SKILL CLASS RC11

FINALL ASSIGNMENT

VECTORIAL ENCODINGS OF QUALITATIVE DOMAINS

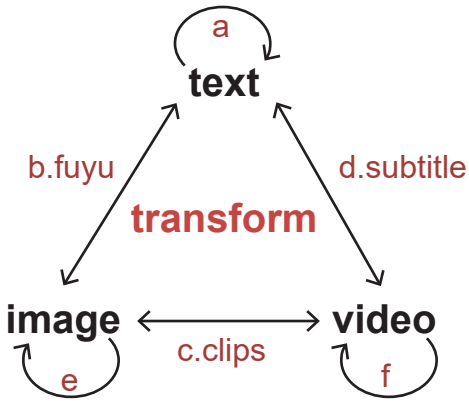
Student Number: 23121106

Github Link: https://github.com/UD-Skills-2023-24/RC11_23121106/tree/main/FinalAssignmentJulian
OneDrive Link: https://1drv.ms/f/c/104eac0089d3ee11/EmnhbjSx9LJMvBvAviCRjfsBmtp8YkX2zzPshQKjevWSsg?e=sqdvqR
Colab Link: https://colab.research.google.com/drive/1VU3M1xXO3sDyzKvLDs8rviVi2fLHAd2p?usp=sharing

---

Overall thinking:

The information between the three domains can be transformed by different models and vectorization methods.



Transformations between same domains:

a. Text & Text: Use models like LSA, TF-IDF to convert text into vectors, and build SOM for searching between texts.
e. Image & Image: Analyze color relationships and other features of images using models, convert them into vectors, and build SOM.
f. Video & Video: Similar to image-to-image, extract key frames at regular intervals from videos, use image retrieval methods to convert videos into vectorized representations for searching.

Transformations between different domains:

b.Text & Image: Use the Fuyu model to convert images into text descriptions, then compare them with other texts for finding related image information.
d.Text & Video: There are two methods: using the Fuyu model to describe video key frames in text or using subtitles as an intermediary to match with text. The latter method was adopted in this assignment.
c.Image & Video: Extract key frames from videos, search between images, and then use these key frames as anchors to reconstruct the entire video.

By establishing these transformation relationships, matching searches can be performed within three domains, enabling cross-domain information retrieval.

The construction approach of the code:

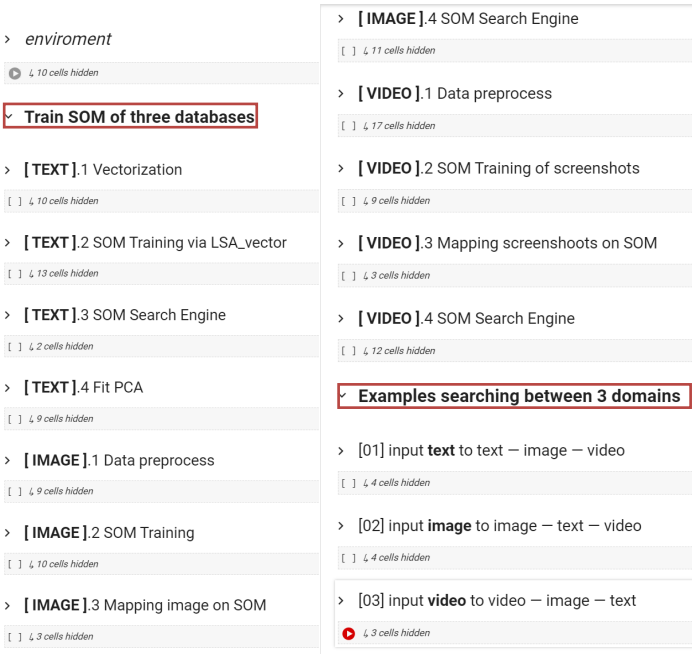
The process is divided into two main parts:

Part One: Data Preprocessing and Vectorization

In this phase, the data undergoes preprocessing and is vectorized. Three types of data (text, images, and videos) are processed separately. Each data type is then used to create a Self-Organizing Map (SOM) or another similarity-based search engine.

Part Two: Three Examples

Three examples are provided, each demonstrating how matching data is obtained when using text, images, or videos as input.



1. Text SOM Training

1.1 Comparison between vectorisation techniques: LSA and TF-IDF

LSA and TF-IDF, as techniques for text vectorization, have different characteristics and suitable for different situation.

TF-IDF, know as its name Term Frequency-Inverse Document Frequency, is a statistical measure used to evaluate the importance of a word in a document, by calculateing the frequency of a term in a document and asjusting it by the term cross all documents. It has the advantage of being simple and efficient. But, it does not capture the semantic meaning of words and it is not good at dealing with synonymy and polysemy.

LSA analyzes terms by producing a set of concepts related to the documents and terms. It is based on SVD)and reduces the dimensionality of the term-document matrix. It can captures the underlying semantic structure ,because it considers the context in which words occur. However, LSA is computationally expensive.

Based on the limited training data of assignment, i chose LSA. LSA can still provide meaningful representations even with a smaller corpus, as it captures the semantic relationships between words and documents, making it more robust in handling small datasets.

```
novel_model_lsa.search('memory in madrid likes a dream', n=5)
```


```
[{'paragraph': "'I wondered whether that was a compliment or a condemnation.'We can't be seen together, Daniel. Not like this, in
full view of everyone.''If you like, we can go into the bookshop. There's a coffeepot in the back room and—''No.",
  'nr': 1567,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'},
 {'paragraph': ' The case, like Salvador Jausa's lips, was sealed forever a few months later. Barcelona's high society observed that
nothing like this had ever happened in the history of the city, and that the likes of rich colonials and other rabble arriving from
across the pond was ruining the moral fibre of the country.',
  'nr': 1619,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'},
 {'paragraph': ' The female, on the other hand - and this is pure science - heats up like an iron, slowly, over a low heat, like a
tasty stew. But then, once she has heated up, there's no stopping her. Like the steel furnaces in Vizcaya.',
  'nr': 879,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'},
 {'paragraph': ' The hand of the angel pierced his chest, spearing him, the accursed soul driven out like black vapour, falling like
frozen tears over the mirror of frozen water.I collapsed then, unable to keep my eyes focused any longer.',
  'nr': 3295,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'},
 {'paragraph': ' 'You like Sugus sweets, don't you?'On our circuitous path back to the exit, we passed the real undertaker and his
two cadaverous assistants carrying a cheap pine coffin, rope, and what looked suspiciously like a recycled shroud.',
  'nr': 1970,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'}]
```


LSA Output

```
novel_model_tfidf.search('memory in madrid likes a dream', n=5)
```

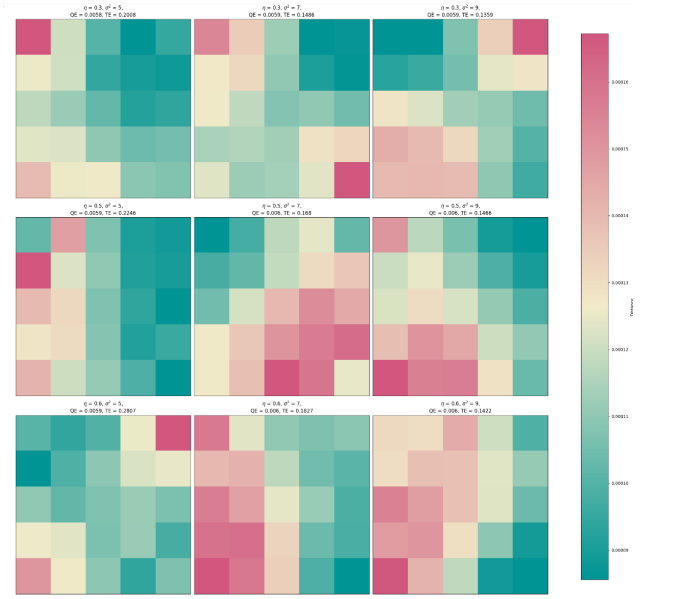

```
[{'paragraph': " We all need friends. Even you.''That kid doesn't have friends and never will. He has the heart of a spider. And if
you don't believe me, time will tell. I wonder what he dreams about. . . ?'Miquel Moliner could not know that Francisco Javier's
dreams were more like his friend Julian's than he would ever have thought possible.",
  'nr': 1477,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'},
 {'paragraph': ' I was stiff from sleeping on the ground. "What did you do? Wake up?" Bill asked. "Why didn\'t you spend the night?"
I stretched and rubbed my eyes. "I had a lovely dream," Bill said. "I don\'t remember what it was about, but it was a lovely
dream.',
  'nr': 713,
  'ID': 'the-sun-also-rises',
  'type': 'epub'},
 {'paragraph': "'For the love of Christ, Master Daniel, have you no shame? Jesus. Mary, and Joseph. Some people never learn . . .'In
her embarrassment Bernarda beat a hasty retreat, and I hoped that once the effects of the brandy wore off, the memory of what she
had seen would also fade from her mind, like the traces of a dream.",
  'nr': 2024,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'},
 {'paragraph': "Dear Julian:This morning I found out through Jorge that you did in fact leave Barcelona to go in pursuit of your
dreams. I always feared that those dreams would never allow you to be mine, or anyone else's.",
  'nr': 942,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'},
 {'paragraph': ' Fermin Romero de Torres was engrossed in the show and the chocolates. When the lights went on at the end of the
film, I felt as if I were waking from a bad dream and was tempted to imagine that the man in the stalls had been a mere illusion, a
trick of memory.',
  'nr': 597,
  'ID': 'The-Shadow-of-the-Wind',
  'type': 'epub'}]
```

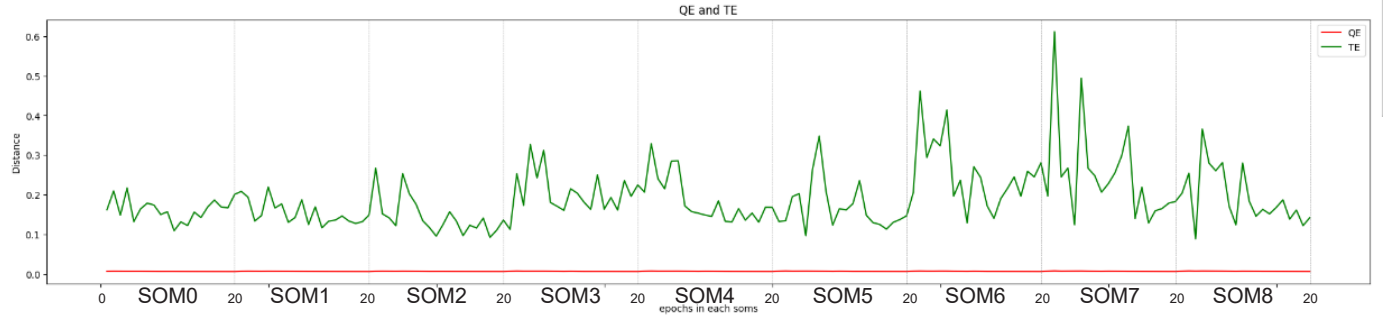

TF-IDF Output

1.2 Train SOM

I trained 9 SOMs using different combinations of learning rates and radii, recording the values of QE and TE for each SOM during training epochs from 0 to 20. Then, I plotted these values on a line chart for visualization.

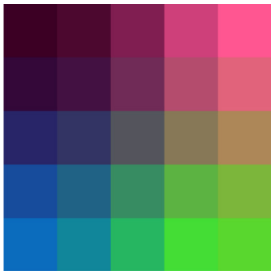
From these charts, I selected the SOM with the lowest TE and QE values as my final choice. This selection process took into account not only the topological performance of the SOM but also its performance in data representation and spatial mapping.



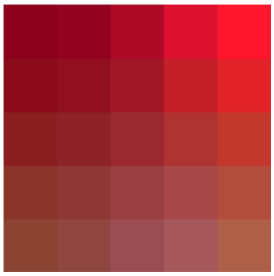


1.3 Fit PCA in Different Way

Fit PCA on all cells of the SOM:
For this approach, PCA is performed directly on the vectorized representations of all cells in the SOM grid. Each cell's vector representation is considered independently. PCA is applied to reduce the dimensionality of each cell's vector representation to three components, which are then used as the RGB values for visualization. So, it may capture more local variations within each cluster represented by a cell. But, it may not capture the overall structure of the data as effectively, especially if some cells have fewer data points or if there is high variance between cells.



Fit PCA on the entire training set:
In this method, PCA is applied to the entire training set and then applying it to each cell of the SOM, captures the global structure of the data. By using PCA on the entire training set, it considers the relationships between all data points and provides a more holistic understanding of the data distribution. However, it may not capture local variations within each cell as effectively as the first approach.



1.4 Built text SOM searching engine

I define a function to activate som and return the top n results with the highest similarity in the BMU. It displays information about the active SOM and text.

```python
def search_textSOM(sentence, n=5):
    print("query:", sentence)

    # Vectorize the input sentence
    vectorized_sentence = np.squeeze(novel_model_lsa.vectorize(sentence))

    # Activate the SOM and find the best matching unit
    activated_SOM = SOMlib.activate(SOM_t, vectorized_sentence)
    top_index = np.argmax(activated_SOM)
    g, h = SOMlib.find_BMU(SOM_t, vectorized_sentence)
    top_coordinate = (g, h)

    # Plot the activated SOM
    fig, ax = plt.subplots(figsize=(3, 3))
    plt.imshow(activated_SOM, cmap=cmap_diverging, interpolation='none')
    plt.axis('off')
    plt.show()

    #print("best matched unit:", top_coordinate, "\n")

    # Extract the best-matched n paragraphs of the most matched cells
    SOMcontents = []
    train_data = novel_model_lsa.vector_matrix
    for i in range(len(SOM_t)):
      row = []
      for j in range(len(SOM_t[0])):
        row.append([])
      SOMcontents.append(row)
    for f, fi in zip(train_data, novel_model_lsa.paragraphs):
        g, h = SOMlib.find_BMU(SOM_t, f)
        SOMcontents[g][h].append(fi)

    matched_paragraphs = []
    similarities = {}

    # Calculate the cosine similarity between text paragraphs and the input sentence
    for paragraph_info in SOMcontents[g][h]:
        paragraph = paragraph_info['paragraph']
        paragraph_info_copy = paragraph_info.copy()
        # Vectorize the paragraph
        paragraph_vector = np.squeeze(novel_model_lsa.vectorize(paragraph))
        # Calculate cosine similarity
        similarity_score = cosine_similarity([vectorized_sentence], [paragraph_vector])
        # Store similarity score
        similarities[paragraph] = (paragraph_info_copy, similarity_score)

    # Select the most similar text paragraphs
    sorted_paragraphs = sorted(similarities.items(), key=lambda x: x[1][1], reverse=True)
    matched_paragraphs = [(paragraph_info, similarity_score) for paragraph, (paragraph_info, similarity_score) in sorted_paragraphs
    print(matched_paragraphs)

    return matched_paragraphs
```

Example usage:

query : python is a difficult skill

[({'paragraph': ""You touch a single hair of his and I swear I'll—"You scare me to bits, really. I just shat my pants.'Fermin swallowed, as if to hold in all the courage that was seeping out of him. 'Those wouldn't be the same sailor-boy pants that your esteemed mother, the Illustrious Kitchen Maid, made you wear? That would be a shame; I'm told the outfit really suited you.", 'nr': 1985, 'ID': 'The-Shadow-of-the-Wind', 'type': 'epub'}, array([[0.55058058]]))]

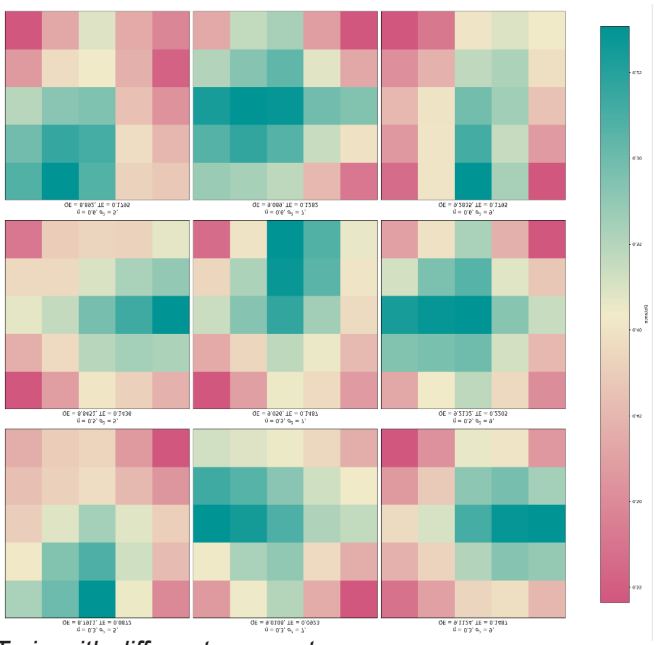
2. Image SOM Training

2.1 Data preprocess

On the one hand, use the fuyu model to convert images into descriptive text, store them in csv, and extract them into dictionaries in python.
fuyu code link: https://colab.research.google.com/drive/18r6YFqWyUDLmYri99wx-UllBt7xferHW?usp=sharing
fuyu results link: https://github.com/UD-Skills-2023-24/RC11_23121106/tree/main/FinalAssignmentJulian/database/fuyu_image
On the other hand, MobileNet model was used to extract image features.

2.2 Train SOM

Similar to the text, only the results are shown.



Train with different parameters

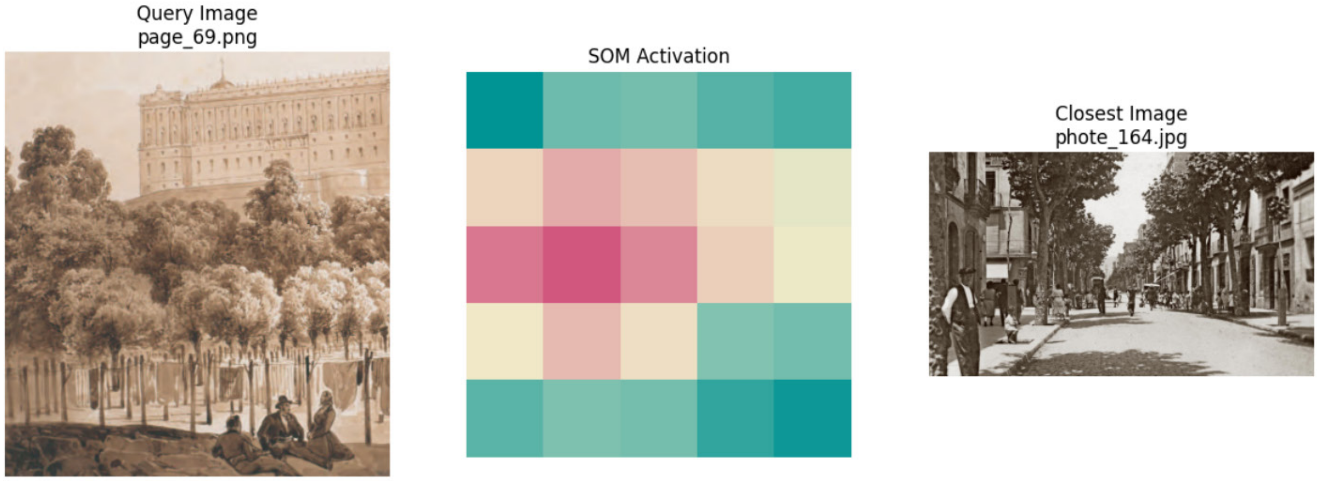
Map the image on the SOM

2.3 Built image SOM searching engine

Input image:
I defined a function that takes the path of the input image and returns the active som, the most similar image, and the fuyu description.

```
def search_imageSOM(query_image_Path):

    query_image_name = os.path.basename(query_image_Path)
    path_prefix = "/content/drive/MyDrive/Fianl_Julian/database/image/"
    query_feature = processImage(query_image_Path, model)

    activatedSOM = SOMEuc.activate(train_data, SOM_i, query_feature)
    g, h = SOMEuc.find_BMU(SOM_i, query_feature)
    #print(g,h)
    closest_image_index = get_closest_image(g, h)
    closest_image_path = path_prefix + cell[closest_image_index]['image']
    #print(cell[closest_image_index])

    query_image = mpimg.imread(query_image_Path)
    closest_image = mpimg.imread(closest_image_path)

    # Display the images
    fig, axs = plt.subplots(1, 3, figsize=(15, 5))
    axs[0].imshow(query_image)
    axs[0].set_title('Query Image\n' + query_image_name)
    axs[0].axis('off')
    axs[1].imshow(activatedSOM, cmap=cmap_diverging)
    axs[1].set_title('SOM Activation')
    axs[1].axis('off')
    axs[2].imshow(closest_image)
    axs[2].set_title('Closest Image\n' + cell[closest_image_index]['image'])
    axs[2].axis('off')
    plt.show()
    print("Description of the closest image:", cell[closest_image_index]['description'])

    return cell[closest_image_index]
```

```
# Example usage:
query_image_Path = "/content/drive/MyDrive/Fianl_Julian/database/input_image/page_69.png"

closest_info = search_imageSOM(query_image_Path)
```




```
Description of the closest image: A group of men, women, and children standing and walking on a sidewalk
```

Input text:
Use TextModel.Search that accepts text query and returns the most similar image, and the fuyu description.

```
[ ] CSV_model = TextModel(CSV_FILES,vectorization='lsa',min_df=3)

    fuyu_search_result = CSV_model.search('memory in madrid likes a dream', n=3)

[ ] show_fuyu_results(fuyu_search_result)
```

```
file name: phote_138.jpg
Paragraph: In the picture, there is a large crowd of people standing next to a lamppost. Among them, a man is hold

---------------
```

3. Video SOM Training

3.1 Data preprocess

First, frames of the movie is extracted at equal intervals, saved as a picture, and then som will do the frame SOM.
Use MobileNet model to extract frames features.

Second, extract the information of subtitles from .srt files.
Merge subtitle information with frames information.

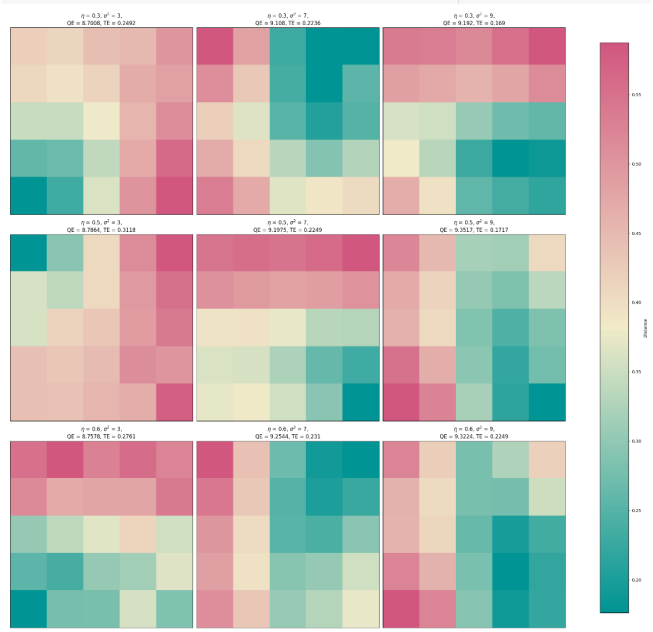
Finally, i got:

```
clipsPairsFeature[24:27]
```

```
[{'frame_count': 2880,
  'image': 'All_About_My_Mother_2880.jpg',
  'film_name': 'All_About_My_Mother.mkv',
  'paragraph': ' ',
  'feature': array([0.54660475, 0.11187976, 0.86779994, ..., 0.2247148 , 1.2517923 ,
         0.69793344], dtype=float32)},
 {'frame_count': 3000,
  'image': 'All_About_My_Mother_3000.jpg',
  'film_name': 'All_About_My_Mother.mkv',
  'paragraph': "I'm going to call. ",
  'feature': array([1.6628255 , 0.11484616, 0.17633484, ..., 0.9095515 , 2.4134054 ,
         0.6958458 ], dtype=float32)},
 {'frame_count': 3120,
  'image': 'All_About_My_Mother_3120.jpg',
  'film_name': 'All_About_My_Mother.mkv',
  'paragraph': 'TRANSPLANT COORDINATION ',
  'feature': array([1.1568251 , 0.22795494, 0.5901655 , ..., 0.5252316 , 2.8821852 ,
         0.28715444], dtype=float32)}]
```

3.2 Train SOM

Similar to the image, only the results are shown.



3.3 Built image SOM searching engine

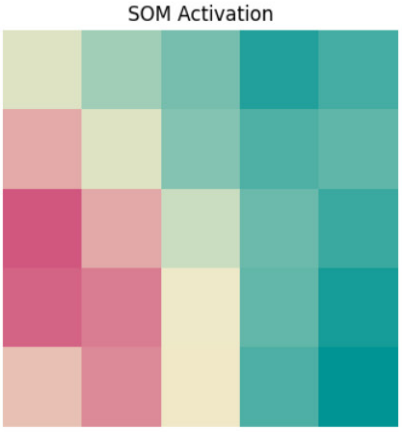
Input image:

Defined a function to extract a clip from a fram.

Set searching function that takes the path of the input image and returns the closet frame, subtitle (if it contains) and download the clips of film (due to Colab can not directly show video in the output area).

```
def search_videoSOM(query_image_Path, outputname):
    query_image_name = os.path.basename(query_image_Path)
    path_prefix = "/content/drive/MyDrive/Fianl_Julian/database/film_screenshot/"
    query_feature = processImage(query_image_Path, model)

    train_data = screenshot_features
    activatedSOM = SOMEuc.activate(train_data, SOM_v, query_feature)
    a, b = SOMEuc.find_BMU(SOM_v, SOMEuc.normalise(train_data, query_feature))
    #a, b = SOMEuc.find_BMU(SOM_v, query_feature)
    #print(a,b)

    closest_image_index = get_closest_frame(a, b)
    closest_image_path = path_prefix + cell_frame[closest_image_index]['image']
    #print(closest_image_path)

    query_image = mpimg.imread(query_image_Path)
    closest_image = mpimg.imread(closest_image_path)

    # Get the closest image frame count
    closest_frame_count = cell_frame[closest_image_index]['frame_count']

    # Load the film
    film_path_prefix = "/content/drive/MyDrive/Fianl_Julian/database/video/"
    film_name = cell_frame[closest_image_index]['film_name']
    video_path = film_path_prefix + film_name

    # Calculate end frame
    end_frame = closest_frame_count + 120

    # Output path with unique filename based on query image name and timestamp
    output_filename = f"{outputname}_{query_image_name}_{closest_frame_count}_{end_frame}.mp4"
    output_path = os.path.join("/content/drive/MyDrive/Fianl_Julian/output_clips/", output_filename)

    extract_video_clip(video_path, closest_frame_count, end_frame, output_path)

    return cell_frame[closest_image_index]
```

```
def extract_video_clip(video_path, start_frame, end_frame, output_path):

    cap = cv2.VideoCapture(video_path)
    cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

    fps = cap.get(cv2.CAP_PROP_FPS)
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    fourcc = cv2.VideoWriter_fourcc(*'MP4V')
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    for i in range(start_frame, end_frame):
        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
        ret, frame = cap.read()
        if not ret:
            break
        out.write(frame)

    cap.release()
    out.release()

    print(f"Video clip extracted and saved to {output_path}")
```




```
subtitle:
Video clip extracted and saved to /content/drive/MyDrive/Fianl_Julian/output_clips/test_page_216.png_62880_63000.mp4
```

Input text:

Use TextModel.Search that accepts text query and returns the most similar frame, subtitle and downloand the clip.

```
test_search_film_results = pkl_text_model.search('night like a dream', n=3)
```

```
show_frame_results(test_search_film_results,"testsubtitle")
```

```
frame: All_About_My_Mother_111120.jpg
Paragraph: If I bore you, you can pretend to snore... like this.
Video clip extracted and saved to /content/drive/MyDrive/Fianl_Julian/output_clips/testsubtitle_1111
---------------
```

```
frame: All_About_My_Mother_130440.jpg
Paragraph: or what he's like...  or how he behaved towards her.
Video clip extracted and saved to /content/drive/MyDrive/Fianl_Julian/output_clips/testsubtitle_1304
---------------
```

4. Examples Searching between 3 Domains

4.1 input text to text — image — video

01. After vectorization, text similarity is matched directly to find the most matched paragraph
02. Match the paragraph with the yufu description of the picture to get the matching picture
03. Activate the matching image in clips som, find the BMU and get the movie slice

Finally, i got:

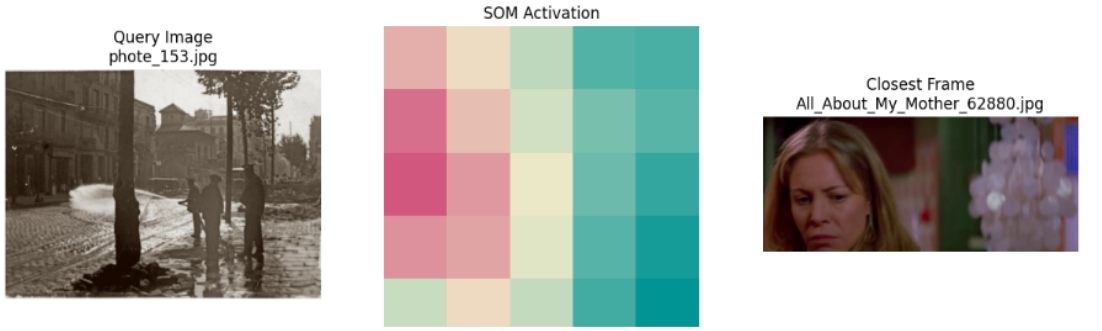
```python
def from_text_search(query, outFilmName):
    # text to text
    query_1 = search_textSOM(query, n=1)[0][0]['paragraph']

    # text to image
    imagename = show_fuyu_results(CSV_model.search(query_1, n=1))

    # Construct query path
    query_2 = f"/content/drive/MyDrive/Fianl_Julian/database/image/{imagename}"

    # image to video
    test_search_film_info = search_videoSOM(query_2, outFilmName)

    return test_search_film_info
```

```python
n = 1
for query in query_list:
  print(f"\n ------[ SEARCHING EXAMPLE {n} ]----------------------------------
  n += 1
  from_text_search(query, "example01")
```

```
query: The sun rose over the horizon, casting its golden rays across the tranquil meadow, as birds chirped in harmony, welcoming the new day.
```

```
best matched unit: (4, 2)

[({'paragraph': ' You could not be upset about anything on a day like that. That was the last day before the fiesta. 15 At noon of Sunday, the 6th of July, the fiesta exploded. There is no other w
```

```
file name: phote_153.jpg
Paragraph: In the picture, three men are standing on a street corner, possibly on a rainy day. They are standing next to a pole and a tree, possibly discussing something or waiting for someone.

---------------
1/1 [==============================] - 0s 57ms/step
```



* Examples of the following two paths are not shown here, details can be found in the ipynb file

4.2 input image to image — text — video

01. image is activated in image SOM to get the matching image
02. Activate text SOM with the fuyu description of the matching image to get the most matched paragraph
03. Search for movie subtitles with matching paragraphs to get clips

4.3 input video to video — image — text
01. Match the features of the 2 videos to get similar clips
02. Use the key frame of similar clips to search the image SOM and get the matching picture
03. Activate text SOM with the fuyu description of matching image to get the corresponding paragraph

som outputs of clips
Github Link: https://github.com/UD-Skills-2023-24/RC11_23121106/tree/main/FinalAssignmentJulian/output_clips

5. Use Separate .py Files with Classes

Github Link: https://github.com/UD-Skills-2023-24/RC11_23121106/tree/main/FinalAssignmentJulian/model

SOM.py and SOM_Euclid.py are used to train SOMs. The distinction lies in the definition of functions: one calculates the Euclidean distance, while the other computes the cosine distance. In this assignment, I employed cosine distance to train SOMs for text data.Because cosine similarity can better reflect semantic similarity when processing such data, it is more suitable for modeling and searching textual data.
And Euclidean distance for SOMs trained on images and frames extracted from videos. Because it is more able to capture the geometric distance relationship of samples in the feature space.

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

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

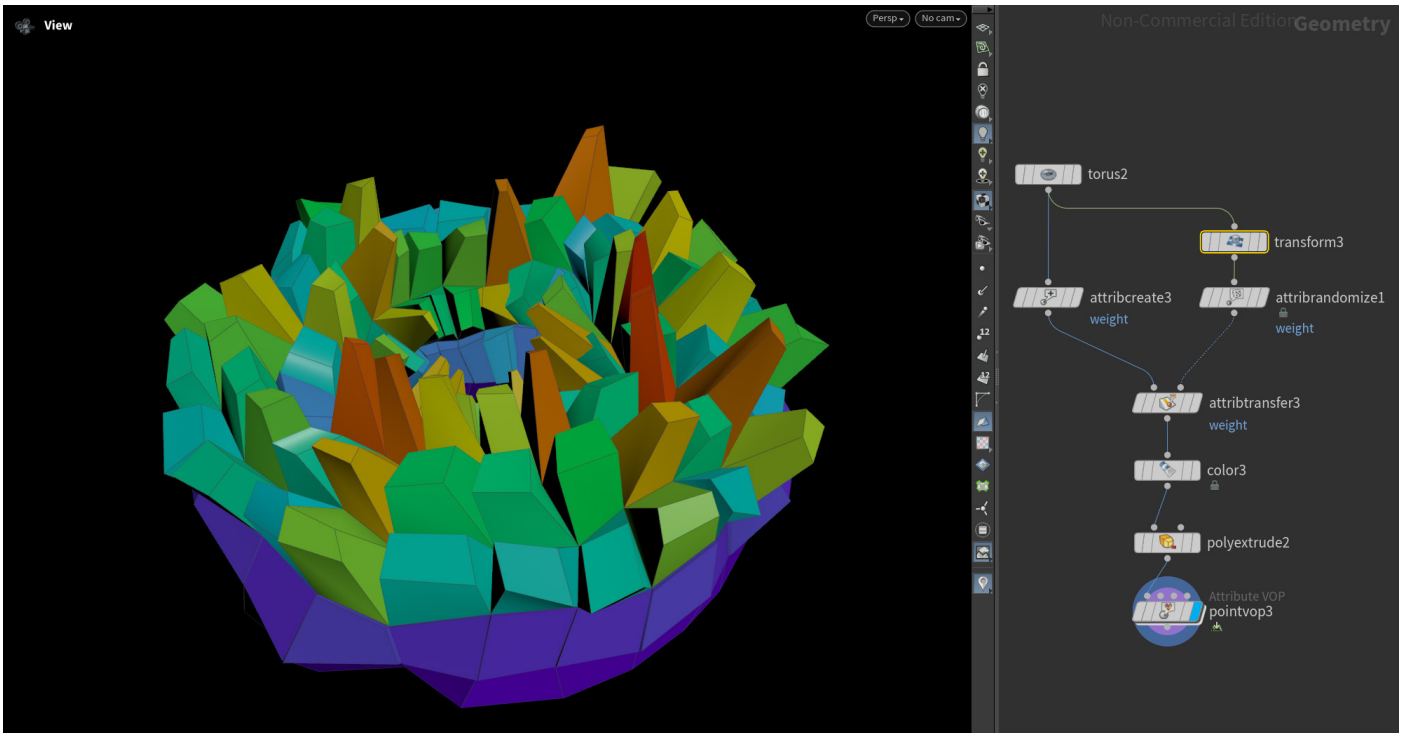
text_model.py is used to deal with text data, it can read different types of text files (such as pdf, epub, csv, str), split the paragraph. And can choose two techniques (lsa, tfidf) to vectorise the data and achieve the function of searching between the text.

```python
class TextModel:
    def __init__(self, files, vectorization='lsa', dimension=200, min_df=2):
        self.vectorization = vectorization
        self.paragraphs = []

        for f in files:
            ID = os.path.splitext(os.path.basename(f))[0]
            filetype = f.split('.')[-1]
            if filetype == 'epub':
                paragraph = read_epub_paragraphs(f, ID, 'epub')
            elif filetype == 'pdf':
                paragraph = read_pdf_paragraphs(f, ID, 'pdf')
            elif filetype == 'csv':
                paragraph = read_csv_paragraphs(f, ID,'csv')
            elif filetype == 'pkl':
                paragraph = read_pkl_paragraphs(f)
            else :
                print("This file type cannot be read")
            self.paragraphs.extend(paragraph)


        self.preprocessed_paragraphs = preprocess(p['paragraph'] for p in self.paragraphs)


        if self.vectorization == 'tfidf':
            self.tfidf_vectorizer = TfidfVectorizer(min_df=min_df)
            self.vector_matrix = self.tfidf_vectorizer.fit_transform(self.preprocessed_paragraphs)
        elif self.vectorization == 'lsa':
            self.tfidf_vectorizer = TfidfVectorizer(min_df=min_df)
            self.tfidf_matrix = self.tfidf_vectorizer.fit_transform(self.preprocessed_paragraphs)
            self.svd = TruncatedSVD(n_components=dimension, algorithm='randomized')
            self.vector_matrix = self.svd.fit_transform(self.tfidf_matrix)

        self.nnModel = NearestNeighbors(n_neighbors=10,
                                        metric='cosine',
                                        algorithm='brute',
                                        n_jobs=-1)
        self.nnModel.fit(self.vector_matrix)
```

RC11 Houdini Assignment
23121106

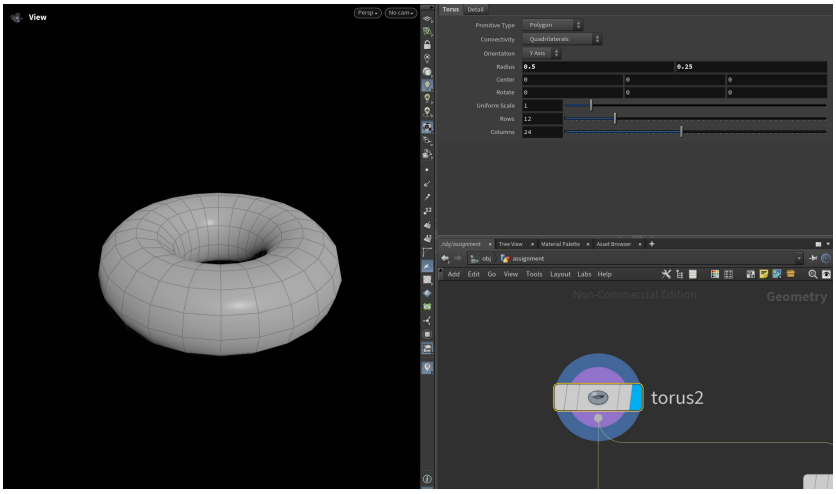
Github Link: https: https://github.com/UD-Skills-2023-24/RC11_23121106/tree/main/FinalAssignmentJoris
OneDrive Link: https://1drv.ms/f/c/104eac0089d3ee11/ErgZTtjPImhOl1tfOFOAHLABCT-CASBRoFR0OQ0v0iFIbw?e=fjR4Rk

1 Houdini Fundamentals
[Example 1]



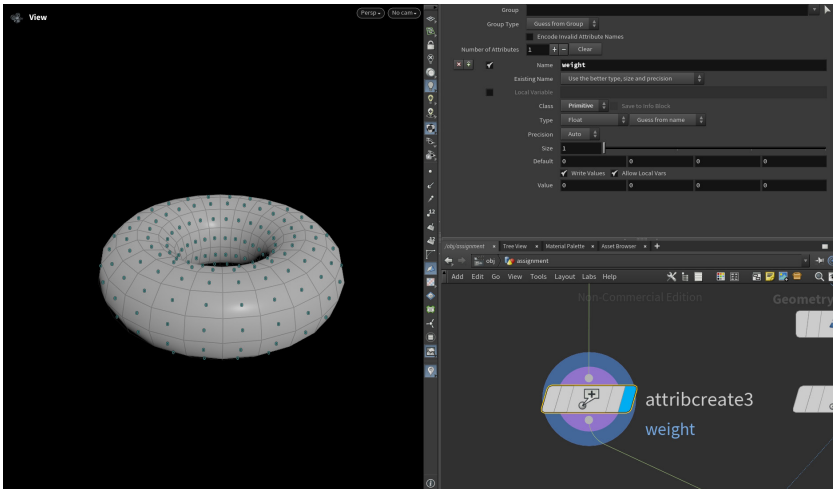
1: Torus SOP

This SOP is a geometry node that generates a torus shape.



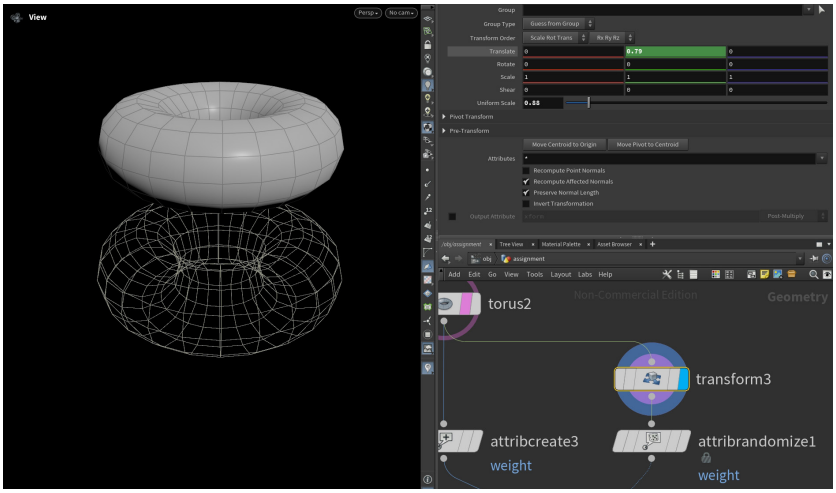
2: AttribCreate SOP

This SOP is a node used to create or modify attributes on geometry. It adds the specified attribute, "weight", to the geometry with the initial values "0.0". This attribute can then be used in downstream nodes for various purposes such as shading, deformation, or simulation.



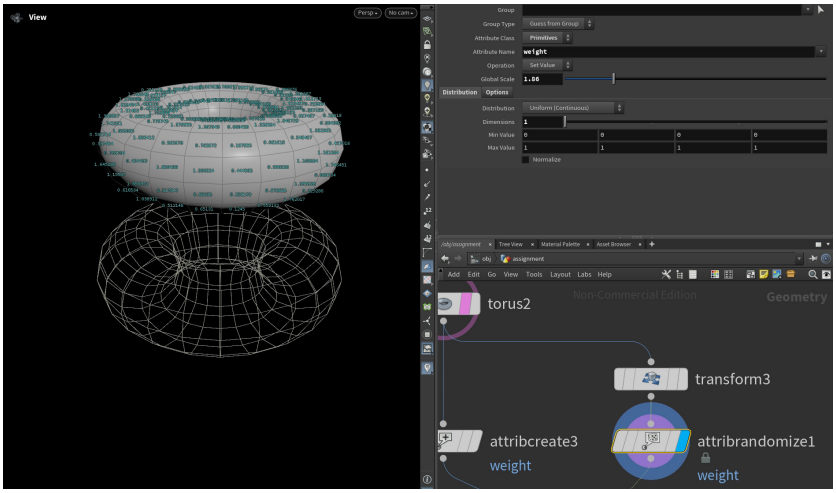
3: Transform SOP

The Transform SOP in Houdini is a node used for translating, rotating, scaling, or generally transforming geometry in 3D space. In this step, I move the original torus up by 0.79 units in the Y-axis using the Transform SOP.



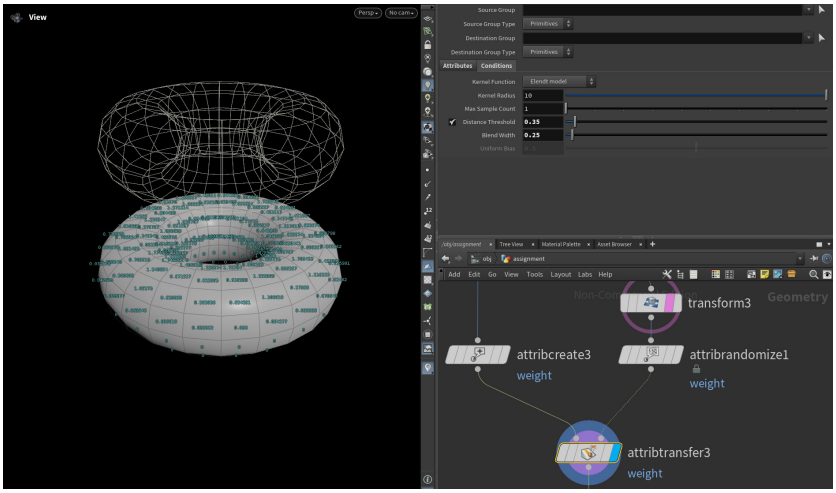
4: Attribrandomize SOP

It is a node used to assign random values to attributes on geometry.I gave the attribute "weight" a new global numerical value. These values can be useful for various purposes such as procedural modeling, simulation, or shading.



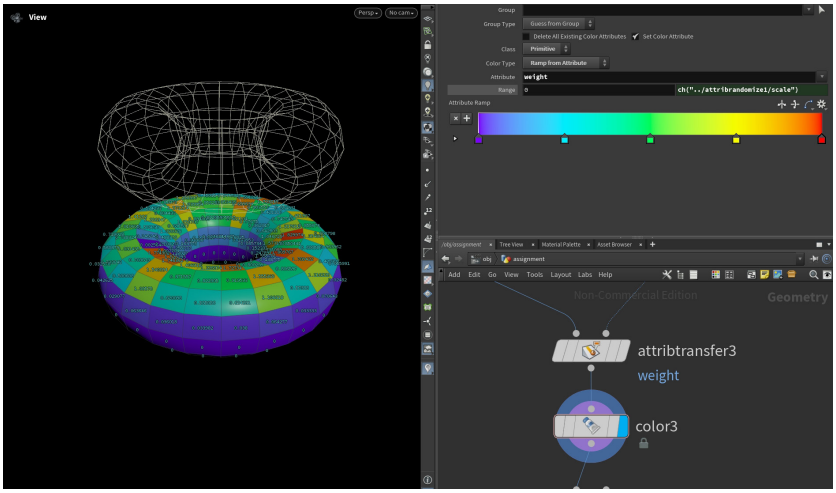
5: AttribTransfer SOP

It is a node used to transfer attribute values from one piece of geometry to another based on their spatial relationships. By connecting the output of the AttribCreate and AttribRandomize nodes to the input of the AttribTransfer SOP, I transfered the "weight" attribute from this original geometry to another geometry.



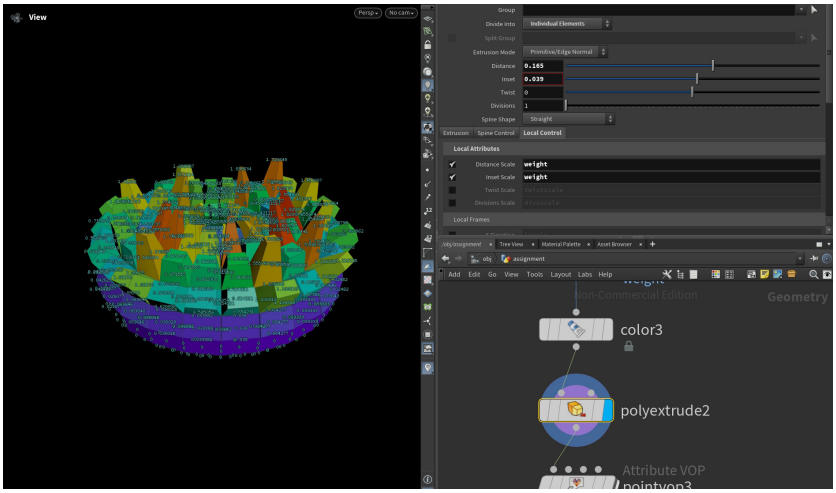
6: Color SOP

It is a node used to assign color attributes to geometry based on various criteria. I colored the "weight" attribute using the Color SOP in order to assigned colors to the geometry based on the values of the "weight" attribute, making it easier to interpret or visualize the data.



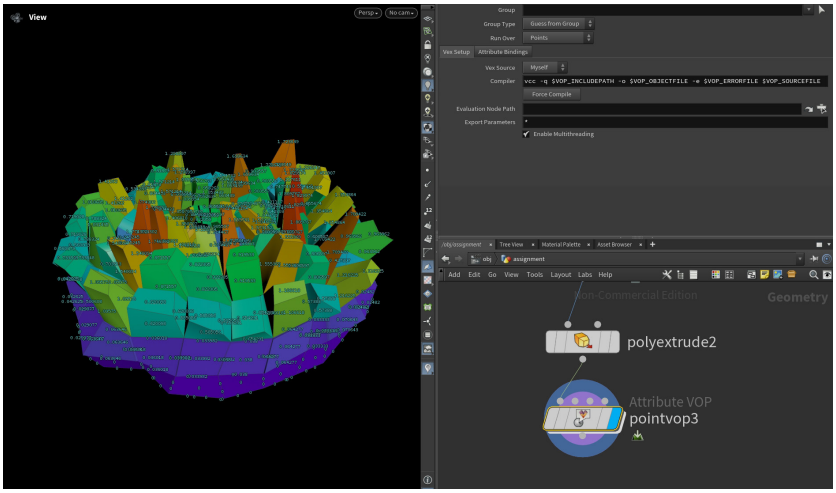
7: PolyExtrude SOP

It is a node used for extruding geometry, creating depth and volume by pushing faces, edges, or vertices outward. I extruded individual elements based on the "weight" attribute. Elements with higher "weight" values will be extruded further and with a larger inset, vice versa.

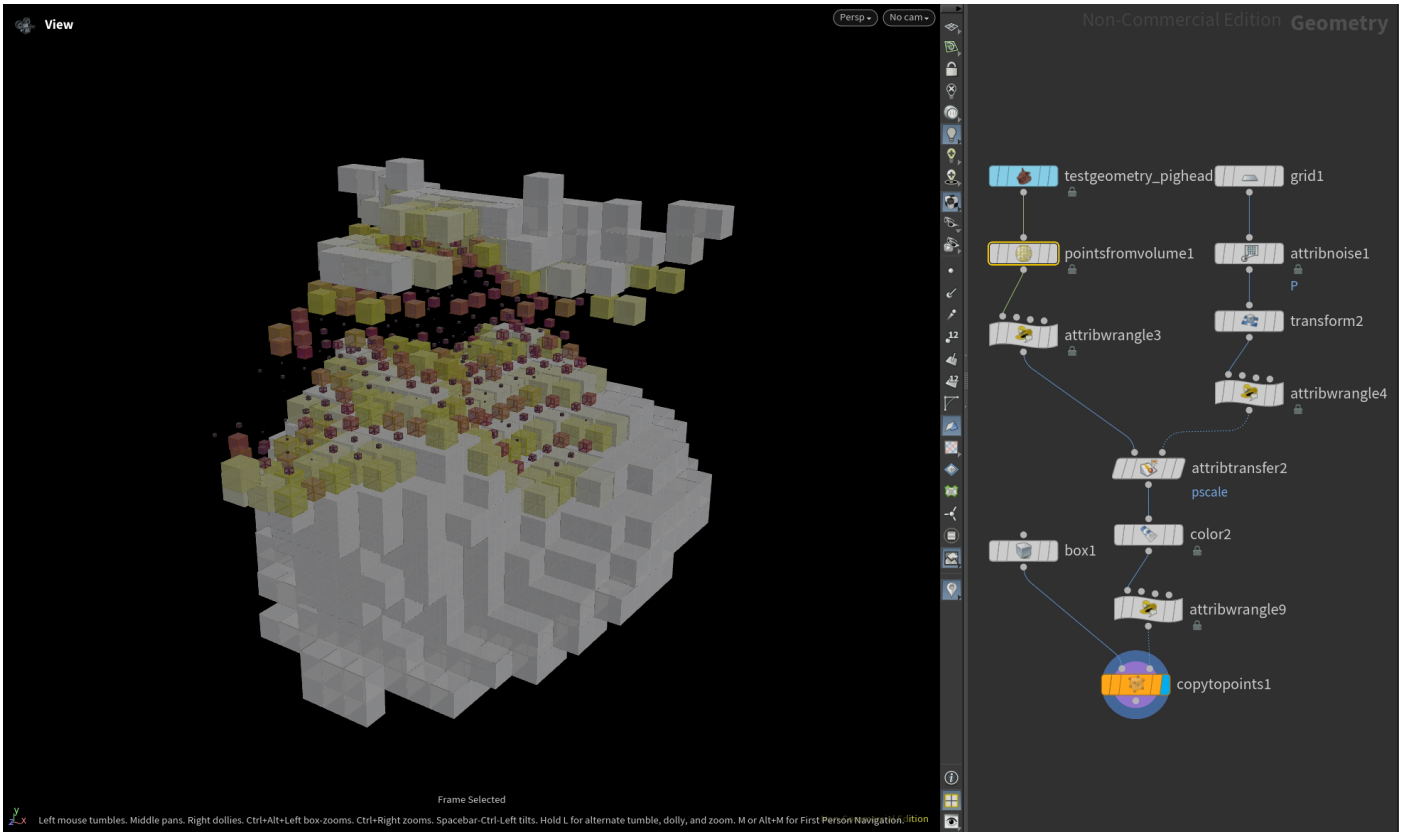


8: Point VOP SOP

It is a node that allows you to manipulate point attributes using a visual programming interface. I added noise to the entire geometry using the PointVOP SOP.

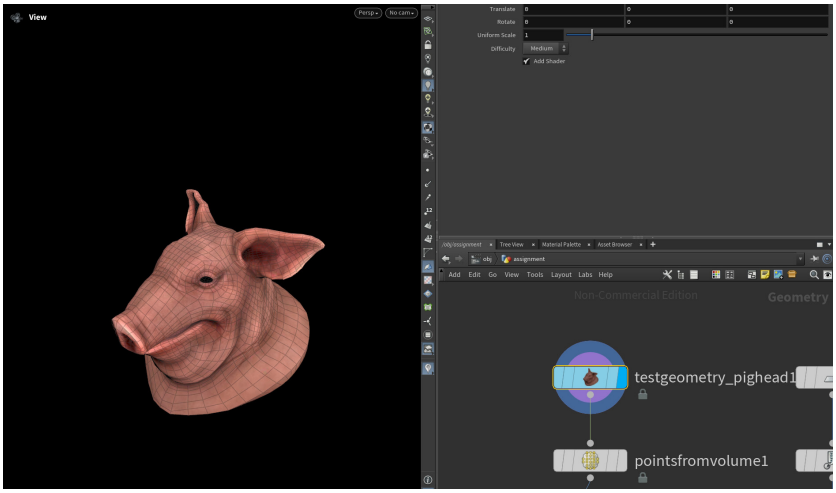


[Example 2]



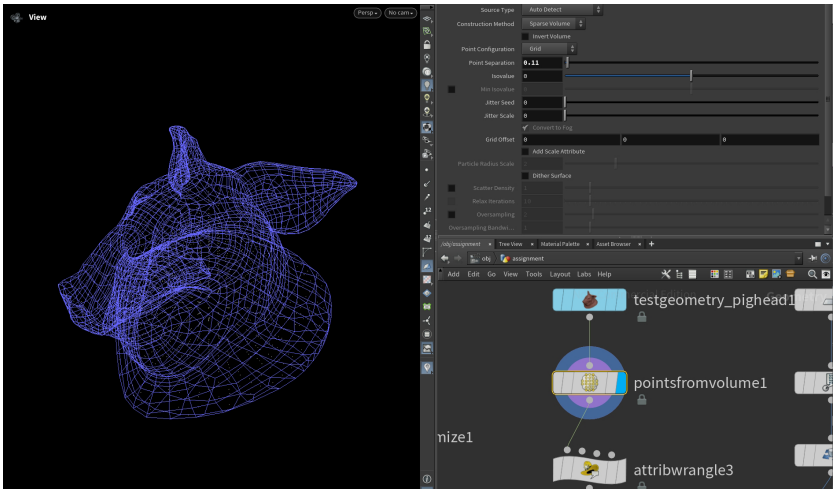
1: TestGeometry_Pig SOP

It is a node that creates a basic pig-shaped model that can be used for various tasks such as experimenting with different modeling techniques.



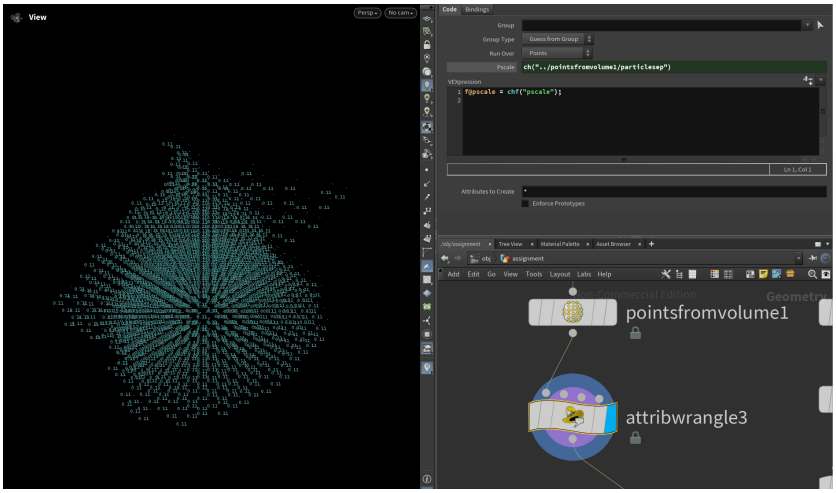
2: PointsfromVolum SOP

It is a node used to generate point clouds from volumetric data. I used the SOP to transfer a pig to points, converting the pig-shaped geometry into a point cloud representation based on the volume of the pig's shape.



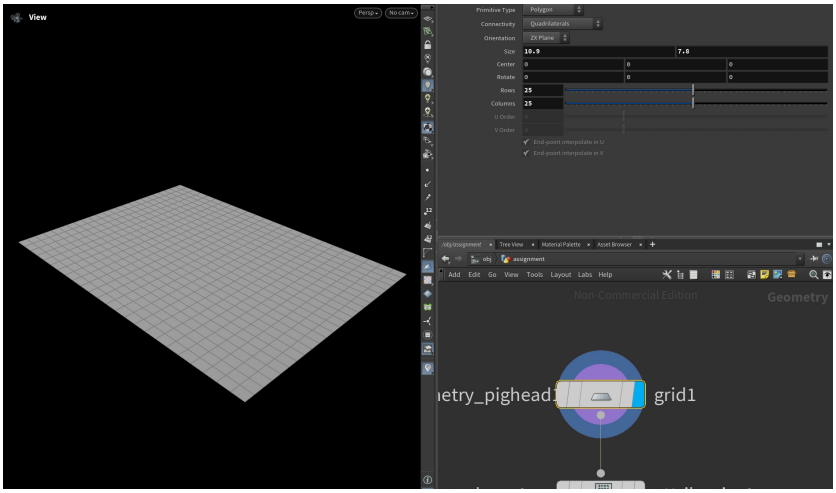
3: AttribWrangle SOP

It is a node used for manipulating attributes on geometry using VEX language. The expressionsets the "pscale" attribute of each point in the geometry to the value of the "pscale" channel parameter. So, I can control the size of individual points in the geometry by adjusting the "pscale" parameter in the node's parameters panel.



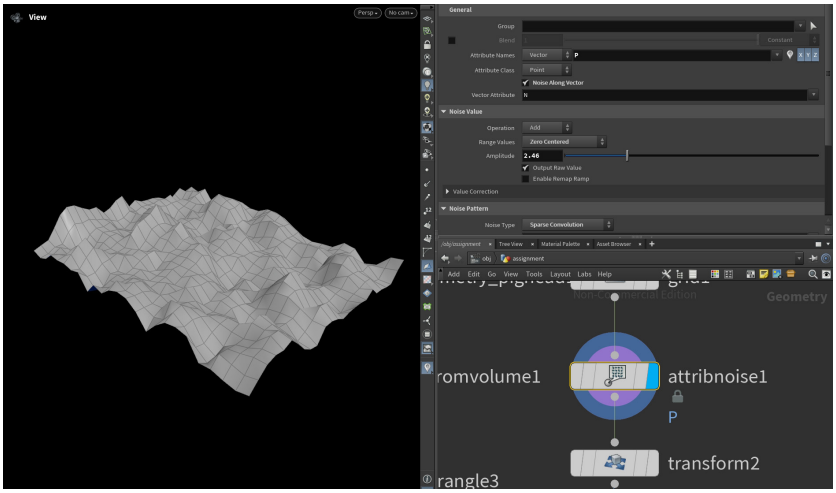
4: Grid SOP

It is a node used to generate a flat grid of quadrilateral polygons. Creat a new geometry with the Grid SOP in order to transfer points' scale (pscale) later.



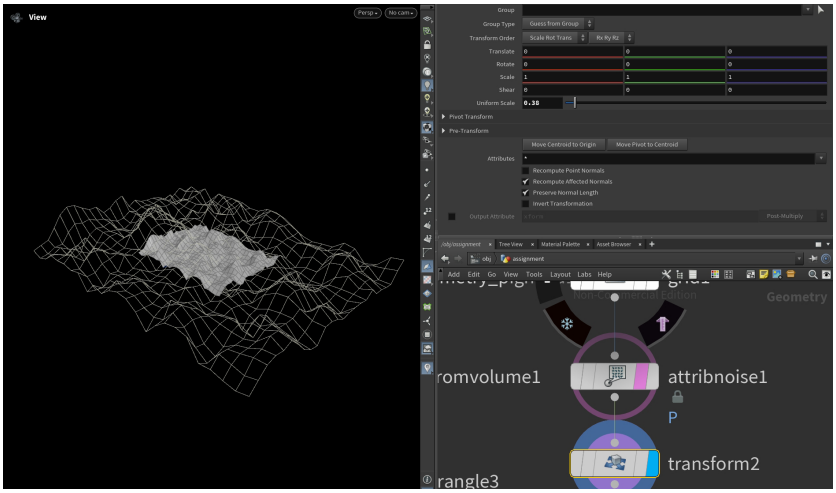
5: Attribnoise SOP

It is a node used to add noise to attributes on geometry, adding variation and randomness to attributes.



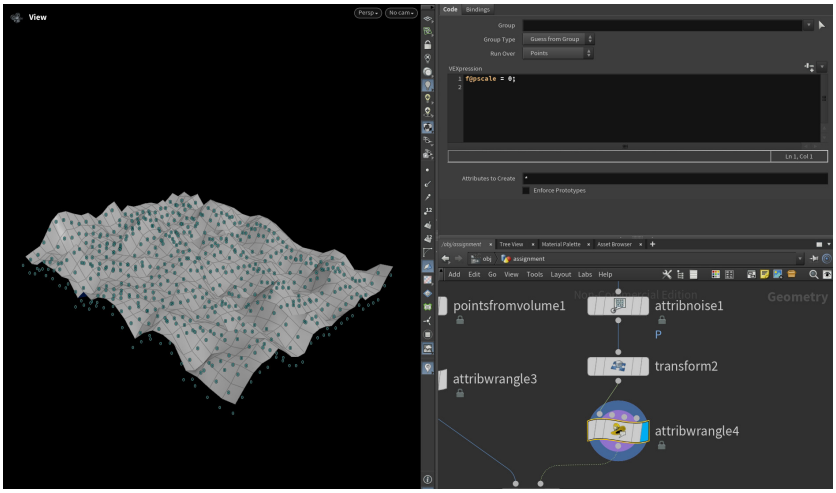
6: Transform SOP

The Transform SOP is used to shrink a noised grid, the goal is to adjust its size and shape to better fit and complement another piece of geometry, the pig's head. Additionally, enhance the visibility of the noise effect, making the undulations more pronounced.



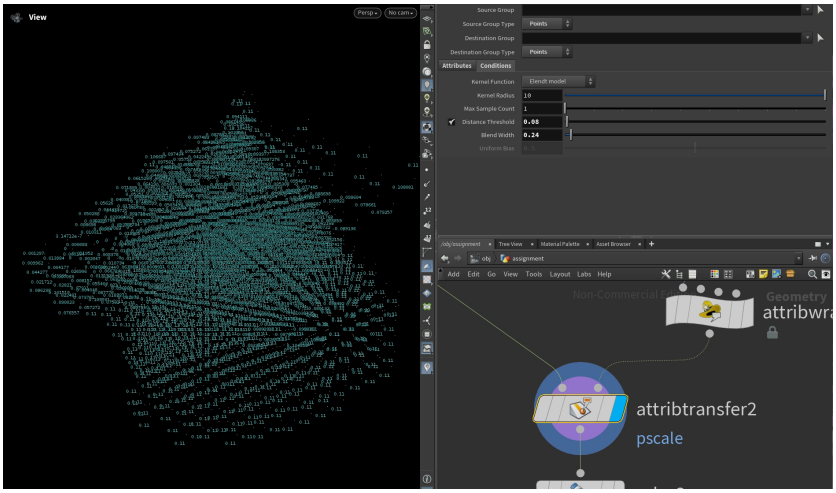
7: AttribWrangle SOP

Manipulate attributes on geometry using VEX (Vector Expression) language to set the "pscale" attribute of all points in the noised grid to 0.



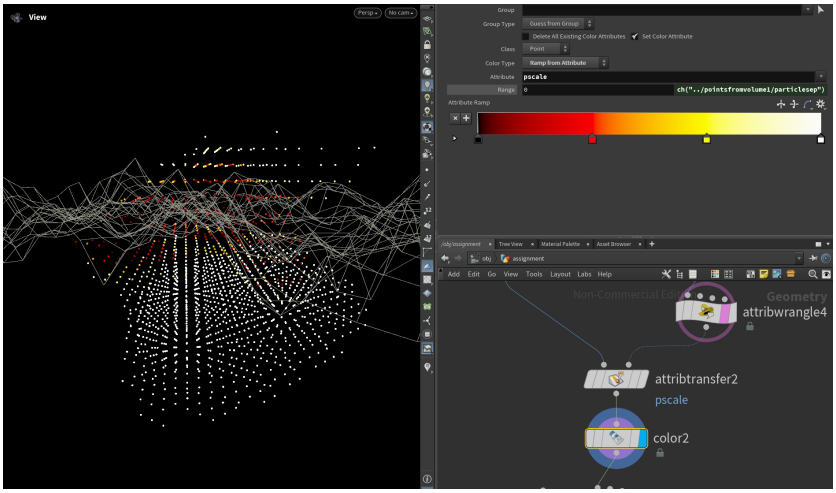
8: AttribTransfer SOP

The AttribTransfer SOP in Houdini is used to transfer attributes between two sets of geometry. I transfered the "pscale" attribute from the grid to the pig, allowing me to control the scale of the pig based on the vale of "pscale" of the grid geometry.



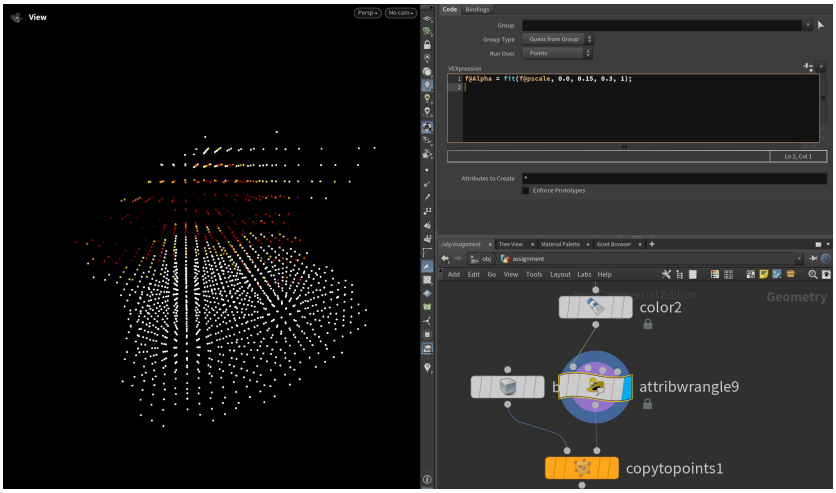
9: Color SOP

Assign colors to geometry, using the "pscale" attribute to determine the colors of the geometry.



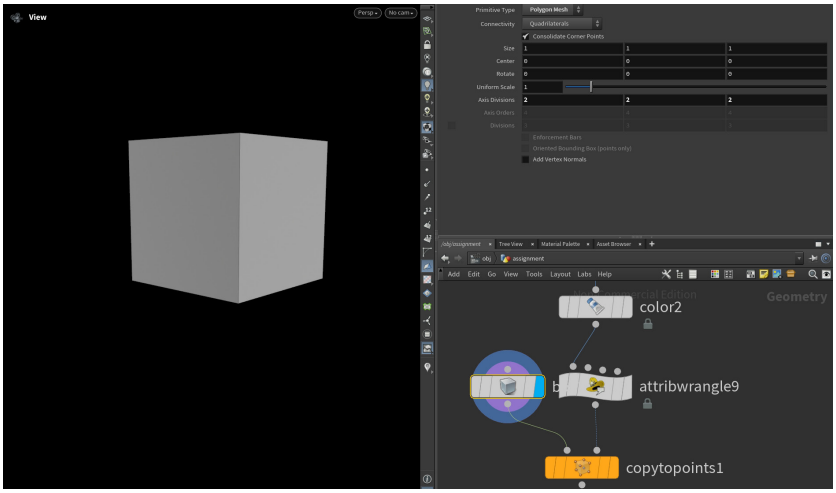
10: AttribWrangle SOP

Manipulate attributes on geometry. Remap the values of the "pscale" attribute to a new range and assign the result to the "Alpha" attribute. So, smaller points might have lower transparency (closer to 0.3), while larger points might have higher transparency (closer to 1).



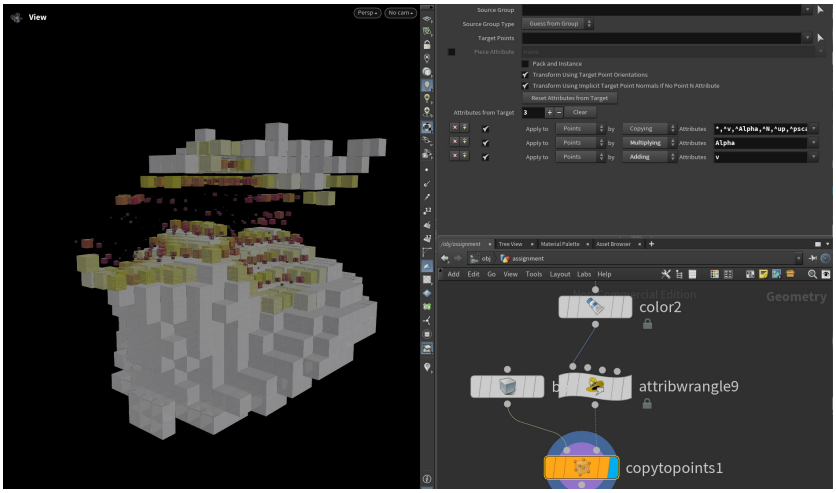
11: Box SOP

Create a simple box-shaped geometry for further operations.



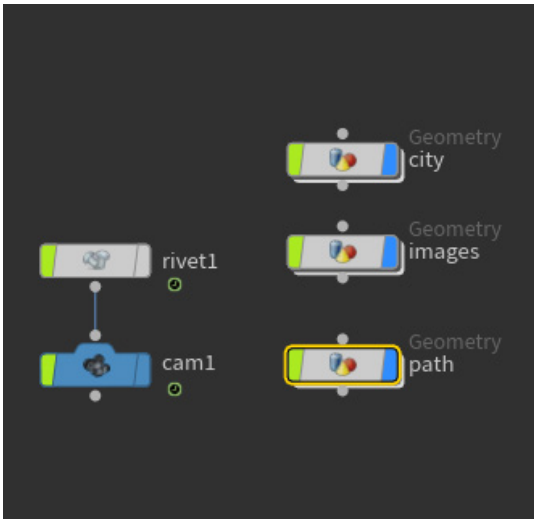
12: CopytoPoints SOP

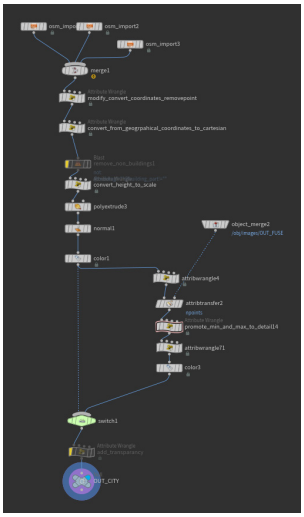
This node is is used to duplicate geometry onto the points of another piece of geometry. I placed copies of the box onto each point of the pig.

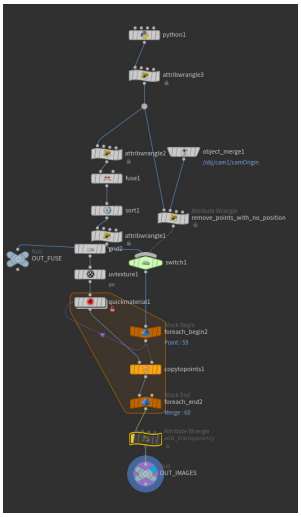


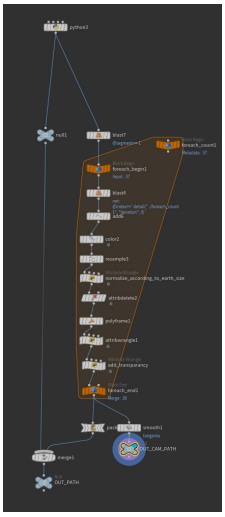
2 Visualizing GPS and Image Metadata

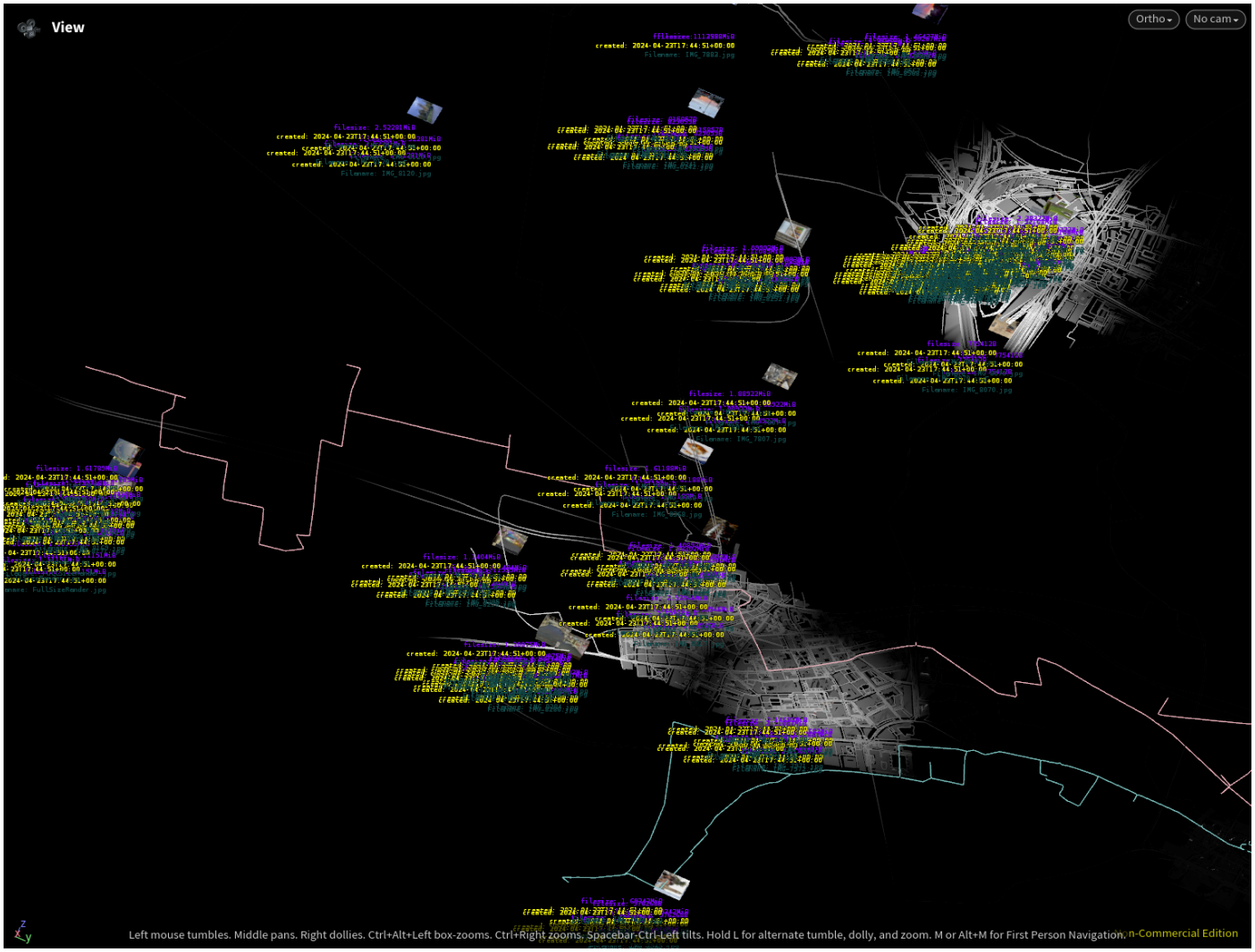
For this assignment, I chose London as the focal point.
Within Houdini, I imported images with geolocation data, along with London's OMS file and GPS data. The objective was to showcase a day's worth of my travels within London.
Due to limitations with my phone's Google Maps application, I couldn't activate the timeline feature to record my itinerary. Consequently, I resorted to using alternative software to temporarily track new GPS data. As a result, there may be discrepancies between my images and the GPS route, hindering precise alignment.

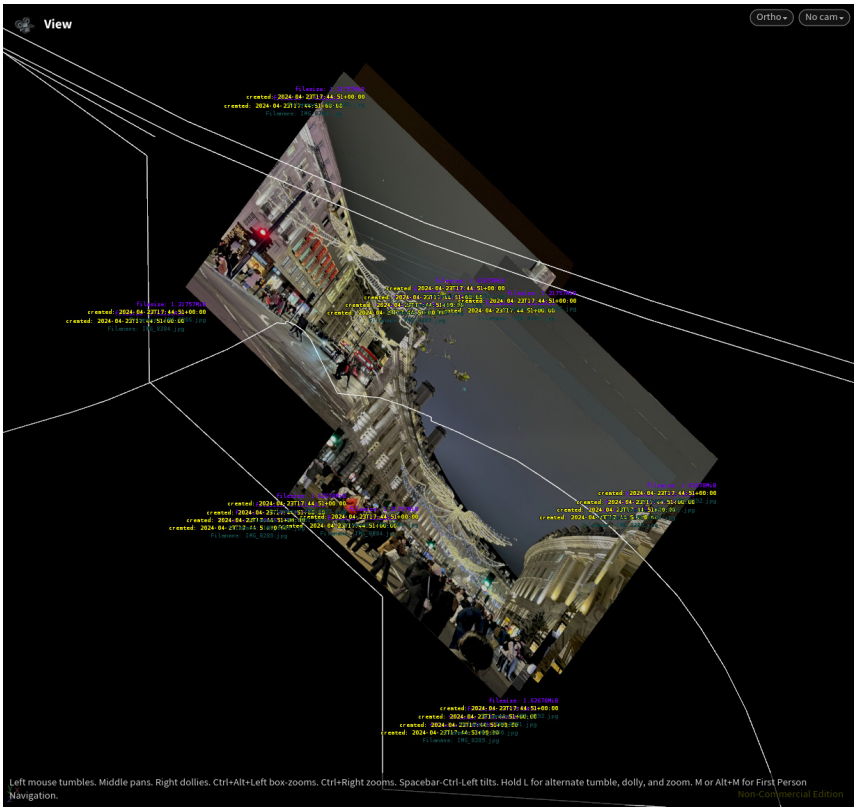


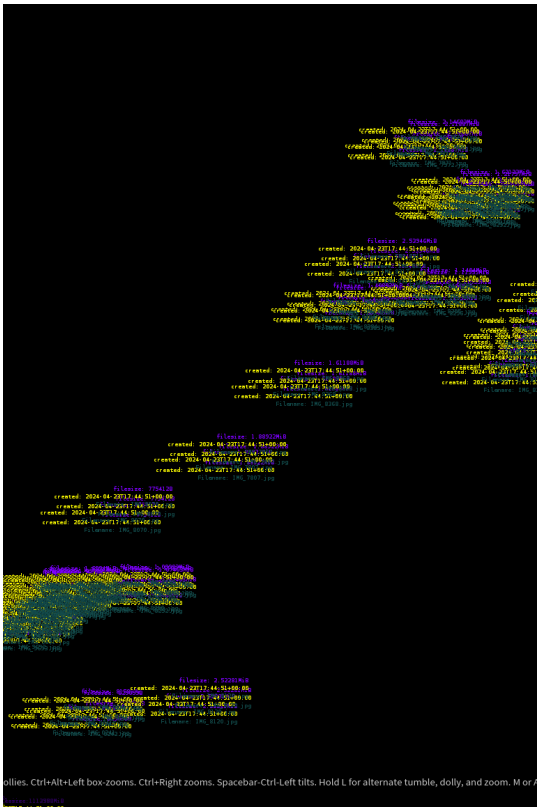


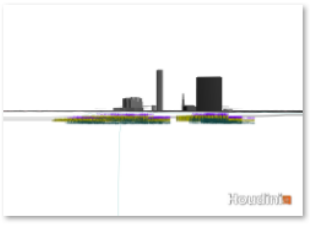




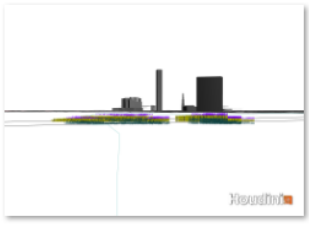




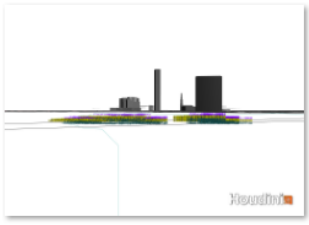




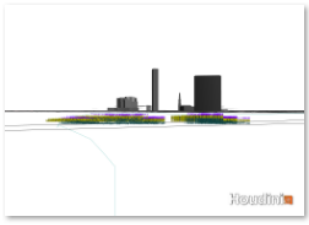
coverlay_100.png



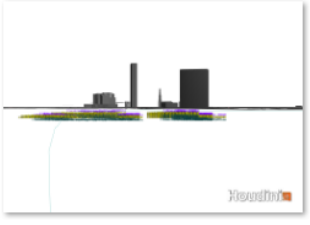
coverlay_101.png



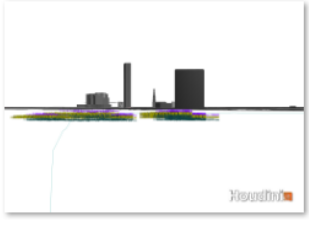
coverlay_102.png



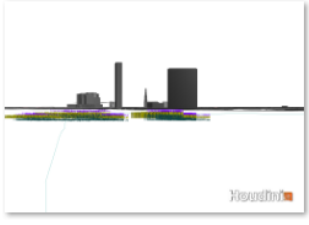
coverlay_103.png



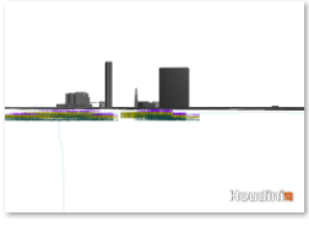
coverlay_108.png



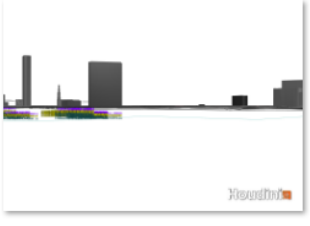
coverlay_109.png



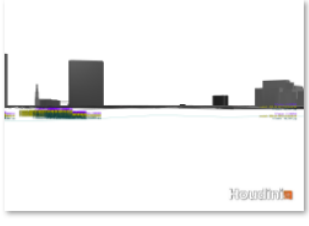
coverlay_110.png



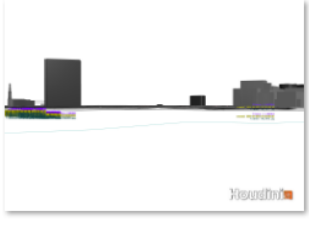
coverlay_111.png



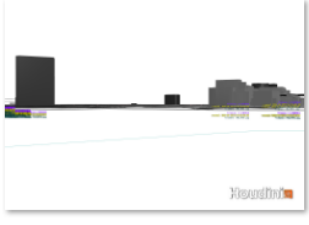
coverlay_116.png



coverlay_117.png



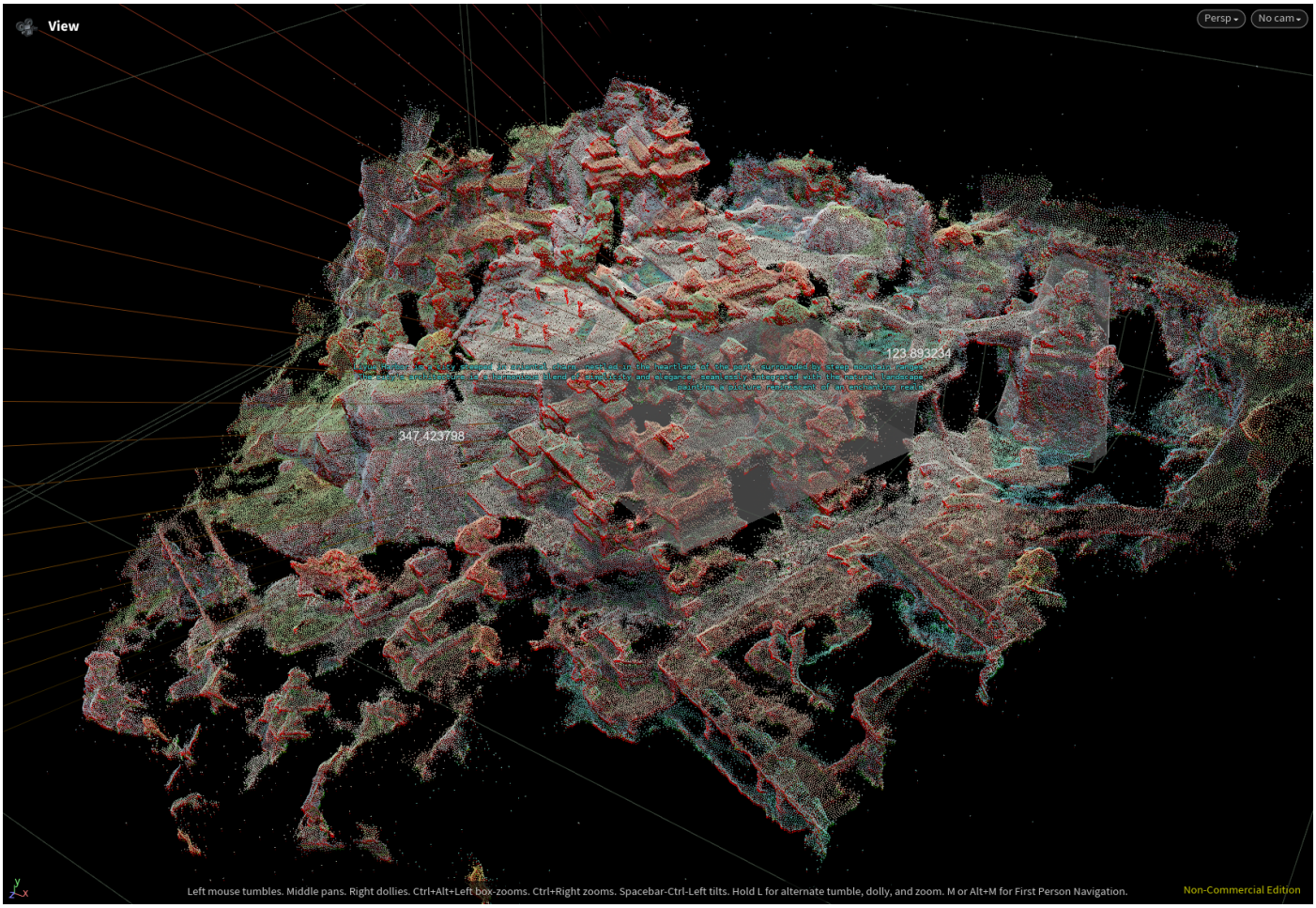
coverlay_118.png

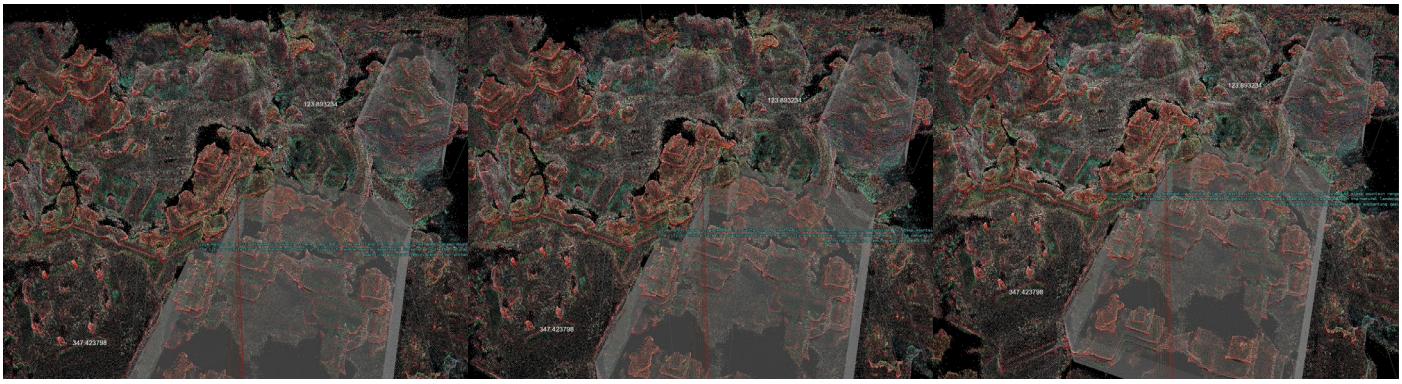


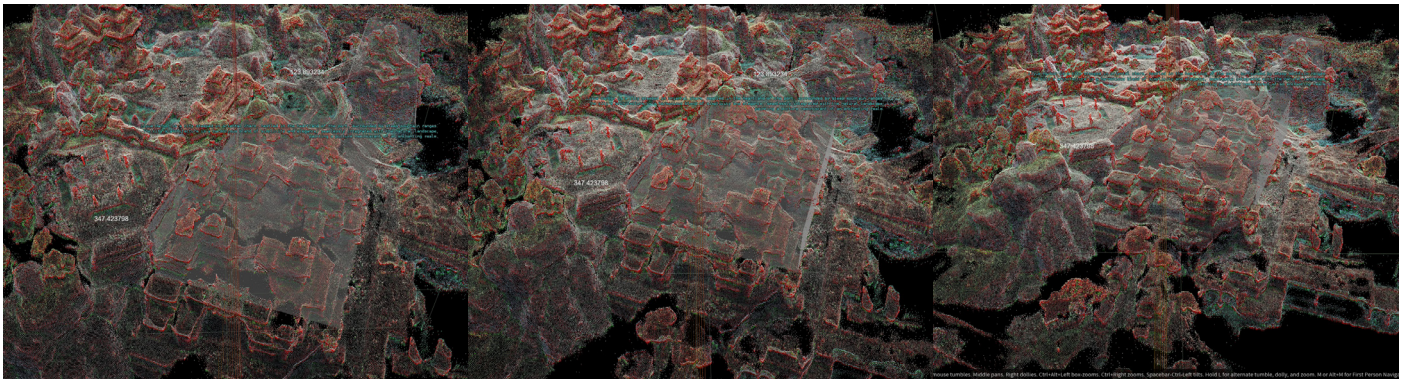
coverlay_119.png

3 video to 3D Model

For this assignment, I transformed the virtual landscapes from an animation into digital models. The scenes were extracted from Genshin, a game renowned for its rich terrain and architecture. In Houdini, I aimed to offer clearer insights into the relationship between buildings and terrain, as well as to analyze the volume of the structures.

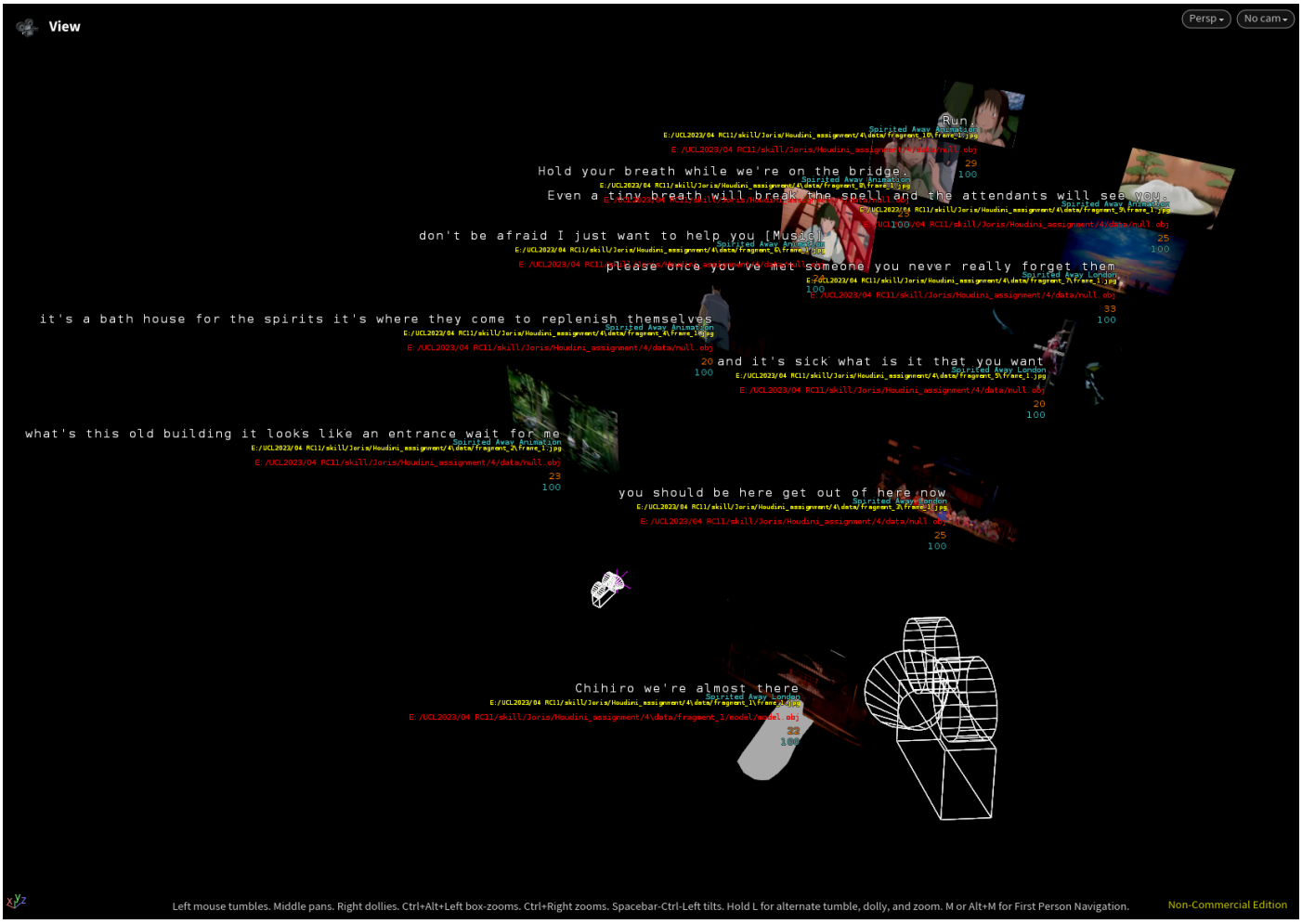




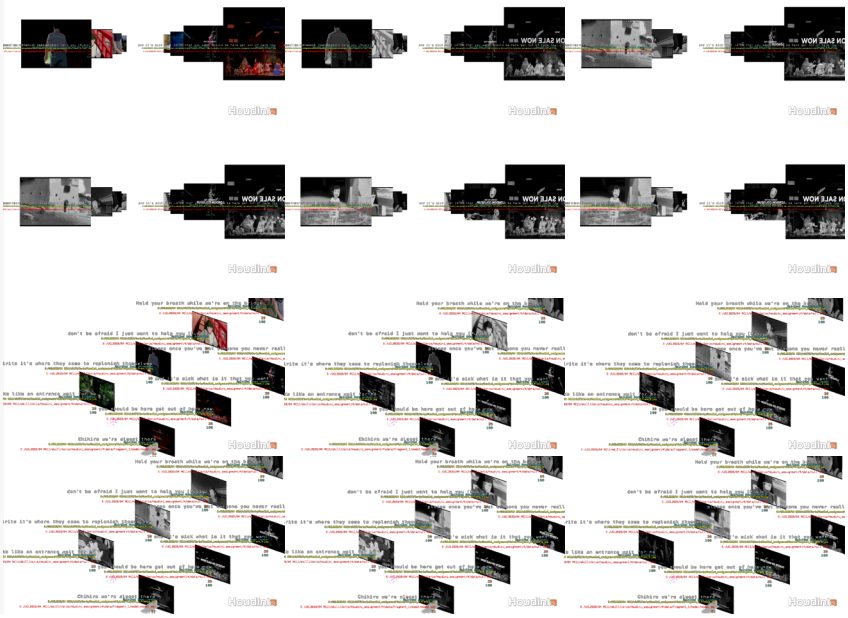


4 Visualizing JSON in Houdini

For this assignment, I sourced film clips from my Python skills learning journey. Utilizing image features to match segments of multiple videos for similarity assessment, I selected two categories of videos: the original animated version and the theatrical performance of "Spirited Away." I organized the most relevant frames from each video category into separate columns in Houdini. This arrangement allows for a straightforward visual comparison between the two categories from a unified perspective.




```
reconstruct = {
  "folder": [
    {
      "paragraph": "Chihiro we're almost there",
      "filmID": "Spirited Away London",
      "image_name": "frame",
      "model_path": "model.obj",
      "image_n": 22,
      "time": 100,
      "film_path": "Spirited Away London  Tickets On Sale Now.mp4"
    },
    {
      "paragraph": "what's this old building it looks like an entrance wait for me",
      "filmID": "Spirited Away Animation",
      "image_name": "frame",
      "model_path": "",
      "image_n": 23,
      "time": 100,
      "film_path": "Spirited Away - Official Trailer.mp4"
    },
    {
      "paragraph": "you should be here get out of here now",
      "filmID": "Spirited Away London",
      "image_name": "frame",
      "model_path": "",
      "image_n": 25,
      "time": 100,
      "film_path": "Spirited Away London  Tickets On Sale Now.mp4"
    },
    {
      "paragraph": "it's a bath house for the spirits it's where they come to replenish themselves",
      "filmID": "Spirited Away Animation",
      "image_name": "frame",
      "model_path": "",
      "image_n": 20,
      "time": 100,
      "film_path": "Spirited Away - Official Trailer.mp4"
    },
    {
      "paragraph": "and it's sick what is it that you want",
      "filmID": "Spirited Away London",
      "image_name": "frame",
      "model_path": "",
      "image_n": 20,
      "time": 100,
      "film_path": "Spirited Away London  Tickets On Sale Now.mp4"
    },
    {
      "paragraph": "don't be afraid I just want to help you [Music]",
      "filmID": "Spirited Away Animation",
      "image_name": "frame",
      "model_path": "",
```

Complexity Assignment

23121106

Github Link: https://github.com/UD-Skills-2023-24/RC11_23121106/tree/main/FinalAssignmentCeel

Exercise One

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

In [7]:
```
import numpy as np

def square_matrix_multiply(A, B):
    n = len(A)
    C = [[0] * n for row in range(n)] # Initialize result matrix C

    for i in range(n):  # Iterate through rows of A
        for j in range(n):  # Iterate through columns of B
            for k in range(n):  # Iterate through elements of rows of A and columns of B
                C[i][j] += A[i][k] * B[k][j]  # Compute the product and accumulate it in C

    return C
```

In [8]:
```
# Example usage:
A = np.array([[2,4],[0,5]])
B = np.array([[3, 2],[1,6]])

result = square_matrix_multiply(A, B)
print(result)
```

```
[[10, 28], [5, 30]]
```

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

I think the asymptotic time complexity of the matrix multiplication algorithm above is O(n^3).

'for i = 1 to n do'----------n times;

'for j = 1 to n do'----------n times;

'cij = 0 for k=1 to n do'----------n times;

' cij = cij + aik × bkj'----------n times;

So, O(n *n* n * n) = O(n^3)

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

4. Compare with built-in multiplication functions;

In [9]:
```
#To compare the algorithms on different sizes of matrices,
#we can generate random matrices of varying sizes
#and measure the time taken by each algorithm to multiply them

import time

# create a function to generate a random matrix of size n x n
def generate_random_matrix(n):
    return [[np.random.randint(20) for _ in range(n)] for _ in range(n)]

#create different size of matrices to compare
sizes = [50, 100, 200]
for size in sizes:
    A = generate_random_matrix(size)
    B = generate_random_matrix(size)

    #Measure time with built-in multiplication functions
```

```
    start_time = time.time()
    np_result = np.dot(A, B) # np.dot(A, B) used for list; A@B used for Numpy arrays
    numpy_time = time.time() - start_time

    # Measure time for nested list implementation
    start_time = time.time()
    list_result = square_matrix_multiply(A, B)
    list_time = time.time() - start_time

    print(f"Matrix size: {size}x{size}")
    print(f"NumPy time: {numpy_time:.3f} s")
    print(f"Nested list time: {list_time:.3f} s")
    print()
```

```
Matrix size: 50x50
NumPy time: 0.000 s
Nested list time: 0.011 s

Matrix size: 100x100
NumPy time: 0.000 s
Nested list time: 0.112 s

Matrix size: 200x200
NumPy time: 0.007 s
Nested list time: 0.886 s
```

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

In [10]:
```
# multiplying three matrices:

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

In [29]:
```
# Example usage:
A = [[2, 1, 13], [11, 3, 12], [6, 0, 10]]
B = [[8, 2, 18], [17, 4, 10], [14, 10, 9]]
C = [[15, 10, 0], [1, 9, 0], [19, 18, 18]]

list_result = multiply_matrices(A, B, C)
print(list_result)

print()
np_result = np.dot(np.dot(A, B), C)
print(np_result)
```

```
[[6460, 6326, 2934], [11143, 10504, 6048], [6694, 6452, 3564]]

[[ 6460  6326  2934]
 [11143 10504  6048]
 [ 6694  6452  3564]]
```

Exercise Two

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

This algorithm, Square-Matrix-Multiply-Recursive (SMMRec), is a recursive implementation of matrix multiplication.

1. "SMMRec(A,B):": accepts two matrices A and B as input.
2. "n = nr of rows of A": Define a variable n which is equal to the number of rows of A, also the size of A.
3. "let C be a new n × n matrix": Create a new square C with the same dimensions as A and B.
4. "if n == 1 then":If the size of the matrix is 1 × 1, the following operations are performed; otherwise, the chunked multiplication of the matrix is performed.

5. "quarter matrices A, B, and C":Divide the matrices A, B and C into four equal-sized sub-matrices.
6. "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)": Calculate the value of C11 C12 C21 C22 separately.

recursiveness: The base case of the recursion is when the size of the matrices becomes 1 × 1, the algorithm computes the product directly without further recursion.And in each recursion step, the algorithm divides the input matrices into four equal-sized submatrices. It then recursively multiplies these submatrices until the base case is reached, where it computes the product directly.

divide-and-conquer:

1. **Divide**:The algorithm divides the input matrices A, B, and C into four equal-sized submatrices.
2. **Conquer**: It recursively multiplies these submatrices. Each multiplication involves recursively calling the SMMRec function until the base case is reached, and then directly computing the product.
3. **Combine**: After computing the products of submatrices, the algorithm combines them to form the final result matrix C. The combination involves adding or summing the products of corresponding submatrices to form the corresponding submatrices of C.

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.

In [11]:
```python
def split_matrix(A):
    mid = len(A) // 2
    A11 = [row[:mid] for row in A[:mid]]
    A12 = [row[mid:] for row in A[:mid]]
    A21 = [row[:mid] for row in A[mid:]]
    A22 = [row[mid:] for row in A[mid:]]
    return A11, A12, A21, A22

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

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

def SMMRec(A, B):
    n = len(A)
    if n == 1:
        # Base case: direct multiplication of 1x1 matrices
        return [[A[0][0] * B[0][0]]]
    else:
        # Split matrices into quarters
        A11, A12, A21, A22 = split_matrix(A)
        B11, B12, B21, B22 = split_matrix(B)

        # Recursive calls to compute submatrix products
        C11 = matrix_add(SMMRec(A11, B11), SMMRec(A12, B21))
        C12 = matrix_add(SMMRec(A11, B12), SMMRec(A12, B22))
        C21 = matrix_add(SMMRec(A21, B11), SMMRec(A22, B21))
        C22 = matrix_add(SMMRec(A21, B12), SMMRec(A22, B22))

        # Combine submatrix products into result matrix
        return combine_matrices(C11, C12, C21, C22)
```

In [57]:
```python
# Example usage:
A = [[9, 7, 10, 12], [8, 17, 2, 2], [13, 7, 15, 19], [19, 13, 2, 12]]
B = [[15, 0, 13, 3], [7, 18, 16, 16], [19, 14, 16, 2], [15, 8, 8, 5]]
```

```
C = SMMRec(A, B)
print("SMMRec result:\n", C, "\n")

np_result = np.dot(A, B) # Check if the results of SMMRec are correct
print("np result:\n", np_result)
```

```
SMMRec result:
 [[554, 362, 485, 219], [307, 350, 424, 310], [814, 488, 673, 276], [594, 358, 583, 329]]

np result:
 [[554 362 485 219]
 [307 350 424 310]
 [814 488 673 276]
 [594 358 583 329]]
```

Straightforward Steps: Base case, splitting matrices and combining matrices are some straightforward steps.

Steps with Complexity:

1. matrix addition: Matrix addition involves iterating over corresponding elements of the matrices and adding them together. In the matrix_add() function, we iterate over the rows and columns of the matrices, accessing corresponding elements and adding them together.
2. recursive call: Complexity is managing indices to access elements of matrices correctly. In the pseudocode, the algorithm computes the product of submatrices by performing addition and multiplication on their corresponding elements.

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.

Base case B(n): O(1) because it involves a simple multiplication operation between two scalars.

Divide D(n)： The divide step involves splitting the input matrices into quarters. Since this step involves iterating over all elements of the matrices, its time complexity is O(n^2)

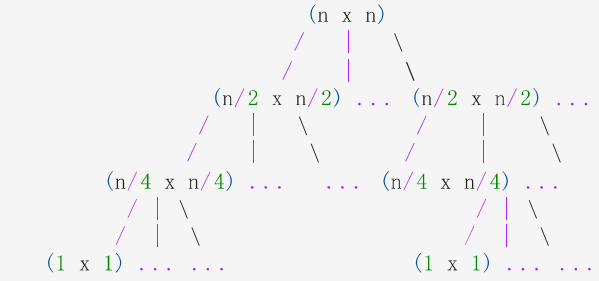
Conquer R(n)： The conquer step involves recursively multiplying submatrices until the base case is reached. The size of the matrices reduces by half in each recursive call. Then in each recursive call, the size reduces to n/2. Therefore, the recursive conquer step has a complexity of T(n/2).

Combine C(n): This step requires iterating over each element of the result matrix and copying or summing the corresponding elements from the submatrix products. Its time complexity is O(n^2).

Overall: Overall time complexity: if base case: T(n) = O(1); else: O(n^2) + T(n/2) + O(n^2)

4. Do a tree analysis;

In [ ]:



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

In [29]:
```
# Test and compare the algorithms
sizes = [16,64,128]  # Different sizes of matrices to test
for size in sizes:
    A = generate_random_matrix(size)
    B = generate_random_matrix(size)

    # Measure time for recursive algorithm
```

```
    start_time = time.time()
    result_recursive = SMMRec(A, B)
    recursive_time = time.time() - start_time

    # Measure time for non-recursive algorithm (for comparison)
    start_time = time.time()
    result_non_recursive = np.dot(A, B)
    non_recursive_time = time.time() - start_time

    print(f"Matrix size: {size}x{size}")
    print(f"Recursive time: {recursive_time:.3f} s")
    print(f"Non-recursive time: {non_recursive_time:.3f} s")
    print()
```

```
Matrix size: 16x16
Recursive time: 0.010 s
Non-recursive time: 0.000 s

Matrix size: 64x64
Recursive time: 0.335 s
Non-recursive time: 0.000 s

Matrix size: 128x128
Recursive time: 2.818 s
Non-recursive time: 0.000 s
```

Exercise Three

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

1. **addition and subtraction complexity**: These two operation are relatively straightforward and have a complexity of O(n^2),because each element in the result matrix is computed independently by adding or subtracting corresponding elements from the input matrices. The time complexity scales linearly with the number of elements in the matrices.
2. **Multiplication Complexity**: Multiplication is more complicated.The time complexity of O(n^3) for two n × n matrices. Because each element in the result matrix is computed by taking the dot product of a row from the first matrix and a column from the second matrix. So,the time complexity of matrix multiplication grows cubically with the size of the matrices.

2. Do a complexity analysis of the Strassen algorithm.

Base case B(n): Same as Algorithm 2, the base case complexity remains O(1) as it involves a simple multiplication operation between two scalars.

Divide D(n)： Splitting the input matrices into quarters requires O(n^2) time, as it involves creating four submatrices each with size n/2 × n/2, similar to Algorithm 2.

Conquer R(n)： In Strassen's algorithm, there are 7 recursive calls (P1 to P7) instead of 8 as in Algorithm 2. Each recursive call computes the product of two submatrices with size n/2 × n/2 using Strassen's algorithm. Therefore, the complexity of the recursive step is 7 * T(n/2), where T(n/2) is the time complexity of multiplying two n/2 × n/2 matrices recursively.

Combine C(n): Combining the submatrix products into the final result matrix involves addition and subtraction operations on the intermediate matrices (S1 to S10) and the products (P1 to P7). Similar to Algorithm 2, combining the submatrix products requires O(n^2) time.

Overall: Overall time complexity: if base case: T(n) = O(1); else: O(n^2) + 7 * T(n/2) + O(n^2)

Difference: The main difference lies in the conquer step.Algorithm 2 involves 8 recursive multiplications, but Strassen's algorithm reduces the number of recursive multiplications to 7. As a result, Strassen's algorithm

achieves a better asymptotic time complexity, especially for large matrix sizes

3. Implement and test the algorithm;

In [30]:
```
def matrix_sub(A, B):
    return [[A[i][j] - B[i][j] for j in range(len(A))] for i in range(len(A))]

def Strassen(A, B):
    n = len(A)
    if n == 1:
        # Base case: direct multiplication of 1x1 matrices
        return [[A[0][0] * B[0][0]]]
    else:
        # Split matrices into quarters
        A11, A12, A21, A22 = split_matrix(A)
        B11, B12, B21, B22 = split_matrix(B)

        # Calculate intermediate matrices S1 to S10
        S1 = matrix_sub(B12, B22)
        S2 = matrix_add(A11, A12)
        S3 = matrix_add(A21, A22)
        S4 = matrix_sub(B21, B11)
        S5 = matrix_add(A11, A22)
        S6 = matrix_add(B11, B22)
        S7 = matrix_sub(A12, A22)
        S8 = matrix_add(B21, B22)
        S9 = matrix_sub(A11, A21)
        S10 = matrix_add(B11, B12)

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

        # Calculate quarters of result matrix C
        C11 = matrix_add(matrix_sub(matrix_add(P5, P4), P2), P6)
        C12 = matrix_add(P1, P2)
        C21 = matrix_add(P3, P4)
        C22 = matrix_sub(matrix_sub(matrix_add(P5, P1), P3), P7)

        # Combine submatrix products into result matrix
        return combine_matrices(C11, C12, C21, C22)
```

In [31]:
```
# Example usage:
A = [[9, 7, 10, 12], [8, 17, 2, 2], [13, 7, 15, 19], [19, 13, 2, 12]]
B = [[15, 0, 13, 3], [7, 18, 16, 16], [19, 14, 16, 2], [15, 8, 8, 5]]

C = Strassen(A, B)
print("SMMRec result:\n", C, "\n")

np_result = np.dot(A, B) # Check if the results of SMMRec are correct
print("np result:\n", np_result)
```

```
SMMRec result:
 [[554, 362, 485, 219], [307, 350, 424, 310], [814, 488, 673, 276], [594, 358, 583, 329]]

np result:
 [[554 362 485 219]
 [307 350 424 310]
 [814 488 673 276]
 [594 358 583 329]]
```

In [34]:
```
# Test the Strassen algorithm
sizes = [16, 64, 128]  # Different sizes of matrices to test
for size in sizes:
    A = generate_random_matrix(size)
    B = generate_random_matrix(size)

    # Measure time for Strassen's algorithm
```

```
    start_time = time.time()
    result_strassen = Strassen(A, B)
    strassen_time = time.time() - start_time

    # Measure time for recursive algorithm
    start_time = time.time()
    result_recursive = SMMRec(A, B)
    recursive_time = time.time() - start_time

    # Measure time for non-recursive algorithm (for comparison)
    start_time = time.time()
    result_non_recursive = np.dot(A, B)
    non_recursive_time = time.time() - start_time

    print(f"Matrix size: {size}x{size}")
    print(f"Strassen's time: {strassen_time:.3f} s")
    print(f"Recursive time: {recursive_time:.3f} s")
    print(f"Non-recursive time: {non_recursive_time:.3f} s")
    print()
```

```
Matrix size: 16x16
Strassen's time: 0.002 s
Recursive time: 0.012 s
Non-recursive time: 0.000 s

Matrix size: 64x64
Strassen's time: 0.441 s
Recursive time: 0.366 s
Non-recursive time: 0.000 s

Matrix size: 128x128
Strassen's time: 3.250 s
Recursive time: 2.714 s
Non-recursive time: 0.000 s
```

4. Discuss other optimisations;

In [43]:
```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import requests
from io import BytesIO

image_url1 = "https://www.mdpi.com/mathematics/mathematics-09-02033/article_deploy/html/images/mathemat
response1 = requests.get(image_url1)
img1 = Image.open(BytesIO(response1.content))
img_array1 = np.array(img1)

plt.imshow(img_array1)
plt.axis('off')
plt.show()
print("Figure 1. Winograd's convolution in two dimensional case.")
```

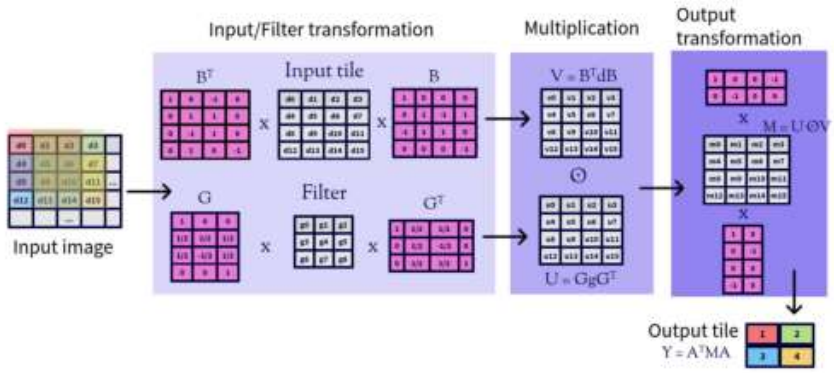


Figure 1. Winograd's convolution in two dimensional case.

Winograd's Algorithm:

Winograd's algorithm, is another algorithmic improvement over standard matrix multiplication and Strassen's algorithm. Like Strassen's algorithm, Winograd's algorithm aims to reduce the number of arithmetic operations required for matrix multiplication.

This section provides further details of the implementation of the Winograd algorithm. At the top level, the algorithm starts with an input matrix, which is divided into tiles of $\alpha \times \alpha$ elements. Then, these tiles are processed separately. In fact, the processing of different tiles can take place in parallel and this is an important source of parallelism. The processing of each tile is done using the formulas, making space for the following sequence of four steps.

1. Filter transformation
2. Input transformation
3. Multiplication
4. Output transformation

Winograd's algorithm applies certain mathematical transformations to the input matrices and intermediate results to simplify the multiplication process, achieving a further reduction in the number of arithmetic multiplications compared to Strassen's algorithm. It accomplishes this by decomposing the traditional matrix multiplication operation into smaller sub-operations that require fewer multiplications.In the meantime, Winograd's algorithm precomputes certain intermediate values that can be reused across multiple multiplications. By leveraging these precomputed values, the algorithm reduces redundant computations and achieves better computational efficiency.