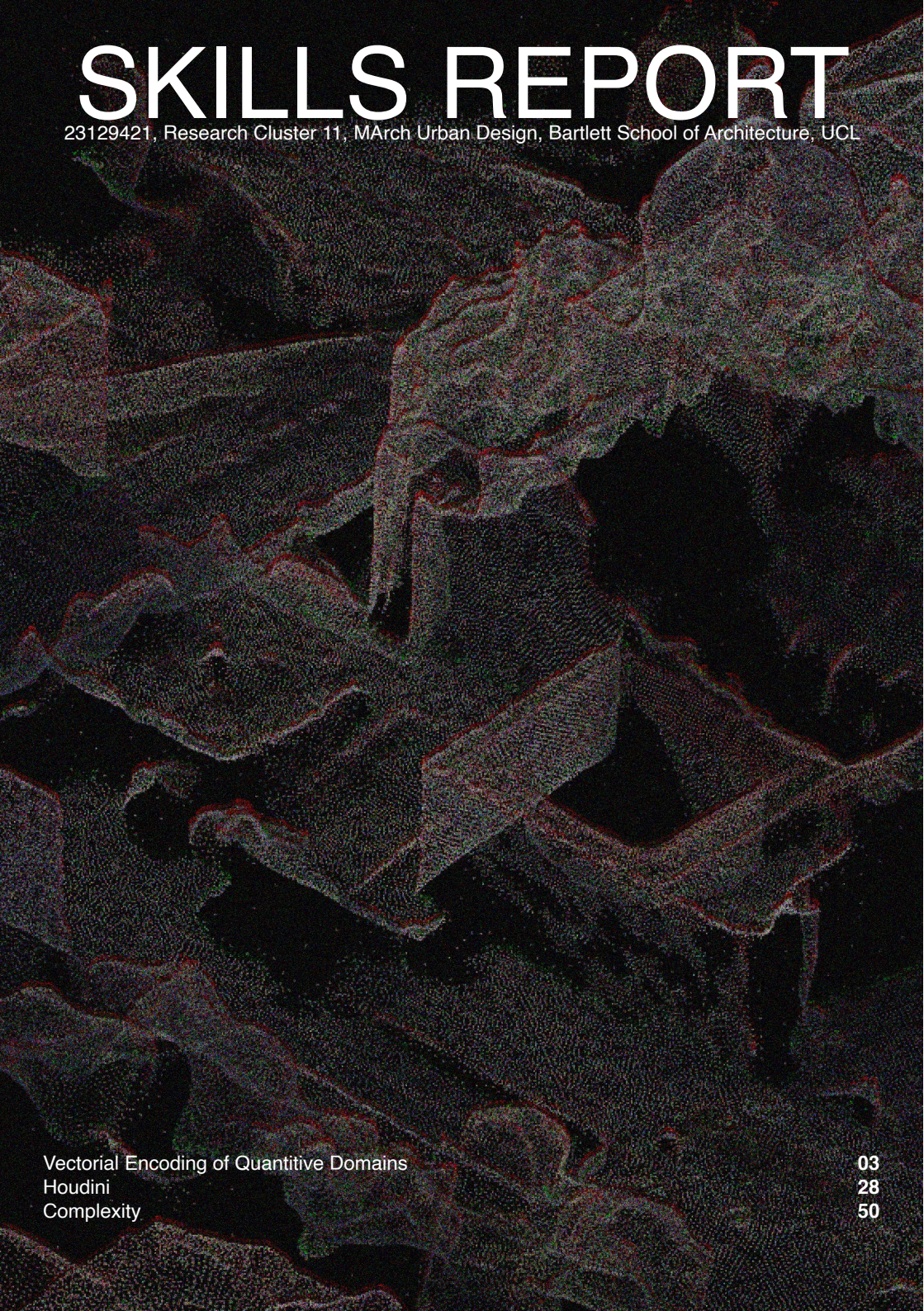
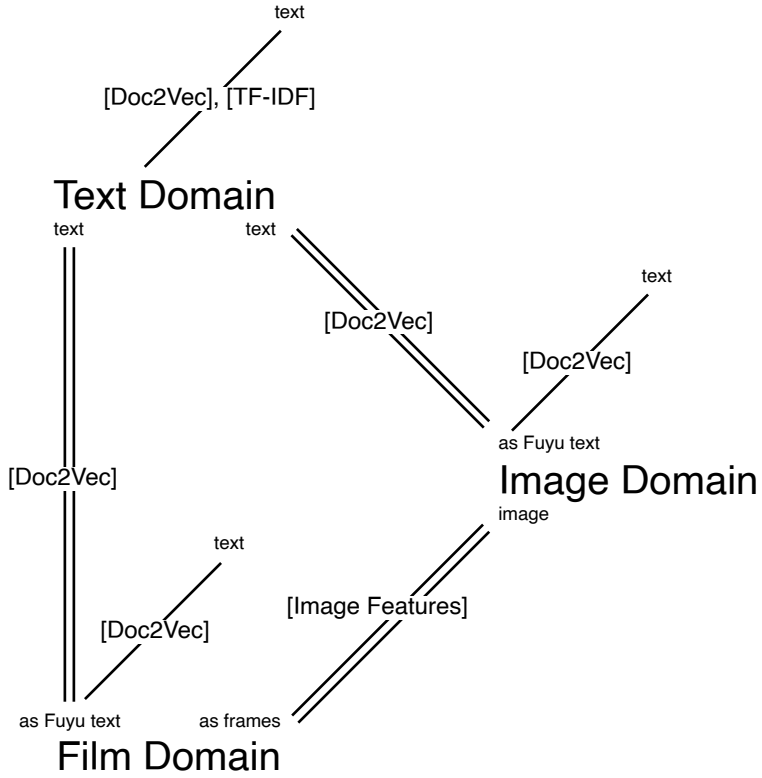
SKILLS REPORT

23129421, Research Cluster 11, MArch Urban Design, Bartlett School of Architecture, UCL



VECTORIAL ENCODING OF QUANTITIVE DOMAINS

01



System Description

In this experiment, a kind of two-way arbitrary matching of pictures, videos, and films is established.
Also, a random text query entered externally can be found in the three domains.

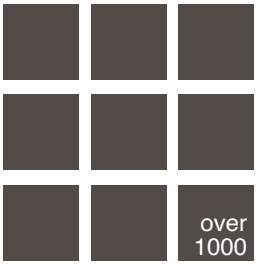
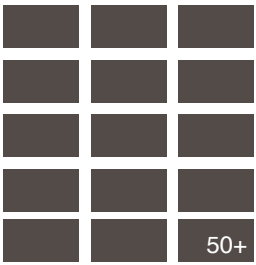


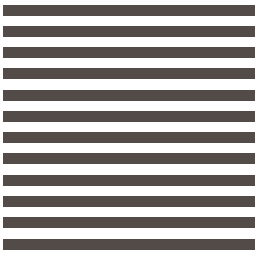
Image domain

Keyframing a Madrid walking tour video (100min total in length) by 6 seconds. Over 1000 keyframes generated as image domain.



Film domain

Over 50 Independent Films batch downloaded using Pytube from Youtube.

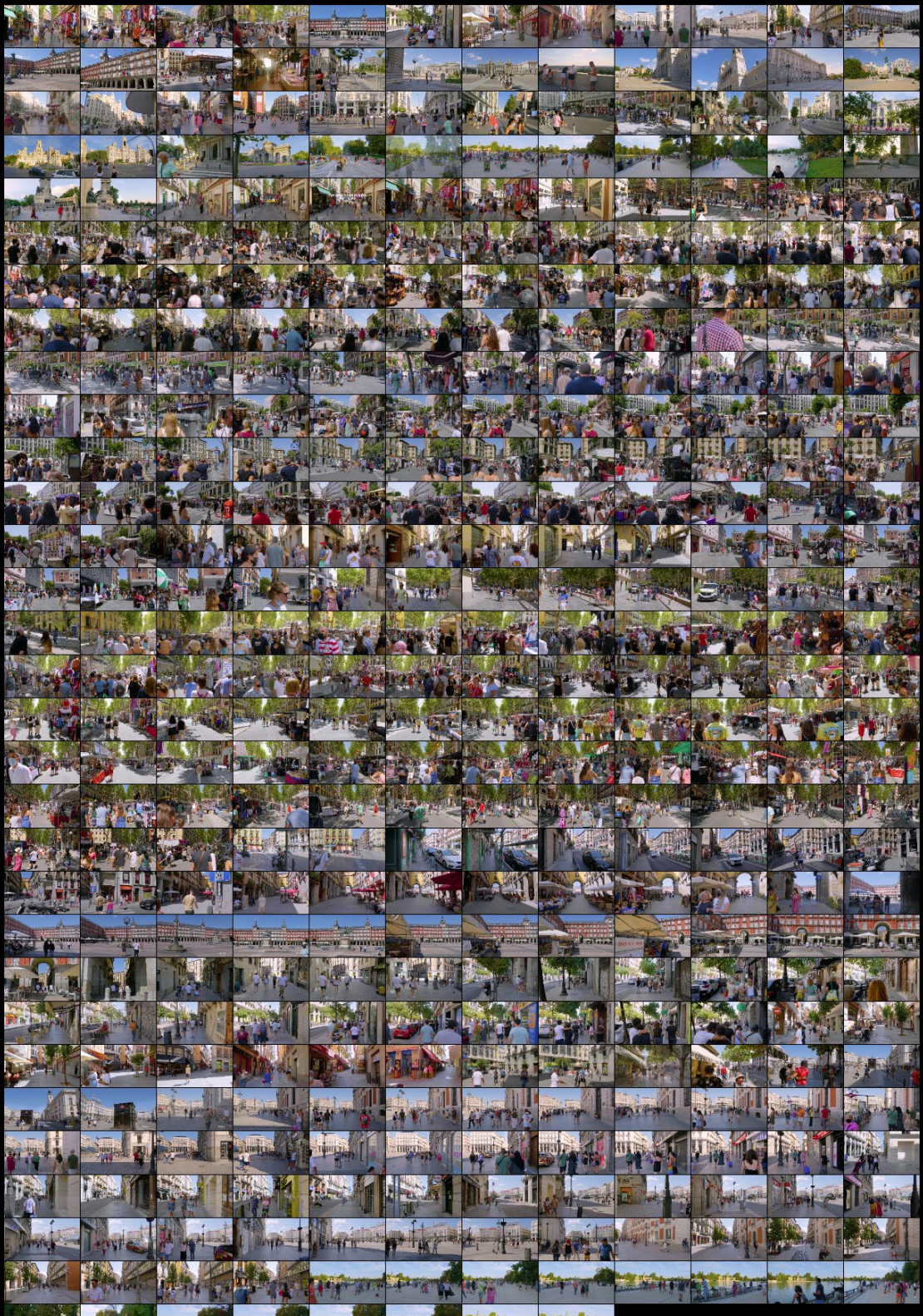


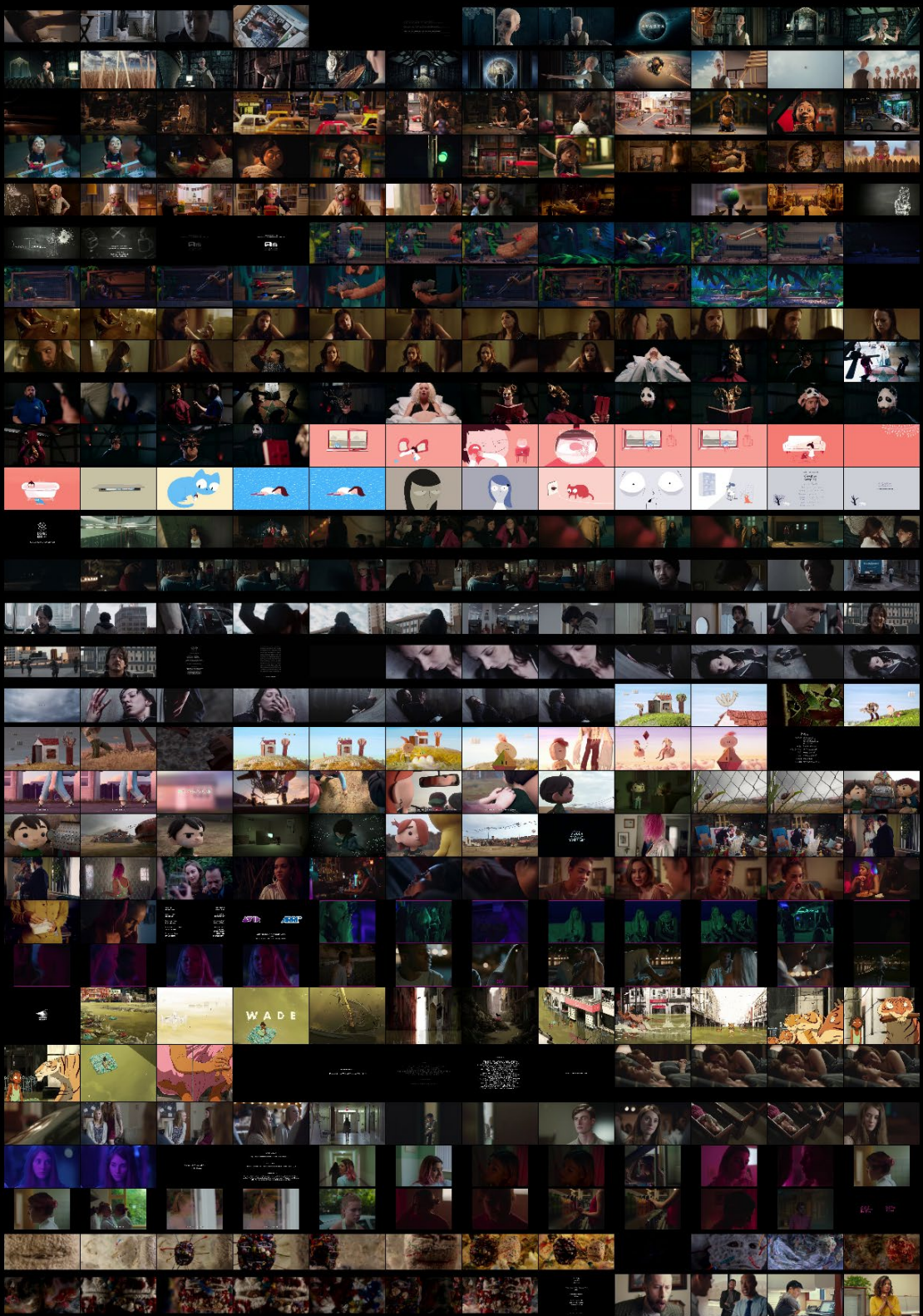
Text domain

A Dictionary of Anarchy and Dictatorship: News from Nowhere, Nineteen Eighty-four, Brave New World.

Data Selection & Collection

For my image domain, I selected walking tour video video slices of Madrid separately, as it covers a large number of well-known locations and street scenes in Madrid.
For my film domain, I selected independent film videos, which are more responsive to non-mainstream, countercultural or subcultural scenes than mainstream films.
For the text domain, I integrated several books as a ‘lexicon’ as an experiment to see if I could activate Utopian or Dystopian content.




WADE

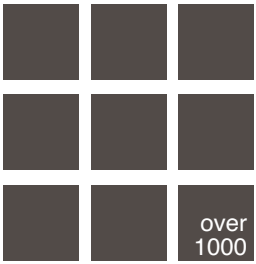
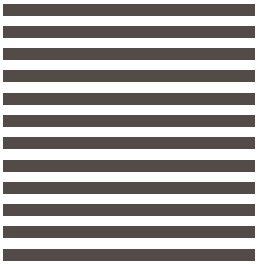
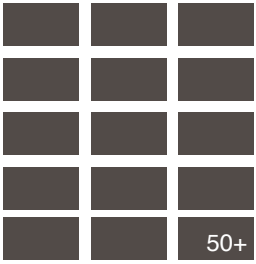


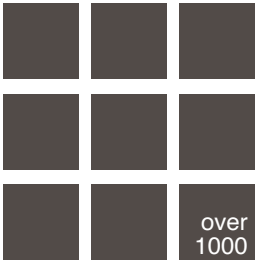
Image domain



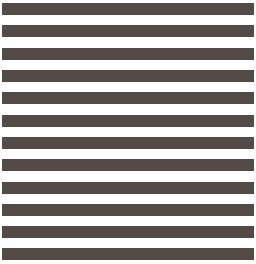
Text: Image to Fuyu



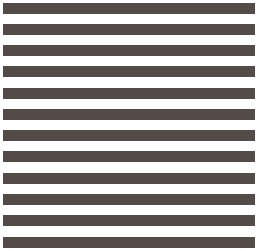
Film domain



Frames: Film random framing



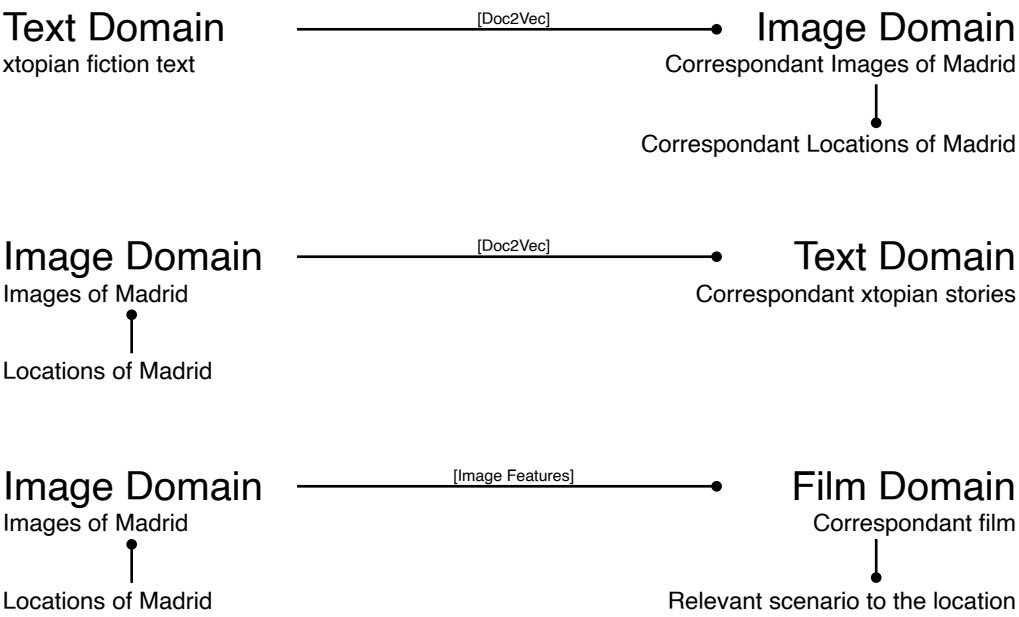
Text: frames to Fuyu



Text domain

Data Pre-process

In order to have a smoother and more accurate transformation between more datasets, I firstly processed the images descriptively using the Fuyu model. Twenty frames were randomly selected from each individual film to be used as the material for matching pictures and videos. These frames are then described by the Fuyu model so that the text can be smoothly matched to the corresponding video.



Text Domain - Image Domain - Film Domain

Text Domain - Film Domain - Image Domain

Image Domain - Text Domain - Film Domain

Image Domain - Film Domain - Text Domain

Film Domain - Text Domain - Image Domain

Film Domain - Image Domain - Text Domain

Film Domain
Indie Films

[Image Features]

Image Domain
Correspondant scenario and behaviour

Correspondant Locations of Madrid

Film Domain
Indie Films

[Doc2Vec]

Text Domain
Correspondant xtopian fiction text

Text Domain
xtopian fiction text

[Doc2Vec]

Film Domain
Correspondant film scene

Relevant space or human behaviour

Dynamic System

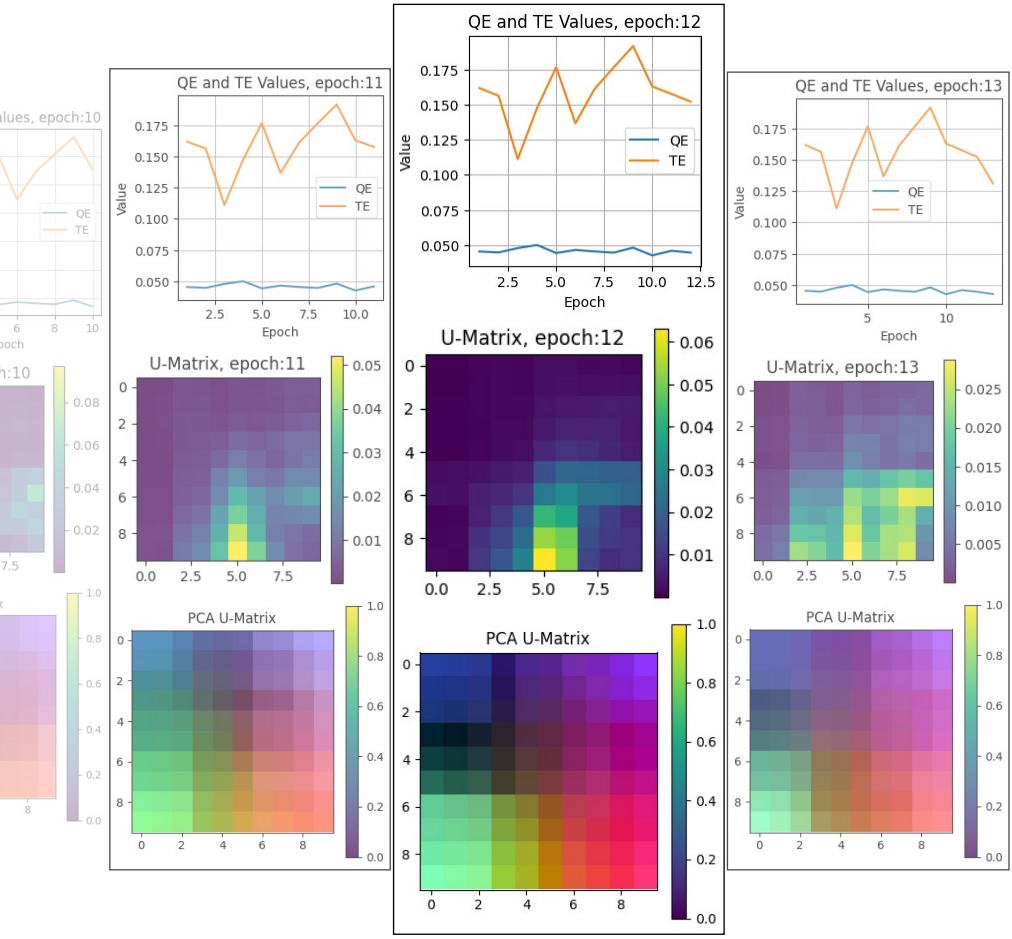
By matching and transforming the data, it is possible to achieve functions such as **selecting a site** based on the behaviour of a text or an non-mainstream film, **finding a corresponding urban environments and human behaviour in pictures** based on the reality counterparts, and **re-contextualising an existing text**.

Text SOM - [Doc2Vec]

Doc2Vec vectorise the text.

```
sentences=processed_text
tagged_data = [TaggedDocument(words=sentence.split(), tags=[str(i)]) for i, sentence in enumerate(sentences)]
model1 = Doc2Vec(vector_size=300, min_count=1, epochs=10)
model1.build_vocab(tagged_data)
model1.train(tagged_data, total_examples=model1.corpus_count, epochs=model1.epochs)
sentence_vectors = [model1.infer_vector(sentence.split()) for sentence in sentences]
#u_matrix = np.array(sentence_vectors)
```

Train the sentence_vectors SOM
Finding epoch with most decent QE and TE Values



Epoch: 12, QE: 0.0484, TE: 0.1492

Build a data dictionary, data_dict, to store the correspondence between text and its vectors. It is then desirable to use this dictionary to find the best matching unit (BMU) at query time.

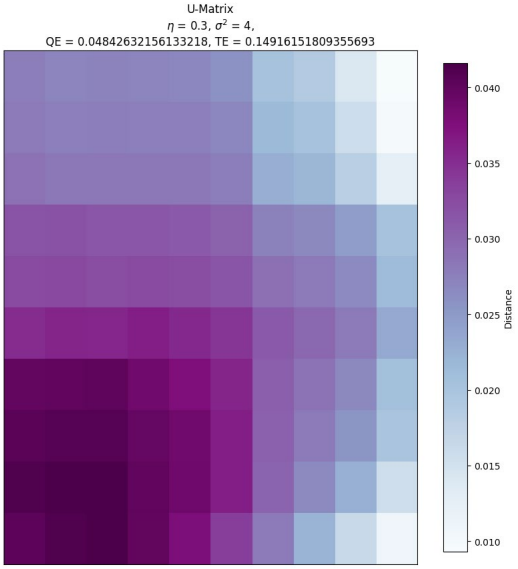
```
data_dict = []
for i in range(len(loaded_text_som_model)):
    row = []
    for j in range(len(loaded_text_som_model[0])):
        row.append([])
    data_dict.append(row)
vectortextPairs=[]
for i in range(0,len(paragraphs)):
    vectortext={}
    vectortext['text']=paragraphs[i]
    vectortext['vector']=sentence_vectors[i]
    vectortextPairs.append(vectortext)
for i in vectortextPairs:
    g,h = find_BMU1(loaded_text_som_model,i['vector'])
    data_dict[g][h].append(i)
```

searchtextsom takes a query as input, preprocesses it, infers its vector representation using a pre-trained model (model1), activates a self-organizing map (SOM) using the inferred query vector, and visualizes the activated SOM. Then, it finds the Best Matching Unit (BMU) on the SOM grid for the query vector, calculates the cosine similarity between the query vector and the vectors associated with the BMU on the SOM, and identifies the most similar text corresponding to the BMU. Finally, it returns the most similar text based on cosine similarity as the result.

```
def searchtextsom(query):
    result = []
    query = [query]
    preprocessed_query = preprocess(query)
    query_vector = model1.infer_vector(preprocessed_query[0].split())
    activatedSOM = activate1(sentence_vectors, loaded_text_som_model, query_vector)

    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.BuPu, aspect='auto')
    plt.title(f'U-Matrix\n$\eta$ = {0.3}, $\sigma^2$ = {4},\nQE = {QE}, TE = {TE}')
    plt.colorbar(im, shrink=0.95, label='Distance')
    plt.xticks([])
    plt.yticks([])
    plt.show()

    g, h = find_BMU1(loaded_text_som_model, query_vector)
    print((g, h))

    similarities = {}
    max_similarity = -1
    biggest = None
    for i in data_dict[g][h]:
        vector_array = np.array([i['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarity = cosine_similarity(vector_array, [loaded_text_som_model[g][h]])[0][0]
        similarities[i['text']] = similarity
        if biggest is None or similarity > max_similarity:
            max_similarity = similarity
            biggest = i['text']

    result = biggest.replace("'", "")
    return result
```

searchtextsom(‘A parliament of counter-culture.’)

<Figure size 640x480 with 0 Axes>



(3,3)

[‘ One could assume that Withers and his associates were now in disgrace, but there had been no report of the matter in the Press or on the telescreen. That was to be expected, since it was unusual for political offenders to be put on trial or even publicly denounced.’]

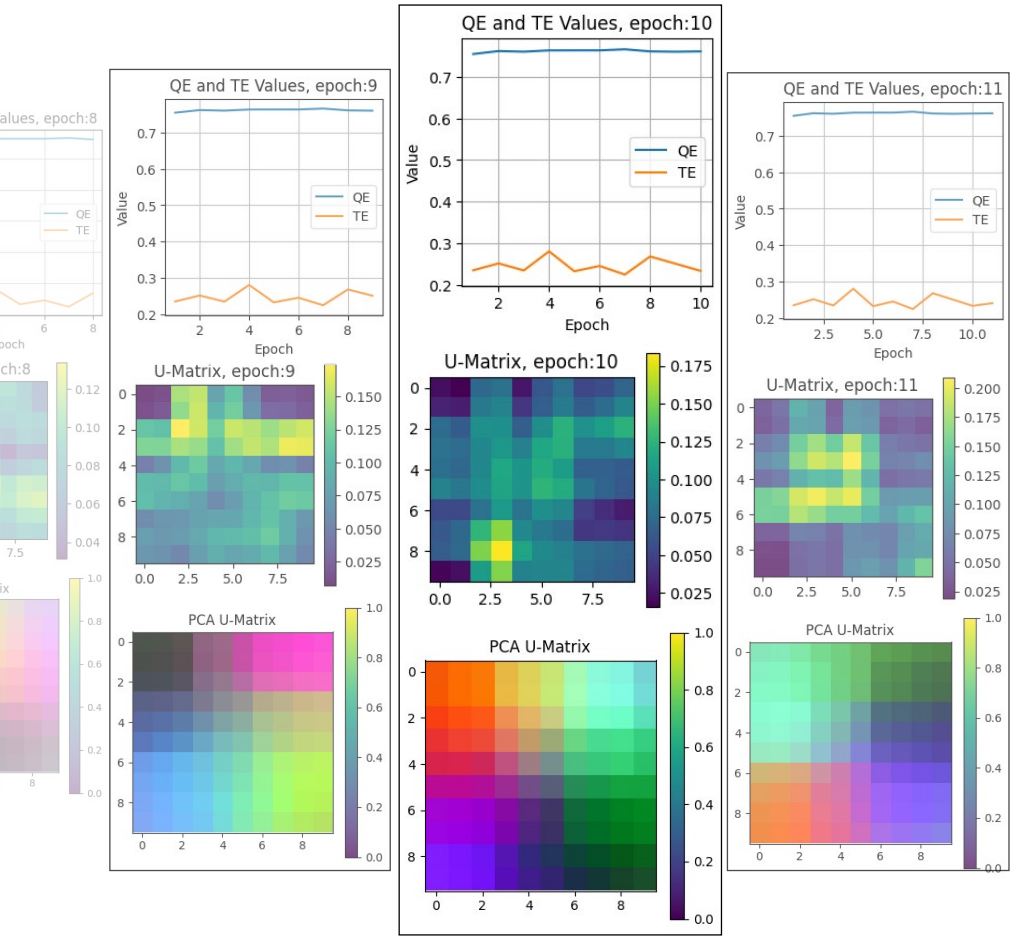
Text SOM - [TF-IDF]

TF-IDF vectorise the text.

```
vectorizer = TfidfVectorizer()
TFidf_vectors = vectorizer.fit_transform(processed_text)
```

```
svd = TruncatedSVD(n_components=300)
svd_u_matrix = svd.fit_transform(TFidf_vectors)
```

Train the svd_u_matrix SOM
Finding epoch with most decent QE and TE Values



Epoch: 10, QE: 0.7605, TE: 0.2339

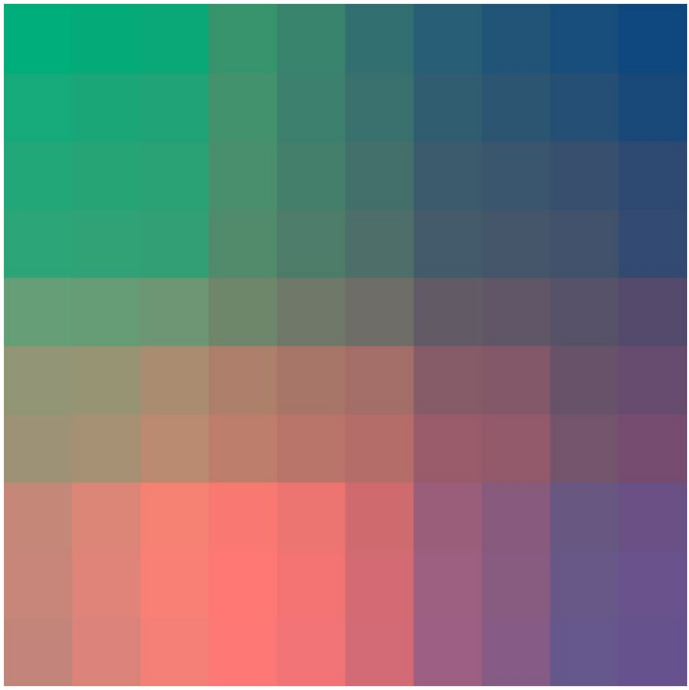
Construct a data dictionary similar to the previous one, but this time for TF-IDF feature vectors instead of raw text vectors.

```
Tfidf_data_dict = []
for i in range(len(TFidf_loaded_som_model)):
    row = []
    for j in range(len(TFidf_loaded_som_model[0])):
        row.append([])
    Tfidf_data_dict.append(row)
vectorTFidftextPairs=[]
for i in range(0,len(paragraphs)):
    vectorTFidftext={}
    vectorTFidftext['text']=paragraphs[i]
    vectorTFidftext['vector']=svd_u_matrix[i]
    vectorTFidftextPairs.append(vectorTFidftext)
for i in vectortextPairs:
    g,h = find_BMU1(TFidf_loaded_som_model,i['vector'])
    Tfidf_data_dict[g][h].append(i)
```

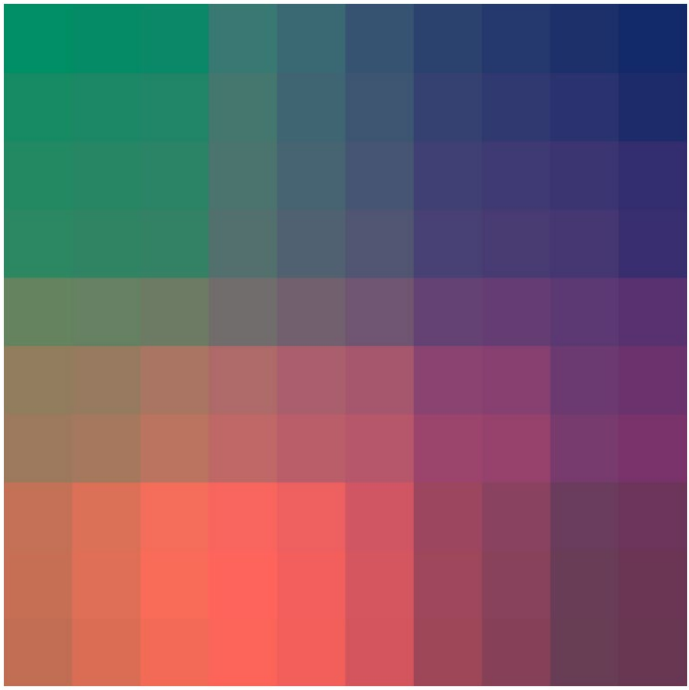
Fit PCA on All Cells

```
som_height, som_width, vector_dimension = loaded_text_som_model.shape
num_nodes = som_height * som_width

flattened_som = loaded_text_som_model.reshape(num_nodes, vector_dimension)

pca = PCA(n_components=3)
colourComponents = pca.fit_transform(flattened_som)



colourComponents=normalise(colourComponents,colourComponents)

rgb_values = (colourComponents * 255).astype(int)

som_colors = np.zeros((som_height, som_width, 3), dtype=np.uint8)

for i in range(som_height):
    for j in range(som_width):
        node_idx = i * som_width + j
        som_colors[i, j] = rgb_values[node_idx]



fig, ax = plt.subplots(figsize=(som_width, som_height))

for i in range(som_height):
    for j in range(som_width):
        color = som_colors[i, j] / 255.0
        ax.add_patch(plt.Rectangle((j, i), 1, 1, color=color))

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

Fit PCA on all cells



Fit PCA on entire training

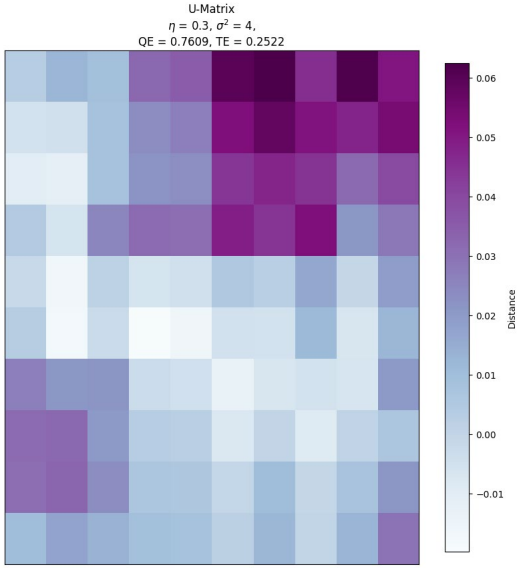


searchTFidftextsom is intended to perform text search using a combination of TF-IDF (Term Frequency-Inverse Document Frequency) vectorization, dimensionality reduction with Singular Value Decomposition (SVD), and Self-Organizing Map (SOM).

```
def searchTFidftextsom(query):
    result=[]
    query=[query]
    preprocessed_query=preprocess(query)
    vectorizer = TfidfVectorizer()
    TFidf_vectors = vectorizer.fit_transform(processed_text)
    query_vector = vectorizer.transform(preprocessed_query)
    svd = TruncatedSVD(n_components=300)
    svd_u_matrix = svd.fit_transform(TFidf_vectors)
    svd_query_u_matrix = svd.transform(query_vector).flatten()
    unit = find_BMU1(TFidf_loaded_som_model, svd_query_u_matrix)
    print(unit)
    activatedSOM = activate1(svd_u_matrix, TFidf_loaded_som_model, svd_query_u_matrix)
    fig = plt.figure()
    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.BuPu, aspect='auto')
    plt.title(f'U-Matrix\n$\eta$ = {0.3}, $\sigma^2$ = {4},\nQE = {QE}, TE = {TE}')
    plt.colorbar(im, shrink=0.95, label='Distance')
    plt.xticks([])
    plt.yticks([])
    plt.show()
    similarities = {}
    for i in Tfidf_data_dict[g][h]:
        vector_array = np.array([i['vector']])
        vector_array = vector_array.reshape(1, -1)
        similarities[i['text']] = cosine_similarity(vector_array, [TFidf_loaded_som_model[g][h]])
    biggest = None
    max_similarity = similarities[Tfidf_data_dict[g][h][0]['text']]
    for i in Tfidf_data_dict[g][h]:
        sim = similarities[i['text']]
        if sim > max_similarity:
            max_similarity = sim
            biggest = i['text']
    result.append(biggest)
    return result
```

searchTFidftextsom(‘A parliament of counter-culture.’)

<Figure size 640x480 with 0 Axes>



(9, 0)

TFIDF
[‘ The steel door opened with a clang. A young officer, a trim black–uniformed figure who seemed to glitter all over with polished leather, and whose pale, straight–featured face was like a wax mask, stepped smartly through the doorway.’]

Doc2Vec
[‘ One could assume that Withers and his associates were now in disgrace, but there had been no report of the matter in the Press or on the telescreen. That was to be expected, since it was unusual for political offenders to be put on trial or even publicly denounced.’]

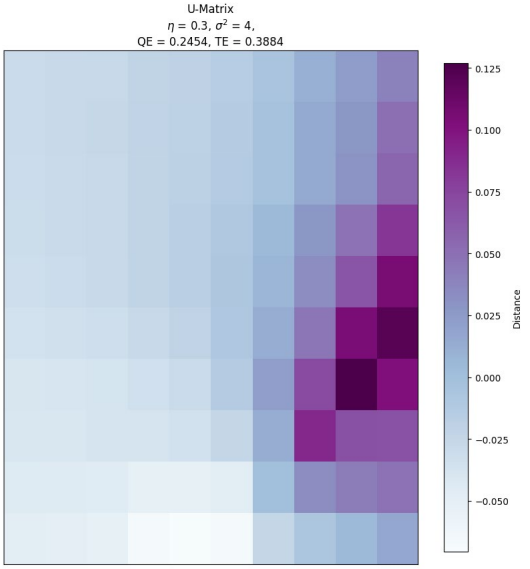
Doc2Vec is able to understand the semantics better, and it seems that Doc2Vec is also faster to process in real applications compared to TF-IDF.

Madrid Fuyu SOM - [Doc2Vec]

Please find the correspondant detailed code in the notebooks.

In this system the function of searching for matching images by text is implemented, first the text will find the best match in the Madrid Fuyu text self-organising mapping. Then it is returned to the CSV source file for matching to get the file name.

searchmadridfuyusom('Its merely a question of self–discipline, reality–control. But in the end there wont be any need even for that. The Revolution will be complete when the language is perfect. Newspeak is Ingsoc and Ingsoc is Newspeak, he added with a sort of mystical satisfaction.')



<Figure size 640x480 with 0 Axes>

['In the image, there is a street scene with several people walking down the sidewalk and cars parked along the street. Among them, there are multiple pedestrians walking down the sidewalk and one person carrying a handbag.\n\nThere are also several cars parked along the street, with one car closer to the left side of the scene and another car further back. Additionally, there is a traffic light visible ']

Text Domain
xtopian fiction text

[Doc2Vec]

Image Domain
Correspondant Images of Madrid

Correspondant Locations of Madrid

Its merely a question of self–discipline, reality–control. But in the end there wont be any need even for that. The Revolution will be complete when the language is perfect. Newspeak is Ingsoc and Ingsoc is Newspeak, he added with a sort of mystical satisfaction.

Correspondant keyframe: [‘keyframe_775.jpg’]

['In the image, there is a street scene with several people walking down the sidewalk and cars parked along the street. Among them, there are multiple pedestrians walking down the sidewalk and one person carrying a handbag.\n\nThere are also several cars parked along the street, with one car closer to the left side of the scene and another car further back. Additionally, there is a traffic light visible ']

Madrid Image SOM - [Features]

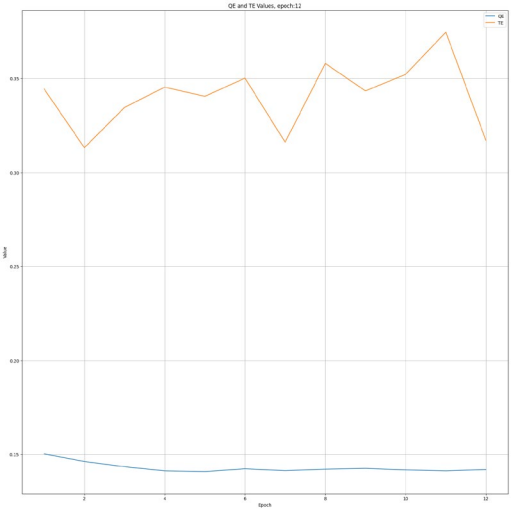
This code utilize a pre-trained MobileNet model from TensorFlow Keras for feature extraction on images. It constructs a list of dictionaries (madridfeatureImagePairs), where each dictionary contains an image filename and its corresponding extracted features.

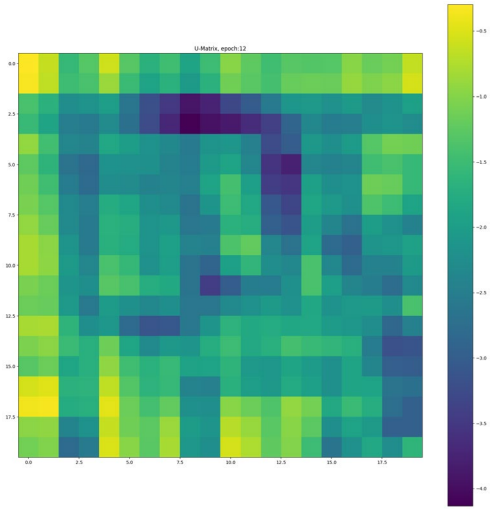
```
model3 = tf.keras.applications.mobilenet.MobileNet(
# The 3 is the three dimensions of the input: r,g,b.
  input_shape=(224, 224, 3),
  include_top=False,
  pooling='avg'
)
```

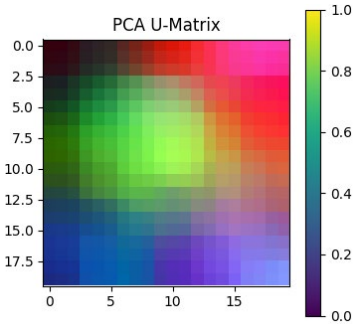
```
madridfeatures = []
for m in madridFiles:
    if m != '.DS_Store':
        path = os.path.join('D:\\FinalAssignment\\Images', m)
        f = processImage(path, model3)
        madridfeatures.append(f)
```

```
madridfeatureImagePairs = []
for i in range(len(madridfeatures)):
    featureImage = {}
    featureImage['image'] = madridFiles[i]
    featureImage['feature'] = madridfeatures[i]
    madridfeatureImagePairs.append(featureImage)
```

Train the madridfeatures SOM
Finding epoch with most decent QE and TE Values







Build a Data dictionary.

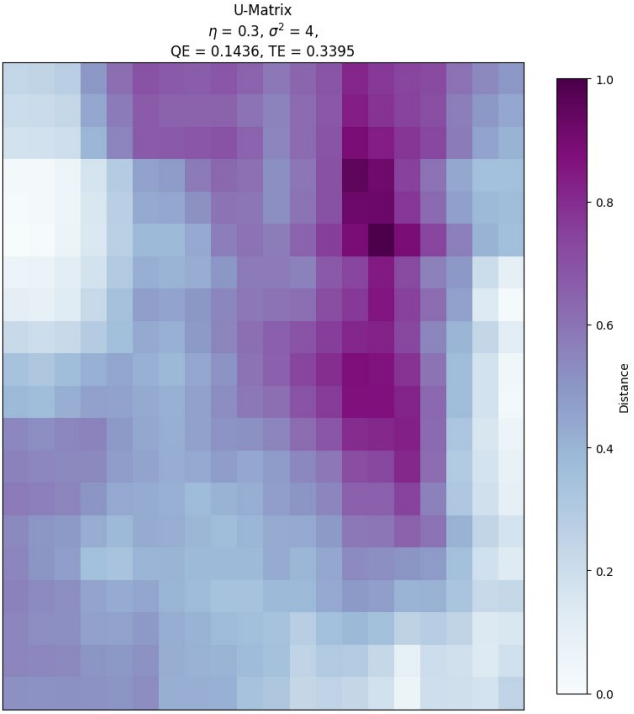
```
madrid_dict = []
for i in range(len(loaded_madrid_som_model)):
    row = []
    for j in range(len(loaded_madrid_som_model[0])):
        row.append([])
    madrid_dict.append(row)
for i in madridfeatureImagePairs:
    query=i['feature']
    g,h = find_BMU(loaded_madrid_som_model,query)
    madrid_dict [g][h].append(i)
```

searchmadridpicturesom function

```
def searchmadridpicturesom(query):
    result=[]
    query_features=[]
    path = query
    q_f = processImage(path, model3)
    query_features.append(q_f)
    activatedSOM = activate(madridfeatures, loaded_madrid_som_model, query_features)
    fig = plt.figure()
    plt.figure(figsize=(10, 10))
    im = plt.imshow(activatedSOM, cmap=cm.BuPu, aspect='auto')
    plt.title(f'U-Matrix\n$\eta$ = {0.3}, $\sigma^2$ = {4},\nQE = {QE}, TE = {TE}')
    plt.colorbar(im, shrink=0.95, label='Distance')
    plt.xticks([])
    plt.yticks([])
    plt.show()
    g,h=find_BMU(loaded_madrid_som_model,query_features)
    closest_image_index=get_closest_image(g,h)
    result.append(madrid_dict[g][h][closest_image_index]['image'])
    return result
```

searchmadridpicturesom('D:\\FinalAssignment\\MoviesKeyframes\\A SXSW Dark Comedy Short The Voice In Your Head_frame_11396.jpg')

1/1 [==============================] - 0s 73ms/step
<Figure size 640x480 with 0 Axes>



['keyframe_88.jpg']

Film Domain
Indie Films

[Image Features]

Image Domain
Correspondant scenario and behaviour

Correspondant Locations of Madrid

D:\\FinalAssignment\\MoviesKeyframes\\A SXSW Dark Comedy Short The Voice In Your Head_frame_11396.jpg

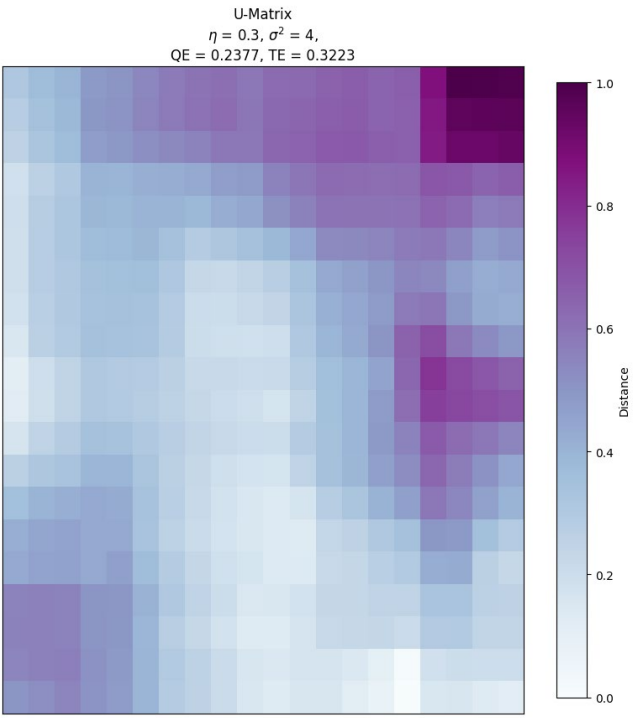
['keyframe_88.jpg']

IndieFilm Frame SOM - [Features]

Please find the correspondant code in the notebooks.

searchindiefilmframessom('D:\\FinalAssignment\\Images\\keyframe_11.jpg')

1/1 [==============================] - 0s 64ms/step
<Figure size 640x480 with 0 Axes>



['Good Intentions Award-Winning Stop-Motion Animated Short Film_frame_2030.jpg']

Image Domain
Images of Madrid
Locations of Madrid

[Image Features]

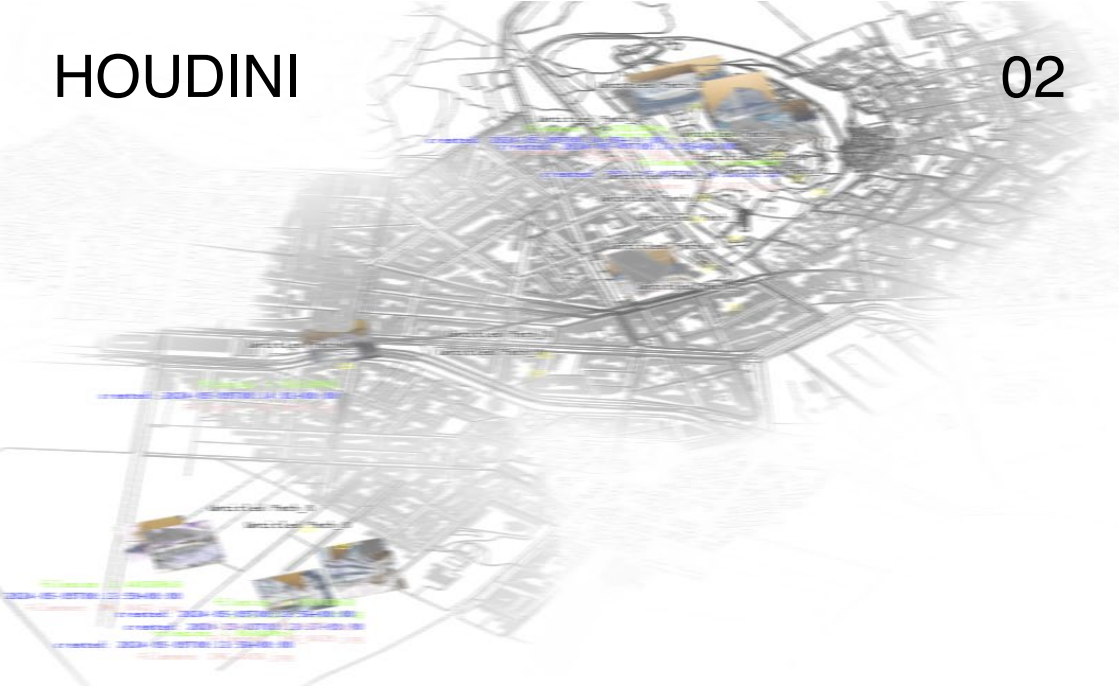
Film Domain
Correspondant film
Relevant scenario to the location

D:\\FinalAssignment\\Images\\keyframe_11.jpg

['Good Intentions Award-Winning Stop-Motion Animated Short Film_ frame_2030.jpg']

HOUDINI

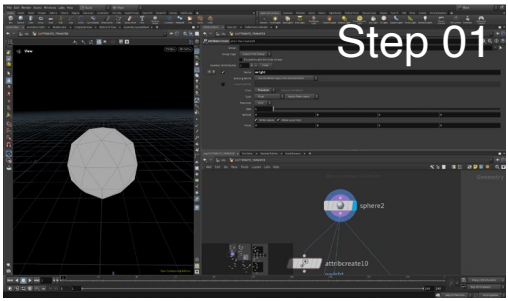
02



2.1.1 Houdini Fundamental ‘Dissolve’

Step 01 Sphere SOP

The Sphere SOP in Houdini is a tool for making sphere shapes. User can decide how big the sphere is and how detailed it should be by adjusting how many rows and columns it has. It allows primitive type of NURBS, polygon, mesh, etc.

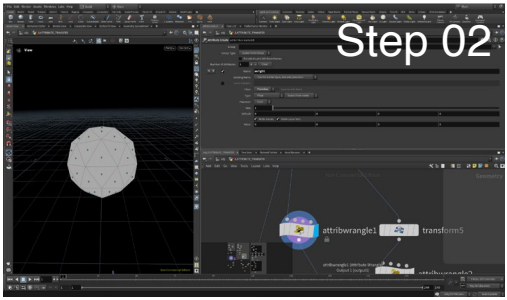


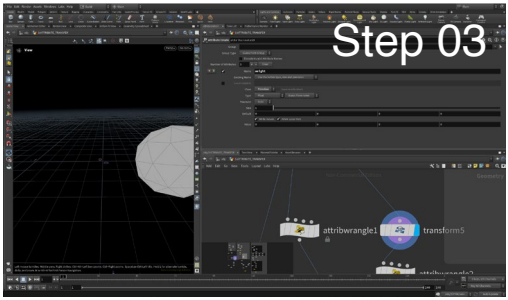
Step 02 Attribute Wrangle SOP
The attribute is of type float as described by the ”f@” declaration. The attribute is set to run over primitives. This is because later in the node setup, we will use a PolyExtrude node where we will need a primitive attribute to drive the extrusion value. The value is set to 0 (which will be interpreted as 0.0 because of the float declaration) because later, we can interpolate between 0 and 1.

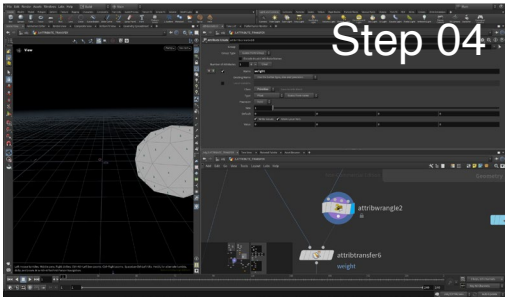
Step 03 Transform SOP
Then, we lay down a ”Transform” SOP.
This will be the attractor; it is just a duplicate of the original sphere. The distance between this duplicate (attractor) and the original sphere will dictate the final extrusion value.

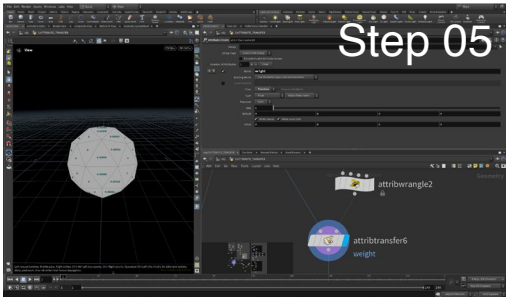
Step 04 Attribute Wrangle SOP
The same as the previous Wrangle SOP, only with the opposite value. The reason we interpolate between 0 and 1 is because it is easier to work with normalizedvalues than arbitrary values.

Step 05 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 canhave 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).







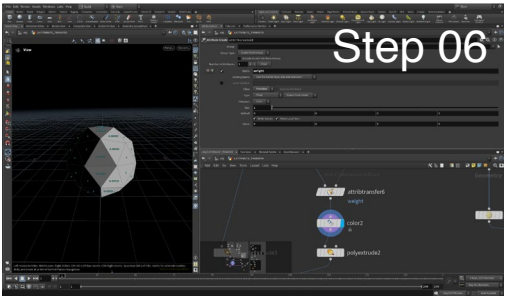


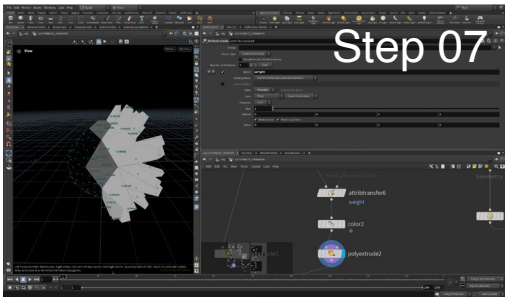
Step 06 Color SOP
A visualisation of the ”weight” primitive attribute according to the ”viridis” colorscheme. Because we normalized our values between 0 and 1, it is more stable if we would ever change distances, or geometry inputs.

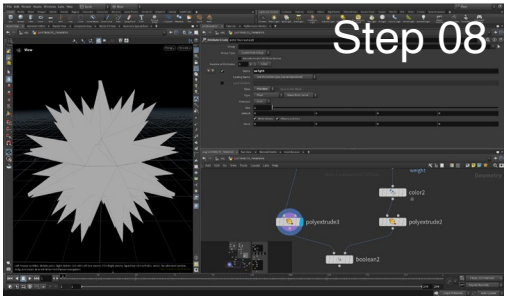
Step 07 PolyExtrude SOP
Using the ”weight” value in d”distance scale”, we use the weight value as a multiplier of the extrusion value. If we set the distance value to 0, the extrusion is 0 (0 x weight). If we set the distance value to 1, we get the full weight value as extrusion (1 x weight)

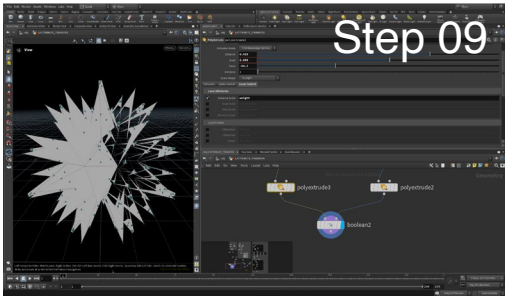
Step 08 PolyExtrude SOP
Directly polyextrude against the original sphere to get a multi-tipped hedgehog shape by changing the values of the parameters distance, inset, twist.

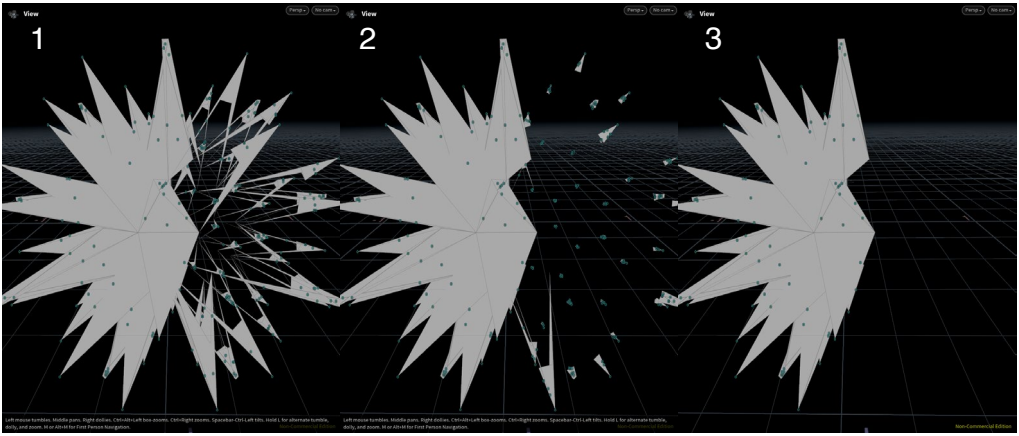
Step 09 Boolean SOP
The Boolean is a tool used in Houdini to combine two or more geometries together or take parts from each other. It is based on Boolean logic principles and can perform operations such as union, intersection or difference sets.

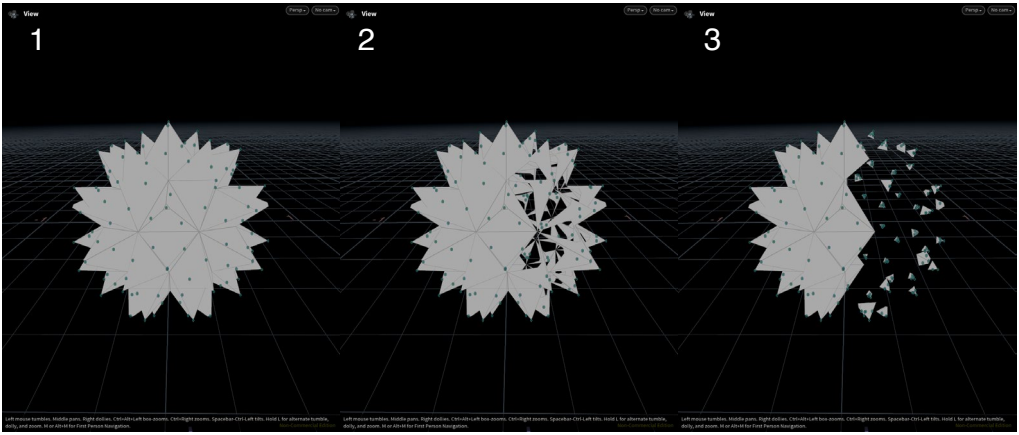


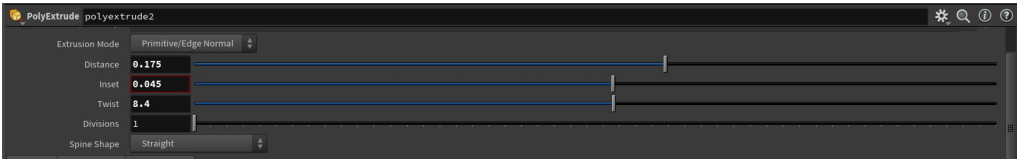










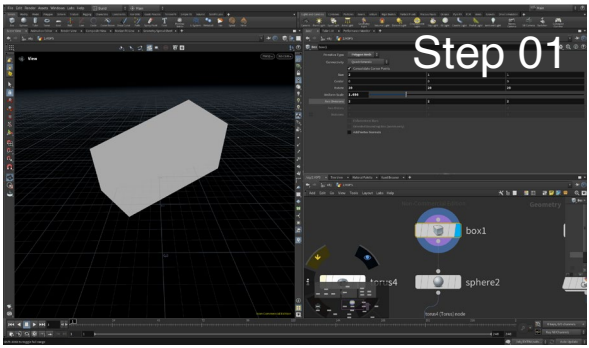


By changing the Distance parameter in Polyextrude, it is possible to control the stage at which the sphere ‘melts’ and, in conjunction with previous parameter changes, the direction in which it melts.

2.1.2 Houdini Fundamental ‘Erosion’

Step 01 Box SOP

The Box SOP in Houdini is a tool for creating box-shaped geometry. We adjust its size by changing its dimensions in three directions (length, width, and height) and position it in 3D space.

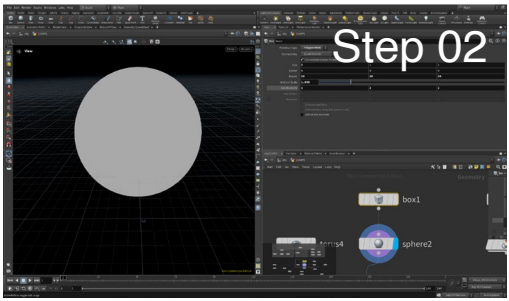


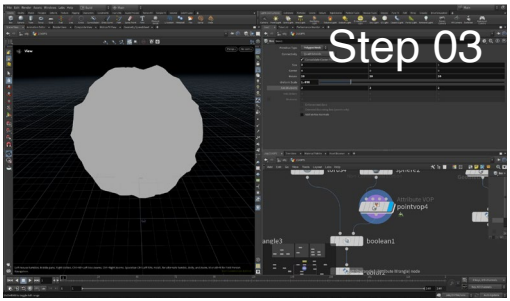
Step 02 Sphere SOP
The Sphere SOP in Houdini is a tool for making sphere shapes. User can decide how big the sphere is and how detailed it should be by adjusting how many rows and columns it has. It allows primitive type of NURBS, polygon, mesh, etc.

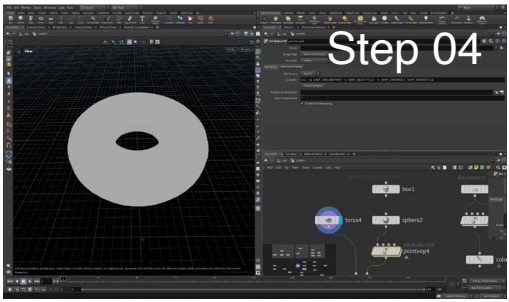
Step 03 Point VOP
The Point VOP (Vector Operation) in Houdini is a node used for working with point attributes in a procedural manner. It allows users to manipulate attributes such as position, color, and velocity at each individual point of a geometry. The primitive type is Polygon Mesh. By connecting the Box-Sphere-Point VOP, a perturbed ball shape is rendered.

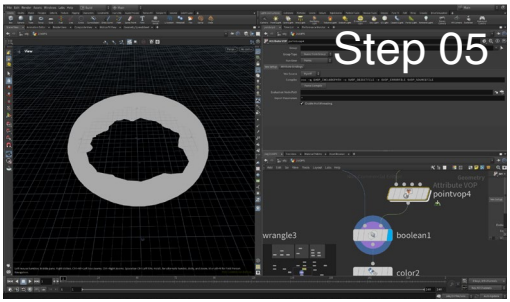
Step 04 Torus SOP
The Torus SOP in Houdini is a tool for creating a torus-shaped geometry, which is like a donut or a ring shape. **Here, Torus is the subject of erosion.**

Step 05 Boolean SOP
The Boolean is a tool used in Houdini to combine two or more geometries together or take parts from each other. It is based on Boolean logic principles and can perform operations such as union, intersection or difference sets.



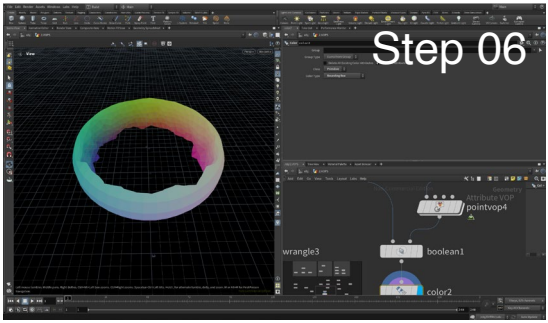


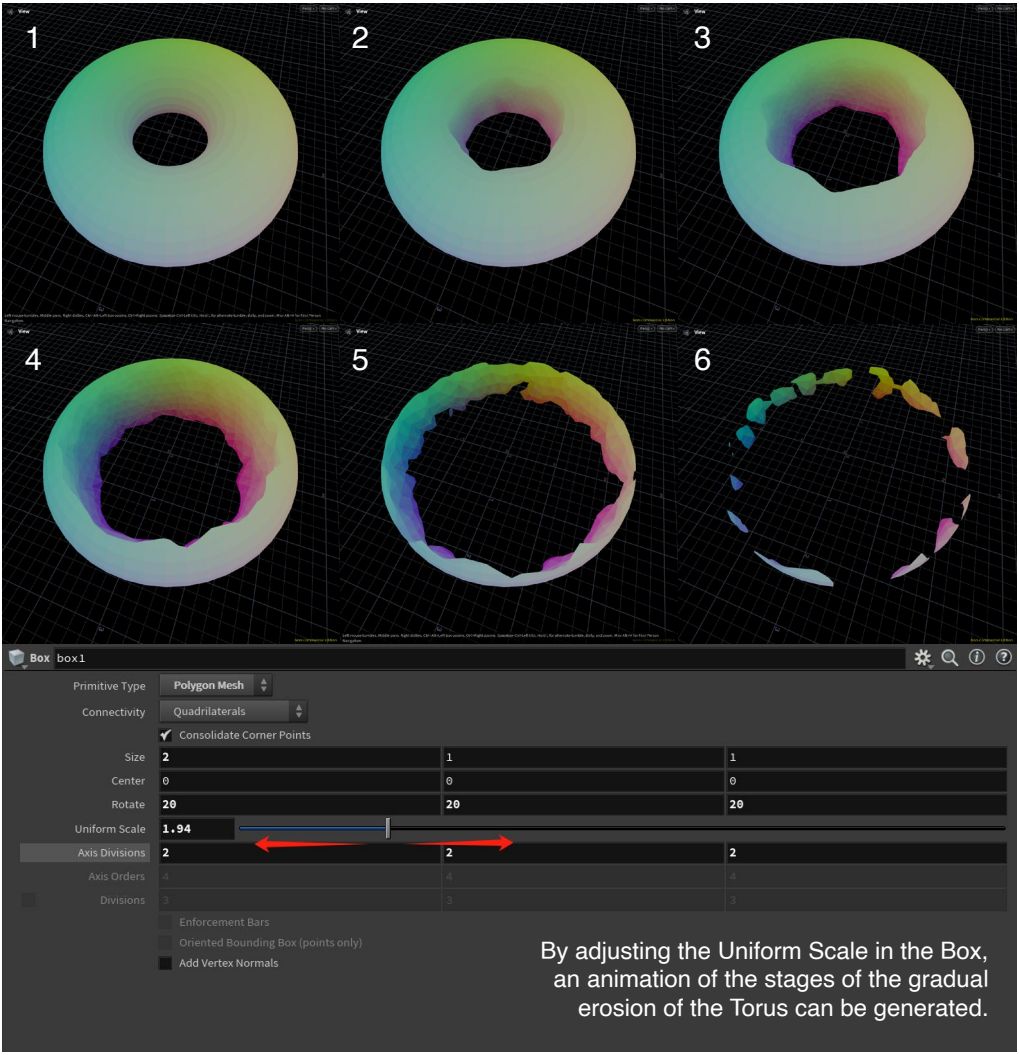




Step 06 Color SOP

The class is ‘Primitive’ and the color type is ‘Bounding box’, this step is to more intuitively allow shapes to be displayed.

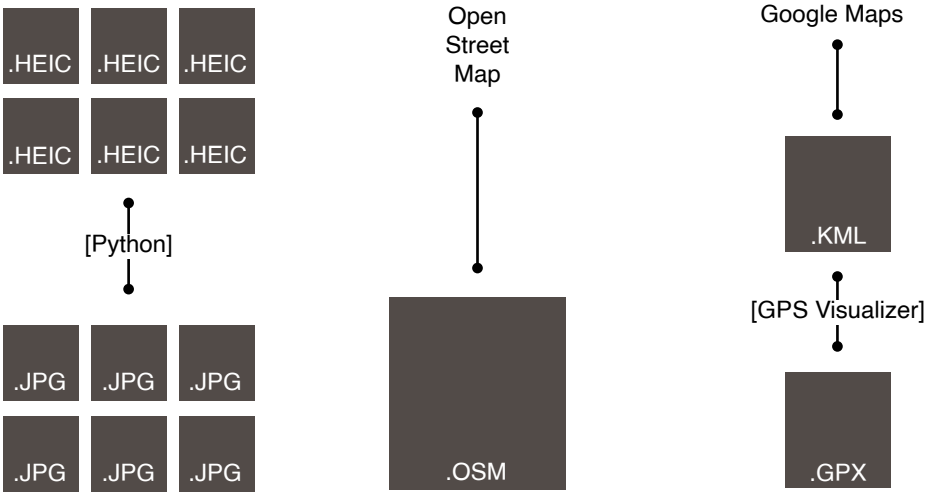




By adjusting the Uniform Scale in the Box, an animation of the stages of the gradual erosion of the Torus can be generated.

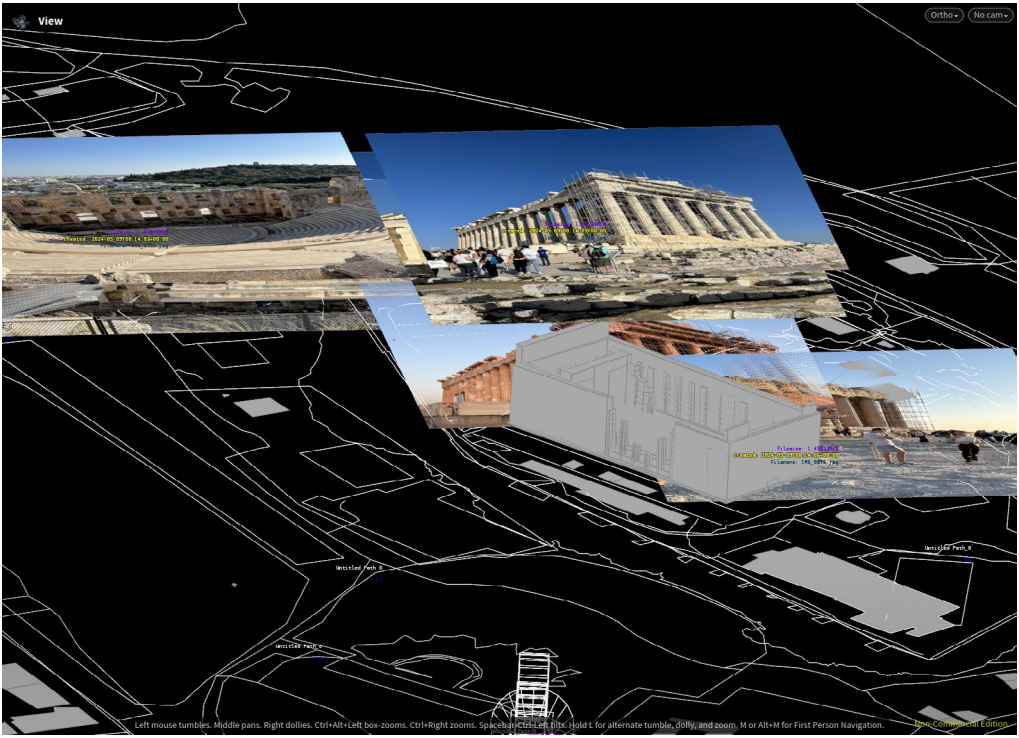
A 3D visualisation of the photos and their locations is generated by importing a gpx file that records the travel footprints, a heic file that records the photo images and their location at the time, and an OSM file that has been supplied with the map by OpenStreetMap.

2.2 Visualize GPX Data

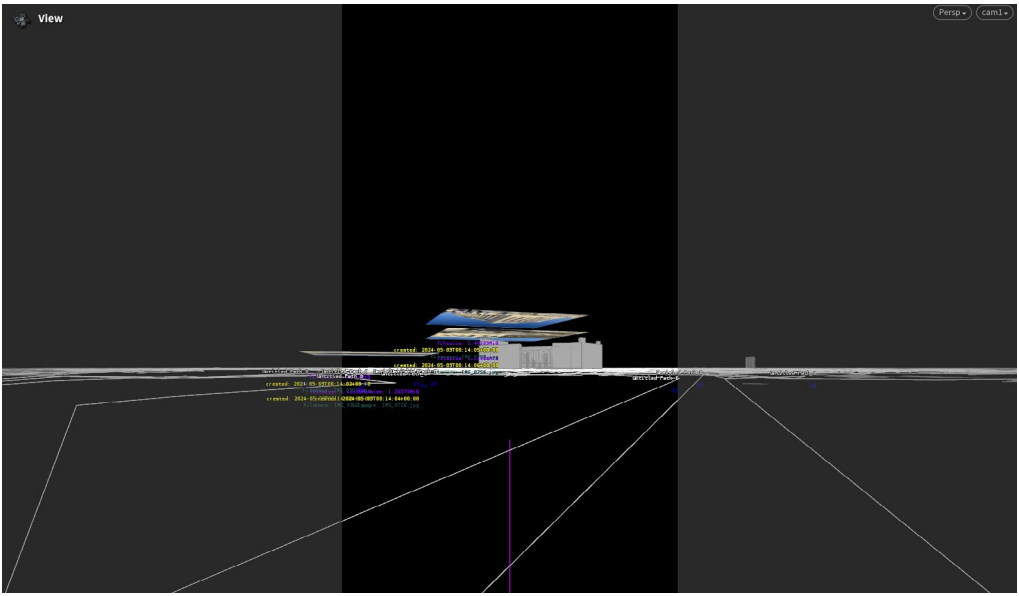


Step 01 Obtain and preprocess all the files required

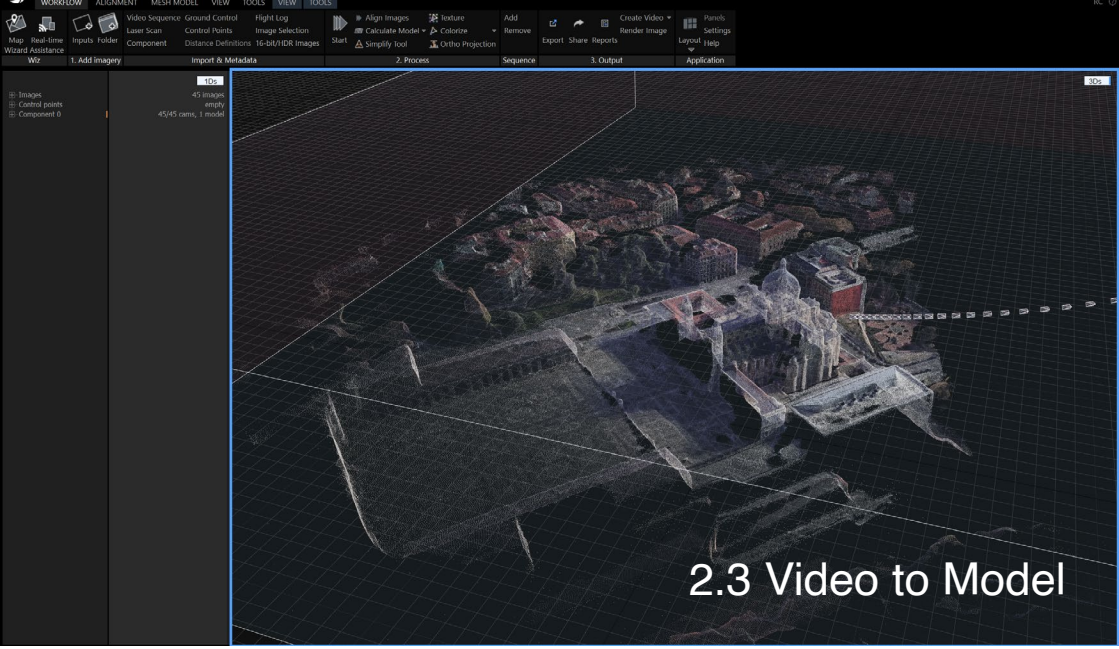
Step 02 Import the files and compose into a visualized map



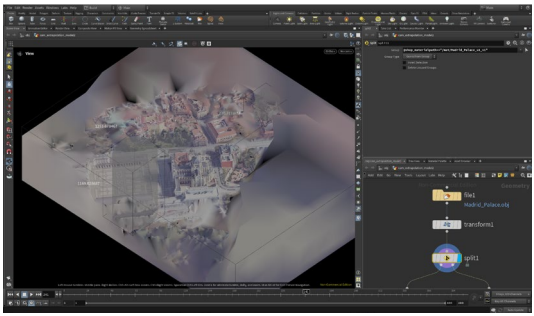
Acropolis data visualization



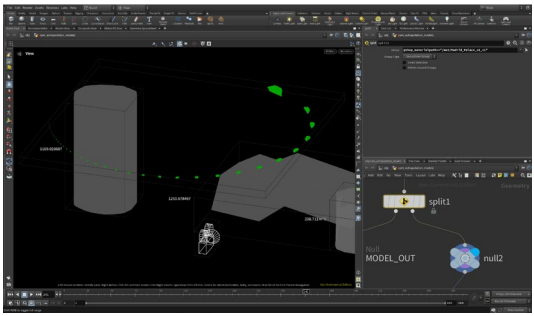
Camera Path



STEP 01 Reproduce Geometry of Madrid Royal Palace
Google Earth - Python - Capture Reality
In this step, I first screen recorded a google earth dynamic view of Madrid Royal Palace, which is one of the relevant site to our project, and then use python to keyframe the video. Finally, import all of the photos to Capture Reality and convert it in to an obj model.

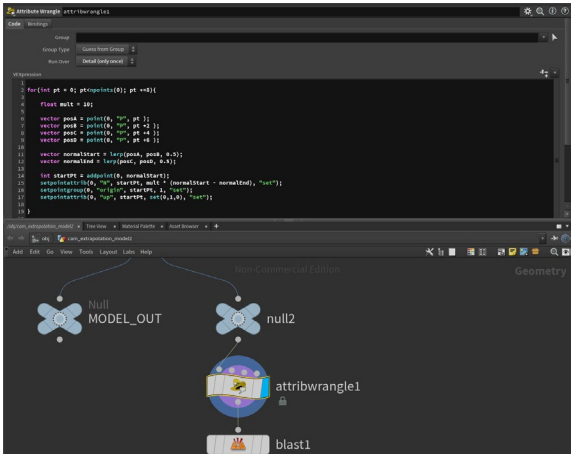


STEP 02 File SOP & Transform SOP
First link the obj file to Houdini, and then transform it into proper position.



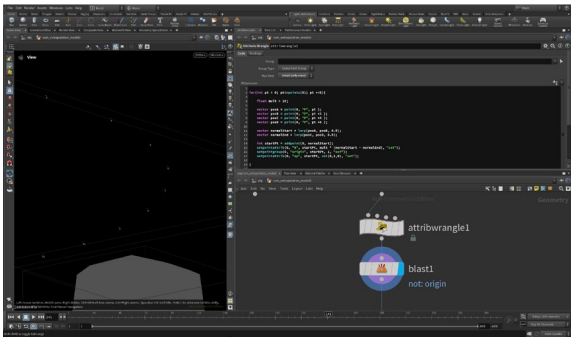
STEP 03 Split SOP
@shop_materialpath==”/mat/Madrid_Palace_u1_v1”

STEP 04 Attribute Wrangle SOP



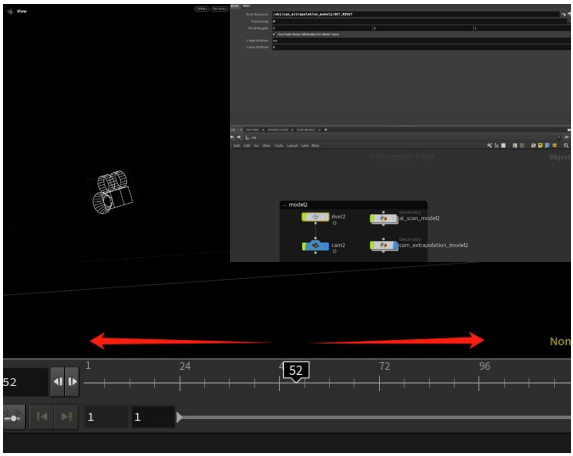
STEP 05 Blast SOP

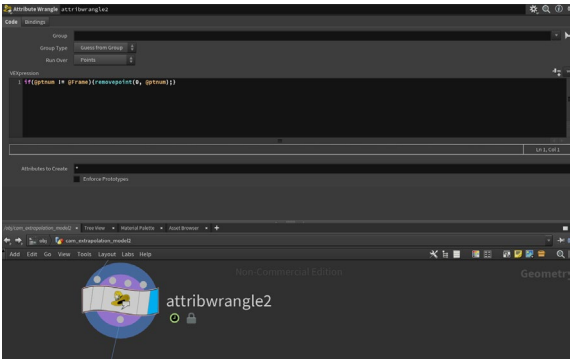
SOP node that allows you to delete or filter out specific elements from your geometry based on certain criteria.



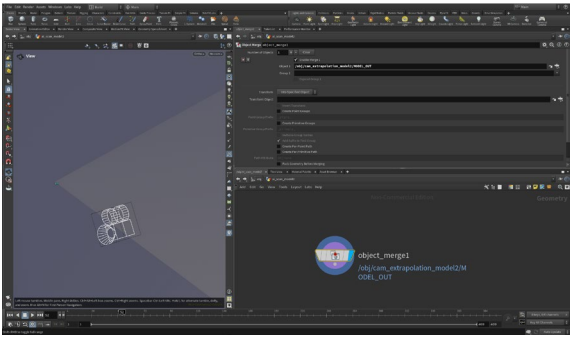
STEP 06 CAM + RIVET

The CAM node is used to create motion paths for cameras in Houdini. It allows animators to define complex camera movements along spline curves or other paths. The RIVET node is used to attach objects (such as geometry or nulls) to surfaces in Houdini. It allows for precise placement of objects on surfaces, even as the surfaces deform or animate.

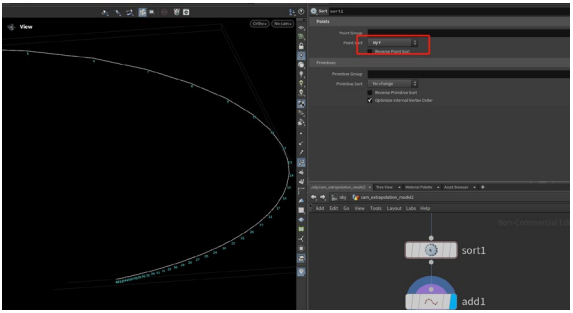




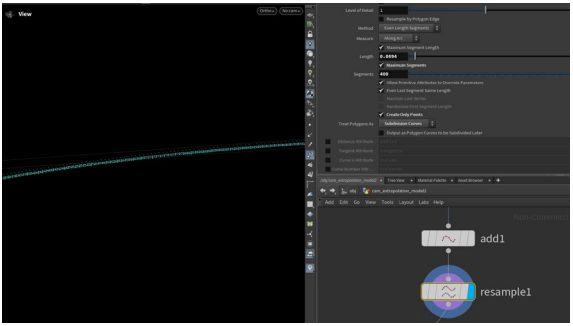
STEP 07 Attribute Wrangle SOP



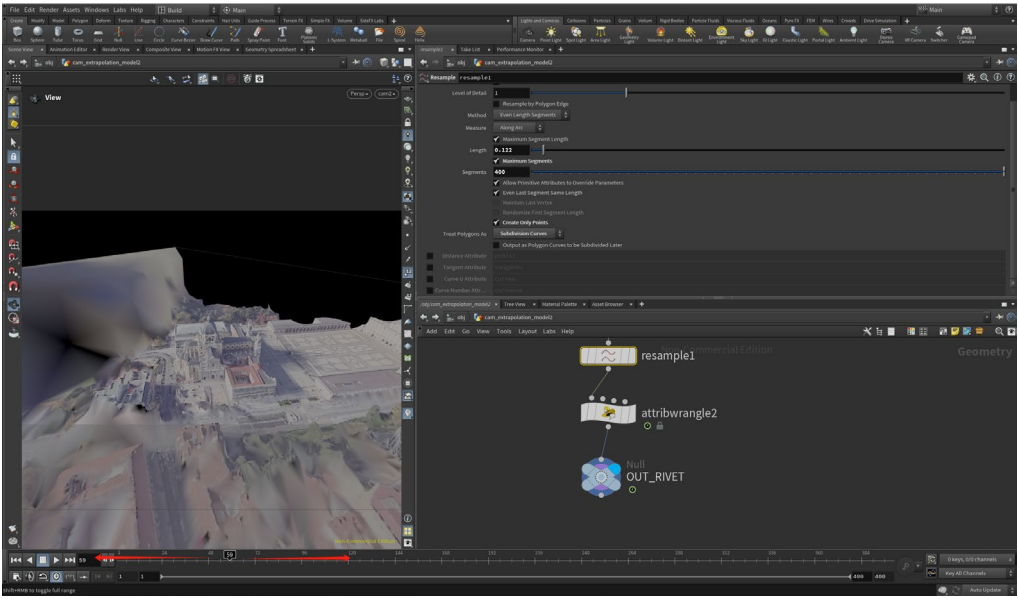
STEP 08 A new geometry container



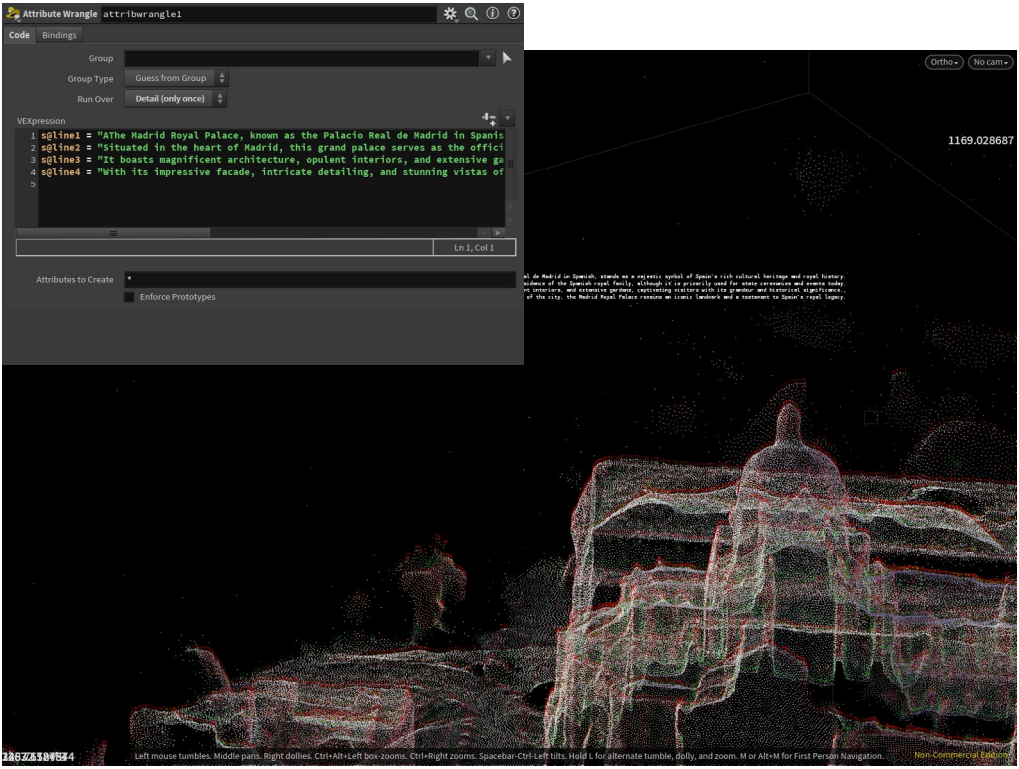
STEP 09 Sort & Add SOP
In my condition, the Point Sort is ‘By Y’ to generate proper Add SOP.

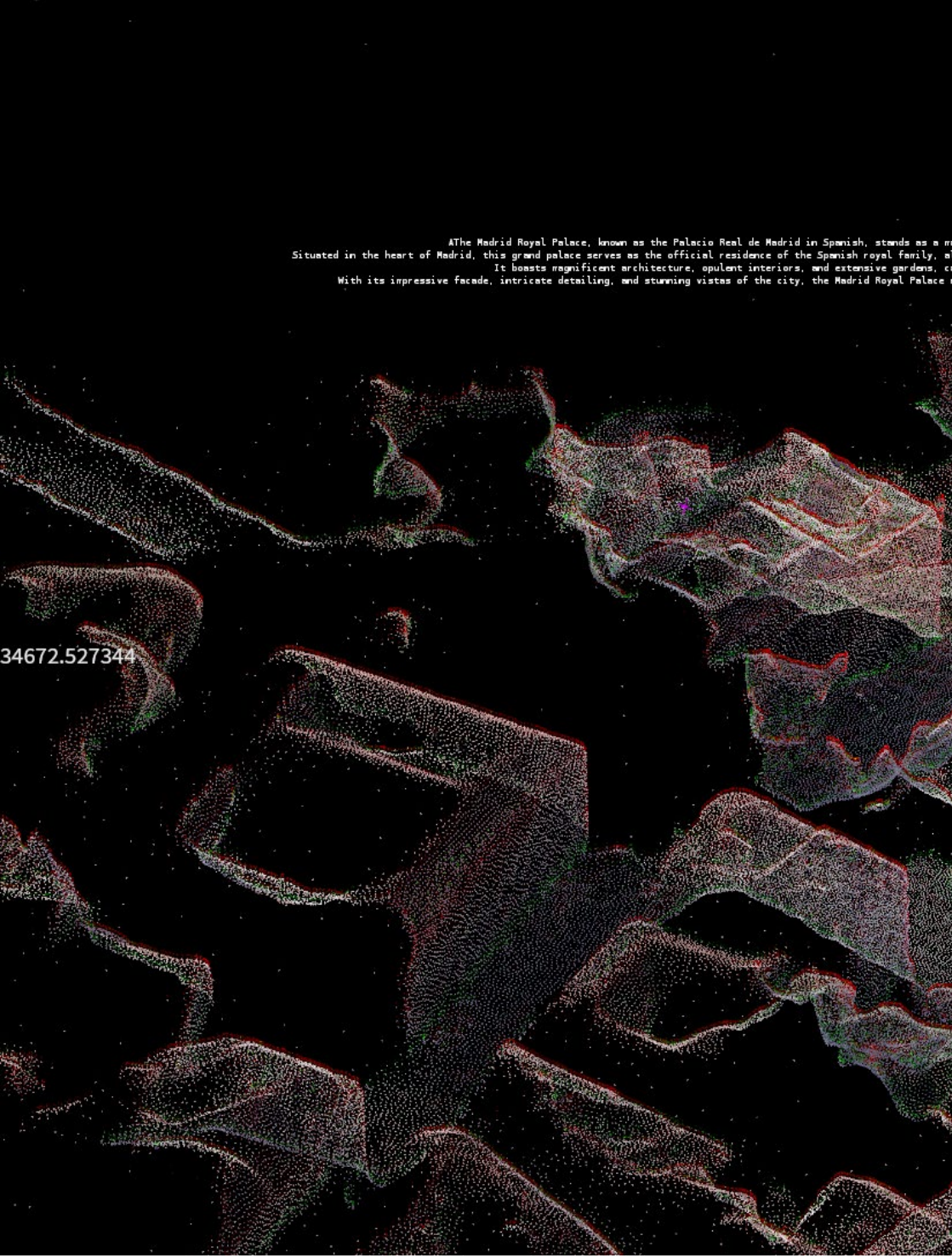


STEP 10 Resample SOP
The Resample SOP in Houdini is a powerful tool used for altering the point density of geometry, which can be useful for various purposes such as simplifying geometry, refining details, or achieving specific effects.



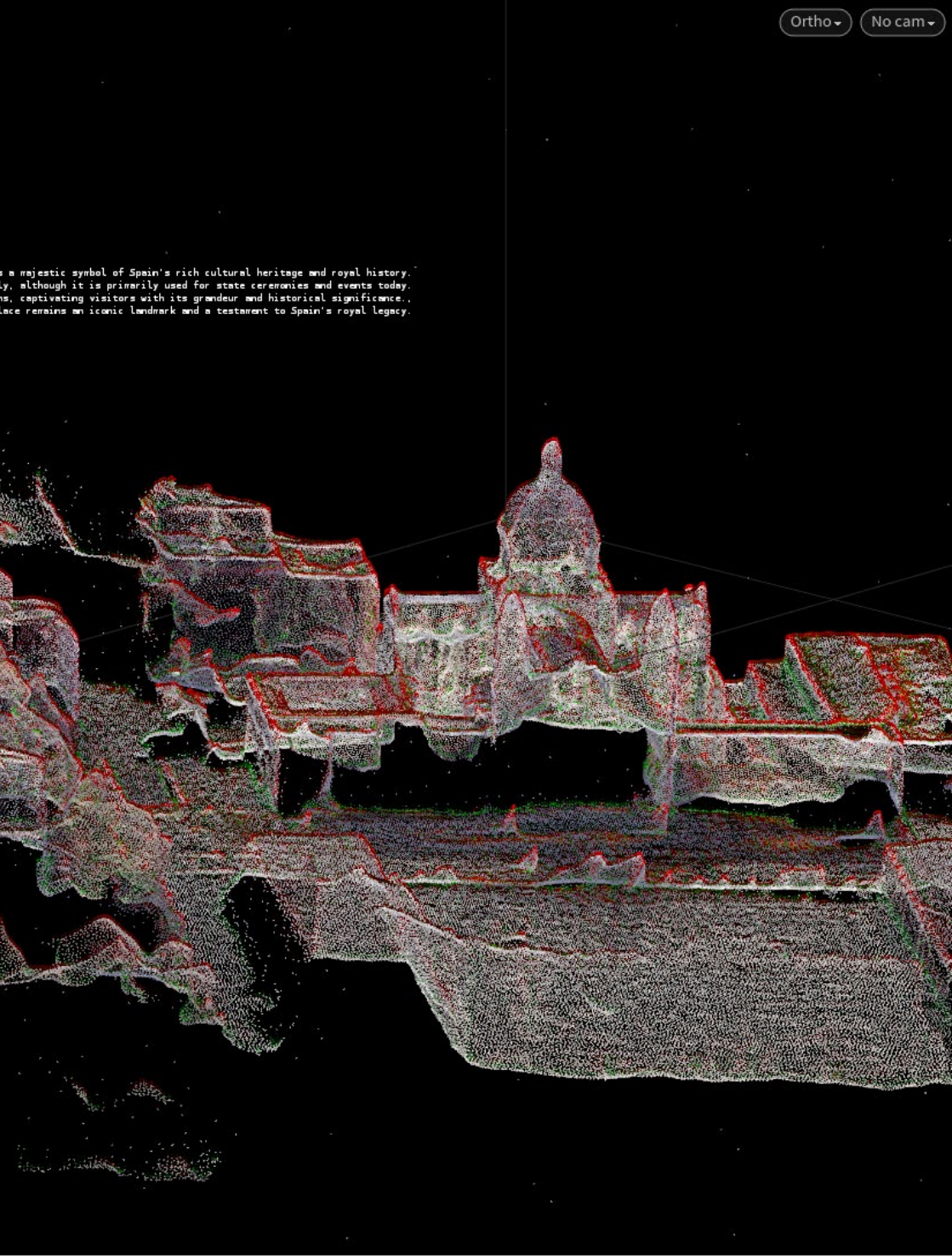
STEP 11 Camera Testing




AThe Madrid Royal Palace, known as the Palacio Real de Madrid in Spanish, stands as a m
Situated in the heart of Madrid, this grand palace serves as the official residence of the Spanish royal family, a
It boasts magnificent architecture, opulent interiors, and extensive gardens, c
With its impressive facade, intricate detailing, and stunning vistas of the city, the Madrid Royal Palace
34672.527344


ajestic symbol of Spain's rich cultural heritage and royal history.
Although it is primarily used for state ceremonies and events today.
aptivating visitors with its grandeur and historical significance.,
remains an iconic landmark and a testament to Spain's royal legacy.


View
AThe Madrid Royal Palace, known as the Palacio Real de Madrid in Spanish, stands as
Situated in the heart of Madrid, this grand palace serves as the official residence of the Spanish royal fami
It boasts magnificent architecture, opulent interiors, and extensive garde
With its impressive facade, intricate detailing, and stunning vistas of the city, the Madrid Royal Pa
34672.527344

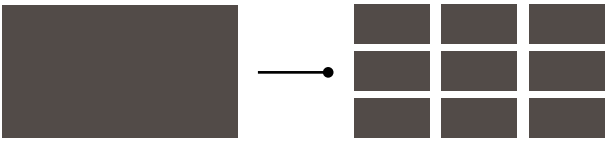

Ortho
No cam
s a majestic symbol of Spain's rich cultural heritage and royal history.
ly, although it is primarily used for state ceremonies and events today.
ns, captivating visitors with its grandeur and historical significance.,
lace remains an iconic landmark and a testament to Spain's royal legacy.

2.3 Visualizing JSON

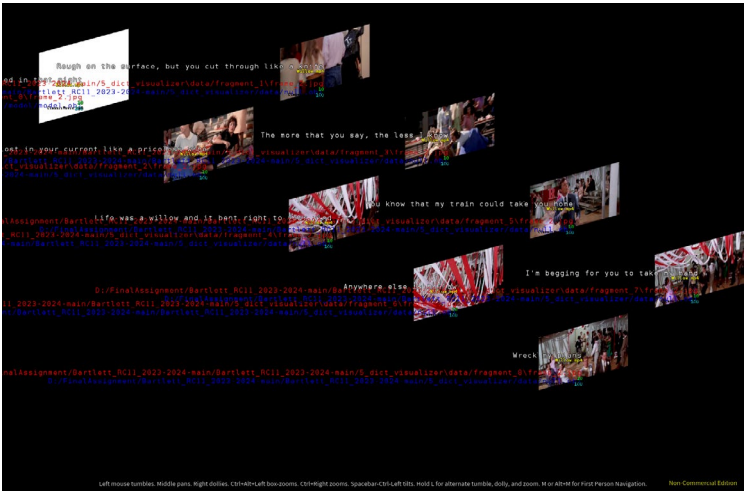

```
"folder": [
  {
    "paragraph": "I'm like the water when your ship rolled in that night",
    "filmID": "Willow.mp4",
    "image_name": "frame_1.jpg",
    "model_path": "model.obj",
    "image_n": 10,
    "time": 100,
    "film_path": "ClassicMovie.mp4"
  },
  {
    "paragraph": "Rough on the surface, but you cut through like a knife",
    "filmID": "Willow.mp4",
    "image_name": "frame_1.jpg",
    "model_path": "",
    "image_n": 10,
    "time": 100,
    "film_path": "ClassicMovie.mp4"
  },
  {
    "paragraph": "Lost in your current like a priceless wine",
    "filmID": "Willow.mp4",
    "image_name": "frame_1.jpg",
    "model_path": "",
    "image_n": 10,
    "time": 100,
    "film_path": "ClassicMovie.mp4"
  },
  {
    "paragraph": "The more that you say, the less I know",
    "filmID": "Willow.mp4",
    "image_name": "frame_1.jpg",
    "model_path": "",
    "image_n": 10,
    "time": 100,
    "film_path": "ClassicMovie.mp4"
  },
  {
    "paragraph": "Life was a willow and it bent right to your wind",
    "filmID": "Willow.mp4",
    "image_name": "frame_1.jpg",
    "model_path": "",
    "image_n": 10,
    "time": 100,
    "film_path": "ClassicMovie.mp4"
  },
  {
    "paragraph": "You know that my train could take you home",
    "filmID": "Willow.mp4",
```


In this attempt, I loaded the project related lyrics and movie frames into the model so that the movie frame material could be visualised and analysed in bulk by visualising the JSON through Houdini.

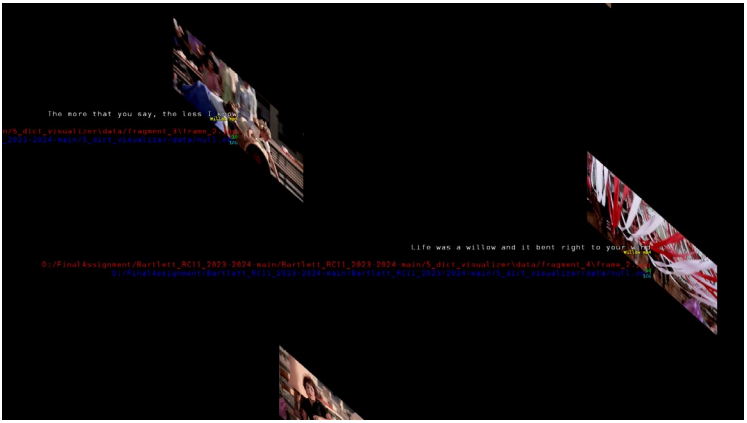
Prepare JSON file



Film domain Frames



Output



Output

3.1.1 Implement the algorithm in Python

```
import numpy as np

def square_matrix_multiply(A, B):
    n = A.shape[0]
    C = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i, j] += A[i, k] * B[k, j]
    return C



A = np.array([[3, 4, 2], [4, 6, 7],[1, 2, 4]])
B = np.array([[6, 4, 2], [8, 2, 3],[1, 3, 4]])



result = square_matrix_multiply(A, B)



print(result)
```

```
[[52. 26. 26.]
 [79. 49. 54.]
 [26. 20. 24.]]
```

Analysis of Time Complexity
The time complexity analysis for this algorithm stays consistent with the previous explanation. The algorithm employs three nested loops, iterating through each element of the resulting matrix C. In each iteration, it calculates the dot product of a row in matrix A with a column in matrix B.

In detail:
The outer loop (indexed by i) runs n times, where n represents the size of the matrices.
The middle loop (indexed by j) iterates n times for each iteration of the outer loop.
The inner loop (indexed by k) runs n times for each iteration of the middle loop.
This arrangement results in a total of n * n * n = n^3 scalar multiplications being executed.
Consequently, the time complexity of this algorithm is denoted as O(n^3).

The rationale behind this assessment remains consistent with the earlier explanation. The algorithm's structure comprises three nested loops, each iterating n times, thereby leading to a cubic time complexity. Each loop iteration entails n scalar multiplications, cumulatively amounting to n^3 scalar multiplications. Consequently, the time complexity indeed stands at O(n^3).

3.1.2 Implement the algorithm with nested lists

```
import time
```

```
nestedlist1 = np.random.rand(10, 10).tolist()
```

```
nestedlist2 = np.random.rand(10, 10).tolist()
```

```
nestedlist3 = np.random.rand(100, 100).tolist()
```

```
nestedlist4 = np.random.rand(100, 100).tolist()
```

```
def square_matrix_multiply(A, B):
    n = len(A)
    C = [[0 for i in range(n)] for i 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
```

```
start_time = time.time()
square_matrix_multiply(nestedlist1, nestedlist2)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 0.000946044921875 seconds
```

```
start_time = time.time()
square_matrix_multiply(nestedlist3, nestedlist4)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 0.35797929763793945 seconds
```

Triple Nested Loops: The algorithm employs three nested loops to iterate over the elements of the matrices A and B. Each loop iterates n times, where n is the size of the matrices. This structure results in a cubic time complexity, as there are three nested loops, each iterating n times.

Scalar Multiplications: Within the innermost loop, there's a constant number of operations performed (multiplications and additions) to compute the product of corresponding elements of row i of matrix A and column j of matrix B. Since this operation is executed n^2 times (once for each element in the resulting matrix), the total number of scalar multiplications is proportional to n^3.

Overall Complexity: Combining the effects of the triple nested loops and the scalar multiplications, the overall time complexity of the algorithm is O(n^3).

3.1.3 Comparison

Comparing execution times (in seconds) for custom vs. built-in multiplication:
This code will generate random matrices of different sizes, multiply them using both the custom algorithm with nested lists and the built-in multiplication function (np.dot()), and compare their execution times.

```
def square_matrix_multiply(A, B):
    n = len(A)
    C = [[0 for i in range(n)] for i 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

def generate_random_matrix(n):
    return [[np.random.rand() for _ in range(n)] for _ in range(n)]

def measure_custom_time(matrix_size):
    A = generate_random_matrix(matrix_size)
    B = generate_random_matrix(matrix_size)

    start_time = time.time()
    square_matrix_multiply(A, B)
    end_time = time.time()

    return end_time - start_time

def measure_builtin_time(matrix_size):
    A = np.random.rand(matrix_size, matrix_size)
    B = np.random.rand(matrix_size, matrix_size)

    start_time = time.time()
    np.dot(A, B)
    end_time = time.time()

    return end_time - start_time

matrix_sizes = [10, 50, 100, 200, 500]

print("Comparing execution times (in seconds) for custom vs. built-in multiplication:")
print("{:<10} {:<15} {:<15}".format("Matrix Size", "Custom Time", "Built-in Time"))
for size in matrix_sizes:
    custom_time = measure_custom_time(size)
    builtin_time = measure_builtin_time(size)
    print("{:<10} {:<15.6f} {:<15.6f}".format(size, custom_time, builtin_time))
```

[14]

```
Comparing execution times (in seconds) for custom vs. built-in multiplication:
Matrix Size Custom Time     Built-in Time
10          0.001002        0.018447
50          0.043015        0.000000
100         0.238083        0.002000
200         2.158756        0.002033
500         32.722635       0.007995
```

As shown above, built-in is a more effecient choice.

Multiply more than two matrices / perform other operations on matrices
This code will compare the execution times for multiplying multiple matrices using both the custom algorithm with nested lists and the built-in multiplication function (np.dot()).

```
def multiply_multiple_matrices(matrices):
    result = matrices[0]
    for i in range(1, len(matrices)):
        result = np.dot(result, matrices[i])
    return result

def measure_multiple_matrices_time(matrix_size, num_matrices):
    matrices = [np.random.rand(matrix_size, matrix_size) for _ in range(num_matrices)]

    start_time = time.time()
    multiply_multiple_matrices(matrices)
    end_time = time.time()

    return end_time - start_time

def measure_custom_multiple_matrices_time(matrix_size, num_matrices):
    matrices = [generate_random_matrix(matrix_size) for _ in range(num_matrices)]

    start_time = time.time()
    result = matrices[0]
    for i in range(1, len(matrices)):
        result = square_matrix_multiply(result, matrices[i])
    end_time = time.time()

    return end_time - start_time

num_matrices = 3

print("\nComparing execution times (in seconds) for custom vs. built-in multiplication of {} matrices:".format(num_m
print("{:<10} {:<15} {:<15}".format("Matrix Size", "Custom Time", "Built-in Time"))
for size in matrix_sizes:
    custom_time = measure_custom_multiple_matrices_time(size, num_matrices)
    builtin_time = measure_multiple_matrices_time(size, num_matrices)
    print("{:<10} {:<15.6f} {:<15.6f}".format(size, custom_time, builtin_time))
```

[16]

...

```
Comparing execution times (in seconds) for custom vs. built-in multiplication of 3 matrices:
Matrix Size Custom Time     Built-in Time
10          0.001084        0.001062
50          0.077531        0.000000
100         0.577909        0.002001
200         4.467828        0.003000
500         67.795220       0.012995
```

As shown above, built-in is a more effecient choice.

3.2 Implement the recursive algorithm

Algorithm Description

01. Recursiveness:
The algorithm repetitively splits the matrices into smaller segments until they reach a minimal size, typically 1x1, where a direct multiplication operation can be executed.

02. Divide-and-Conquer:
Divide:
Initially, the algorithm divides the input matrices into four equal-sized quarters, setting the stage for further processing.
Conquer:
It subsequently tackles each submatrix pair recursively, multiplying them until they reach the base case.
Combine:
After recursive multiplication, the algorithm combines the results from individual submatrices to generate the final product.

Implementation
The translation of the pseudocode into Python functions primarily involves direct mapping, albeit with some complexity in handling matrix splits and combines, necessitating careful index management.

Complexity Analysis
Base Case:
Multiplying 1x1 matrices involves one operation, hence O(1) time complexity.
Divide Step:
Dividing matrices into quarters necessitates copying elements, taking O(n^2) time.
Conquer Step:
The recursive multiplication of submatrices involves eight recursive calls, each dealing with matrices of size n/2, leading to a per-call time complexity of T(n/2).
Combine Step:
Combining submatrices involves element-wise addition, taking O(n^2) time.
Overall:
The overall time complexity is deduced as O(n^3) from the recurrence relation T(n) = 8T(n/2) + O(n^2), reflecting the need for additions and multiplications at each recursion level within the divide-and-conquer framework. Despite its recursive nature, the divide-and-conquer approach helps manage the problem by breaking it into smaller, more manageable subproblems.

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

```python
A = np.random.rand(128,128)
B = np.random.rand(128,128)
```

```python
start_time = time.time()
square_matrix_multiply_recursive(A,B)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 7.632548093795776 seconds
```

```python
start_time = time.time()
square_matrix_multiply(A,B)
end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")
```

```
Execution time: 1.4195423126220703 seconds
```

3.3 Strassen algorithm

Reflection on Matrix Operations Complexity
Matrix multiplication typically involves more complex operations than addition and subtraction. Multiplying two matrices involves the scalar product of rows and columns, resulting in more computational steps compared to addition or subtraction, which only requires adding or subtracting corresponding elements.

Complexity Analysis of Strassen Algorithm
Base Case:The base case remains the same as the recursive algorithm in Exercise 2, involving a single multiplication operation, hence O(1).
Divide Step:Splitting matrices into quarters takes O(n^2) time, similar to the recursive algorithm.
Conquer Step:Strassen's algorithm performs seven recursive multiplications of matrices of size n/2 × n/2. Each of these recursive calls deals with matrices of half the size of the original matrices, leading to a time complexity of T(n/2) per call.
Combine Step:After the recursive multiplications, there are additional steps (S1 to S10) involving additions and subtractions, totaling 10 extra operations.
The final step combines the submatrices into the resulting matrix C, which also involves additions and subtractions.
Overall Complexity:The recurrence relation for Strassen's algorithm is T(n) = 7T(n/2) + O(n^2), as there are seven recursive calls each dealing with matrices of size n/2 × n/2; Using the Master Theorem or other methods, the overall time complexity is deduced as O(n^log2(7)) ≈ O(n^2.81); Compared to the naive recursive algorithm, Strassen's algorithm reduces the number of recursive multiplications but introduces additional overhead due to extra addition and subtraction operations.

[Adaptation and Explanation of Changes]
The implemented code adapts the Strassen algorithm by incorporating the splitting, combining, and recursive multiplication operations. Instead of eight recursive multiplications as in Exercise 2, Strassen's algorithm performs seven recursive multiplications. Additionally, it includes extra steps (S1 to S10) for additions and subtractions before the recursive multiplications and after to calculate the final quarters of matrix C. These changes affect the overall time complexity, reducing the number of recursive multiplications while introducing extra overhead for additional operations.
Overall, Strassen's algorithm optimizes matrix multiplication by reducing the number of recursive multiplications, which leads to improved performance for large matrices. However, the algorithm's effectiveness depends on various factors, such as the size of the matrices and the efficiency of the implementation.

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


def strassen_matrix_multiply(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]
    else:
        # Splitting and recombining matrices
        mid = n // 2
        A11, A12, A21, A22 = split_matrix(A, mid)
        B11, B12, B21, B22 = split_matrix(B, mid)

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

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

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

GitHub links:

01_VECTORIAL ENCODINGS OF QUALITATIVE DOMAINS/1.2_SelfOrganizingMap/**1.2_FullNotebook.ipynb**
https://github.com/UD-Skills-2023-24/RC11_23129421/blob/2fbf6ad93d8326ec47eeb16f62f05849e5554937/01_VECTORIAL%20ENCODINGS%20OF%20QUALITATIVE%20DOMAINS/1.2_SelfOrganizingMap/1.2_FullNotebook.ipynb

03_COMPLEXITY/**Complexity.ipynb**
https://github.com/UD-Skills-2023-24/RC11_23129421/blob/2fbf6ad93d8326ec47eeb16f62f05849e5554937/03_COMPLEXITY/Complexity.ipynb

01_VECTORIAL ENCODINGS OF QUALITATIVE DOMAINS/1.2_SelfOrganizingMap/Notebooks_Saparated/**1.2.1_Text_SOM(Doc2Vec_TF-IDF).ipynb**
https://github.com/UD-Skills-2023-24/RC11_23129421/blob/2fbf6ad93d8326ec47eeb16f62f05849e5554937/01_VECTORIAL%20ENCODINGS%20OF%20QUALITATIVE%20DOMAINS/1.2_SelfOrganizingMap/Notebooks_Saparated/1.2.1_Text_SOM(Doc2Vec_TF-IDF).ipynb

fileExtension links:

RC11_23129421_SKILL.fileExtension
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvxuk_ucl_ac_uk/Ei-U9gVw11ZKsuJlK-HVkiMBqCR4qBe-YMchjqU1g3KXzQ?e=o1RKgd

RC11_23129421_JULIAN.fileExtension
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvxuk_ucl_ac_uk/EurEQrchuFNJh5BPUzSyMj4Bm8DXywllAvoi_IUmmcyxDA?e=HYcfip

RC11_23129421_HOUDINI.fileExtension
https://liveuclac-my.sharepoint.com/:f:/g/personal/ucbvxuk_ucl_ac_uk/EhDP8d9sUh5KkS4nrNsNwgUBLtXHbpsLMdrUGBb3kI12cA?e=JzyqHf