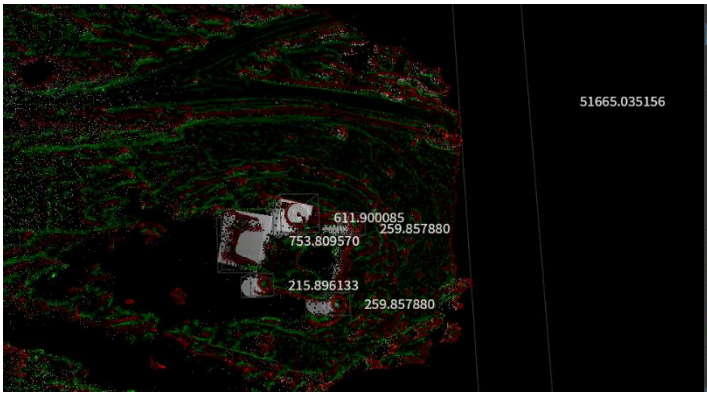
Volume speculation：

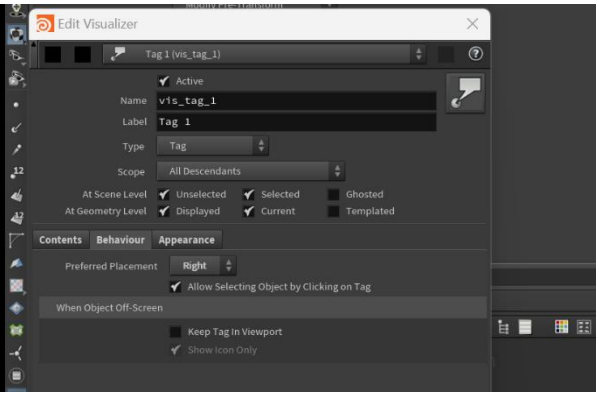
Through the ‘measure curvature’ node, all the boundaries of the site could be seen, and based on these boundaries, I used ‘curve’ and ‘polyextrude’ to generate objects. Finally, I created an ‘attribwrangle’ for adjusting the transparency.





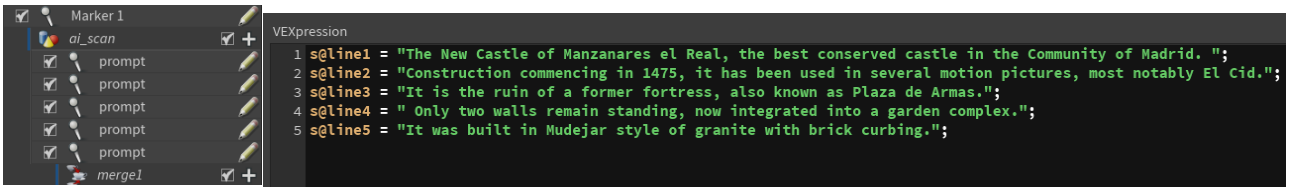


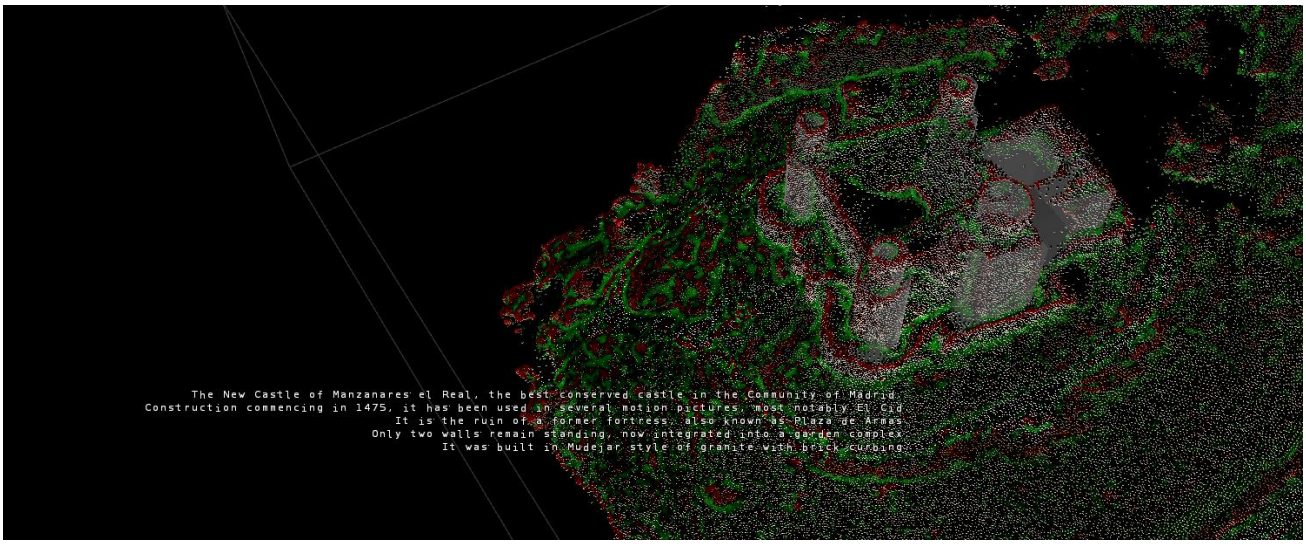




Text description of the venue (from Wikipedia)：

In this part I created Markers which named prompt. Adding an attribwrangle moudle including five lines, which would attribute to the Markers.

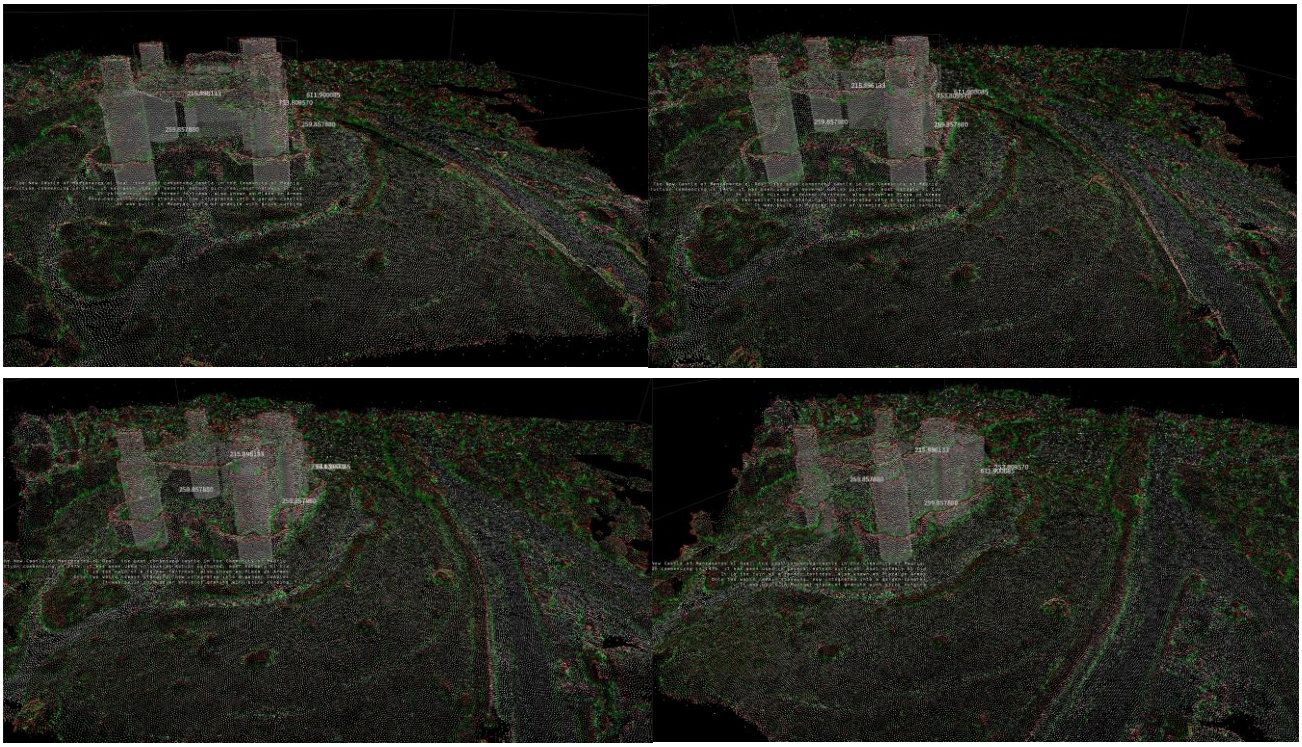




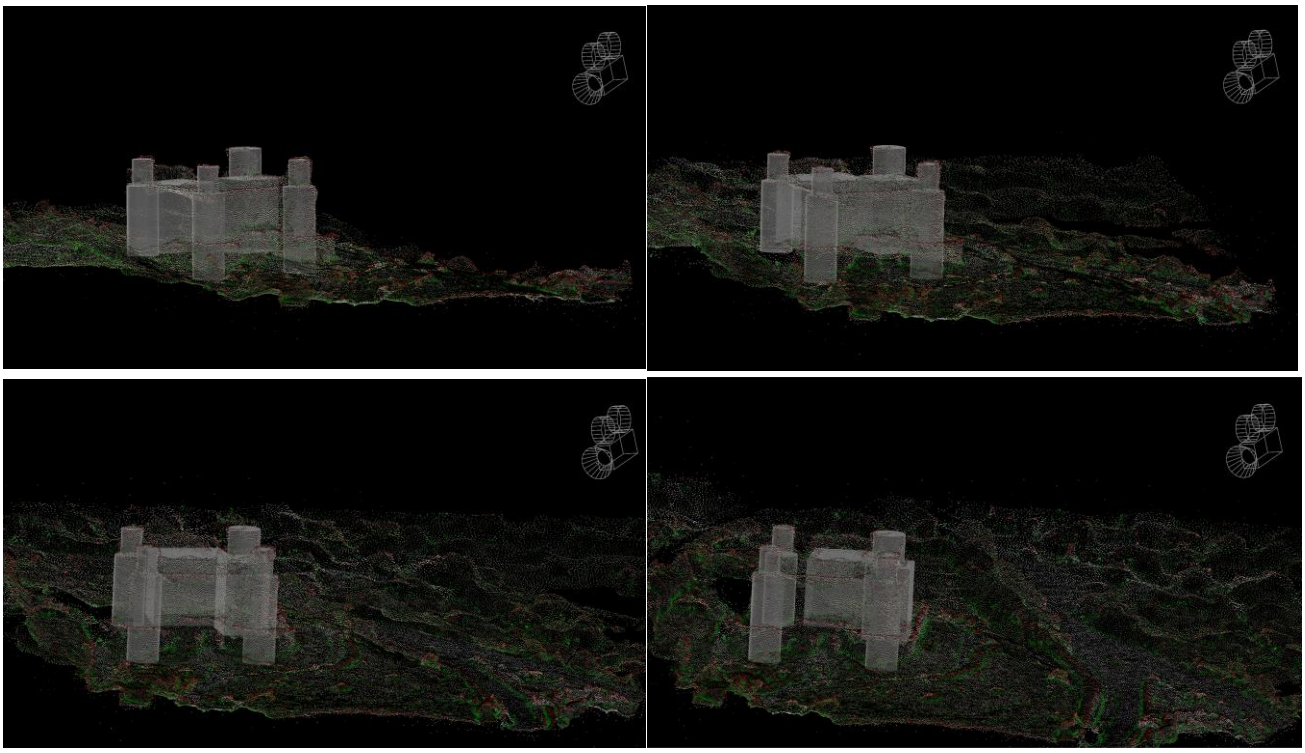
• Visualize your model as a collection of images/renders.

I created two camera positions, and in the second position, you can see the first camera. I took pictures of animation frames at different positions on the progress bar at the bottom of the picture from start to finish.

Cam_01_Examples:



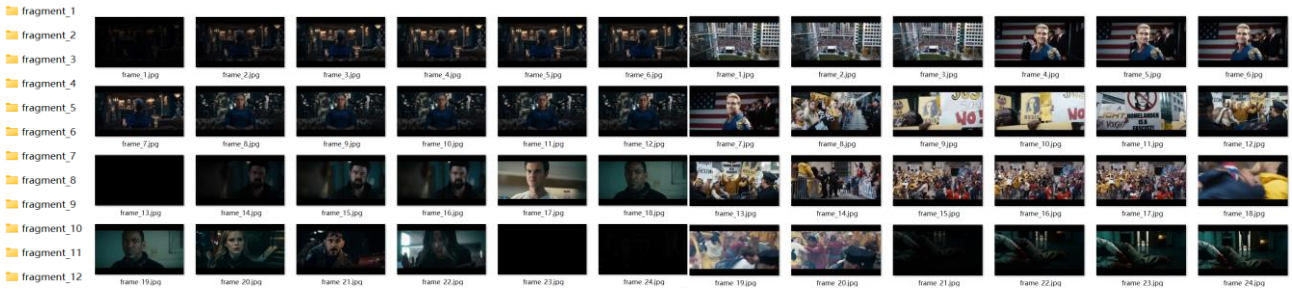
Cam_02_Examples:



4 Visualizing JSON in Houdini

• Extracting video frames

Same as assignment 3, at the beginning I used python code to download the ‘the boys’ trailer from YouTube and split the video into frames. I split the video into 290 frames in total, which corresponds to 12 subtitles.



• Converting and updating JSON data

Next, create a JSON file from the pickle file using code. Additionally, I manually entered video data into a JSON file. The image below displays the outcome of this manual entry. The information is primarily categorized into: ‘film_ID’, ‘paragraphs’, ‘image_name’, ‘model_path’, ‘image_n’, ’time’, ‘film_path’...

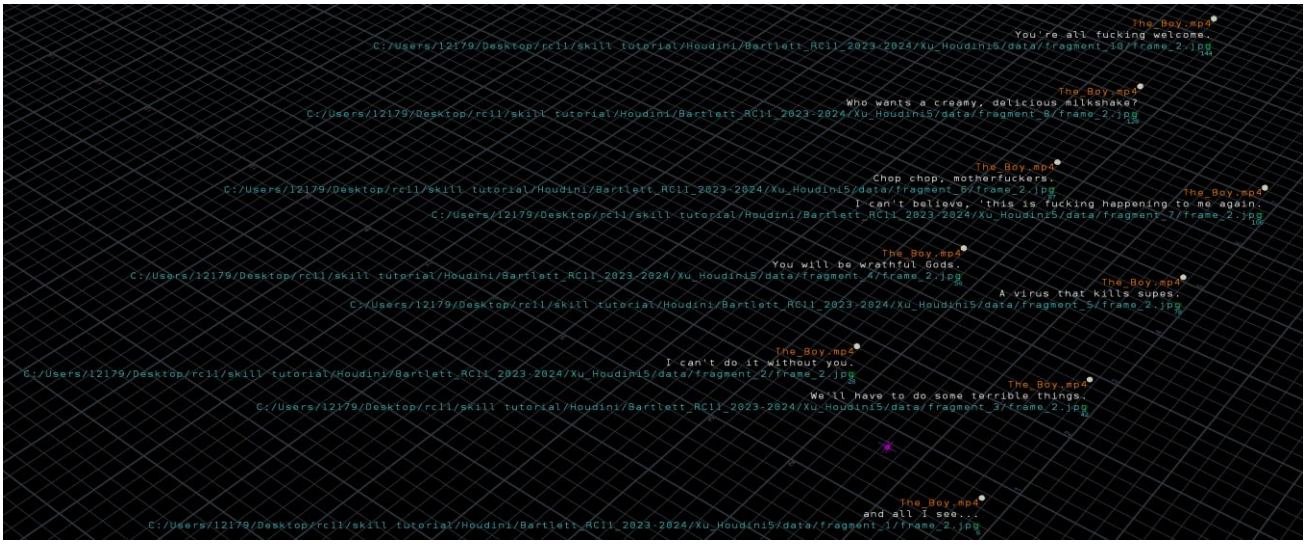

```
{
"folder": [
    {
      "paragraph" : "and all I see...",
      "filmID" : "The Boy.mp4",
      "image name" : "frame",
      "model path" : "Superman.obj",
      "image n" : 30,
      "time" : 6,
      "film path" : "The Boy.mp4"
    },
```

```
{
  "paragraph" : "I can't do it without you.",
  "filmID" : "The Boy.mp4",
  "image name" : "frame",
  "model path" : "",
  "image n" : 28,
  "time" : 28,
  "film path" : "The Boy.mp4"
},
```

```
{
  "paragraph" : "You will be wrathful Gods.",
  "filmID" : "The Boy.mp4",
  "image name" : "frame",
  "model path" : "",
  "image n" : 32,
  "time" : 50,
  "film path" : "The Boy.mp4"
},
```

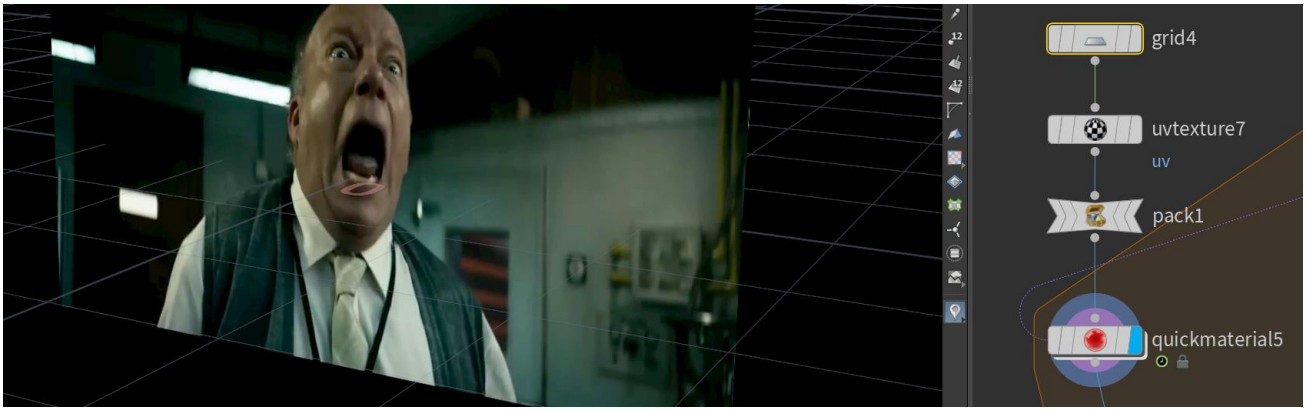
• Processing JSON data for subtitle points in Houdini

In Houdini, creating a ‘python’ node and import Json file. Next, creating points based on the number of subtitles in the Json file and plan the positions of the points. I will divide these points into two columns. In addition, writing code to load the information of Json file (‘film_ID’, ‘paragraphs’...)



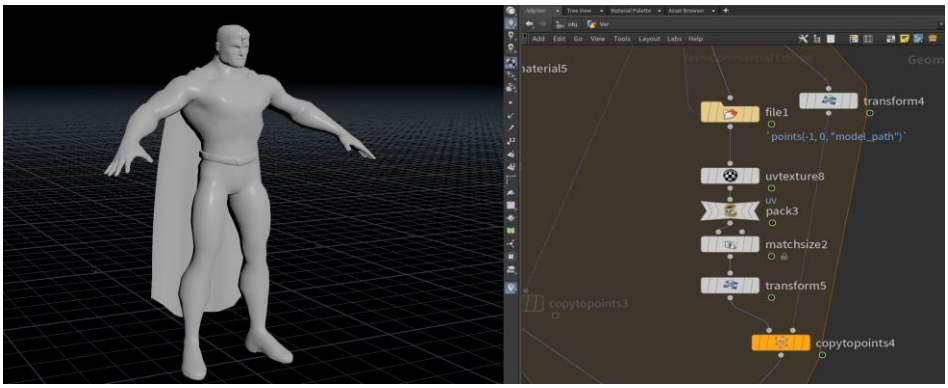
• Input film frames

Setting a grid, the ‘uvtexture7’ is used to ensure the correct display of images on the grid. The ‘pack1’ is used to make the information cleaner. Then set ‘quickmaterial5’ to load the frame images.



• Input Superman model

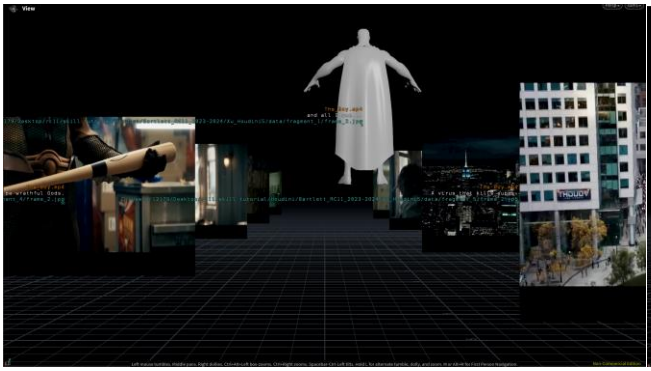
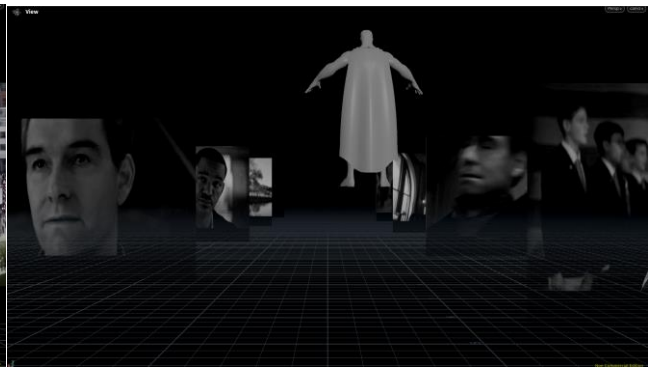
I added the Superman model through the ‘file1’ node, which rotates during playback in Houdini. Set ‘matchsize2’ and ‘transform5’ to add the location of the model and capture these points.

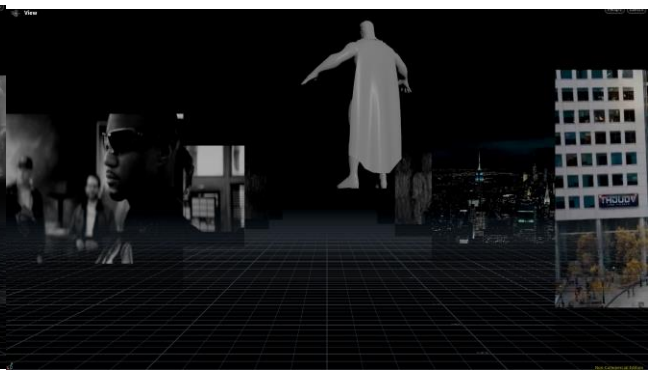


• A collection of images

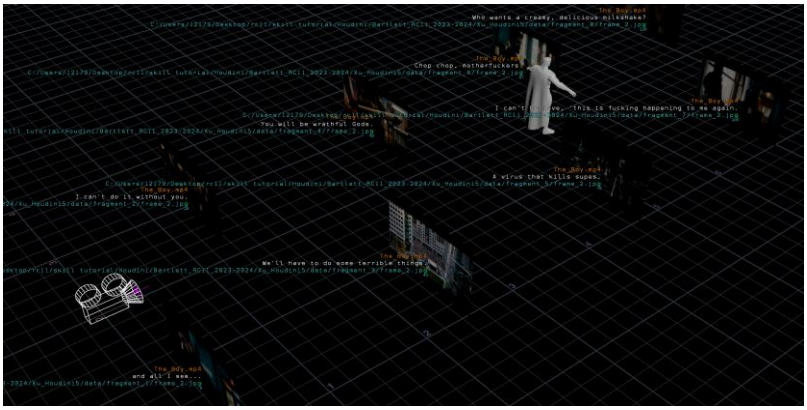
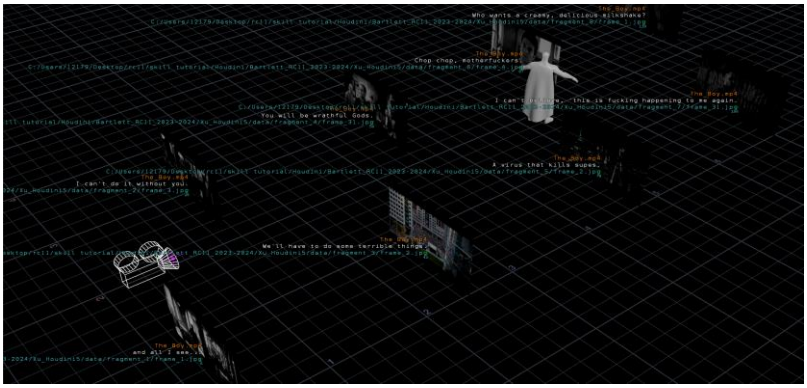
I created two camera positions. I took pictures of animation frames at different positions on the progress bar at the bottom of the picture from start to finish.

Cam_01_Examples:



Cam_02_Examples:

Complexity Assignment

RC11

23103502

GitHub Link: https://github.com/UD-Skills-2023-24/23103502.git

Exercise One

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

Use the NumPy ndarray object for your matrices:

```
import numpy as np


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

Implement with NumPy ndarray with different size and multiple matrices(more than 2):

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.array([[6, 3, 10, 4], [10, 3, 6, 10], [6, 9, 2, 5], [4, 2, 9, 4]])
D = np.array([[i for i in range(1, 6)] for _ in range(5)])

print("Numpy retult:")
print('2 x 2: ', np.matmul(A, B))
print('3 x 3: ', np.matmul(C, C))
print('5 x 5: ', np.matmul(D, D))
print("Nest List retult:")
print('2 x 2: ', square_matrix_multiply(A, B))
print('3 x 3: ', square_matrix_multiply(C, C))
print('5 x 5: ', square_matrix_multiply(D, D))

print('----------------------------------------------------------------------------------------------------')
print('Multiply more than two matrices')
print(np.matmul(np.matmul(A, B), A))
print(square_matrix_multiply(square_matrix_multiply(A, B), A))
```

```
Numpy retult:
2 x 2:  [[19 22]
 [43 50]]
3 x 3:  [[142 125 134 120]
 [166 113 220 140]
 [158  73 163 144]
 [114 107 106  97]]
5 x 5:  [[15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]
 [15 30 45 60 75]]
Nest List retult:
2 x 2:  [[19, 22], [43, 50]]
3 x 3:  [[142, 125, 134, 120], [166, 113, 220, 140], [158, 73, 163, 144], [114, 107, 106, 97]]
5 x 5:  [[15, 30, 45, 60, 75], [15, 30, 45, 60, 75], [15, 30, 45, 60, 75], [15, 30, 45, 60, 75], [15, 30, 45, 6
0, 75]]
----------------------------------------------------------------------------------------------------
Multiply more than two matrices
[[ 85 126]
 [193 286]]
[[85, 126], [193, 286]]
```

Compare with built-in multiplication functions: Built-in multiplication function is faster.

```
from time import time

D = np.array([[i for i in range(128)] for _ in range(128)])

start_time = time()
res_np = np.matmul(D, D)
np_time = time() - start_time

start_time = time()
res_for = square_matrix_multiply(D, D)
for_time = time() - start_time

print(f"Built-in function time: {np_time:.6} s, my method time: {for_time:.6}")
```

```
Built-in function time: 0.00394487 s, my method time: 0.557498
```

Implement with nested list with different size and multiple matrices(more than 2):

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

C = [[6, 3, 10, 4], [10, 3, 6, 10], [6, 9, 2, 5], [4, 2, 9, 4]]

print("Numpy retult:")
print('2 x 2: ', np.matmul(A, B))
print('3 x 3: ', np.matmul(C, C))
print("Nest List retult:")
print('2 x 2: ', square_matrix_multiply(A, B))
print('3 x 3: ', square_matrix_multiply(C, C))

print('----------------------------------------------------------------------------------------------------')
print('Multiply more than two matrices')
print(np.matmul(np.matmul(A, B), A))
print(square_matrix_multiply(square_matrix_multiply(A, B), A))
```

```
Numpy retult:
2 x 2:  [[19 22]
 [43 50]]
3 x 3:  [[142 125 134 120]
 [166 113 220 140]
 [158  73 163 144]
 [114 107 106  97]]
Nest List retult:
2 x 2:  [[19, 22], [43, 50]]
3 x 3:  [[142, 125, 134, 120], [166, 113, 220, 140], [158, 73, 163, 144], [114, 107, 106, 97]]
----------------------------------------------------------------------------------------------------
Multiply more than two matrices
[[ 85 126]
 [193 286]]
[[85, 126], [193, 286]]
```

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

The time complexity of this algorithm is $O(n3)$, as there are three nested loops, each running up to n times.

Algorithm Breakdown:

The algorithm initializes a new square matrix C of dimensions $n \times n$, typically setting all entries to zero. It employs three nested loops:

1. The first loop iterates over each row i of matrix A (from 1 to n).
2. The second loop iterates over each column j of matrix B (also from 1 to n).
3. The third (innermost) loop iterates over the index k (from 1 to n), which is used to compute the dot product of the i -th row of A and the j -th column of B.

Time Complexity Analysis:

The outermost loop runs n times, once for each row of A. The middle loop also runs n times for each iteration of the outer loop, corresponding to each column of B. The innermost loop runs n times for each combination of the outer and middle loops to sum up the products of corresponding elements.

Calculating Total Operations:

Each iteration of the innermost loop involves a multiplication operation. Since the innermost loop runs n times for each pair of i and j, and since both i and j themselves run n times each, the total number of multiplication operations is $n \times n \times n = n^3$.

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?

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

The algorithm is recursive because it solves the matrix multiplication problem by repeatedly calling itself to multiply smaller sub-matrices. Each recursive call further divides the problem into even smaller sub-problems, following the same approach until the base case is reached.

With each recursion step, the matrices are divided into four smaller sub-matrices of half the original size. This process continues until the sub-matrices are reduced to 1×1. By continuously halving the matrix size, the algorithm eventually reduces the problem to its base case, where direct multiplication is performed.

Divide: Break down the complex problem of matrix multiplication into simpler problems involving smaller matrices.

Conquer: Solve these simpler problems recursively, allowing the algorithm to focus on manageable chunks of the original problem.

Combine: Integrate the solutions of the simpler problems to construct the solution to the original problem.

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 split_matrix(A, n):
    A11 = [row[:n] for row in A[:n]]
    A12 = [row[n:] for row in A[:n]]
    A21 = [row[:n] for row in A[n:]]
    A22 = [row[n:] for row in A[n:]]
    return A11, A12, A21, A22

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

def combine_matrix_1(A, B, C, D, n):
    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):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        mid = n // 2
        A11, A12, A21, A22 = split_matrix(A, mid)
        B11, B12, B21, B22 = split_matrix(B, mid)
        # Recursive multiplications
        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))
        # Combining quarters
        C = combine_matrix_1(C11, C12, C21, C22, mid)
        return C
```

```python
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])

print("Numpy retult:")
print('2 x 2: ', np.matmul(A, B))
print('3 x 3: ', np.matmul(C, C))
print("Nest List retult:")
print('2 x 2: ', square_matrix_multiply_recursive(A, B))
print('3 x 3: ', square_matrix_multiply_recursive(C, C))
```

```
Numpy retult:
2 x 2:  [[19 22]
 [43 50]]
3 x 3:  [[ 90 100 110 120]
 [202 228 254 280]
 [314 356 398 440]
 [426 484 542 600]]
Nest List retult:
2 x 2:  [[19, 22], [43, 50]]
3 x 3:  [[90, 100, 110, 120], [202, 228, 254, 280], [314, 356, 398, 440], [426, 484, 542, 600]]
```

Test and compare the practical speed with the non-recursive algorithm:

```python
# Speed Comparation
from time import time

D = np.array([[i for i in range(1, 129)] for _ in range(128)])

start_time = time()
square_matrix_multiply(D, D)
mul_time = (time() - start_time)

start_time = time()
square_matrix_multiply_recursive(D, D)
recursive_time = (time() - start_time)

print(f"Multiply time: {mul_time:.4} s, Recursive time: {recursive_time:.4} s")
```

```
Multiply time: 0.5566 s, Recursive time: 1.688 s
```

3. Do a complexity analysis for the SMMRec algorithm.

First comment on the complexity of the base case, divide step, conquer step, and combine step separately, then put it all together.

- **Divide**: Each matrix of size $n \times n$ is split into four sub-matrices of size $\frac{n}{2} \times \frac{n}{2}$.
- **Conquer**: Recursively multiply these smaller matrices until reaching base cases of 1×1 matrices, where multiplication is straightforward.
- **Combine**: Use matrix addition to combine the results of the recursive multiplications into the final matrix C.

For recursive matrix multiplication, each decomposition results in four subproblems, each of size half of the original. For each pair of sub-matrices, we perform 8 multiplications. The recursive nature yields a time complexity recurrence relation expressed as:

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

Here, $\Theta(n^2)$ accounts for the operations needed to merge the four matrix results back into a larger matrix. This complex recurrence generally resolves to $T(n) = O(n^{log_2^8}) = O(n^3)$ which is no better in complexity than the direct method, but sets the foundation for further optimizations like the Strassen algorithm.

Tree Analysis

1. **Depth of Recursion**:
    - The recursion depth is determined by how many times the matrix can be divided into smaller matrices of size $\frac{n}{2} \times \frac{n}{2}$ until the base case is reached (when n equals 1). The depth is therefore $log_2(n)$.
2. **Work at Each Level**:
    - At the top level, the algorithm performs eight multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices, each requiring $T(/fracn2)$ time.
    - At the second level, each of these subproblems is further divided, resulting in sixteen $\frac{n}{4} \times \frac{n}{4}$ matrix multiplications, each taking $T(\frac{n}{4})$ time.
    - This pattern continues such that each level down the recursion tree, the number of subproblems doubles, but the size of each problem halves.
3. **Total Work**: If $d = log_2(n)$ represents the total number of recursive levels, then the ith level contains 8^i subproblems, each of size $\frac{n}{2^i}$. Therefore, the total work at the ith level is $8^i * T(\frac{n}{2^i})$.

Exercise Three

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

Addition and subtraction of matrices are relatively straightforward operations. For two matrices of the same dimension, $n \times n$, each corresponding element is added or subtracted, which requires n^2 operations in total. Thus, the complexity for both addition and subtraction of matrices is $O(n^2)$.

Matrix multiplication, on the other hand, is significantly more complex. For each element of the resulting $n \times n$ matrix, n multiplications and $n - 1$ additions are required, leading to approximately $2n - 1$ operations. Since there are n^2 elements to compute, the straightforward approach to matrix multiplication has a complexity of $O(n^3)$.

Implement and test the algorithm:

```python
def matrix_add(A, B):
    return [[A[i][j] + B[i][j] for j in range(len(A))] for i in range(len(A))]

def matrix_sub(A, B):
    return [[A[i][j] - B[i][j] for j in range(len(A))] for i in range(len(A))]

def split_matrix(A, size):
    return [
        [A[i][:size] for i in range(size)], [A[i][size:] for i in range(size)],
        [A[i][:size] for i in range(size, len(A))], [A[i][size:] for i in range(size, len(A))]
    ]

def combine_matrix(C11, C12, C21, C22):
    n = len(C11) * 2
    C = [[0]*n for _ in range(n)]
    mid = n // 2
    for i in range(mid):
        for j in range(mid):
            C[i][j] = C11[i][j]
            C[i][j + mid] = C12[i][j]
            C[i + mid][j] = C21[i][j]
            C[i + mid][j + mid] = C22[i][j]
    return C

def strassen_matrix_multiply(A, B):
    n = len(A)
    if n <= 2:  # Use conventional multiplication for small matrices
        return square_matrix_multiply(A, B)
    else:
        mid = n // 2
        A11, A12, A21, A22 = split_matrix(A, mid)
        B11, B12, B21, B22 = split_matrix(B, mid)

        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))

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

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

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

C = [[6, 3, 10, 4], [10, 3, 6, 10], [6, 9, 2, 5], [4, 2, 9, 4]]

print("Numpy retult:")
print('2 x 2: ', np.matmul(A, B))
print('3 x 3: ', np.matmul(C, C))
print("Nest List retult:")
print('2 x 2: ', strassen_matrix_multiply(A, B))
print('3 x 3: ', strassen_matrix_multiply(C, C))
```

```
Numpy retult:
2 x 2:  [[19 22]
 [43 50]]
3 x 3:  [[142 125 134 120]
 [166 113 220 140]
 [158  73 163 144]
 [114 107 106  97]]
Nest List retult:
2 x 2:  [[19, 22], [43, 50]]
3 x 3:  [[142, 125, 134, 120], [166, 113, 220, 140], [158, 73, -29, -12], [114, 107, 330, 207]]
```

Compare the practical speed with the recursive algorithm:

```
# Speed Comparation
from time import time

D = [[i for i in range(1, 2049)] for _ in range(2048)]

start_time_1 = time()
square_matrix_multiply(D, D)
mul_time = (time() - start_time_1)

start_time_2 = time()
square_matrix_multiply_recursive(D, D)
recursive_time = (time() - start_time_2)

start_time_3 = time()
strassen_matrix_multiply(D, D)
strassen_time = (time() - start_time_3)

print(f"Multiply time: {mul_time:.4} s \n Recursive time: {recursive_time:.4} s \n Strassen time: {strassen_time:.4} s")
```

```
Multiply time: 742.0 s
 Recursive time: 6.097e+03 s
 Strassen time: 1.271e+03 s
```

Strassen algorithm is faster than recursive algorithm.

The recursive and Strassen algorithms are often slower than direct triple loop multiplication due to the overhead associated with recursive function calls, increased memory management for storing intermediate results, and less efficient cache utilization, all of which can significantly impact performance, especially for large matrices.

2.Do a complexity analysis of the Strassen algorithm.

Instead of starting from scratch, you can also take your result from Exercise 2 and adapt to the optimization; explain what changes in the complexity formula with these optimizations.

The complexity of the standard recursive algorithm (without Strassen's optimizations) is $T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$, which resolves to $O(n^3)$ as per the Master Theorem.

Strassen's algorithm optimizes matrix multiplication by reducing the number of recursive multiplications from 8 to 7. This method rearranges and combines matrix elements, introducing additional additions and subtractions but reducing the number of multiplications.

The time complexity recurrence for Strassen's algorithm is:

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$$

By applying the Master Theorem to this new recurrence relation, the complexity becomes $T(n) = O(n^{log_2{}^7}) \approx O(n^{2.81})$, This is a substantial improvement over the $O(n^3)$ complexity of the standard recursive multiplication approach, demonstrating the impact of Strassen's optimizations.