BARC0053: Bartlett Architecture Skills Elective (B-Pro Courses) 23/24

Computing Skills Main Assignment

RC11_23135525

SKILLS CLASS RC11 - 23/24

VECTORIAL ENCODINGS OF QUALITATIVE DOMAINS

Submitter: 23135525

– – – – – –

FINAL ASSIGNMENT

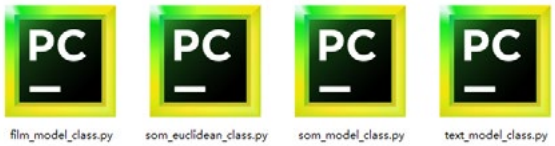
Assignment Files

For the assignment, I have wrote 2 main notebook files for the running coding, and 4 integrated '.py file' packages for calling functions and a datasets folder for data and saving pickles.

One notebook is the working notebook which contains all the codes, functions, input, and the detail steps of how to get the train data or train som or some variables. And the pure notebook is an integrated code document of the former, which is more structured, clear, and more focus on the demonstration.

For the datasets, there are three types of data: texts, images(including image and film frame) and viedo(film). And some trained SOMs, lists, dicts are saved into pickle files in the pickle folder.

[main notebook]

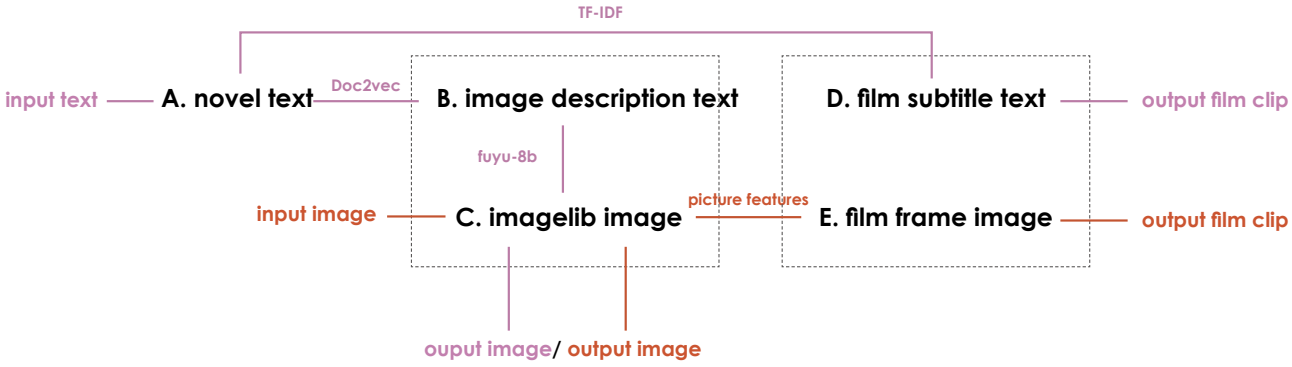


[.py file packages]

[datasets folder]

Outline of the Coding Solution

For this skill class assignment, I try to build **5 differents data SOMs (A-E)** including the 'Novel Text', 'FuyuImage Text', 'Imagelib Image', 'FilmSubtitle Text', 'FilmFrames Image', and attempt to link and integration those datasets within 3 domains (Text, Image, Video). Here shows the The specific vectorisation process.



Input from text:

1. Text query activate novel text som to get most similar novel paragragh.

2. Searching the novel paragraphs in image fuyu text som to find the index of the image and its description.

3. Searching the novel paragraphs in film subtitles som to find the most similar subtitles, and show that film clip.

Input from image:

1. Input image activate image som to get most similar image and its fuyu discription text.

2. Searching the matched image in film frames som to find the most similar frame and show that film clip.

3. Searching the matched image discription text in novel text som to get the most similar novel paragraph.

Outputs Explanation

PART I: Build SOM Datasets

A. Create novel text som and searching function

A.1 Create novel text som

A.1.1 Read the Epub and vectorised them

Load the novel text from the path, and use TextModel import from the text_model_class.py to vectorise it, and it can choose different vectorised mode, here I use the 'doc2vec' as vectorization

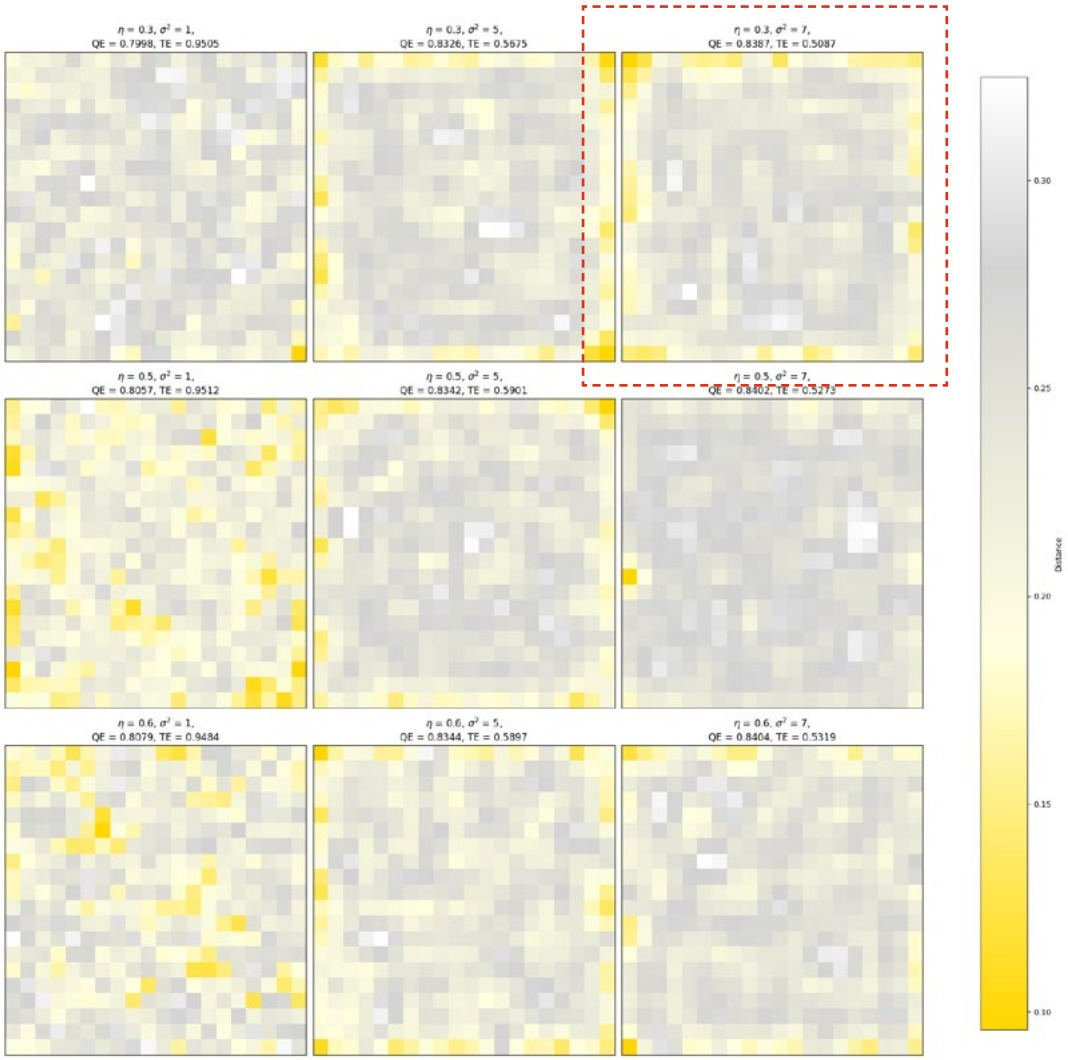
a. Read Text from epub

```
NOVEL_DIR=r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Text'
NOVEL_FILES = []
NOVEL_FILES.extend(glob.glob(os.path.join(NOVEL_DIR, '*.epub')))
```

```
doc2vec_novel_model=TextModel(NOVEL_FILES,vectorization='doc2vec', dimension = 300,epochs=10, min_df=3)
```

A.1.2 Build som

Create a som with Doc2vec with 300 dimension



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.1.3 Link the paragraph with som unit

Then link the paragraphs to their BMUs, and create a data_dict for tracking these info. And I wrote a function to do that, this function is applicable to all text datasets in this notebook.

```
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)
```

```
# Saving into pickle file
pickle_path=r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\pickle\data_dict.pkl'

with open(pickle_path, 'wb') as f:
    pickle.dump(data_dict, f)
```

```
# Loading pickle file
pickle_path=r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\pickle\data_dict.pkl'
with open(pickle_path, 'rb') as f:
    doc2vec_data_dict = pickle.load(f)
```

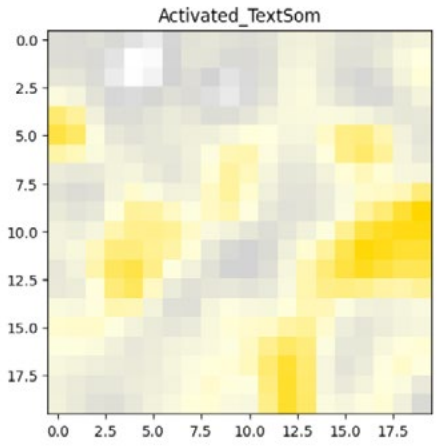
A.2 Create searching function

For this step, I defined a search function to search the text query for som, returning the one result in the BMU that has the smallest distance from the cosine of that lattice vector and print it out. At the same time, it will show the activated som.

```python
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(doc2vec_train_data, som, model.vectorize(query)), cmap=cmap)
        plt.title(f'Activated_TextSom')
        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(lsa_train_data, som, model.vectorize(query)[0]), cmap=cmap)
        plt.title(f'Activated_TextSom')
        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
```

```python
query=query_list[2]['query']
print('The matching query is: ' + query)
som=text_som

train_data=doc2vec_train_data
search_TextSom(som,doc2vec_novel_model,doc2vec_data_dict,query,vectorization ='doc2vec')
```

```
The matching query is: The coffee is bitter and the table is hard.
```

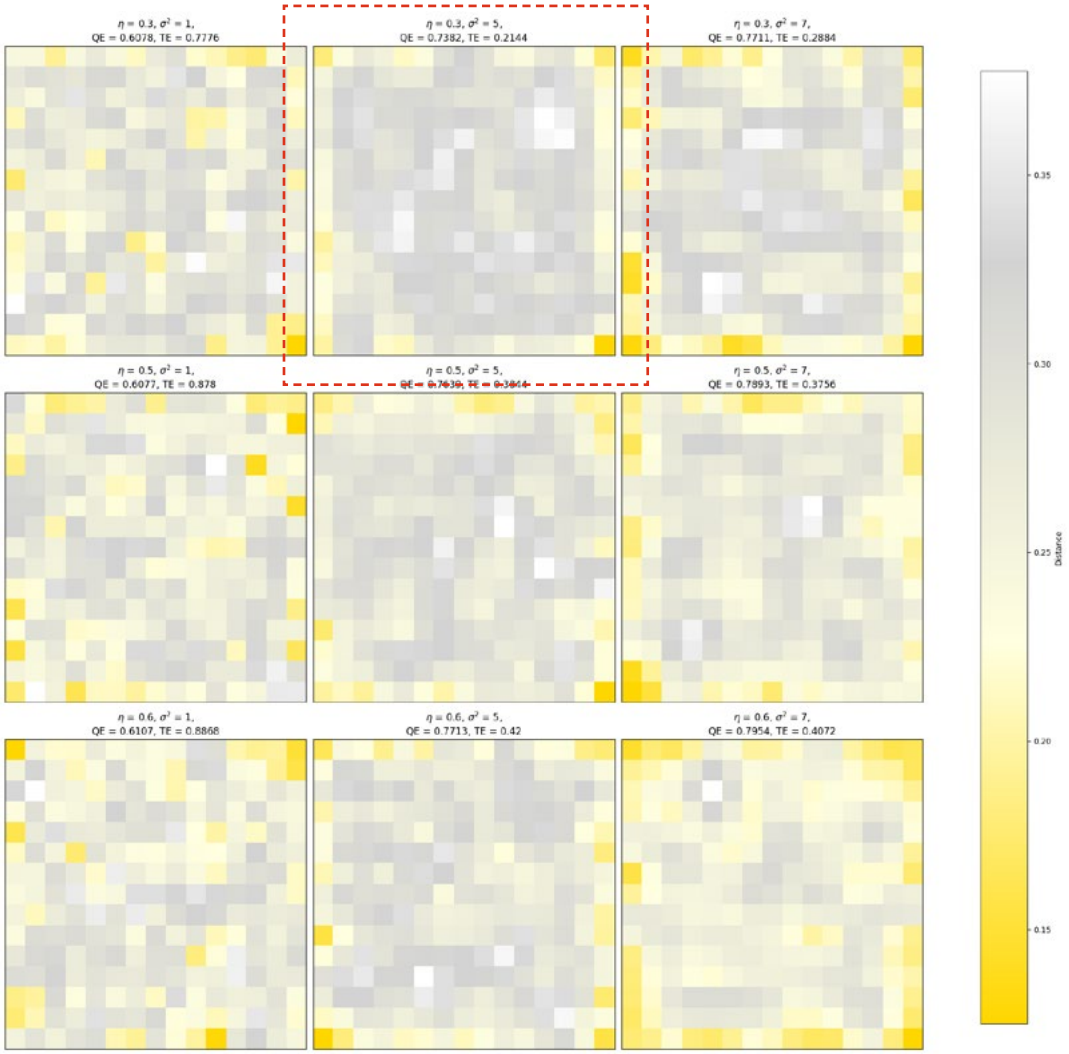



```
The best matching unit(bmu) is: (1, 4)
```

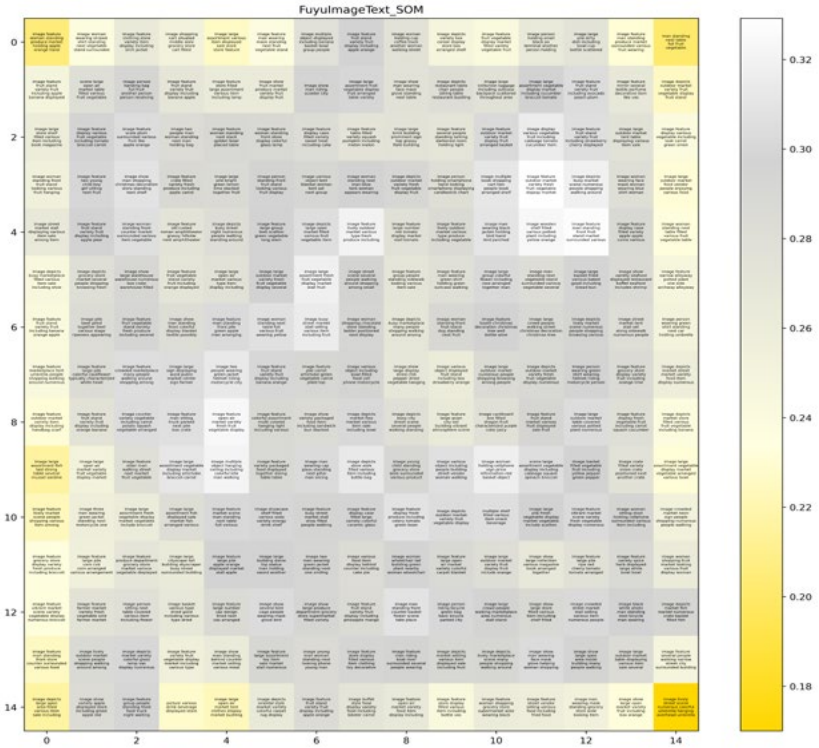
```
[{'paragraph': ' Did they but know, they are in travail for a twin birth. They are troubled with the first movements of a new piety tow
ard fate and a new loyalty toward man. Of the first they know as yet almost nothing, for whenever they experience it, they confuse it w
ith the old complacent love of a loving God.',
  'nr': '1132',
  'ID': 'stapledon-last-men-in-london',
  'type': 'epub'}]
```

B. Create image fuyu text som and searching function

Load ImageLibrary_Text.csv and the specific steps next here are similar to those above, so I won't repeat them again, and will show the som training results directly.



The reason of som selection is similar with the above which is based on QE and TE values, and I chose the som with the index1 here .And the selected som U-Matrix with data mapping on it, is shown below.



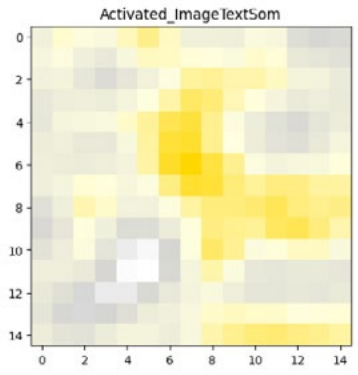
B.2 Create searching function

The function can search the input text query for som, returning the one best matching image discrpition and the index of image then show the activated som and print matched image.

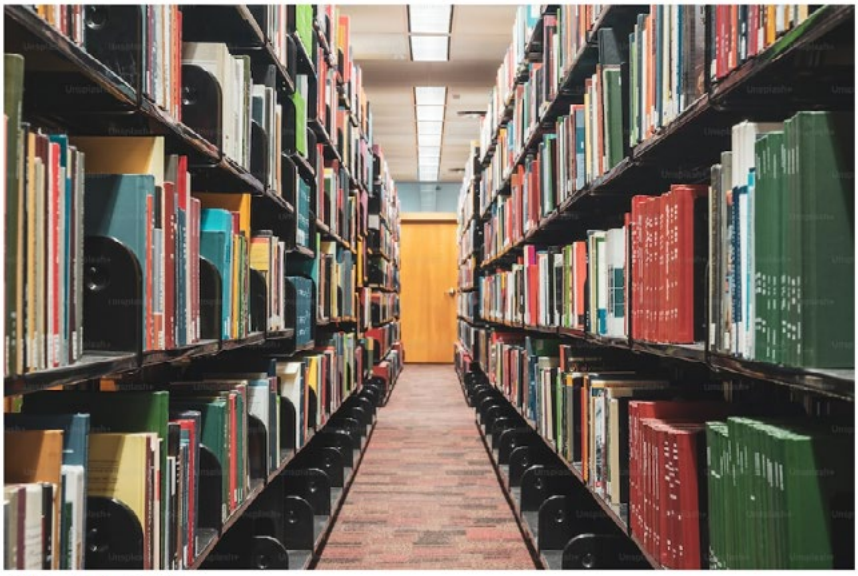
```python
def search_ImageTextSom(som, model, data_dict, image_folder, query='street', vectorization ='doc2vec'):
    result = []
    g, h = 0, 0

    if vectorization == 'doc2vec':
        fig = plt.figure()
        plt.imshow(SOMlib.activate1(ImageText_train_data, som, model.vectorize(query)), cmap=cmap)
        plt.title(f'Activated_ImageTextSom')
        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.title(f'Activated_ImageTextSom')
        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
```

```python
ImageLibrary_folder=r'D:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Image'

query=query_list[7]['query']

print('The matching query is: ' + query)

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

```
The matching query is: Reading is a magical adventure that can transport you to countless worlds.
```




```
The best matching unit(bmu) is: (11, 5)
```



```
[{'filename': 'plaza_301.jpg',
  'text': 'In the image, there is a long row of books lined up on shelves. These books are arranged on shelves that stretch across the lengt
h of the room. The books are arranged in various rows and sections, creating a well-organized and visually appealing scene.'}]
```

C. Create image som and searching function

C.1 Create image som

C.1.1 Load images

Load images from Image Folder in Datasets and preprocess them with image feature 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'
)
```

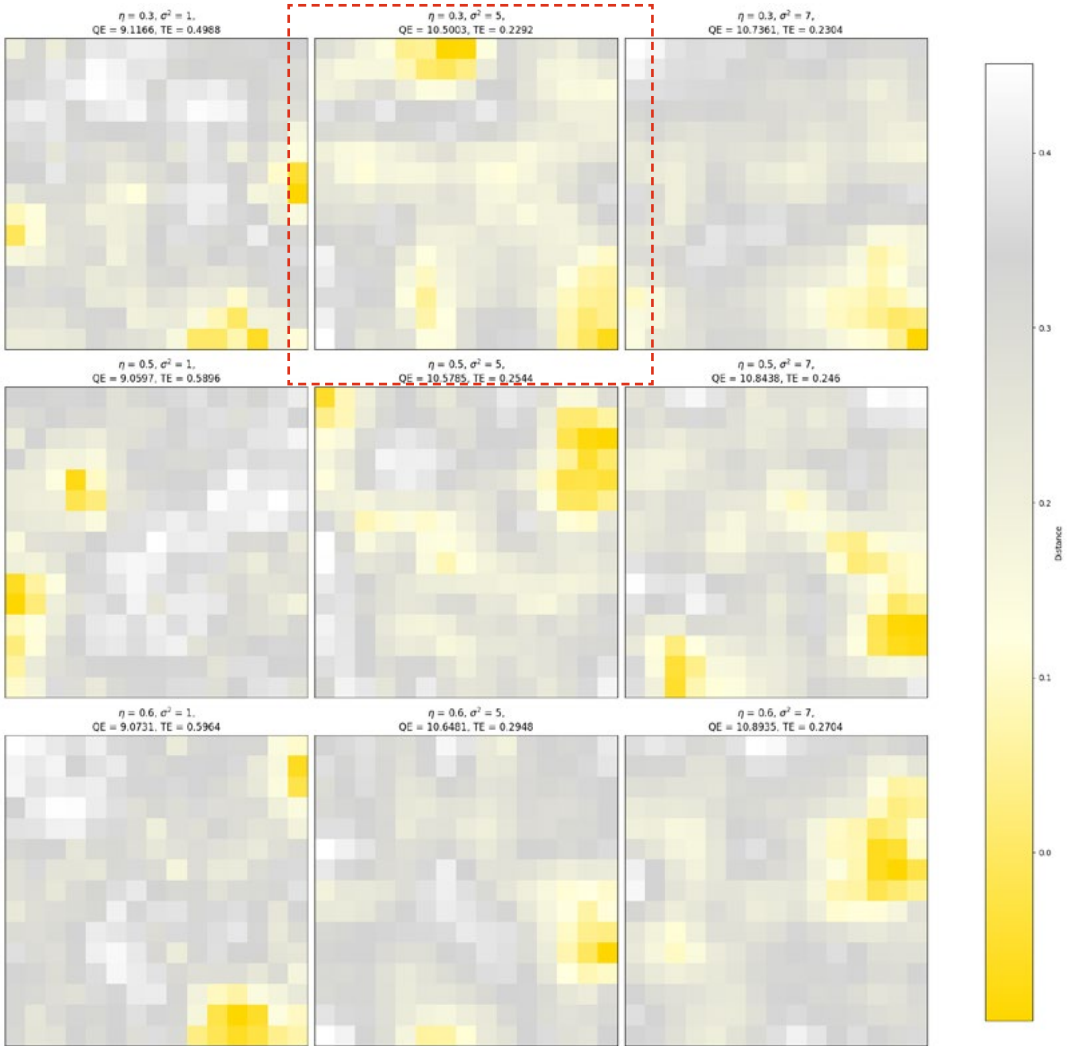
```
Image_path=r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Image'
ImageFiles = os.listdir(Image_path)
```

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

C.1.2 Build som

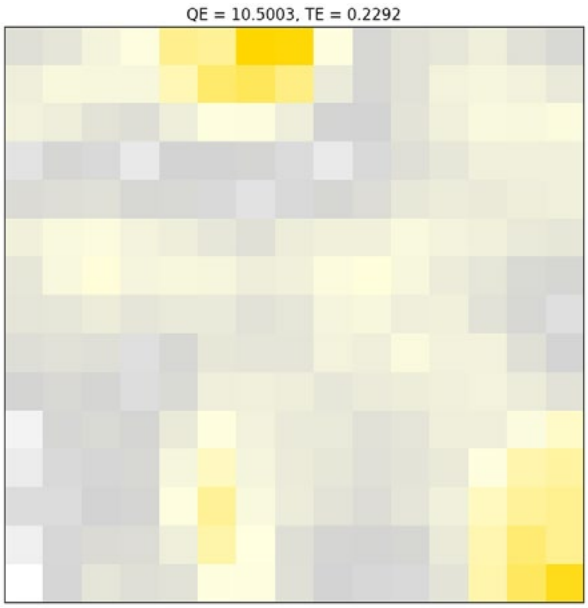
Create a som with feature with 1024 dimension, and normalise the training data before training.

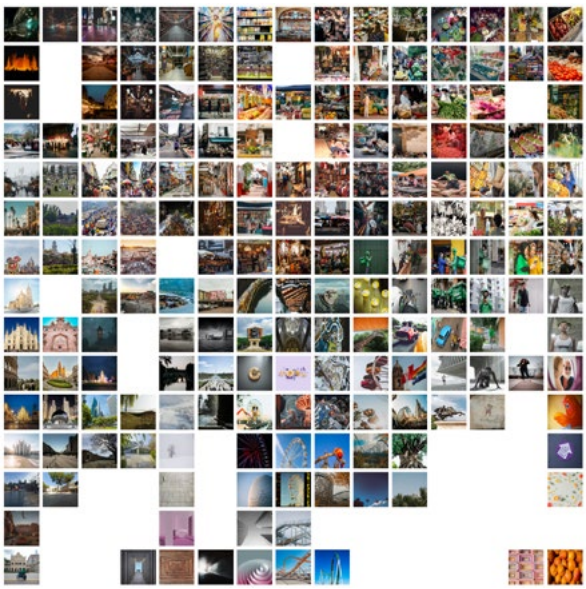
For the 1024 dimension vector matrix, the euclidean distance calculation will betther than cosine, so I import the euclidean distance related function from the som_euclidean_class.py



And I chose the som with the index1

The selected som U-Matrix and image mapping are shown below.





C.2 Create searching function

The Euclidean distance between the unit picture vector and the unit itself is used to calculate the picture that best matches the unit. And the corresponding picture is found by searching 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(Image_train_data, som, query_features), cmap=cmap)
    plt.title(f'Activated_ImageSom')
    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(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
```

D. Create film subtitle som and searching function

D.1 Create film subtitle som

D.1.1 Load subtitle

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
```

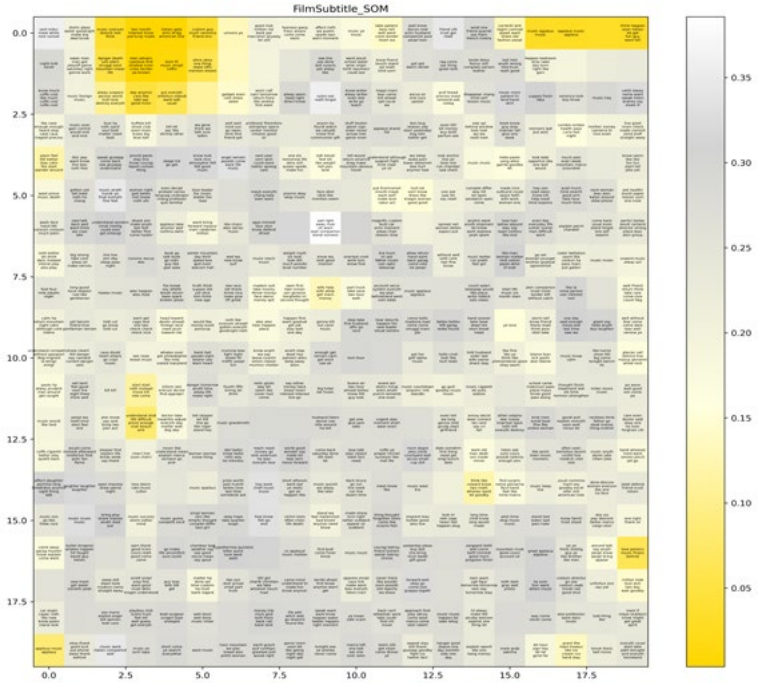
```
film_path = r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Film'
Filmlib.remove_spaces_in_filenames(film_path)
```

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

merged_subtitles
```

D.1.2 Build som

And I chose the som with the index 4

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



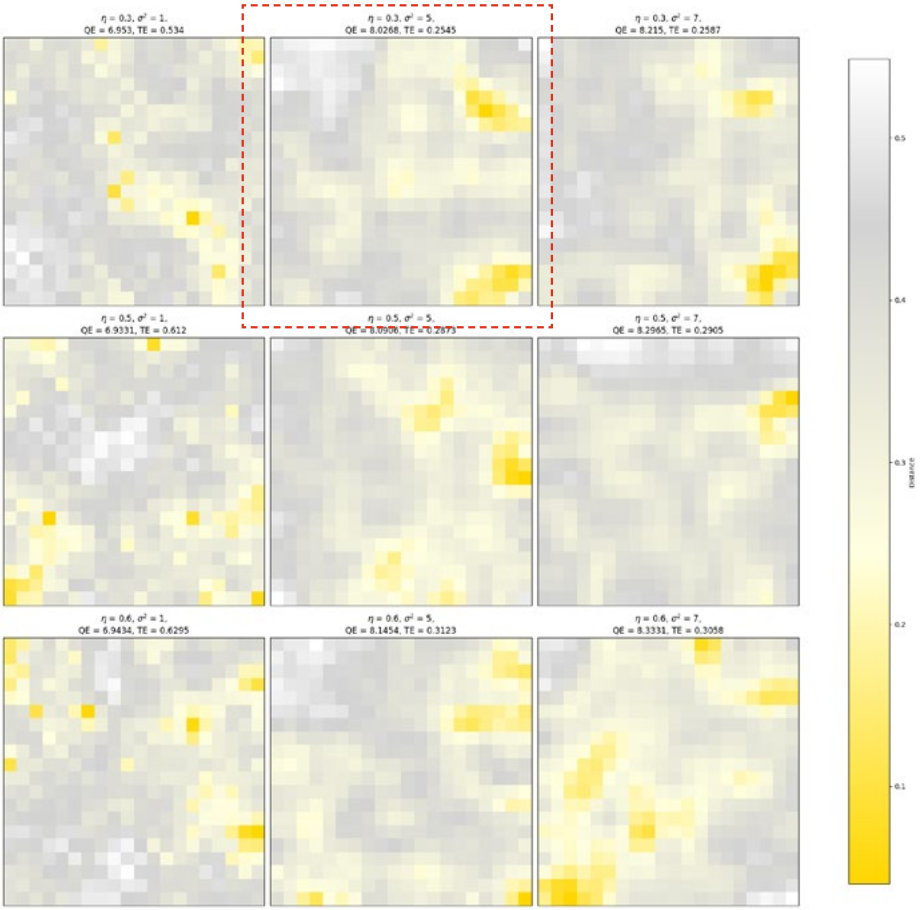
D.2 Create searching function

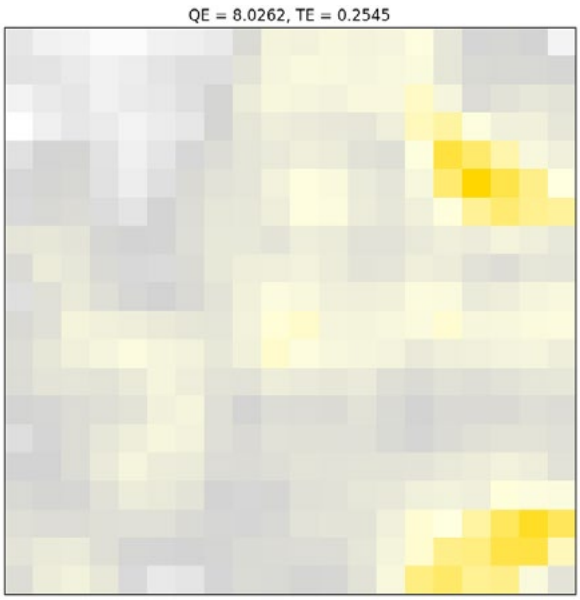
The basic searching idea of this function is similar to the previous text search, but after matching the most similar subtitles, it will return the start time and end time of the subtitles according to the index, and use the ffmpeg command to display the film clips.

```
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.title('Activated_SubtitleSom')
        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
```

E. Create film frames som and searching function

Specific implementation steps as above, the dataset is the frames of the film ech film I get 300 frames images. The following shows the creation of som and selected som.





E.2 Create searching function

The idea is similar to the search for film subtitles above. The difference is that after getting the most similar film frame, the 2 seconds around the frame will be intercepted according to the time of the frame as output matched clip.

```
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(frames_train_data, som, query_features), cmap=cmap)
    plt.title(f'Activated_FramesSom')
    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
```

PART II: Build Searching Engine

The 2 functions here mainly play the part of a tandem where the input query are converted into each other. It is also feasible to use video as input, but the idea is to convert video into frame image and then search, so the essence is the same as using image as input, so I only demonstrated two methods using text and image as input in my assignment, and provides 10 samples for each input.

1.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'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Image'
    image_result=search_ImageTextSom(ImageText_som, ImageText_model, FuyuText_data_dict, ImageLibrary_folder, query, vectorization ='doc2ve
    print('\n'+'The best matching image result is: '+str(image_result)+'\n')

    film_folder=r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\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))
```

2. 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'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\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'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\Film_Frames'
    film_path=r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\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))
```

PART III: Use more than one way to vectorise text databases

I chose Tf-idf for the second vectorised text and used svd for vector reorganisation and dimension modification, in my it shown as 'lsa'. Actually, each time I vectorise the data, I can choose the vectorization mode in the code, I mainly wrote 3 types vectorization, tfidf, lsa(tfidf + svd) and doc2vec. Here I will show a simple example of these 3 vectorization methods with the same query.

```
query='Road? Where is the road? We are going to the future.'
```

TF-IDF Vectorise

```
tfidf_train_data=tfidf_novel_model.vector_matrix

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

```
[{'paragraph': ' We are nearly there now." They were going up the steep hill now. Ranni had put the car into bottom gear, and it growle
d up slowly, the hill-road just as bad as the road they had left. The road wound to right and left in order to make the climbing of the
hill easier.',
  'nr': 105,
  'ID': 'blyton-secret-of-moon-castle',
  'type': 'epub'},
 {'paragraph': ' When the street turned into a road it turned into a road a hundred feet wide: one of those roads which Charles III., wh
en he came to the Spanish throne from Naples, full of beneficent projects and ideals, bestowed upon his unwilling and ungrateful subject
s.',
  'nr': 1320,
  'ID': 'Familiar-Spanish-Travels',
  'type': 'epub'},
 {'paragraph': ' It sprang into a panic-stricken gallop and was off down the road. They were nearly out of sight before the first of the
ir pursuers had run out into the road behind them. Then half a dozen puffs of smoke showed that they were fired on, but an instant later
they were out of sight around a bend in the road.',
  'nr': 424,
  'ID': 'The-strange-people-by-Murray-Leinster',
  'type': 'epub'}]
```

lsa Vectorise

```
lsa_train_data=lsa_novel_model.vector_matrix

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

```
[{'paragraph': ' What fun they were going to have!.',
  'nr': 326,
  'ID': 'blyton-secret-of-moon-castle',
  'type': 'epub'},
 {'paragraph': '" \'That\'s in Italy, ain\'t it?\' \'Hell no! It\'s in Finland! I\'m going to Venice and I\'m going to have a gonDOla of my own,
and ride around in it, and have a bunch of wha\' - juh - call - \'ems to sing Ytalian songs to me.",
  'nr': 361,
  'ID': 'lewis-work-of-art',
  'type': 'epub'},
 {'paragraph': ' "I wear these," he said coldly, "to avoid argument." With a flick she had it off again. "I wasn\'t going to argue—I wa
s only going to thank you." "I can\'t conceive for what. In any case, I don\'t want to be thanked.',
  'nr': 932,
  'ID': 'wharton-here-and-beyond',
  'type': 'epub'}]
```

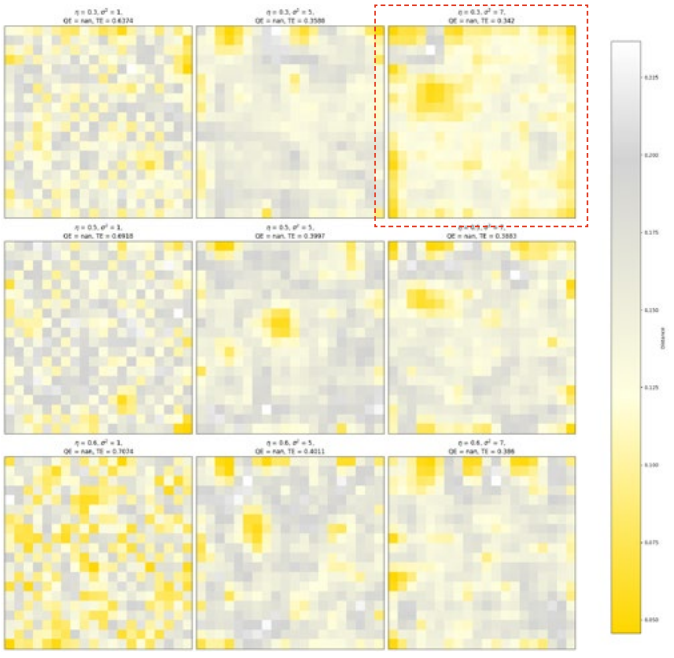
Doc2vec Vectorise

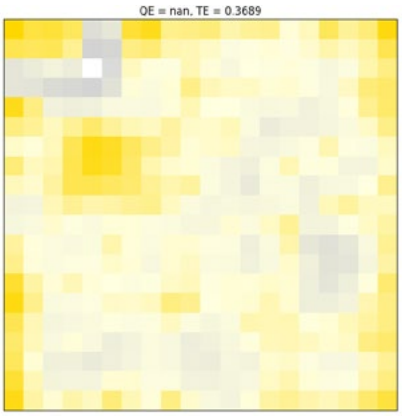
```
doc2vec_train_data=doc2vec_novel_model.vector_matrix

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

```
[{'paragraph': ' They were in the main evidently of the so - called learned professions or the arts—doctors, lawyers, preachers, actors,
writers, with a goodly sprinkling of merchants, manufacturers and young and middle - aged society men, as well as politicians and monied
idlers, generally a little the worse for their pleasures or weaknesses.',
  'nr': 858,
  'ID': 'dreiser-twelve-men',
  'type': 'epub'},
 {'paragraph': ' There is a sufficiently attractive hotel here for transients, and as an allurement to the marine and military leisure o
f Gibraltar, "The Picnic Restaurant," and "The Cabin Tea Room," where no doubt there is something to be had beside sandwiches and te
a.',
  'nr': 1688,
  'ID': 'Familiar-Spanish-Travels',
  'type': 'epub'},
 {'paragraph': '" The police inspector raised his eyes to Osip and asked: "Why is this, brother?" "Show Divine mercy, your honour," Osip
began, growing agitated. "Allow me to say last year the gentleman at Lutorydsky said to me, \'Osip,\' he said, \'sell your hay···you sell
it,\' he said.',
  'nr': 1358,
  'ID': 'chekhov-witch-and-other-stories',
  'type': 'epub'}]
```

With the query 'Road? Where is the road? We are going to the future.' Even though the same query input, the three vectorization results are totally different. But compared to lsa, doc2vec's results are richer, and the matched sentences seem to draw out more meanings. I think this will help with subsequent searches.

I also trained the Text som with lsa vectorization, we can see that the som reult also quite different with doc2vec som result.



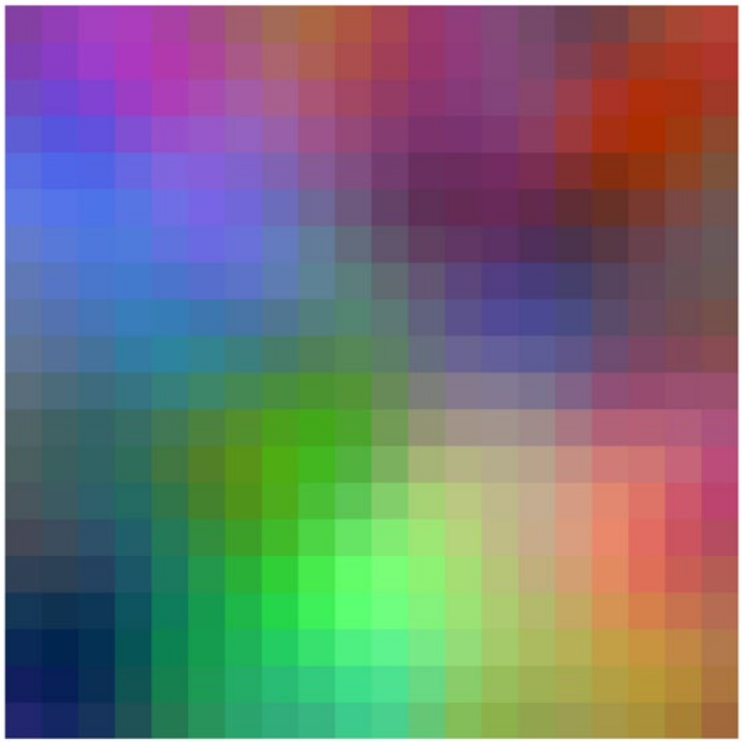


PART IV: Fit PCA

1. Fit PCA on all cells of the SOM

The vector of each cell in Doc2vec som is downscaled to 3 dimensions using PCA and the three dimensions are assigned to that individual cell as RGB corresponding to the colour.

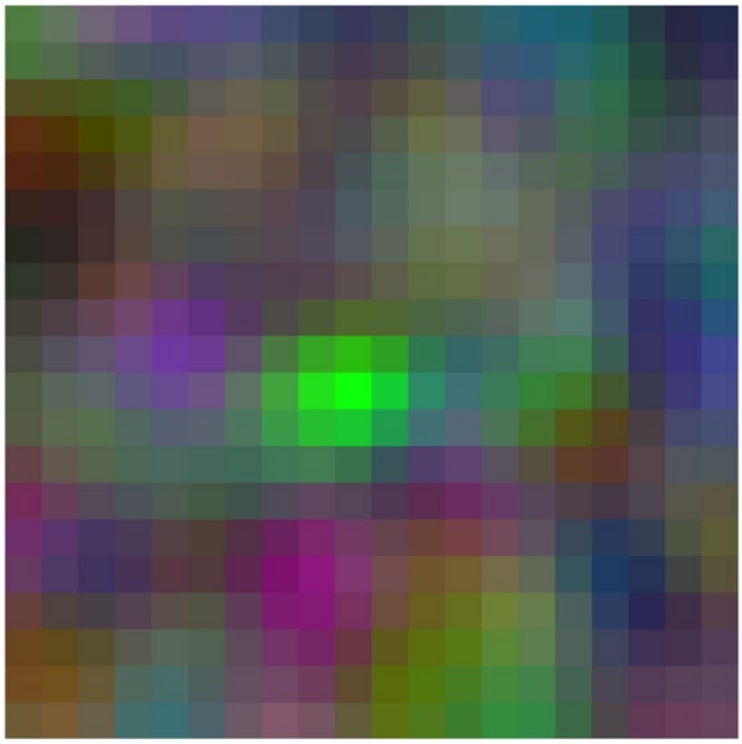
The result display like this:



2. Fit PCA on the entire training set

The vector of each cell in Doc2vec som is downscaled to 3 dimensions refers to the whole database using PCA and the three dimensions are assigned to that individual cell as RGB corresponding to the colour.

And then when render it back to the som , it showed the result shown below:



Appendix: Function Explanation

I have wrote several function for helping me train the data, get the dict or list, here I will select some of my own functions to explain how it works.

1. Query List Generating

```python
# Create a query list
def add_query_to_list(query, query_list):
    j=len(query_list)
    if isinstance(query, str):
        i=len(query_list)
        query_dict={'index':i,'query':query}
        query_list.append(query_dict)
    elif isinstance(query, list):
        for i in range(len(query_list), len(query_list)+len(query)):
            query_dict={'index':i,'query':query[i-j]}
            query_list.append(query_dict)

    return query_list
```

In order to test my search results more conveniently, I need a large number of different input queries, but it is very troublesome to enter or switch manually each time, so I wrote this function to batch record queries into a list, so that I can let gpt to help me generate some queries and add them to the list through this function.

```python
text_list=[
    "Picturesque picnic spots with scenic views near me.",
    "Cozy coffee shops with fireplace ambiance in downtown.",
    "Charming bookstores with hidden reading nooks in the city.",
    "Rooftop bars with panoramic skyline views and live music.",
    "Tranquil parks with winding trails and serene ponds.",
    "Quaint boutiques with vintage charm and eclectic decor.",
    "Intimate jazz clubs with dim lighting and velvet couches.",
    "Art galleries showcasing contemporary works in urban settings.",
    "Hidden speakeasies with secret entrances and craft cocktails.",
    "Botanical gardens with exotic flora and peaceful water features."
]

query_list=add_query_to_list(text_list, query_list)
```

```
{'index': 25,
 'query': 'Essential tips for novice gardeners starting a vegetable garden.'},
{'index': 26, 'query': 'Picturesque picnic spots with scenic views near me.'},
{'index': 27,
 'query': 'Cozy coffee shops with fireplace ambiance in downtown.'},
{'index': 28,
 'query': 'Charming bookstores with hidden reading nooks in the city.'},
{'index': 29,
 'query': 'Rooftop bars with panoramic skyline views and live music.'},
{'index': 30,
 'query': 'Tranquil parks with winding trails and serene ponds.'},
{'index': 31,
 'query': 'Quaint boutiques with vintage charm and eclectic decor.'},
{'index': 32,
 'query': 'Intimate jazz clubs with dim lighting and velvet couches.'},
{'index': 33,
 'query': 'Art galleries showcasing contemporary works in urban settings.'},
{'index': 34,
 'query': 'Hidden speakeasies with secret entrances and craft cocktails.'},
{'index': 35,
 'query': 'Botanical gardens with exotic flora and peaceful water features.'}]
```

As shown in the picture above, I first let gpt write some random queries for me, and then I call this function. These queries will be added to my original query list and their indexes will be recorded.

2. Get the Film Clip around the Frame

```python
def getFilmAroundFrame(path, all_frames_info, result, radius):

    film_folder=path
    film_name=all_frames_info[find_idx(all_frames_info, result)]['movie_file_name']
    film_path=os.path.join(film_folder, film_name)
    frame_name=result[0].split('.png')[0]

    frame_time=all_frames_info[find_idx(all_frames_info, result)]['frame_time']

    film = VideoFileClip(film_path)

    start_time = max(frame_time - radius, 0)
    end_time = min(frame_time + radius, film.duration)

    if start_time < 0:
        start_time = 0

    if end_time > film.duration:
        end_time = film.duration

    #display frame clip
    Filmlib.display_matched_frame(film_path, frame_name, start_time, end_time)
```

This is an optimized version of the code for the film searching session. The key is to intercept the filmclips for a certain duration around the frame through the time point, but I have added a judgment so that it can still working when the matching frame is the first and last frame of the movie.

3. Print Film Subclip

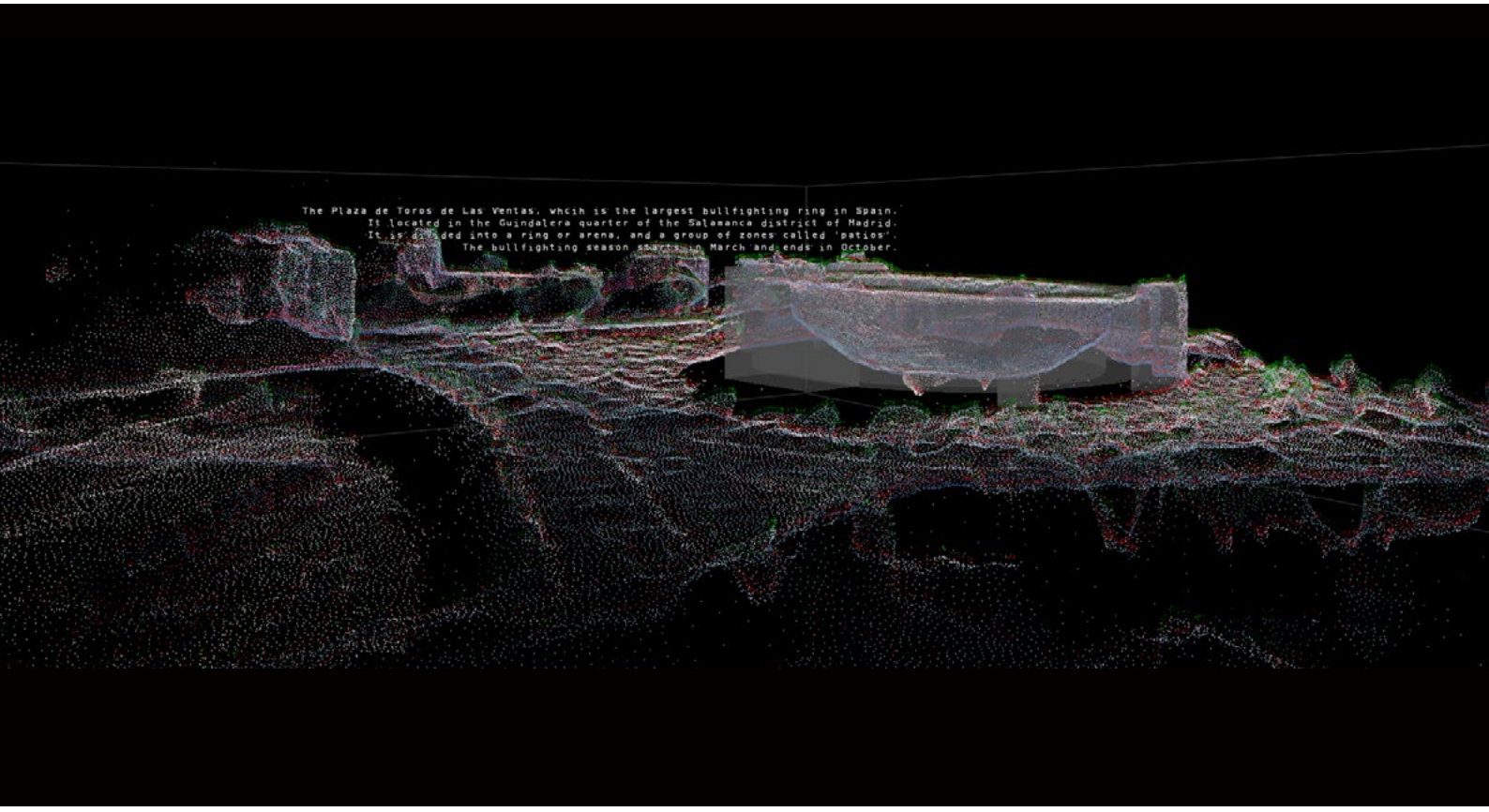
```
def display_matched_filmclip(result, movie_path,query):
    start_time_parts = result[0]['start_time'].split(',')
    start_seconds = int(start_time_parts[0].split(':')[-1]) + 60 * int(start_time_parts[0].split(':')[1]) + 3600 * int(start_time_parts[0].split(':')[0])
    end_time_parts = result[0]['end_time'].split(',')
    end_seconds = int(end_time_parts[0].split(':')[-1]) + 60 * int(end_time_parts[0].split(':')[1]) + 3600 * int(end_time_parts[0].split(':')[0])

    #output_folder=r'E:\Document\Dylan Zhang File\UCL\Term2\Skill_Class\assignment\Julian\Datasets\output_clips'
    #filename= 'matched_clip_{}.mp4'.format(query)
    #output_path = os.path.join(output_folder, filename)
    output_path='matched_clip_{}.mp4'.format(query)

    ffmpeg_command = ['ffmpeg', '-i', movie_path, '-ss', str(start_seconds), '-to', str(end_seconds), '-c', 'copy', output_path]
    subprocess.run(ffmpeg_command, capture_output=True, text=True)

    display(Video(output_path, width=800,embed=True))


def display_matched_frame(movie_path,frame_name,start_time,end_time):

    output_path = 'frame_clip_{}.mp4'.format(frame_name)

    ffmpeg_command = ['ffmpeg', '-i', movie_path, '-ss', str(start_time), '-to', str(end_time), '-c', 'copy',output_path]

    subprocess.run(ffmpeg_command, capture_output=True, text=True)

    display(Video(output_path, width=800,embed=True))
```

In order to be able to search for films in the notebook and display them intuitively, I need a function that can print the matching clips. I integrated these functions into film_class_model.py and used them in the notebook by calling functions. (The calling method is demonstrated in the previous function)

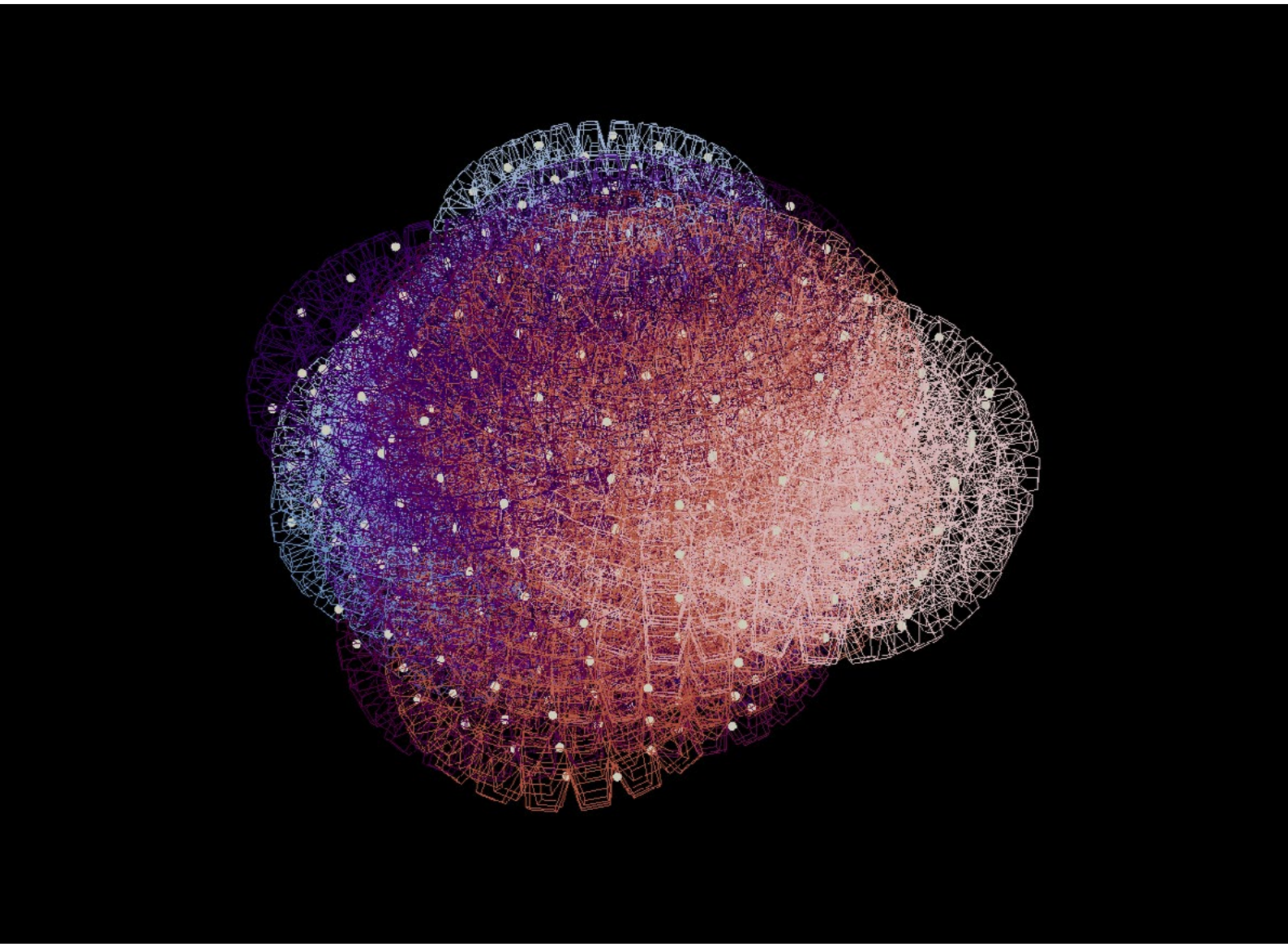
The printing method in the class uses moviepy's subclip and output, but due to environmental issues with my computer, this method is not available, so I used the 'ffmpeg command' instead of moviepy to display movie clips.

HOUDINI ASSESSMENT

RC11_23135525



CONTENT



HOUDINI 1

HOUDINI FOUNDAMENTALS

1 HOUDINI FOUNDAMENTALS

1.1 Setup 1

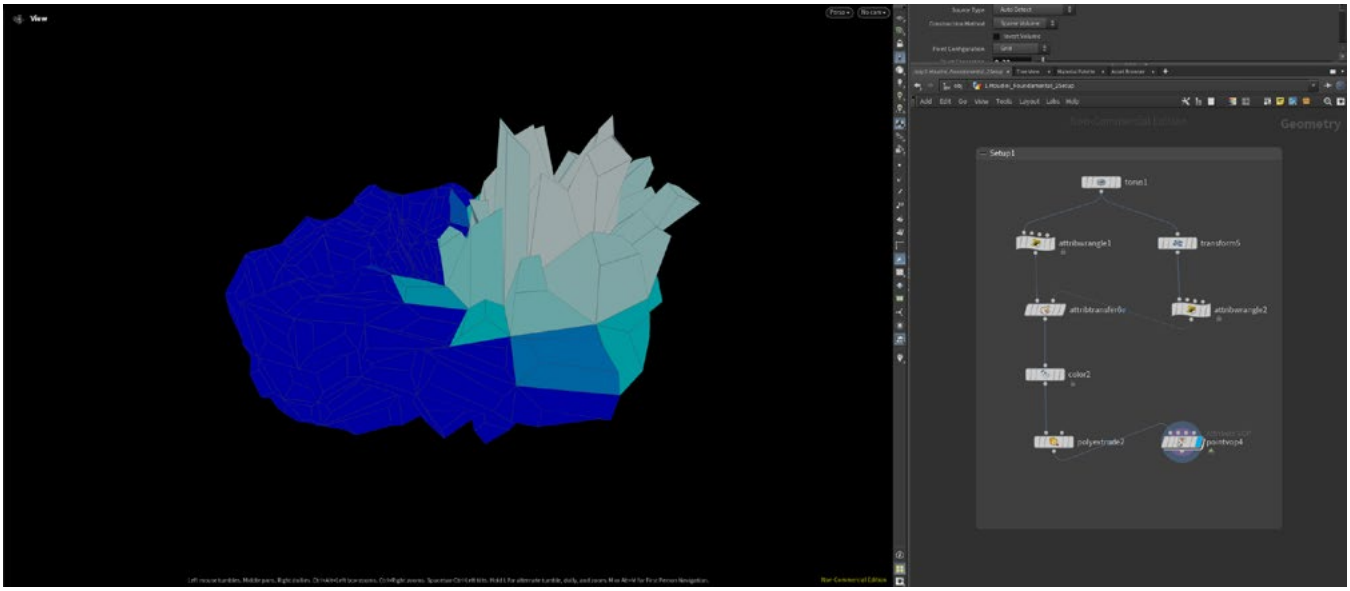
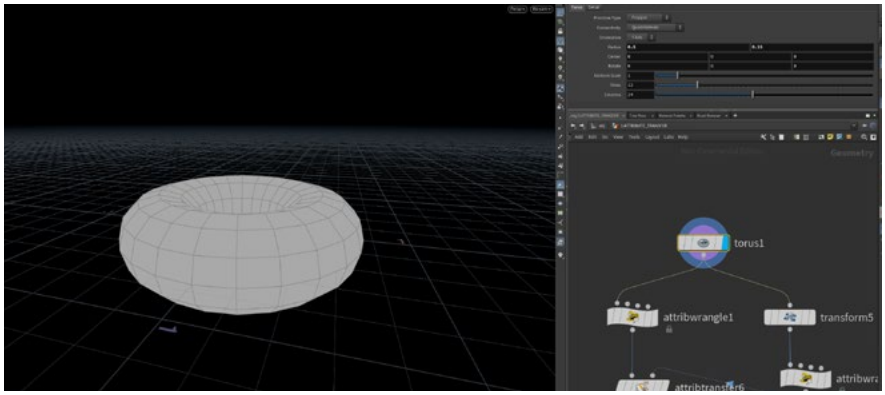


Figure1: Houdini Foundamentals: Setup 1 Overview

Setup 1 Discription

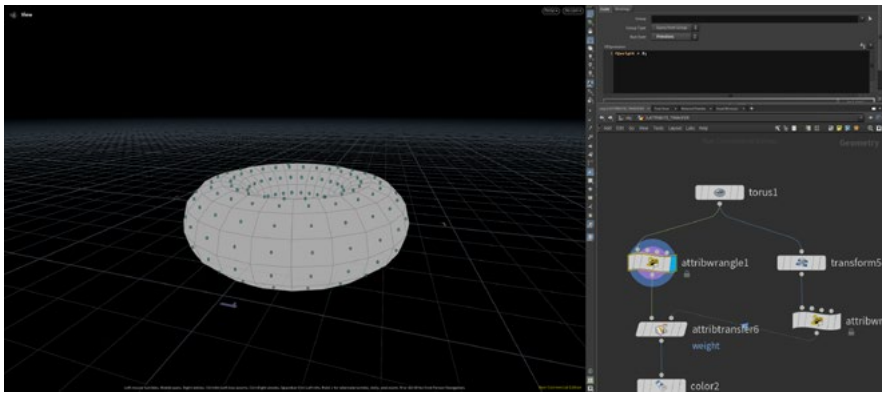
1.1.1: Torus SOP

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



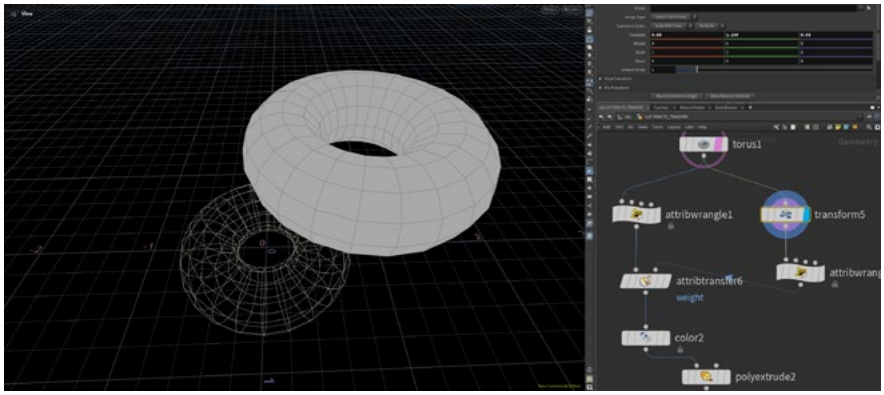
1.1.2: Attribute Wrangle SOP

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



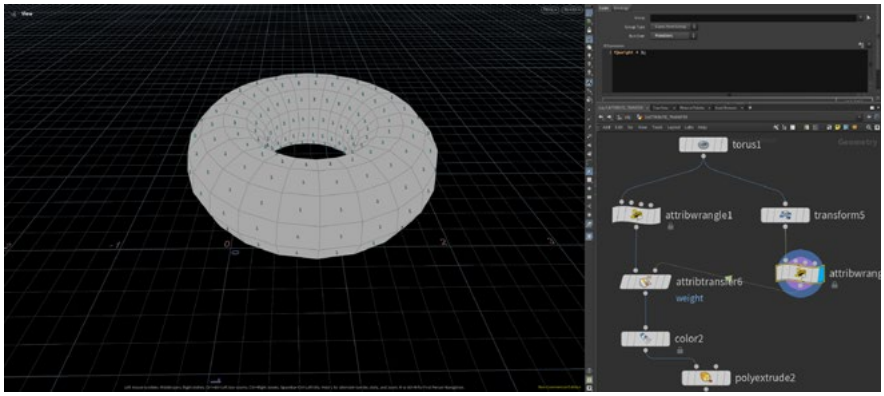
1.1.3: Transform SOP

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



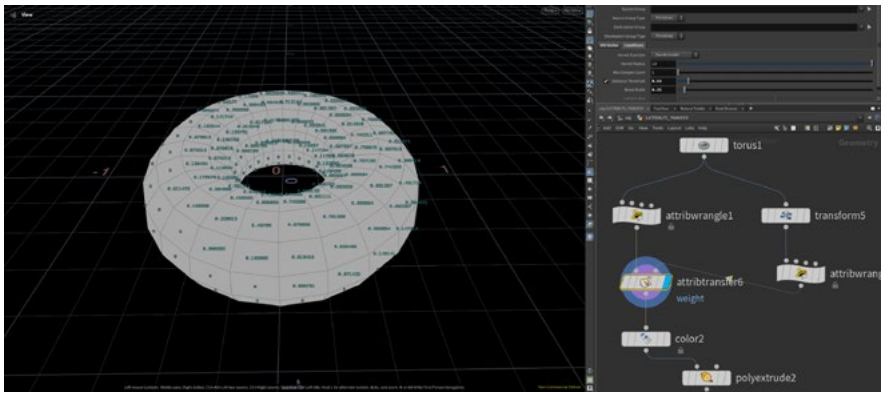
1.1.4: Attribute Wrangle SOP

The same as the previous Wrangle SOP (step 2), only with the opposite value. The reason we interpolate between 0 and 1 is because it is easier to work with normalized values than arbitrary values.



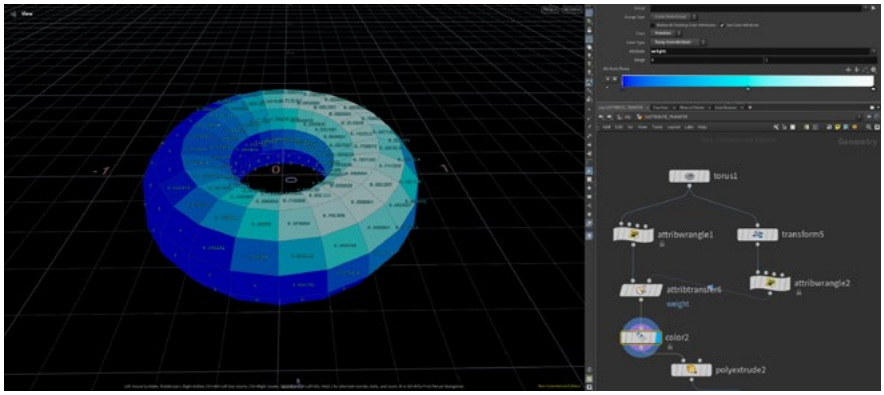
1.1.5: Attribute Transfer SOP

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



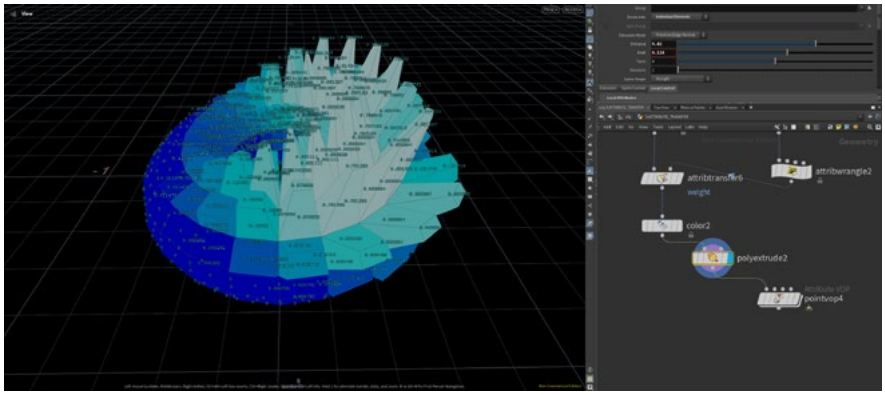
1.1.6: Colour SOP

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



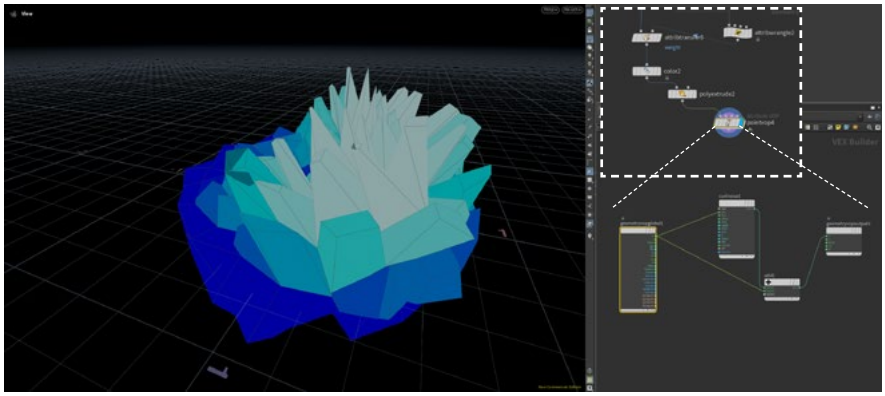
1.1.7: PolyExtrude SOP

Using the "weight" value in d"distance scale", we use the weight value as a multiplier of the extrusion value. If we set the distance value to 0, the extrusion is 0 (0 x weight). If we set the distance value to 1, we get the full weight value as extrusion (1 x weight).



1.1.8: Point Vop SOP

VOP nodes let you define a program (such as a shader) by connecting nodes together. I add curnoise and choose "Perlin noise" as the noise type to make the point shaped like a sound wave and add it to the output.



1.2 Setup 2

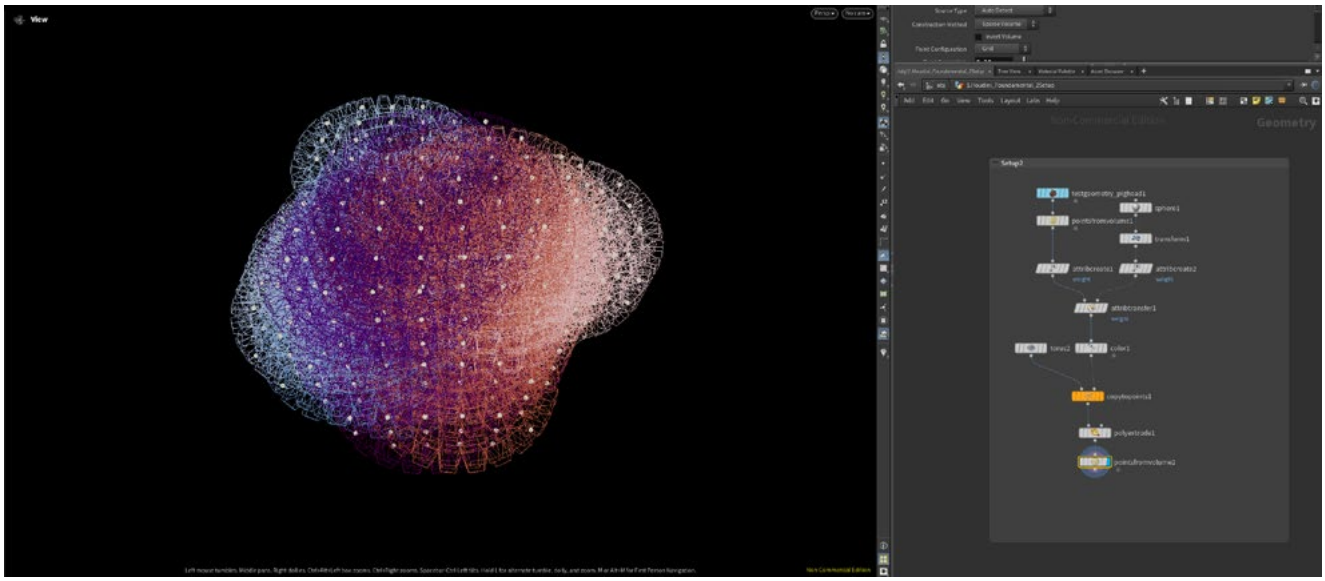
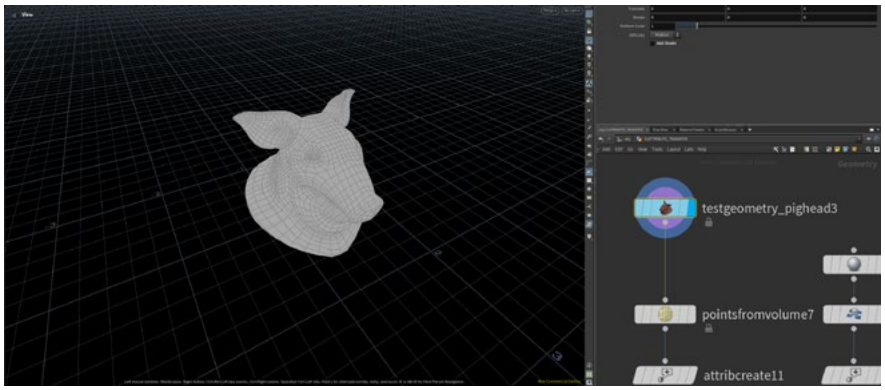


Figure2 : Houdini Foundamentals: Setup 2 Overview

Setup 2 Discription

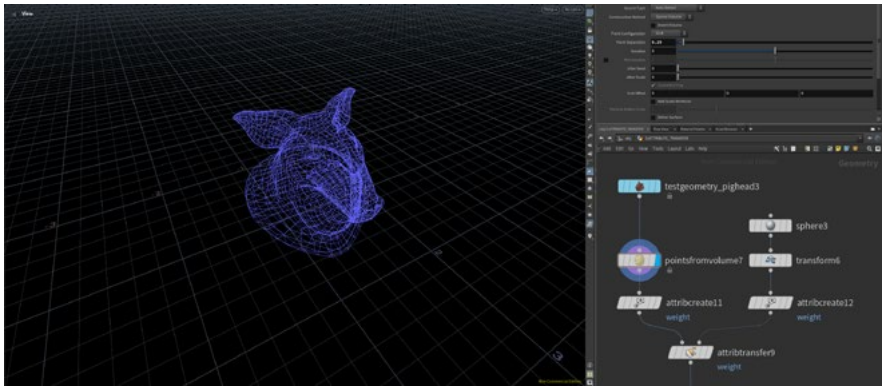
1.2.1: Pighead Geometry

The "pighead" is a test geometry that import from the outer file. It is a complex polygon.



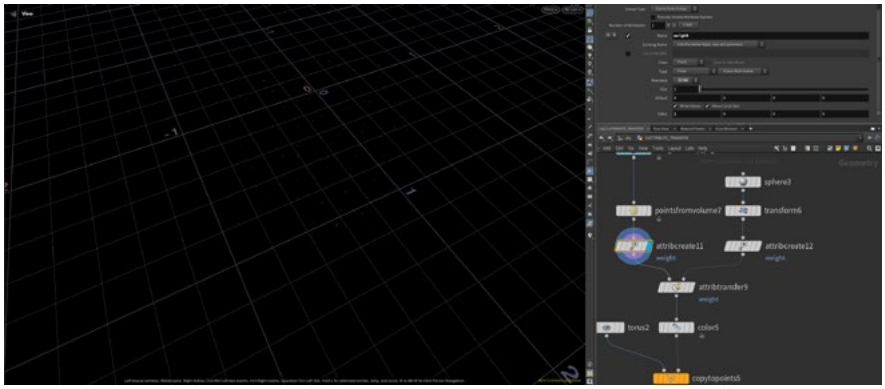
1.2.2: Points from Volume

The points from volume nodes can generate the points based on the input geometry and setting points separation.



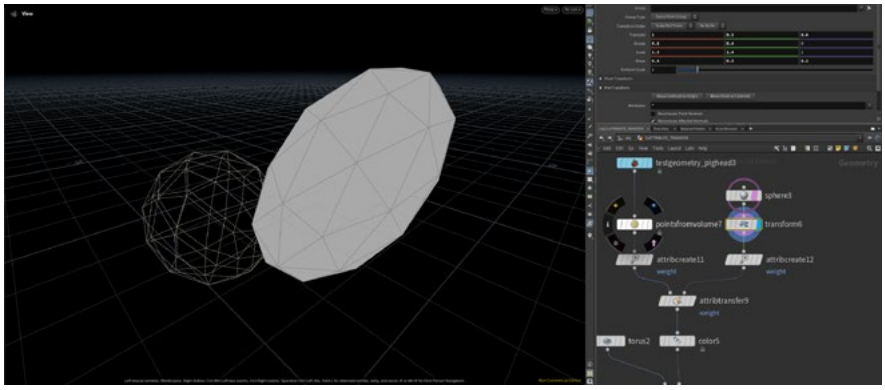
1.2.3: Attribute SOP

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



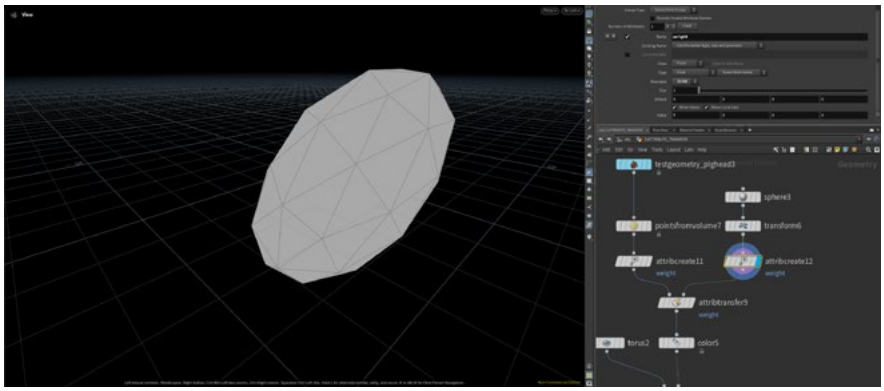
1.2.4: Sphere SOP, Transfer SOP

Add a "sphere" SOP as the transfor reference and transfer it by adjusting its axis and shearing it. So that we can get a more unexpecting result in the following steps.



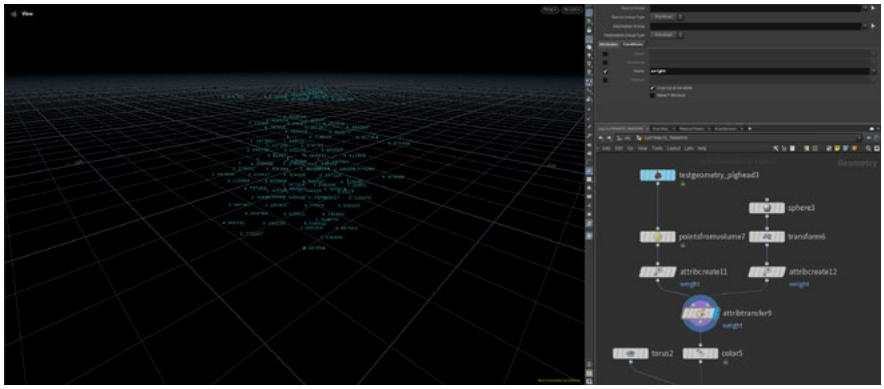
1.2.5: Attribute Sphere

The same with the previous attribute transfer. The node uses the "weight" primitive attribute to write out the distance value between the original sphere and the attractor. By setting the original 'weight' attribute as float, we can have value interpolations between 0 and 1.



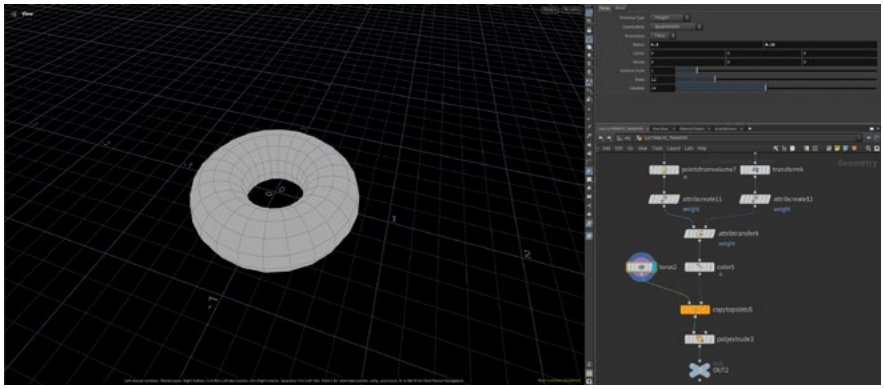
1.2.6: Attribute Transfer SOP

Combined two attribute from the previous steps. So we now turn all the geometry to the points with the value based on the original pighead and the reference geomety the sphere.



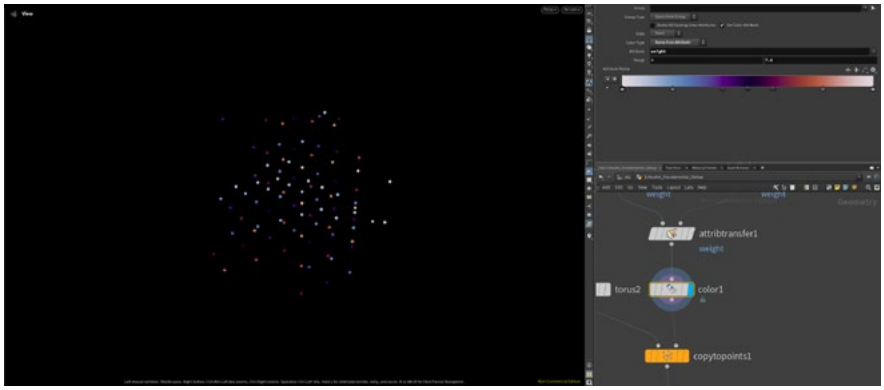
1.2.7: Torus SOP

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



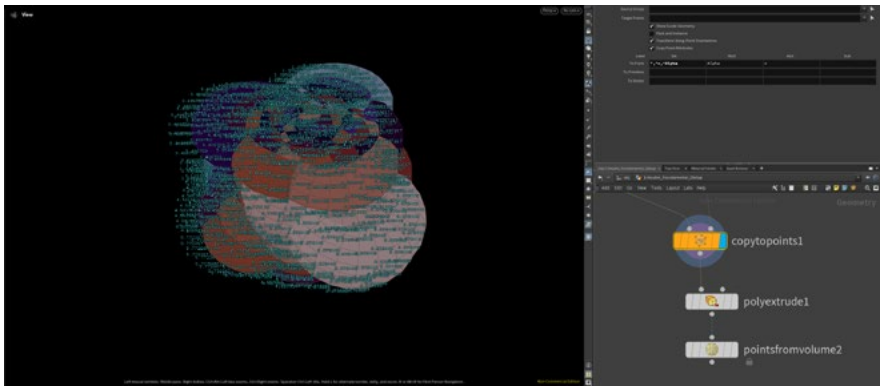
1.2.8: Colour SOP

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



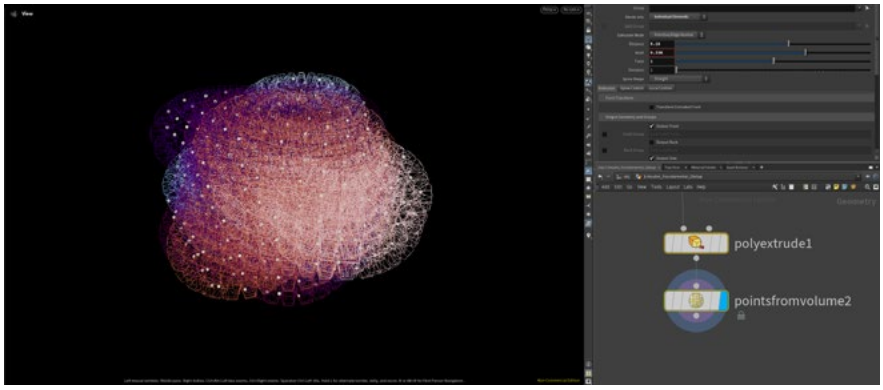
1.2.9: Copy Top Points

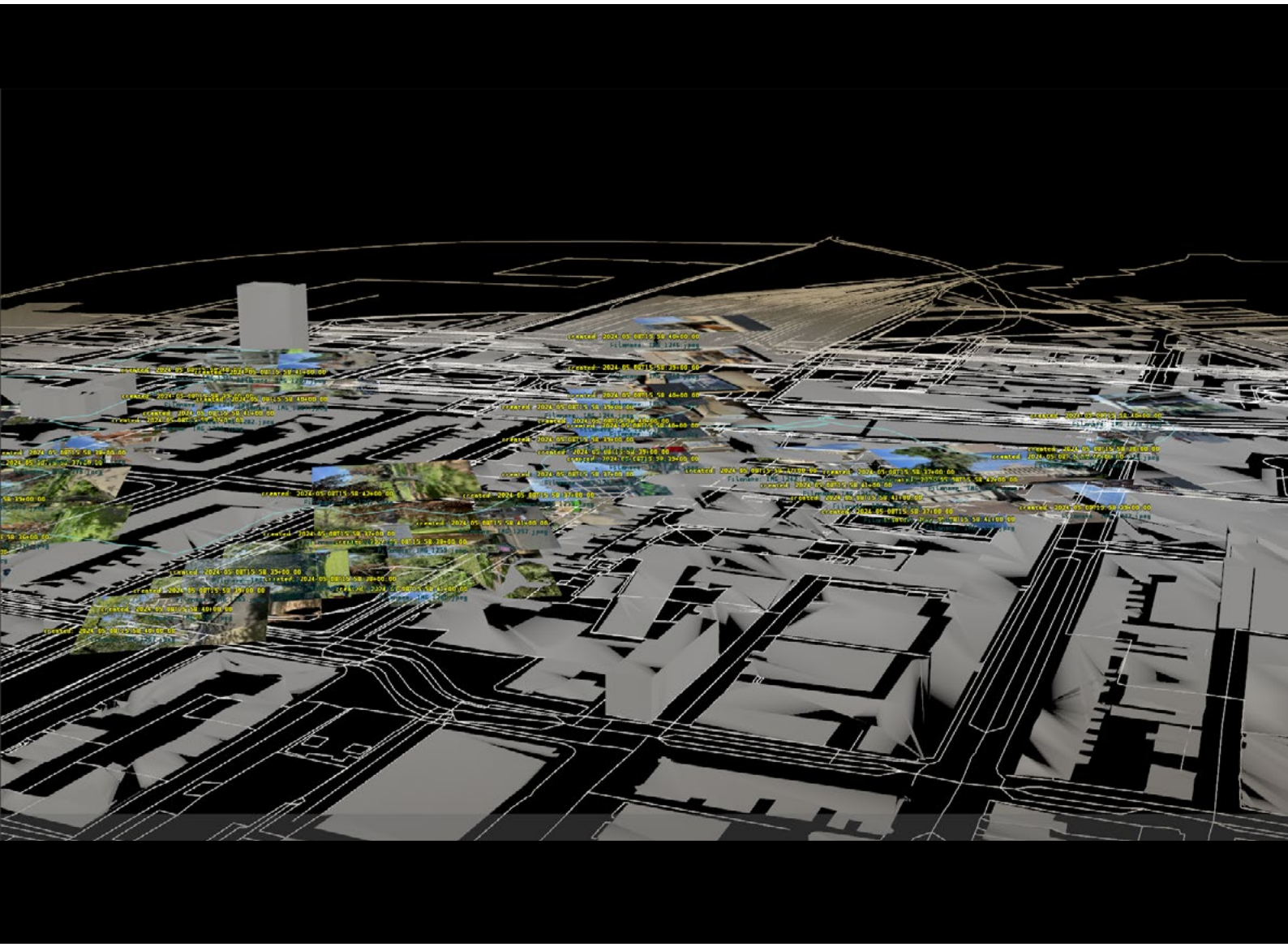
Use the "sphere" SOP as the reference geometry and "Twilight" as the color and transfer the previous points data based on the sphere.



1.2.10: PolyExtrude and turn SOP to Points from Volume

Using the "weight" value in d"distance scale". If we set the distance value to 1, we get the full weight value as extrusion (1 x weight). And then like step 2 generate the points based on the input geometry and setting points separation.





HOUDINI 2

VISUALIZING GPS & IMAGE METADATA

2 Overview

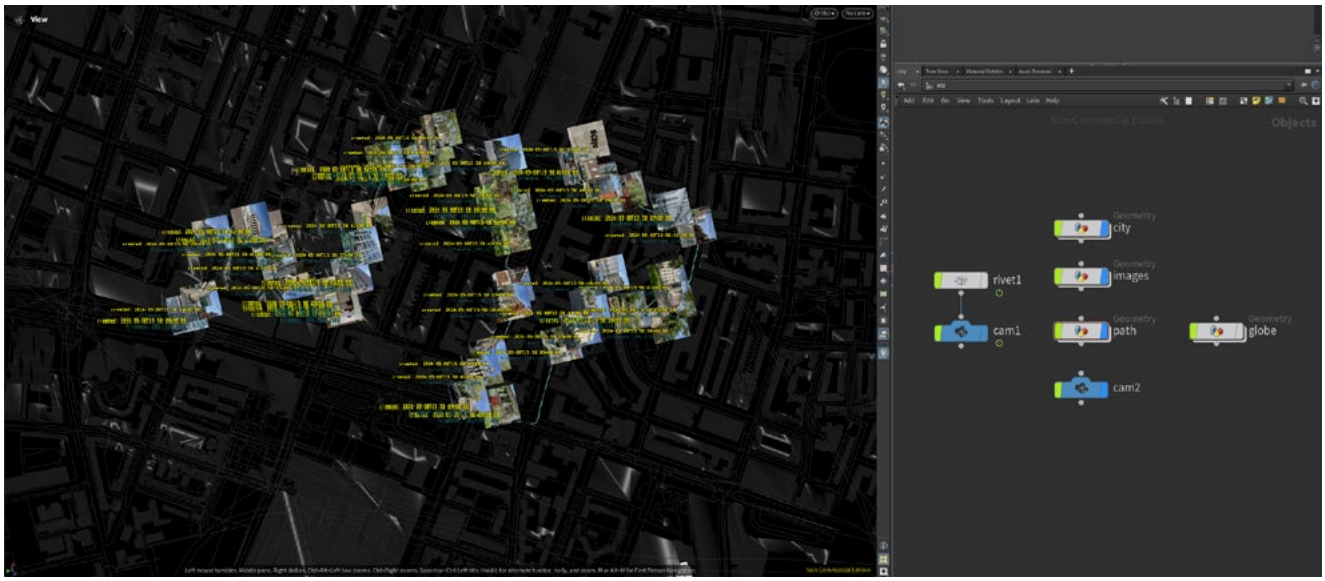
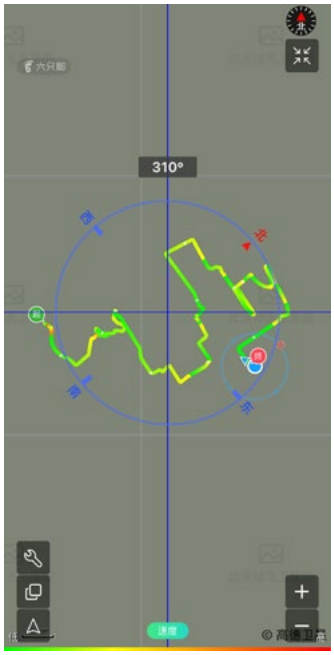


Figure3 : The Overview of Visualising gps and Image Metadata

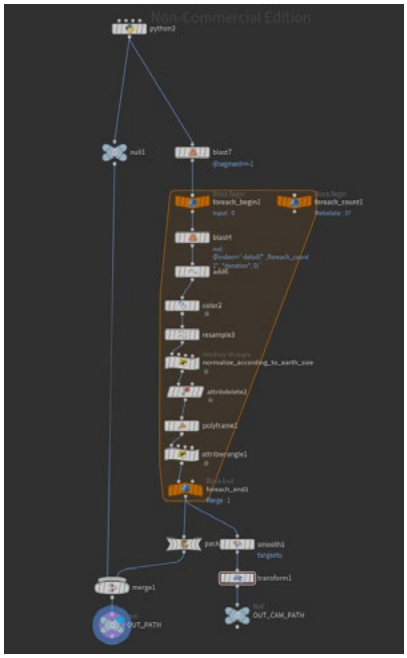
Houdini2 Discription

2.1. Extract GPX data to get path

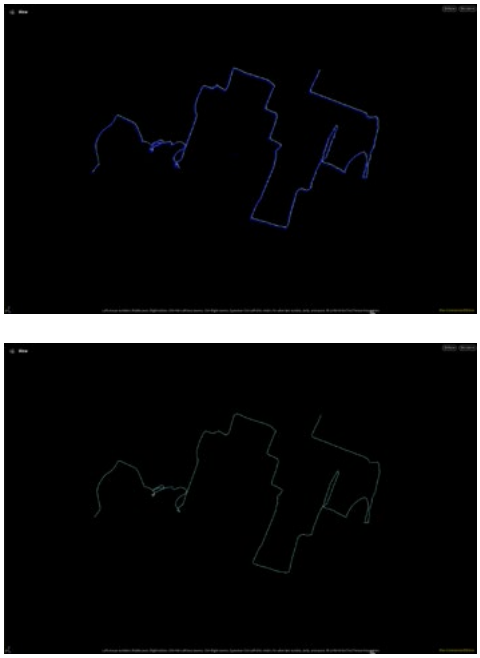
Before visualising the gpx path and image metadata, we need to get the gpx file. Due to my Google Maps timeline is unavailable, so I use the outdoor hiking trail recording software to obtain gpx file (as image1 shows).
Then I upload the gpx file in python module of Houdini. Add modules/functions to ForeachPoint to handle the path, and combine it with Smooth to generate OUT_CAM_PATH, which will be used as the Operator input Rivet1 to match the CAM and path line together.



[gpx recording app]



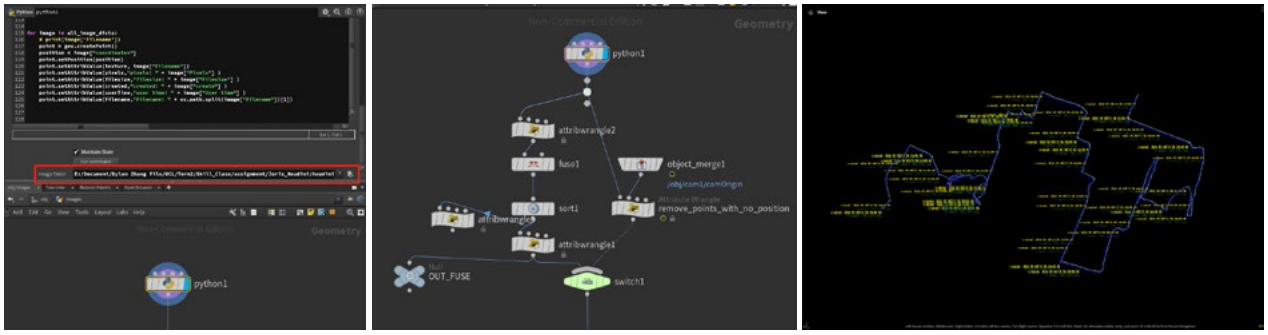
[the path generatng process]



[the path result]

2.2. Loading images (Visualized image data/infomation)

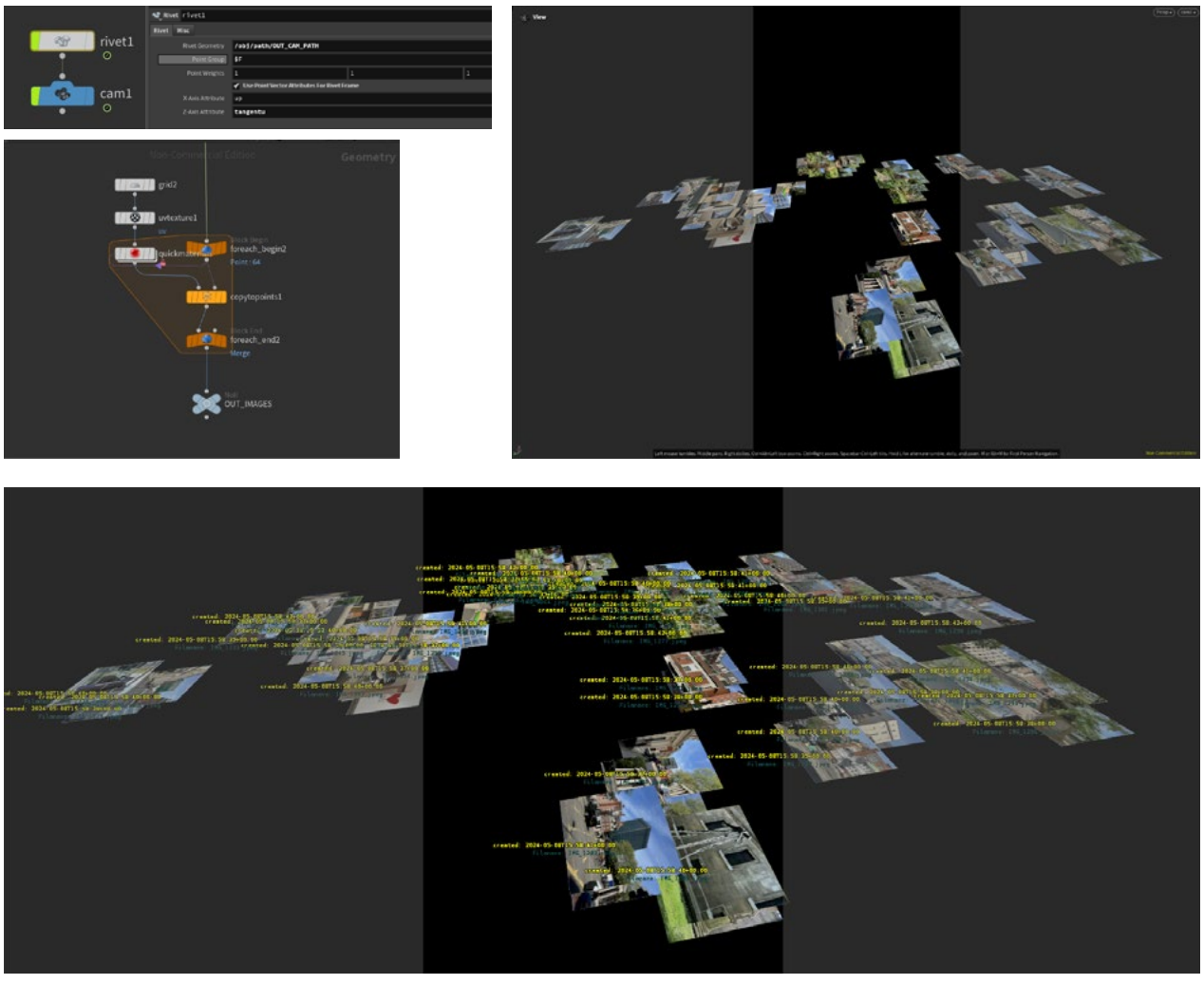
For the Image section, I import the folder where the image is located (in the Python module), and edit the path to this one. Then reading the image information (data) through code, and visualize the information content by adding Marker.



2.3. Loading images (link with points)

Add the Object_merge function, import camOrigin of CAM, and write a attribwangle to delete points without position. Then output OUT_FUSE which record result of fuse_1, and adjust Switch to make images oriented to CAM. The CAM is following the path, by setting OUT_CAM_PATH as Operator uploaded in Rivet_1.

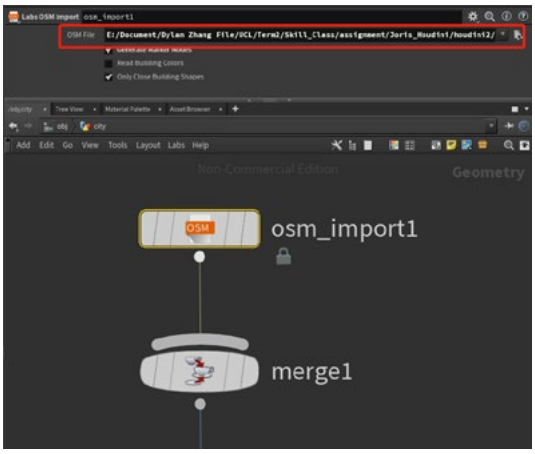
But in order to print the image on those points in the previous steps, I created a Grid, connected UV texture, and adjusted it to z-axis to ensure that the image materials on the grid can be presented correctly. Then connect it to Quickmaterial. In quickmaterial, adding Spare input, and adding ForeachPoint then drag it into Spare input. After that, Copytopoint link the image infomation with each image.



2.4. Extract geographical data (city map)

In order to obtain geographical information of the site, I exported the Camden area where the path is located in OpenStreetMap, and download osm file.

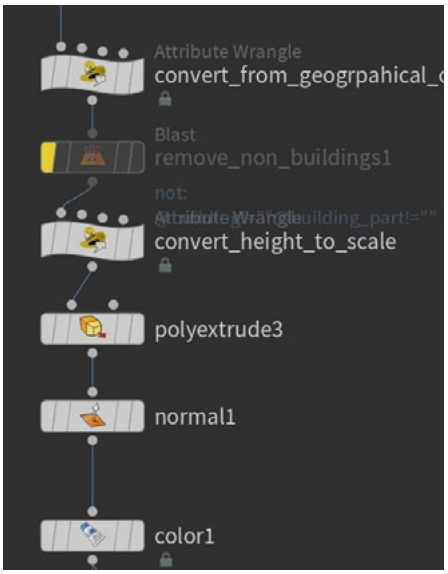
[open street map]

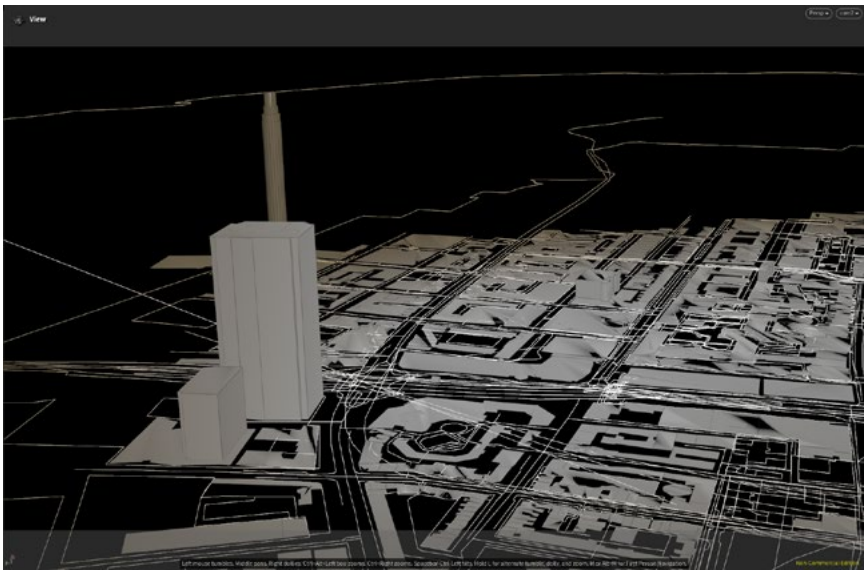


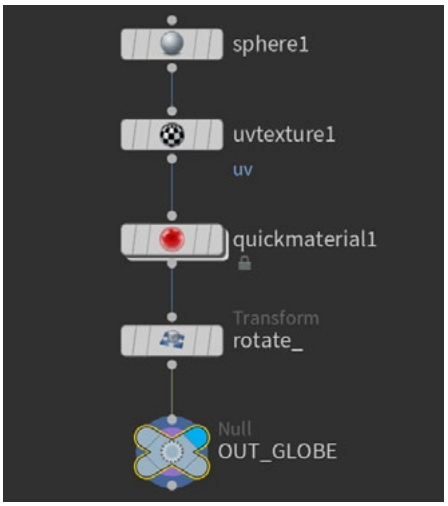
[import osm file]

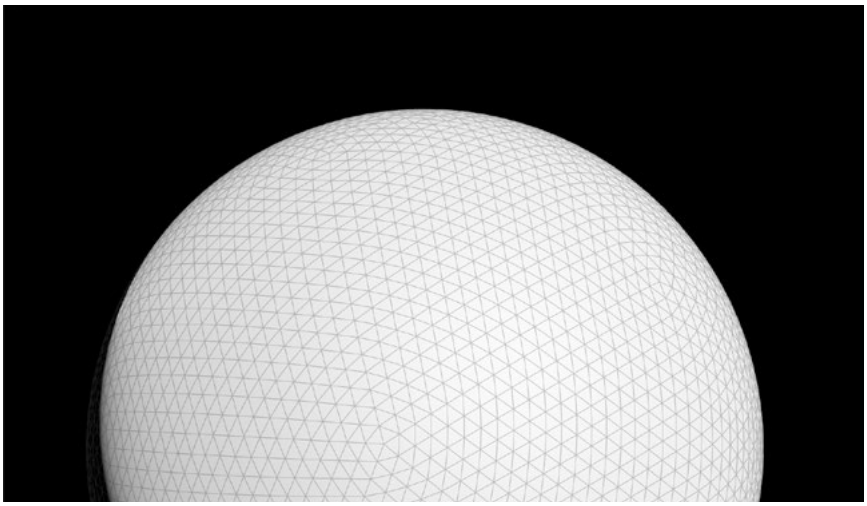
2.5. Generate maps and buildings

After merging Osm files, combine the Attribwangle input code to generate the map and obtain height information, Matching with the Earth (sphere), and then use Polyextrude to generate the volume according data in OSM.



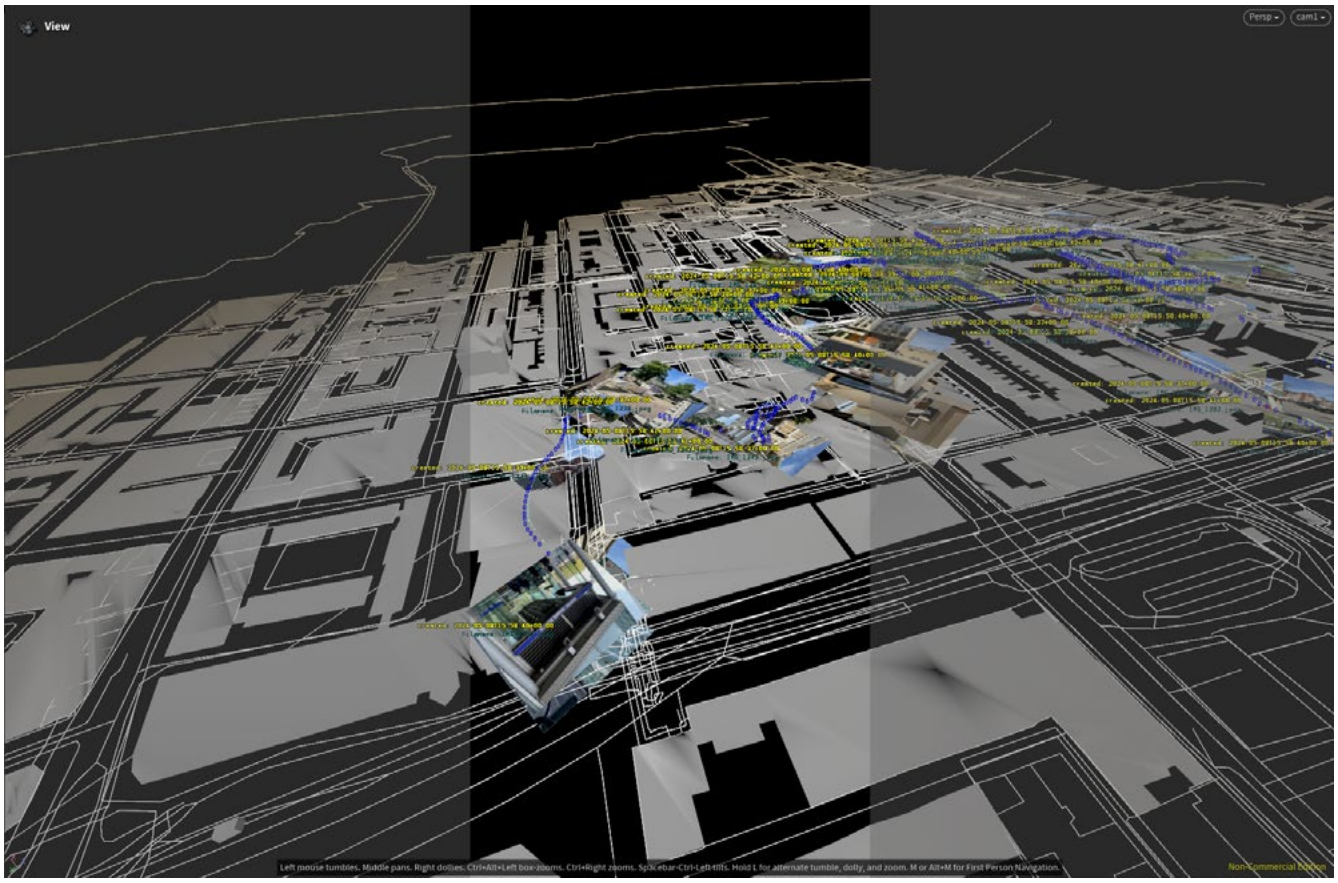




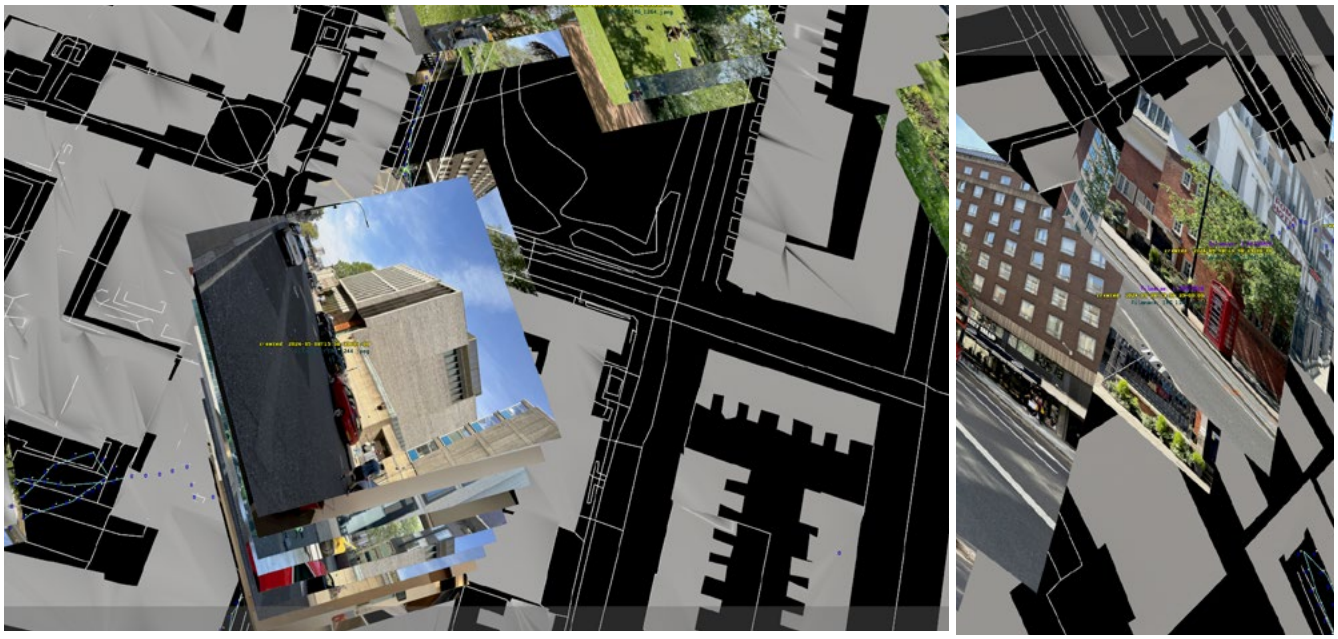


2.6. A series of images depicting the travel. (Performance/Render)

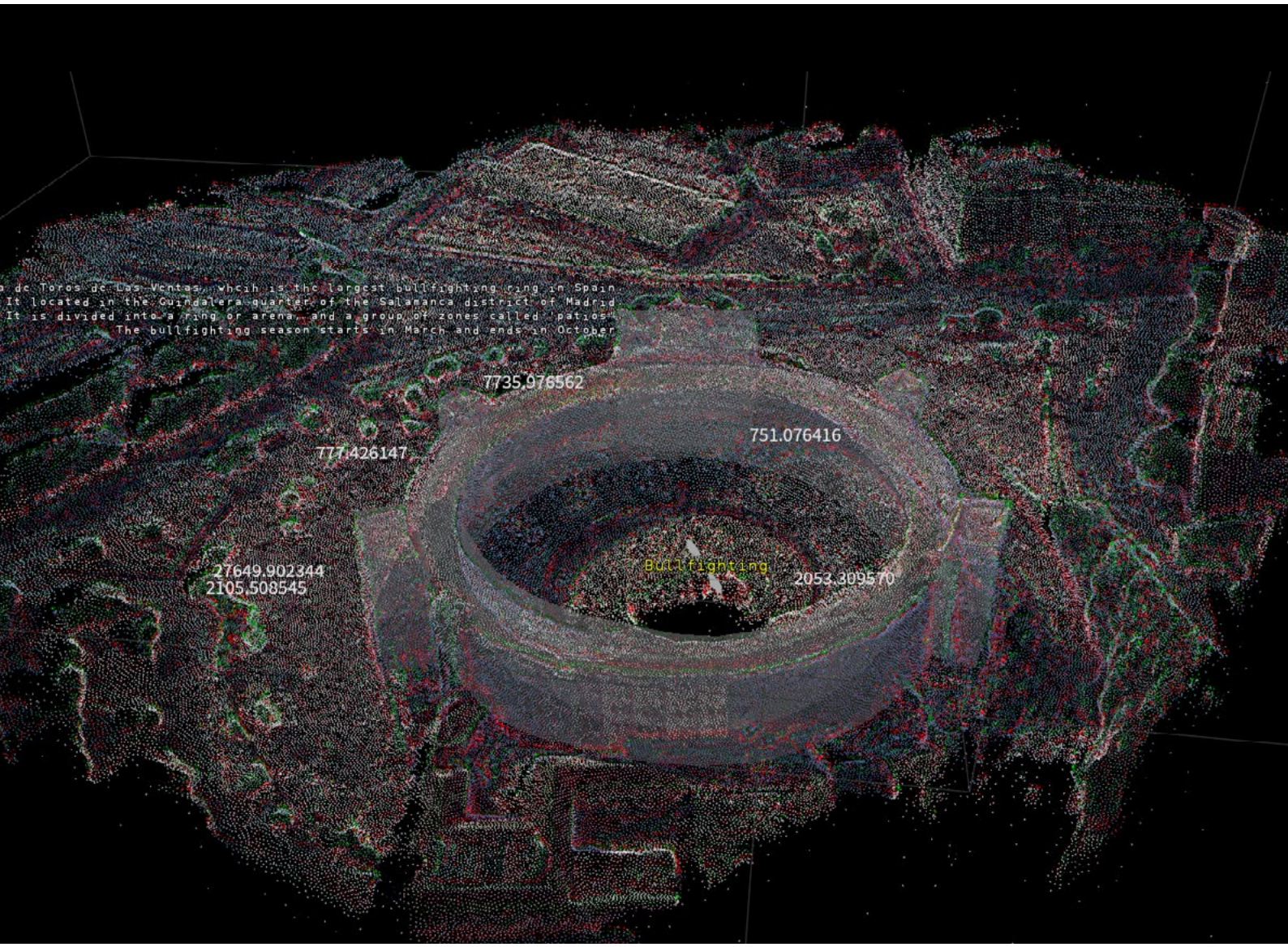
The CAM_1 plays along the walking route. Also, a CAM_2 was set is to show the orthographic view to better see the final effect and details of the whole travel and images. The detailed images will show below.



[CAM_1 path tracking view]



[CAM_2 orthographic view]



HOUDINI 3

3D RECONSTRUCTION

3 Overview

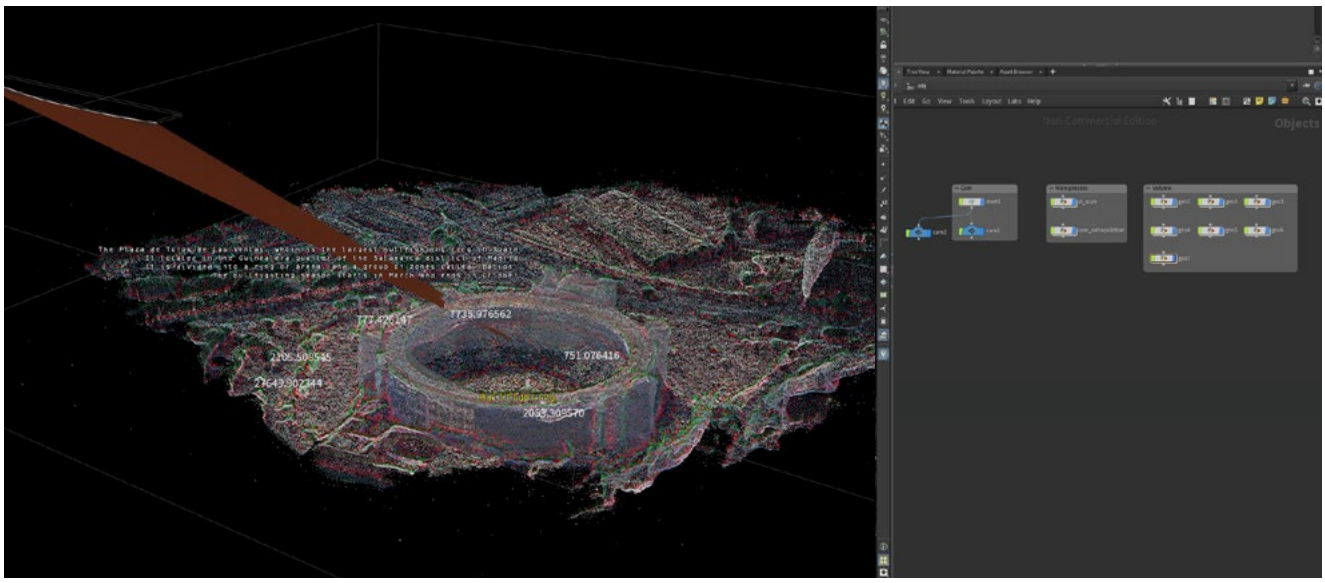
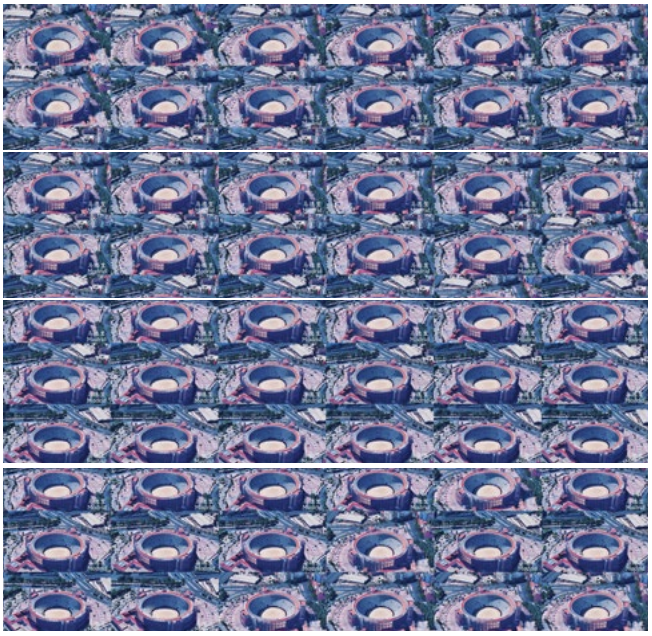


Figure4 : The Overview of 3D Reconstruction

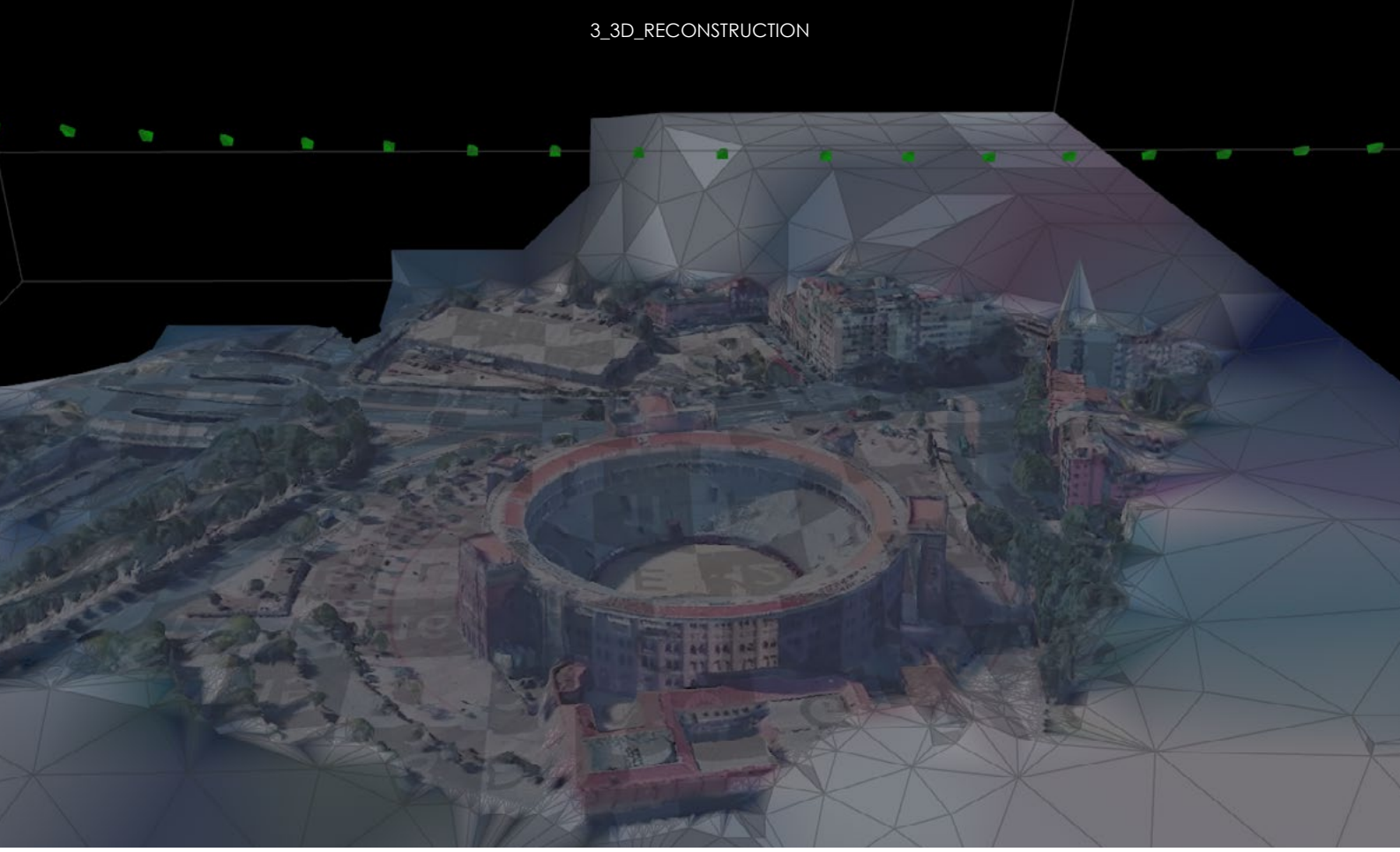
3.1. Generate photogrammetry models

Firstly, the video frames are deconstructed from the found video by code, and the images of these video frames are imported into RealityCapture to generate the reconstructed model. For this part, I chose an overhead video of Madrid's 'plaza de Toros de Las Ventas' from Youtube. Because our design site is located in Madrid; the overhead video contains more information about the site and is more helpful for reconstruction.



[Video frames of the selected site]

[Topview of the selected site]



[The Overview of Import Site Mesh]

3.2. Reconstruct camera path

At first, after reconstructing the 3D model using RealityCapture and exporting the mesh, upload model in houdini, split it into the subsequent processed models and the Camera path to be processed now (shown as figure5).

Then, the next step is creating an attributewangle module and use the code shown in the figure6 to obtain the center point of each camera projectio surface, as well as create corresponding lines for the camera facing the field (shown as figure7).

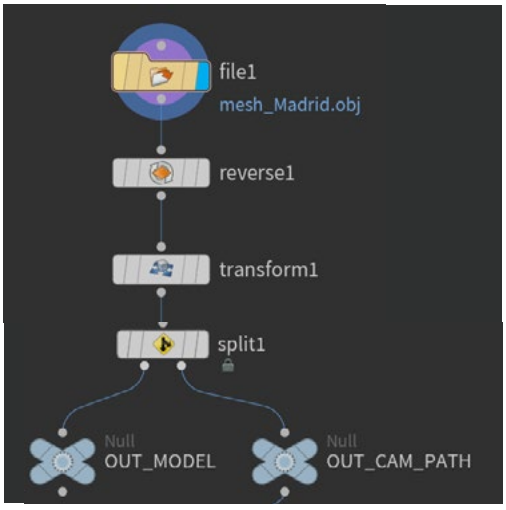


Figure 5

VEXpression

```
for (int pt = 0; pt<npoints(0); pt +=8){


    float mult = 10;

    vector posA = point(0, "P", pt );
    vector posB = point(0, "P", pt +2 );
    vector posC = point(0, "P", pt +4 );
    vector posD = point(0, "P", pt +6 );

    vector normalStart = lerp(posA, posB, 0.5);
    vector normalEnd = lerp(posC, posD, 0.5);

    int startPt = addpoint(0, normalStart);
    setpointattrib(0, "N", startPt, mult *(normalStart - normalEnd), "set");
    //add a point on normalStart, and set a normal vector to that point
    setpointgroup(0, "origin", startPt, 1, "set");
    setpointattrib(0, "up", startPt, set(0,1,0), "set");
}
```

Figure 6

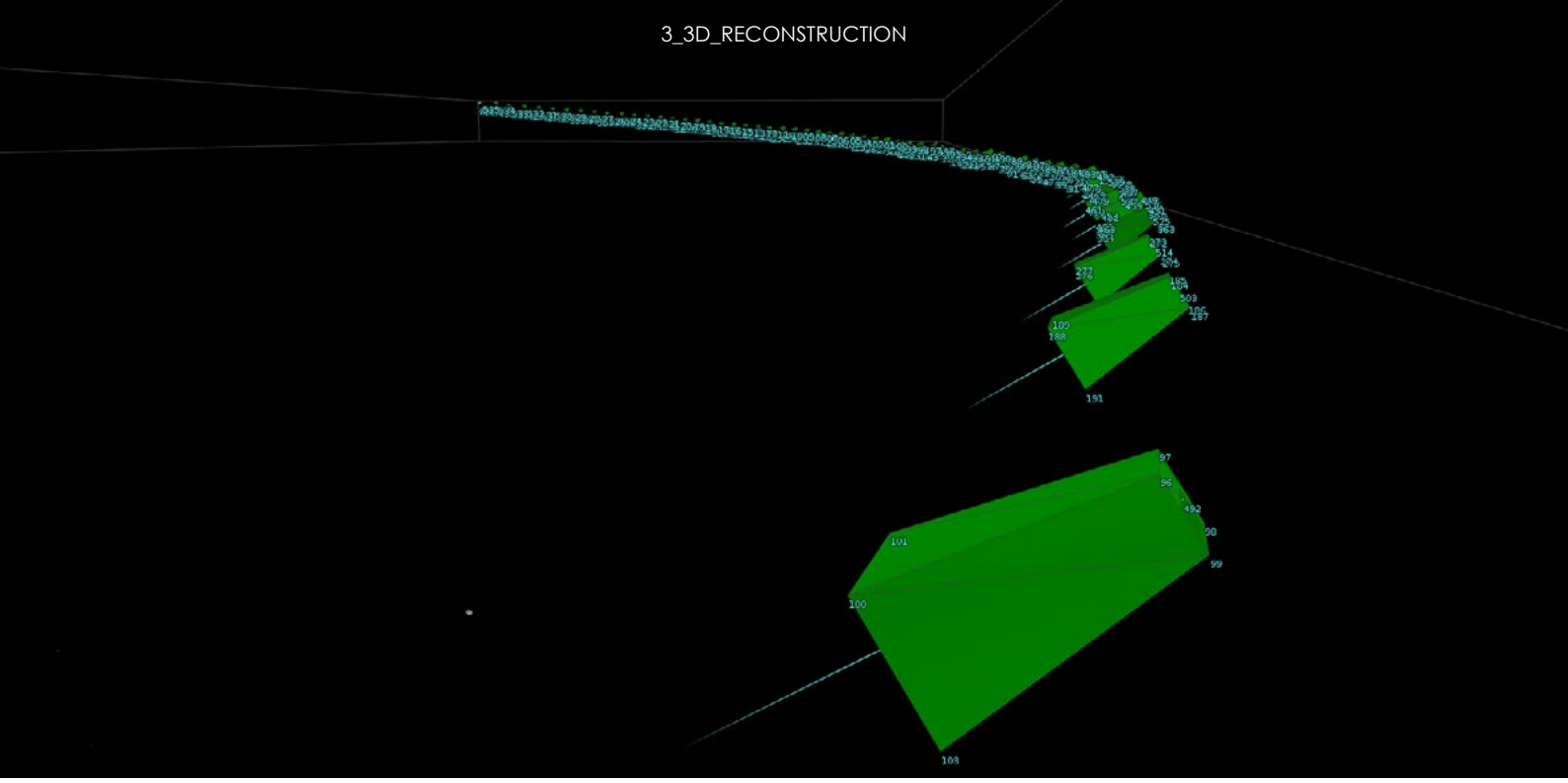
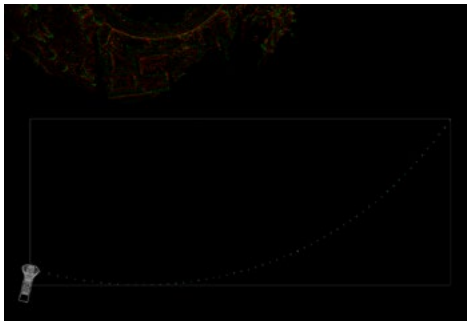
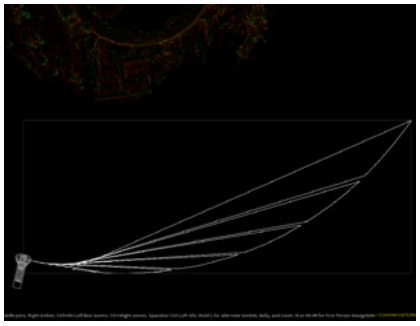


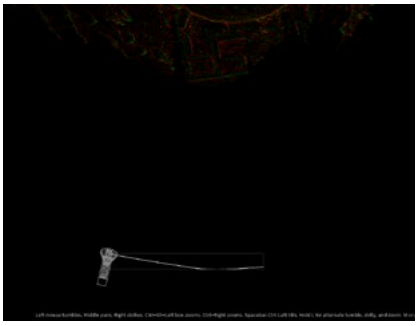
Figure 7

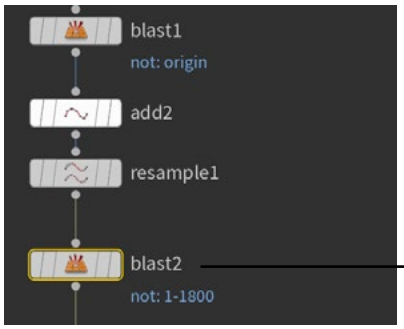
3.2. Reconstruct camera path

This stage is going to reconstructing the camera path. The original video view is surrounging the main building for several time, so that some points' position are overlap. So I need to delete some of the points and shorten the original path to get a new camera path.

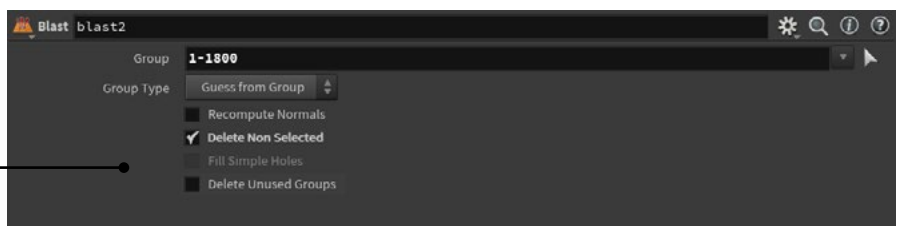








Therefore, I selected non-duplicate points and set them as group 1-1800, and deleted the non selected points through the node blast2.



3.2. Reconstruct camera path

In the previous step, the corresponding lines already created for the camera facing the model. Here I add code in attribwrangle which add Mult, to control the length of those lines (shows in Figure8).

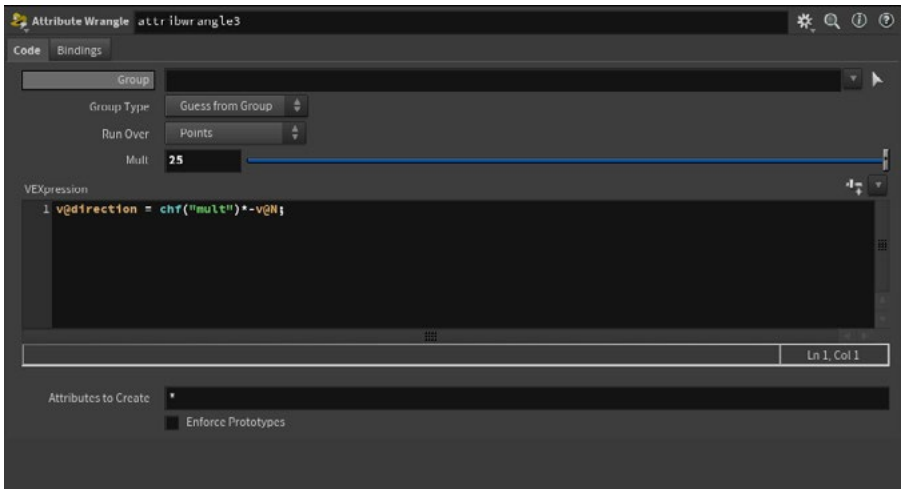
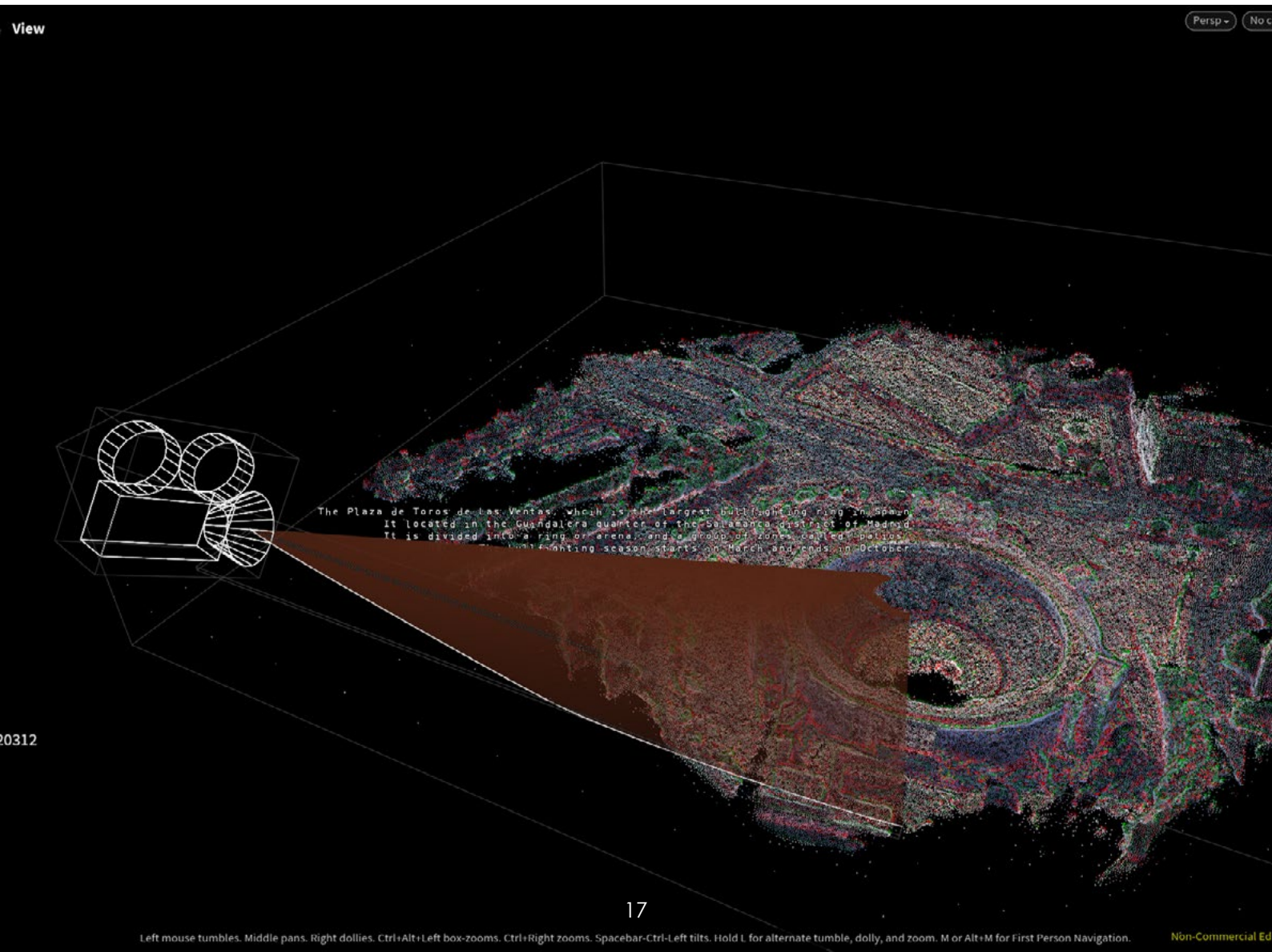


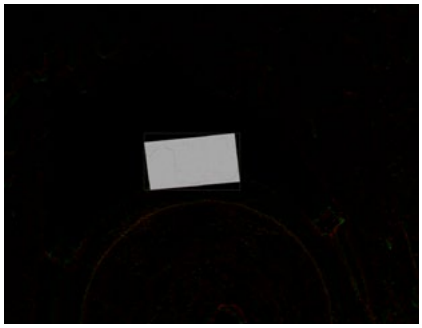
Figure 8

And the final step in the Step of 'Reconstruct camera path' is output the camera path data, which will be used for the CAM part, the CAM would follow the path after this, as the following figure shows.

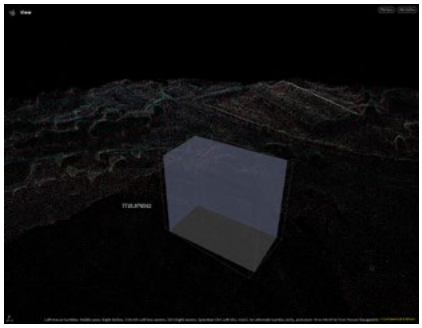


3.3. Volume speculation

Due to the Measure curvature module, 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 two Attribwrangle, one for visualizing volume data, and another attribwrangle for adjusting the transparency of volume objects.

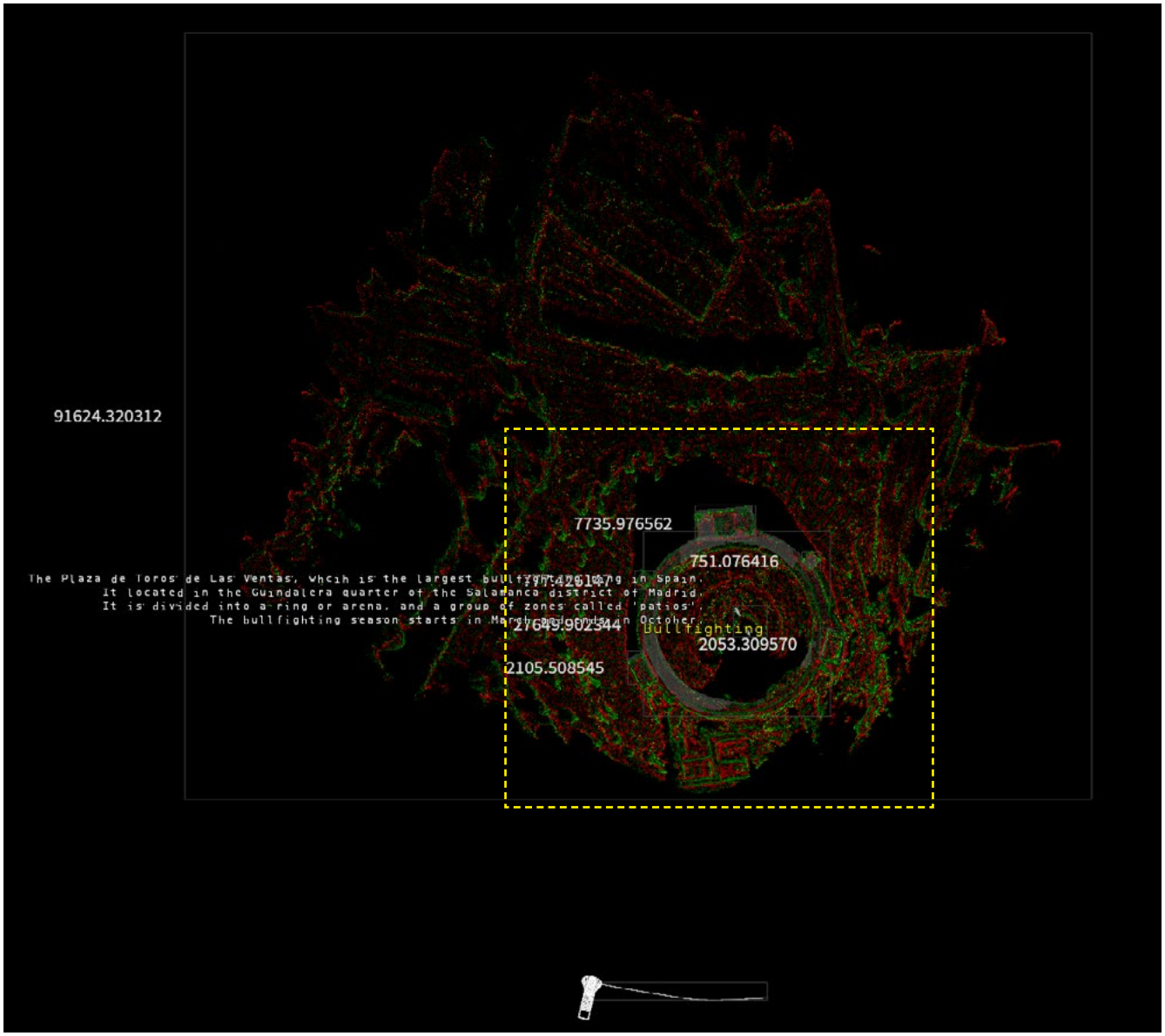


[Draw curve according to point edges]



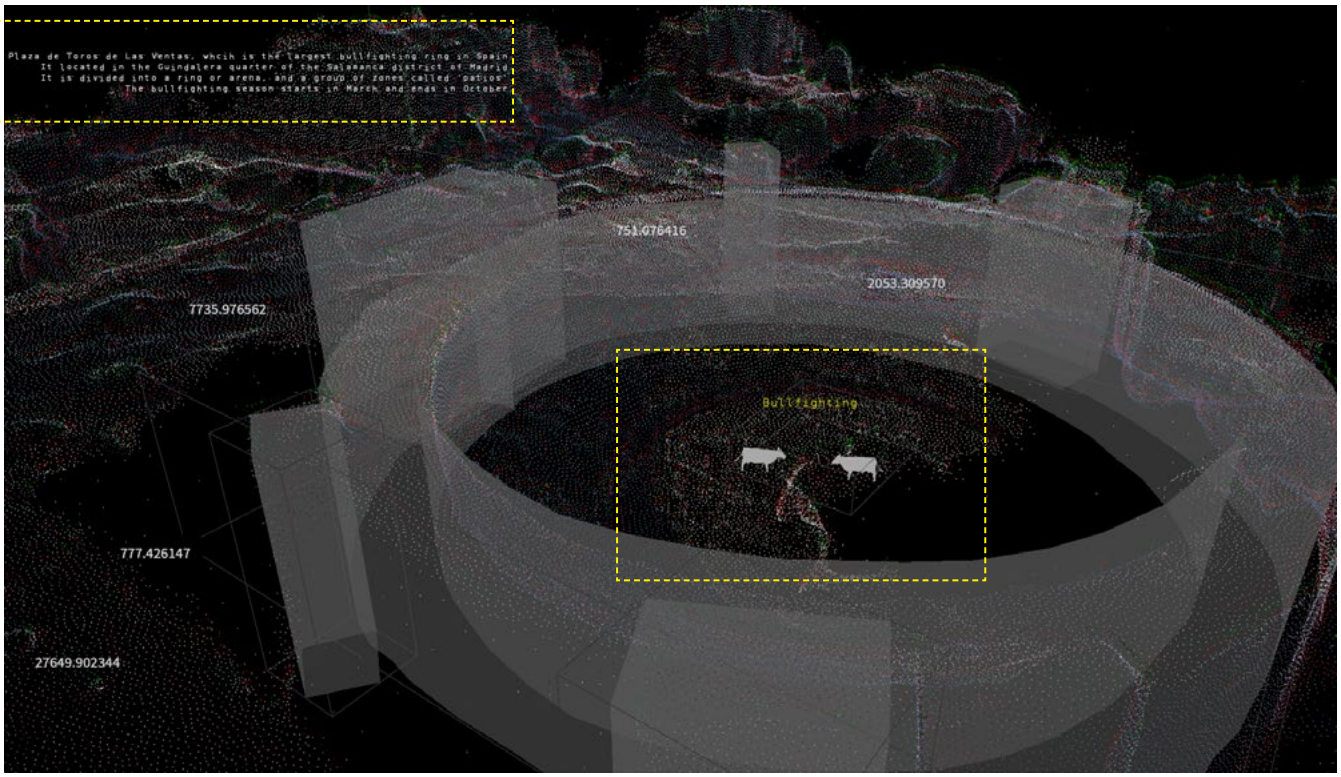
[Extruded geometric block]

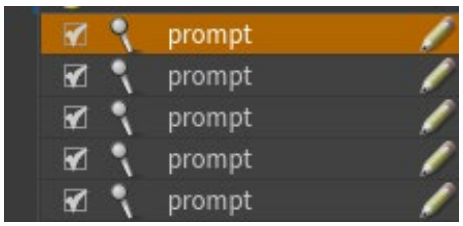
[Extruded geometric block]



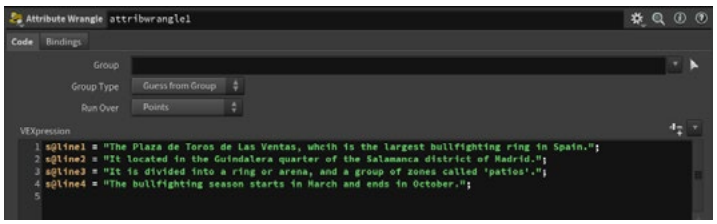
3.4 Adding information to the subject

In the last step, I already add volume data to my objects. In this part I created Markers which named prompt. Adding a attribwrangle moudle including four lines, which would attribute to the Markers. In addition, I also add a extra description for the cows models.





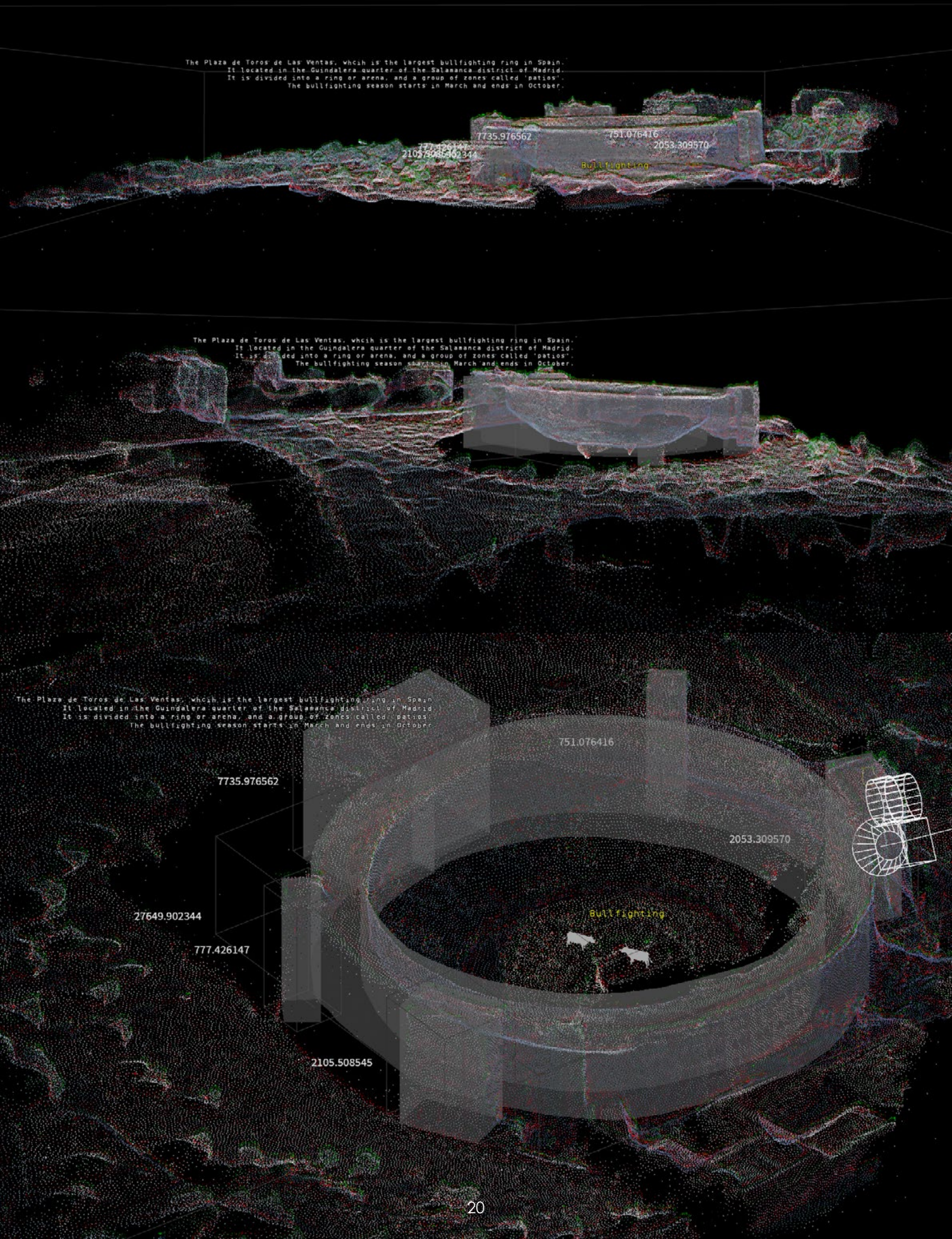
[Showing the promt in Marker]



[Prompt List]

3.5 A Collection Of Images/Renders

I created two camera positions, and in the second position, you can see the first camera.
There are more screenshots showns below:


The Plaza de Toros de Las Ventas, whcih is the largest bullfighting ring in Spain.
It located in the Guindalera quarter of the Salamanca district of Madrid.
It is divided into a ring or arena, and a group of zones called 'patios'.
The bullfighting season starts in March and ends in October.
7735.976562
751.076416
2053.309570
Bullfighting
The Plaza de Toros de Las Ventas, whcih is the largest bullfighting ring in Spain.
It located in the Guindalera quarter of the Salamanca district of Madrid.
It is divided into a ring or arena, and a group of zones called 'patios'.
The bullfighting season starts in March and ends in October.
The Plaza de Toros de Las Ventas, whcih is the largest bullfighting ring in Spain.
It located in the Guindalera quarter of the Salamanca district of Madrid.
It is divided into a ring or arena, and a group of zones called 'patios'.
The bullfighting season starts in March and ends in October.
751.076416
7735.976562
2053.309570
27649.902344
Bullfighting
777.426147
2105.508545

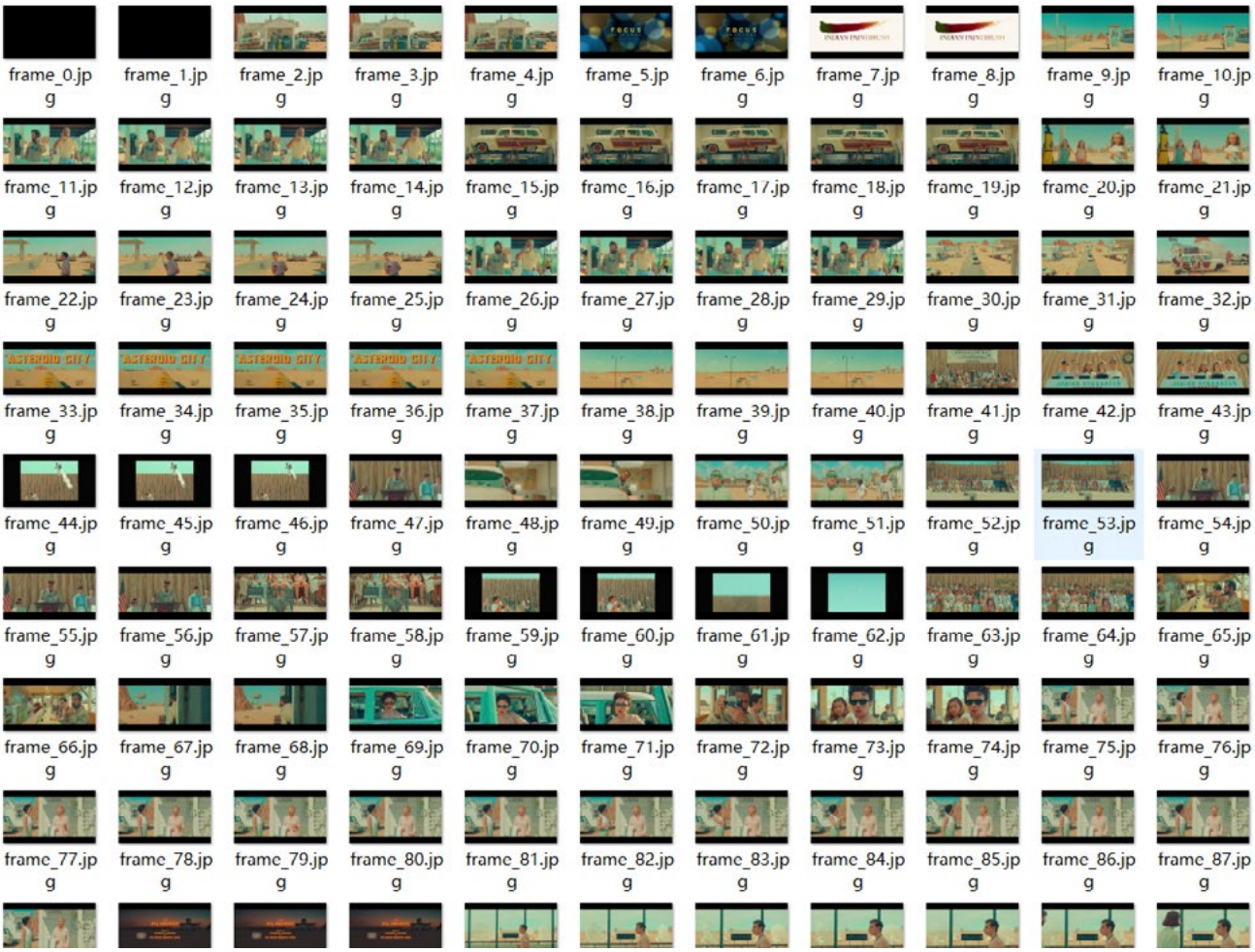
HOUDINI 4

VISUALIZING JSON IN HOUDINI

4.1 Extracting Video Frames

Same as Houdini 3, at the beginning I used python codeto first download the video from Youtube based on the url and split the video into video frames, in this part I split the video into 280 frames in total, which corresponds to 12 subtitles.

The selection of the video was based on our project, our design site was an abandoned site in Madrid that was in a desert-like area, so I chose a movie with a lot of desert scenes, from Wes Anderson's 'Asteroid City '.



[Frames Image Folder]

[Divided 12 subtitles Subfolder]

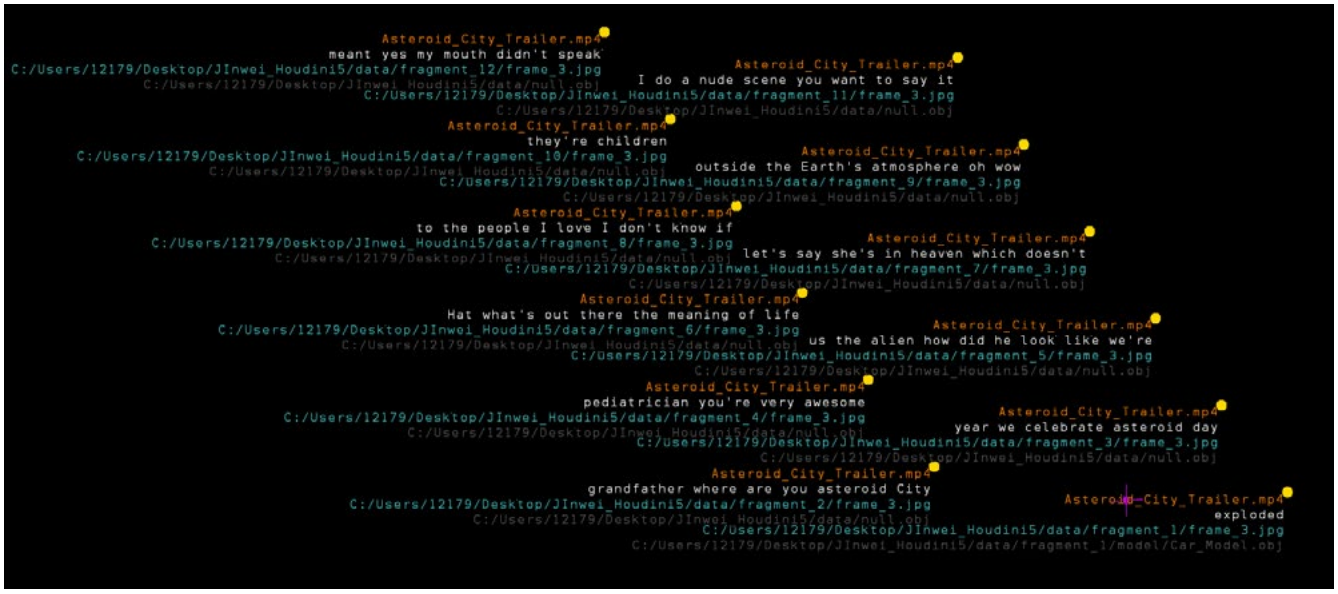
4.1.2 JSON file

Next, generate a JSON file from the pickle file using code. Here, I also attempted to manually input video information into a JSON file. The following image is the result of manual input. The information is mainly divided into: Film ID; Paragraphs; Frame_n; Corresponding model; Time; Film path...


```
{
"folder": [
      {
    "paragraph" : "exploded",
    "filmID" : "Asteroid_City.mp4",
    "image_name" : "frame",
    "model_path" : "Car_Model.obj",
    "image_n" : 19,
    "time" : 7,
    "film_path" : "Asteroid_City_Trailer.mp4"
      },

      {
    "paragraph" : "grandfather where are you asteroid City",
    "filmID" : "Asteroid_City.mp4",
    "image_name" : "frame",
    "model_path" : "",
    "image_n" : 20,
    "time" : 14,
    "film_path" : "Asteroid_City_Trailer.mp4"
      },

      {
    "paragraph" : "year we celebrate asteroid day",
    "filmID" : "Asteroid_City.mp4",
    "image_name" : "frame",
    "model_path" : "",
    "image_n" : 10,
    "time" : 25,
    "film_path" : "Asteroid_City_Trailer.mp4"
      },

      {
    "paragraph" : "pediatrician you're very awesome",
    "filmID" : "Asteroid_City.mp4",
    "image_name" : "frame",
    "model_path" : "",
    "image_n" : 10,
    "time" : 44,
    "film_path" : "Asteroid_City_Trailer.mp4"
      },

```

4.2 Extracting video frames

In Houdini, creating a Python modules and import Json files. 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 column. In addition, writing code to define and load the infomation of json file (Film ID; Paragraphs; Frame_n...)



[Frames data reading from the json]

4.3 Creating camera path

Similar to the method in Houdini 3: 3D Reconstrucion, to create a camera action trajectory. However, the code this time was used to set the route between two points lines.

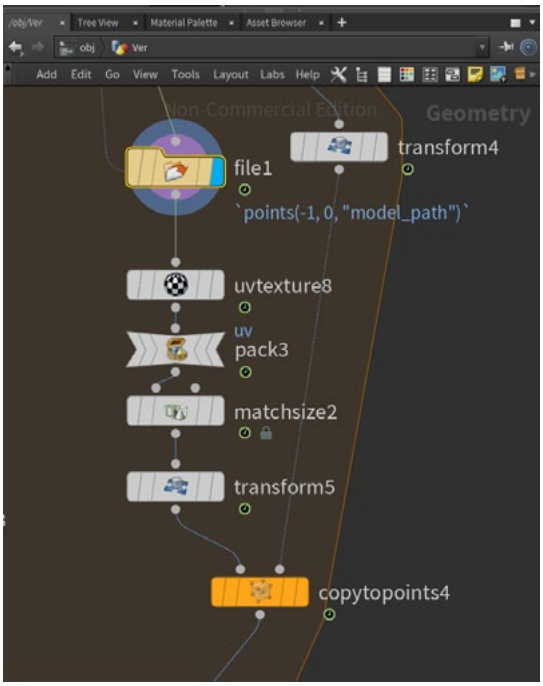
[The camera position]

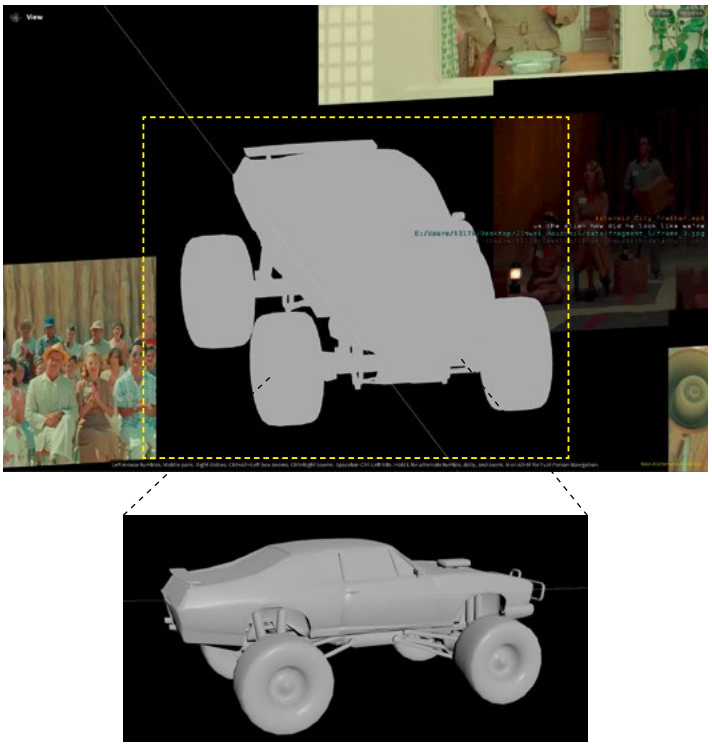
4.4 Input film frames(images)

Setting a grid, the UV texture is used to ensure the correct display of images on the grid. The Pack module is used to make the information more clean (orignally, four points of a grid have the information). Then set Quickmaterial to load the frame images.

4.5 Model input

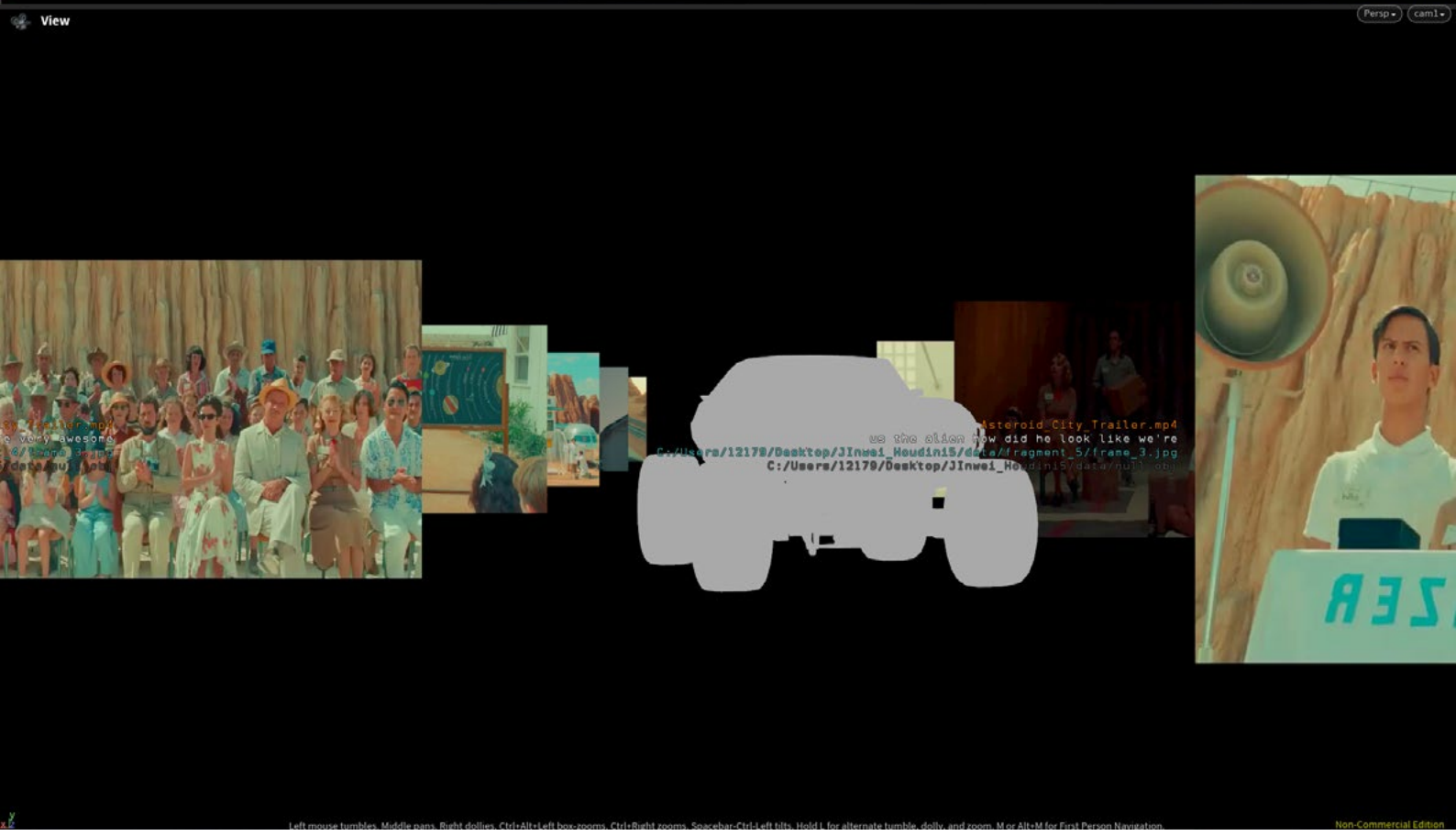
In this part, I also add a model by File module. Matchsize and Transform were set to adject the position of model. and the catch those points. To more fit the film clip I selected, I choose a classic car which is fit the movie scene prop as the model to input



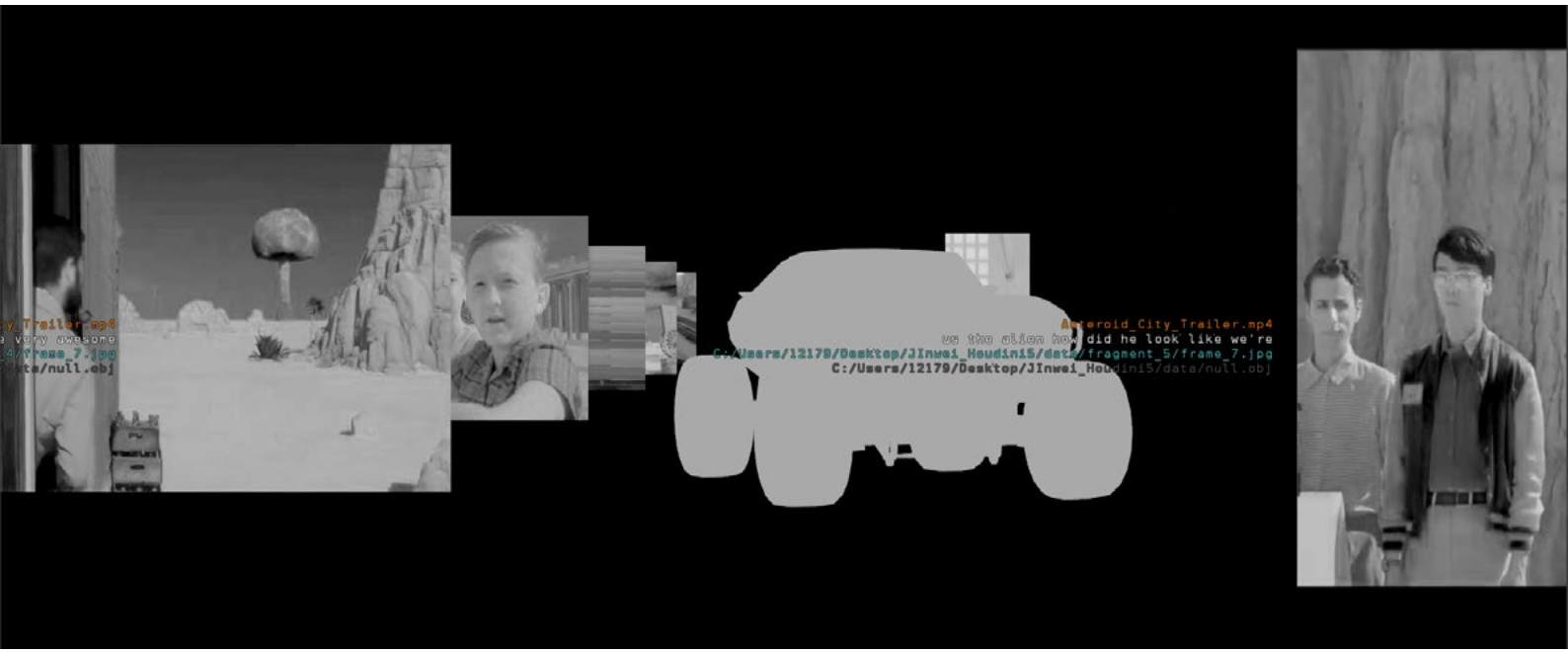


4.6 A Collection Of Images/Renders

Setting another CAM to see the overview. A series of images rendered, and the results are showing below.



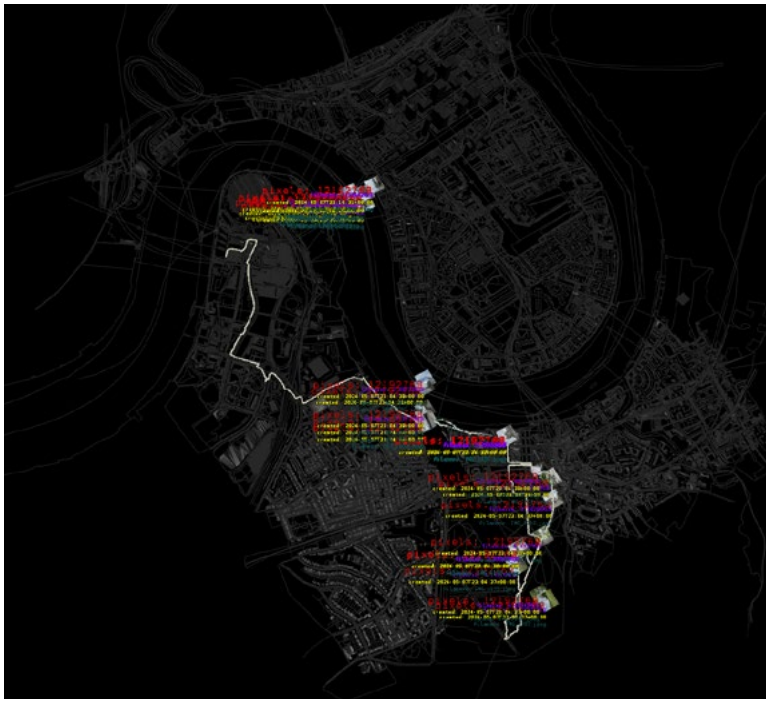
4.6 A Collection Of Images/Renders



5 HOUDINI REVIEW AND REFLECTION

Through the series of Houdini sessions, I learned quite a lot operation of Houdini, and have a preliminary understanding and attempt at the application of Houdini. This assignment was divided into four parts. Through the basic training in the first part, Houudini_Foundamentals, I mastered the adjustment, application and matching of some basic nodes, and understood the basic logic of Houdini to process data and generate models. For the third part and the fourth part I think the most important thing about the 3D reconstruction and the json visualization is that it allowed me to understand and try the camera view adjustment and tracking more skillfully, so that I can build many visual scenes in Houdini.

However, what I struggle with the most is the second part of visualizing gps and image metadata. One is due to the unavailable of google timeline, so I need to use another travelling tracking software to get my travel gpx file. Another difficulty is the setting of longitude. Since the visualising area I selected is London, where is really close to the 0 longitude. What's more special is that in my first attempt, the path I recorded passed through the Greenwich Observatory, where the prime meridian is located, which means that my formation spanned the eastern and western hemispheres. This caused the path and map data I recorded to be swapped with the actual data (as it is shown in images below). The solution was finally solved after adjusting the longitude value in the Python node, but it was this special example that gave me a deeper understanding of the content of this session.



[the path and mapping in Houdini]

[the actual mapping of Greenwich]

SKILLS CLASS RC11 - 23/24

COMPLEXITY ASSIGNMENT

Submitter: 23135525

Exercise One

1. Implement this algorithm in Python.

Use the NumPy ndarray object for your matrices;

Algorithm 1: Square-Matrix-Multiply

```
SMM(A,B):
n = nr of rows of A
let C be a new n × n matrix
for i = 1 to n do
    for j = 1 to n do
        c_ij = 0 for k = 1 to n do
            c_ij = c_ij + a_ik · b_kj
return C
```

```python
import time
import numpy as np


def multiply_matrices(A, B):
    dimension = A.shape[0]
    Result = np.zeros((dimension, dimension), dtype=int)
    for i in range(dimension):
        for j in range(dimension):
            Result[i, j] = sum(A[i, k] * B[k, j] for k in range(dimension))
    return Result
```

Implementation and Comparison

```python
Matrix_A = np.array([[2, 4], [6, 8]])
Matrix_B = np.array([[1, 3], [5, 7]])
Matrix_C = np.array([[3, 2, 7, 5], [10, 7, 3, 6], [5, 1, 3, 6], [2, 3, 8, 7]])
Matrix_D = np.array([[i for i in range(1, 8)] for _ in range(7)])

print("Results using NumPy's matmul:")
start = time.time()
print('2 x 2 Matrix:', np.matmul(Matrix_A, Matrix_B))
print('4 x 4 Matrix:', np.matmul(Matrix_C, Matrix_C))
print('5 x 5 Matrix:', np.matmul(Matrix_D, Matrix_D))
run_time = time.time() - start
print('time: ', run_time)

start = time.time()
print("Results using custom matrix multiplication function:")
print('2 x 2 Matrix:', multiply_matrices(Matrix_A, Matrix_B))
print('4 x 4 Matrix:', multiply_matrices(Matrix_C, Matrix_C))
print('5 x 5 Matrix:', multiply_matrices(Matrix_D, Matrix_D))
run_time = time.time() - start
print('time: ', run_time)

print('Chained Matrix Multiplication:')
print(np.matmul(np.matmul(Matrix_A, Matrix_B), Matrix_A))
print(multiply_matrices(multiply_matrices(Matrix_A, Matrix_B), Matrix_A))
```

```
Results using NumPy's matmul:
2 x 2 Matrix: [[22 34]
 [46 74]]
4 x 4 Matrix: [[ 74  42  88 104]
 [127  90 148 152]
 [ 52  38  95  91]
 [ 90  54 103 125]]
5 x 5 Matrix: [[ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]]
time:  0.0
Results using custom matrix multiplication function:
2 x 2 Matrix: [[22 34]
 [46 74]]
4 x 4 Matrix: [[ 74  42  88 104]
 [127  90 148 152]
 [ 52  38  95  91]
 [ 90  54 103 125]]
5 x 5 Matrix: [[ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]
 [ 28  56  84 112 140 168 196]]
time:  0.0
Chained Matrix Multiplication:
[[248 360]
 [536 776]]
[[248 360]
 [536 776]]
```

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 $O(n3)$

Explanation:

Initializing the result matrix: This step takes constant time $O(1)$ as it involves creating a matrix of size $n \times n$.

For the 'Nested Loops' for matrix multiplication:

1. There are two nested loops iterating over each element of the resulting matrix C,each running from 0 to $n-1$.

2.Inside the nested loops,there's a summation operation which runs n times for each element of C.

3.Thus,the total number of operations inside the nested loops is $n \times n \times n = n^3$.

Therefore, the dominant factor in the time complexity is the cubic term n^3.

Thus, the overall time complexity of the algorithm is $O(n^3)$.

Exercise Two

1. Describe and explain the algorithm. It should contain at least the following:

Algorithm 2: Square-Matrix-Multiply-Recursive

```
SMMRec(A,B):
n = nr of rows of A
let C be a new n × n matrix
if n == 1 then
    c11 = a11 · b11
else
    quarter matrices A, B, and C
    C11 = SMMRec(A11, B11) + SMMRec(A12, B21)
    C12 = SMMRec(A11, B12) + SMMRec(A12, B22)
    C21 = SMMRec(A21, B11) + SMMRec(A22, B21)
    C22 = SMMRec(A21, B12) + SMMRec(A22, B22)
return C
```

- 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).

Explanation:

Recursiveness:

algorithm is recursive, calling itself to perform matrix multiplication on smaller sub-problems. It handles quarter-sized sub-matrices until reaching the base case, where the matrix size reduces to 1x1, simplifying multiplication to a single-element operation.

The recursion divides each matrix into four equal-sized sub-matrices, halving the size with each recursive call until reaching the base case.

Divide-and-Conquer:

Divide:

The algorithm splits the problem of multiplying two n x n matrices into smaller problems by dividing each matrix (A and B) into four sub-matrices.

Conquer:

It recursively solves these smaller multiplication problems, recursively multiplying the sub-matrices until reaching the base case of single-element matrices.

Combine:

After the recursive multiplications, the algorithm combines the results to form the final matrix by concatenating sub-matrix results. The resultant matrix is constructed by concatenating sub-matrices according to their positions.

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
import numpy as np

def partition_matrix(matrix):
    n = len(matrix)
    mid = n // 2
    top_left = [row[:mid] for row in matrix[:mid]]
    top_right = [row[mid:] for row in matrix[:mid]]
    bottom_left = [row[:mid] for row in matrix[mid:]]
    bottom_right = [row[mid:] for row in matrix[mid:]]
    return top_left, top_right, bottom_left, bottom_right

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

def combine_matrices(top_left, top_right, bottom_left, bottom_right):
    top_half = [left + right for left, right in zip(top_left, top_right)]
    bottom_half = [left + right for left, right in zip(bottom_left, bottom_right)]
    return top_half + bottom_half

def recursive_multiply_matrices(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        A11, A12, A21, A22 = partition_matrix(A)
        B11, B12, B21, B22 = partition_matrix(B)
        M1 = add_matrices(recursive_multiply_matrices(A11, B11), recursive_multiply_matrices(A12, B21))
        M2 = add_matrices(recursive_multiply_matrices(A11, B12), recursive_multiply_matrices(A12, B22))
        M3 = add_matrices(recursive_multiply_matrices(A21, B11), recursive_multiply_matrices(A22, B21))
        M4 = add_matrices(recursive_multiply_matrices(A21, B12), recursive_multiply_matrices(A22, B22))
        return combine_matrices(M1, M2, M3, M4)

```

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

print("Results using recursive matrix multiplication:")
print('2 x 2 Matrix:', recursive_multiply_matrices(Matrix_E, Matrix_F))
print('4 x 4 Matrix:', recursive_multiply_matrices(Matrix_G, Matrix_G))

print('Chained Recursive Matrix Multiplication (2 x 2 Matrix):')
chained_result = recursive_multiply_matrices(recursive_multiply_matrices(Matrix_E, Matrix_F), Matrix_E)
print(chained_result)
```

```
Results using recursive matrix multiplication:
2 x 2 Matrix: [[20, 28], [52, 76]]
4 x 4 Matrix: [[90, 100, 110, 120], [202, 228, 254, 280], [314, 356, 398, 440], [426, 484, 542, 600]]
Chained Recursive Matrix Multiplication (2 x 2 Matrix):
[[160, 256], [432, 688]]
```

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

```python
import numpy as np
import time

Large_Matrix = np.array([[i for i in range(1, 65)] for _ in range(64)])

start_time = time.time()
result_basic = multiply_matrices(Large_Matrix, Large_Matrix)
basic_mul_time = (time.time() - start_time)

start_time = time.time()
result_recursive = recursive_multiply_matrices(Large_Matrix.tolist(), Large_Matrix.tolist())
recursive_mul_time = (time.time() - start_time)

print(f"Basic Multiply Time: {basic_mul_time:.4f} s, Recursive Multiply Time: {recursive_mul_time:.4f} s")
```

```
Basic Multiply Time: 0.0715 s, Recursive Multiply Time: 0.4071 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:

The algorithm divides each n x n matrix into four $n/2 \times n/2$ sub-matrices. This step is simple,involving only indexing operations with a constant time complexity $O(1)$.

Conquer:

The conquer step involves eight recursive calls to multiply the smaller $n/2 \times n/2$ sub-matrices until reaching the base case of 1 x 1 matrices, which are directly multiplied in constant time.

Combine:

The results of the recursive multiplications are combined into the final matrix C using matrix addition, which has a complexity of $O(n^2)$.

Overall Complexity:

The recurrence relation for the algorithm is $T(n)=8T(n/2)+O(n^2)$. Solving this gives $T(n)=O(n^3)$, the same as the iterative approach but foundational for more advanced algorithms like Strassen's method. Notably, it paves the way for Strassen's algorithm, which significantly reduces the complexity to approximately $O(n^{\log 27})$,enhancing efficiency for large matrices.

Exercise Three

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

Matrix addition and subtraction are straightforward operations for matrices of size $n \times n$. Each element from one matrix is added or subtracted from the corresponding element of the other matrix, resulting in n^2 total operations, leading to a computational complexity of $O(n^2)$.

Matrix multiplication is more complex. For each element in the resulting $n \times n$ matrix, n multiplications and $n-1$ additions are needed. With n^2 elements to compute, the straightforward (naive) approach has a complexity of $O(n^3)$. Each element requires $2n-1$ operations, which aggregates to a much higher overall computational effort compared to addition or subtraction.

Implement and test the algorithm

Algorithm 3: Square-Matrix-Multiply-Recursive

Strassen(A,B):
n = nr of rows of A
let C be a new $n \times n$ matrix
if $n == 1$ **then**
  $c_{11} = a_{11} \cdot b_{11}$
else
  quarter matrices A, B, and C
  $S_1 = B_{12} - B_{22}$
  $S_2 = A_{11} + A_{12}$
  $S_3 = A_{21} + A_{22}$
  $S_4 = B_{21} - B_{11}$
  $S_5 = A_{11} + A_{22}$
  $S_6 = B_{11} + B_{22}$
  $S_7 = A_{12} - A_{22}$
  $S_8 = B_{21} + B_{22}$
  $S_9 = A_{11} - A_{21}$
  $S_{10} = B_{11} + B_{12}$
  $P_1 = \text{Strassen}(A_{11}, S_1)$
  $P_2 = \text{Strassen}(S_2, B_{22})$
  $P_3 = \text{Strassen}(S_3, B_{11})$
  $P_4 = \text{Strassen}(A_{22}, S_4)$
  $P_5 = \text{Strassen}(S_5, S_6)$
  $P_6 = \text{Strassen}(S_7, S_8)$
  $P_7 = \text{Strassen}(S_9, S_{10})$
  $C_{11} = P_5 + P_4 - P_2 + P_6$
  $C_{12} = P_1 + P_2$
  $C_{21} = P_3 + P_4$
  $C_{22} = P_5 + P_1 - P_3 - P_7$
return C

```python
import numpy as np

def enhanced_matrix_split(matrix):
    size = len(matrix)
    mid_point = size // 2
    quadrant1 = [row[:mid_point] for row in matrix[:mid_point]]
    quadrant2 = [row[mid_point:] for row in matrix[:mid_point]]
    quadrant3 = [row[:mid_point] for row in matrix[mid_point:]]
    quadrant4 = [row[mid_point:] for row in matrix[mid_point:]]
    return quadrant1, quadrant2, quadrant3, quadrant4

def matrix_operation_add(submatrix1, submatrix2):
    return [[submatrix1[i][j] + submatrix2[i][j] for j in range(len(submatrix1))] for i in range(len(submatrix1))]

def matrix_operation_subtract(submatrix1, submatrix2):
    return [[submatrix1[i][j] - submatrix2[i][j] for j in range(len(submatrix1))] for i in range(len(submatrix1))]

def assemble_matrix(q1, q2, q3, q4):
    upper_half = [left + right for left, right in zip(q1, q2)]
    lower_half = [left + right for left, right in zip(q3, q4)]
    return upper_half + lower_half

def optimized_strassen_matrix_multiply(A, B):
    n = len(A)
    if n <= 2:  # Use the direct method for small matrices
        return recursive_multiply_matrices(A, B)
    else:
        mid = n // 2
        A11, A12, A21, A22 = enhanced_matrix_split(A)
        B11, B12, B21, B22 = enhanced_matrix_split(B)

        S1 = matrix_operation_add(A11, A22)
        S2 = matrix_operation_add(B11, B22)
        S3 = matrix_operation_add(A21, A22)
        S4 = B11
        S5 = A11
        S6 = matrix_operation_subtract(B12, B22)
        S7 = A22
        S8 = matrix_operation_subtract(B21, B11)
        S9 = matrix_operation_add(A11, A12)
        S10 = B22
        P1 = optimized_strassen_matrix_multiply(S1, S2)
        P2 = optimized_strassen_matrix_multiply(S3, S4)
        P3 = optimized_strassen_matrix_multiply(S5, S6)
        P4 = optimized_strassen_matrix_multiply(S7, S8)
        P5 = optimized_strassen_matrix_multiply(S9, S10)
        Q1 = matrix_operation_add(matrix_operation_subtract(matrix_operation_add(P1, P4), P5), P2)
        Q2 = matrix_operation_add(P3, P5)
        Q3 = matrix_operation_add(P2, P4)
        Q4 = matrix_operation_subtract(matrix_operation_subtract(matrix_operation_add(P1, P3), P2), P5)

        return assemble_matrix(Q1, Q2, Q3, Q4)
```

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

print("Results using Strassen's algorithm:")
print('2 x 2 Matrix:', optimized_strassen_matrix_multiply(Matrix_H, Matrix_I))
print('4 x 4 Matrix:', optimized_strassen_matrix_multiply(Matrix_J, Matrix_J))
```

```
Results using Strassen's algorithm:
2 x 2 Matrix: [[20, 28], [52, 76]]
4 x 4 Matrix: [[604, 688, 110, 120], [764, 880, 254, 280], [314, 356, 136, 136], [426, 484, 72, 72]]
```

Compare the practical speed with the recursive algorithm

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

start_time = time.time()
result_from_basic = multiply_matrices(Performance_Matrix, Performance_Matrix)
time_for_basic = (time.time() - start_time)

start_time = time.time()
result_from_recursive = recursive_multiply_matrices(Performance_Matrix.tolist(), Performance_Matrix.tolist())
time_for_recursive = (time.time() - start_time)

print(f"Time for Basic Multiplication: {time_for_basic:.4f} s, Time for Recursive Multiplication: {time_for_recursive:.4f} s")
```

```
Time for Basic Multiplication: 0.5441 s, Time for Recursive Multiplication: 3.5106 s
```

2.Do a complexity analysis of the Strassen algorithm.

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

Standard Recursive Matrix Multiplication:

The traditional recursive approach to matrix multiplication has a complexity recurrence of $T(n)=8T(n/2)+O(n^2)$, which resolves to $O(n^3)$ according to the Master Theorem.

Strassen's Algorithm:

Strassen's algorithm reduces the number of recursive matrix multiplications from 8 to **7** by rearranging and combining operations ingeniously, adding more additions and subtractions but fewer multiplications. This results in a new complexity recurrence:

$T(n)=7T(n/2)+O(n^2)$.

Complexity Analysis:

By applying the Master Theorem to Strassen's recurrence, the complexity becomes $T(n)=O(n\log 27) \approx O(n2.81)$.This optimization demonstrates a significant reduction from the $O(n^3)$ complexity of the standard recursive approach, underscoring the efficiency gains achievable through Strassen's method.