UD | RC11 | 2023

SKILL REPORT

Tutor: **Julian Besems**
Ceel Pierik
Joris Putteneers

RC11_22136945

CONTENT

| INTRODUCTION

This report is divided into three distinct yet interconnected parts: computational theory, the application of data vectorization, and 3D modeling. By departing from conventional design methods for urban and architectural spaces and incorporating elements of computational science, we are able to not only accelerate the process of designing large-scale urban projects, but also instigate a paradigm shift in our design thinking.

In this context, images and videos transcend their traditional roles as mere visual components. Through vectorization, they can be transformed into versatile elements that interact with a broad array of applications. This perspective encourages us to view the 3D software tools we have been using from a more technically informed standpoint, expanding our capabilities to manipulate and innovate within these platforms.

As a result, the design field is undergoing a profound transformation. The integration of computational theory, vectorization, and 3D modeling is pushing the boundaries of what is possible, ushering in a new era of design that is faster, more efficient, and more in line with the complexities of large-scale urban projects.

COMPUTATIONAL THEORY

taught by Ceel Pierik

Concepts Explanation

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Ceel_assignment/ConceptExplanation

Applying knowledge

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Ceel_assignment/ApplyingKnowledge

Historical Development Of Computation

For a long time, a computer was just the same as a calculator. Because during that period, it was necessary to start calculating for practical purposes. It used to be mainly for logistics and keeping track of money in goods. One of the first accessories that helped us to keep track of the calculations was the Sumerian abacus during 3rd millennium BC, later, the Chinese abacus during 2nd century BC. Along that line we saw more instruments starting to appear that can calculate, not just numbers in a vacuum, but really a measure quantity, such as length degrees for either nautical or astronomical positions(Rhodes, 2020).

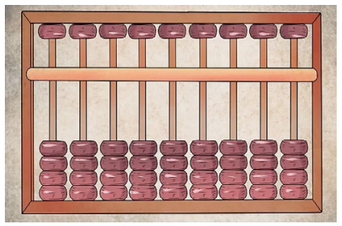


Fig 1. Abacus-World's first calculator History of Computers
Source: https://www.cuemath.com/learn/abacus-history/

Throughout the Middle Ages, more developments happened in the Islamic world, such as astrolabe, torquetum, etc. Then we've got more mechanical inventions that did more than just calculations, such as Book of Ingenious Devices by Banu Musa brothers and the Book of Knowledge of Ingenious Mechanical Devices by Ismail al-Jazari. We started to see the possibilities of automating things and programming things, which made us query what can we do with these mechanical inventions.

After the Middle Ages, we started formalizing our notions of what we mean, which leads to the concept of algorithms.

Algorithms

Algorithms is a finite procedure or a finite set of rules to solve a problem in a step-by-step fashion.

The first part is to have an idea of which steps we want to take to achieve the goal. The second part is the step-by-step fashion, like following a receipt. For example, Sieve of Eratosthenes finds all the prime numbers up to a given limit n.

The idea of algorithms is really the central notion of computability. Instead of just calculating, computer programs can almost do endless number of things as long as the right program is input. This notion of algorithm is then used in something very practical. Like one of the earliest programming machines that we know of, Jacquard machine, which is a mechanical weaving loom which can weave many patterns, and the fabric designs stored on series of punched cards. This idea of punched cards is the key which showed the ability of programming instructions to a machine (2023).

Fig 2. Jacquard loom
Source: https://jollycontrarian.com/index.php?title=Jacquard_loom

Fig 3. Difference Engine No.1
Source: https://collection.sciencemuseumgroup.org.uk/objects/co62243/difference-engine-no-1-difference-engine

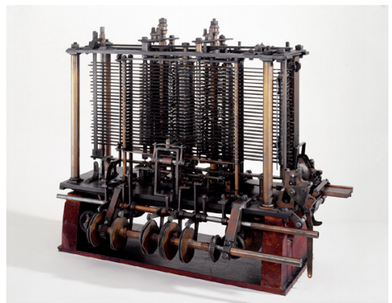


Fig 4. Analytical Engine - Wikipedia
Source: https://en.wikipedia.org/wiki/Analytical_engine

In the 19th century, formal computation started with Charles Babbage's multiple ideas, including the difference engine. The ideas of automating calculation for avoiding mistakes and storing memories were introduced. Later, his analytical engine incorporated mill, store, reader, and printer, which can be seen as a multi-purpose computer than just a calculator. With his assistant Ada Lovelace realized that the numbers that input into the engine can be anything.

Computable Functions

In the early 20th century, the notion of formalization raised lots of development in the mathematical world. If we want to find an algorithm and answer in a positive sense, we just provide the algorithm. However, if we want to prove there's not such a thing, we need to formalize the notion of algorithm.

Computable functions should at least have the following properties:
1. its procedure instructions must be exact and finite in length;
2. for any input in the domain of the function, the function should produce the output (or: halt) in finitely many steps;
3. for any input not in the domain of the function, the function should become stuck or never halt.

Computability is about whether we can computer it, or is there a solution, whereas complexity is about the efficiency, how fast we can compute it. These are three different types of problems in computability:
- decision problem: Is there a solution? Yes/No
- function problem: What is the solution?
- optimisation problem: What is the best solution?

Halting Problem

One of the most famous decision problems is Halting Problem.
Halting Problem: Can we determine for any program and any input whether the program halts on the given input?
In 1936, Turing proved that the answer is 'no'. Think of a computer, when we start up with a random program with a random input, there's no single algorithm that can determine whether the program halts on the given input. There were thousands of problems proved to be incomputable after this, which was done by reducing the problem to the Halting problem.

(General) Recursion

In order to work with the computable functions, there were basically three different models that have been developed within mathematics and computer science: (General) recursive functions, Turing machines, lambda-calculus.

Recursion is defining an object in terms of itself. This idea of recursion generally has smaller and smaller steps or going back defining something in terms of itself. In terms of programming, it is the concept within an infinite loop, the program keeps on running as it would not halt. Recursion is central in the theory of computability. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Ceel_assignment/ConceptExplanation/Skills02Homework-Skill.ipynb)*

Fig 5. Recursion - Wikipedia
Source: https://en.wikipedia.org/wiki/Recursion

A recursive function or algorithm has:
base case(s) prevent the algorithm from keeping on going forever
recursive step(s) defining something in terms of itself

For example:

```
def letterCount(string, character, index=0, count=0):

    if index == len(string):          Base case: if we've reached the end of the string, return the count
        return count

    if string[index] == character:    Recursive step: if the current character matches, increment count
        return letterCount(string, character, index + 1, count + 1)

    else:                             Recursive step: if the current character doesn't match, continue to the next character
        return letterCount(string, character, index + 1, count)

print(letterCount('Freda', 'a'))

Output: 1
```

The general recursive functions are those we can make with a composition of:
1 - basic functions:
• 0 (the function that takes no input and always outputs 0)
• S (the successor function defined by S(n) = n + 1)
• π^i_h (projections, taking n inputs and outputting the i-th)

2 - primitive recursion
Counterexample: Ackermann function

3 - μ-operator
• minimisation or search operation
• it is unbounded, it might never stop if it cannot find a number, it will keep searching
• for-loop vs while-loop, in for-loop, we decide the number of times we are going to loop, in while-loop, we set an ending condition, that it might never met.

Everything we can create with the combine of these, that's called computable.

Deterministic Infinite Automaton (DFA)

A DFA consists of the following 5 parts {Q, Σ, δ, q0, F}, meaning:

• Q state is the set of states; For example, for Fig6. Q = q0, q1
• Σ alphabet: is the alphabet we use for the input; Σ = 0,1
• δ is a transition function between states; δ(q0,1) = q0, δ(q0,0) = q1, δ(q1,1) = q1, δ(q1,0) = q0
• q0 is the starting state of the automaton; q0 = q0
• F is the set of final states. F = q0

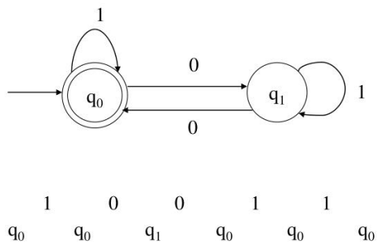


Fig 6. Deterministic Finite State Automata
Source: https://www.slideserve.com/janet/deterministic-finite-state-automata-dfa

Turing Machines

Turing Machine is a hypothetical machine, it can be seen as an idealised computer. We can imagine it to be a one-way infinite tape that divided up into cells, and a tape head. The cells contain either a 1 or nothing ("blank"). There's a head that moves along with the tape, then the data on the cell can be read. Then we can change the cell or move the tape to the left or right.

The Turing Machine is the program, the rule we set to instructions, the algorithm that manipulates this tape. How it operates is very similar to the automaton.

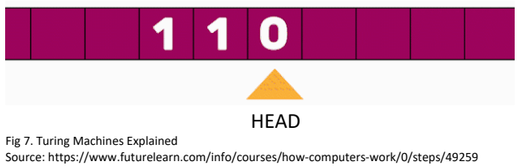


Fig 7. Turing Machines Explained
Source: https://www.futurelearn.com/info/courses/how-computers-work/0/steps/49259

For example,
If we are in state 0 (q0), and the machine reads '1', then it transforms to state 1 (q1), write down a '1' and move to the right,
as we are in the state 1 (q1), and the machine reads '1', then it stays in state 1 (q1), leave a 'blank' and move to the right.

Assume Turing Machines with alphabet 0, 1. Turing machine is finite as it finds its own rules, alphabet, states, movement in one step. We can encode the list of possible rules into a string of 1 and 0. Therefore, we can encode a Turing Machine as a long binary number, which then we can feed them as input to another Turing Machine, then we can create a Universal Turing Machine, the program is then recursive. We can call a system of rules 'Turing complete' if it is at least as strong as a universal Turing machine; that is, if the ruleset can simulate any Turing machine, such as most programming languages: python.

Lambda-calculus

Lambda Calculus is a mathematical system and programming language paradigm. This system is as strong as a Universal Turing Machine, therefore it is Turing complete. It is described as a very simple and ideal programming language. It is a system consisting of:
• Variables (x)
• Abstraction (λx.T), the abstraction rules are basically ways to define mathematical functions
• Application (F M), actually calculation of the function

A function in lambda calculus is expressed as λx.T, where λ is the lambda symbol, x is the parameter, and T is the body of the function. Application of a function to an argument is expressed as (λx.T)M, meaning the function λx.T is applied to the argument M. Variables in Lambda Calculus can take on any function or variable as a value.

For example: $\lambda x.\lambda y.x^2 + y^2$ This fuction means that it takes x and y as two arguments, the output is $x^2 + y^2$. We can use it like this: $f(x, y) = x^2 + y^2$

Complexity vs Computability

Both complexity and computability are important concepts in computational science, yet they address different aspects of computation. Computability refers to whether a problem can be solved only. Whereas complexity focuses on the efficiency of how much time it might take for a problem to be solved, assuming it is computable.

Computability: Can we solve the problem?
Complexity: how fast can we solve the problem?

Computational Complexity

Computational complexity, as opposed to the dictionary meaning, programming complexity, network complexity, Kolmogorov complexity etc., is about the efficiency of solving the problem. If we have multiple algorithms to solve the same problem, complexity theory helps us understand which one is the most efficient in terms of resources like time and space.

In addition, a general problem is the abstract problem, instance represents a concrete example of the problem. If we can solve a problem, we can solve every instance of the problem. However, if we can solve a instance of the problem, that does not mean we can solve the problem in general. As we want to know the complexity of a general algorithm, we are not looking for the complexity of solve an instance of the problem, but solving the general problem with any instance.

Measuring Complexity

When measuring the complexity of general algorithms, there would be problems to measure the execution time or measure how many statements is there for an algorithm. As the measurement depends much on the underlying hardware, the processor architecture, the programming language used, and how does it behave for different sizes. We want to know if algorithms solve general problems with different sizes. Therefore, when measuring complexity, the focus is on how the time or space grows relative to the input.

When measuring the complexity of an algorithm, two key aspects are usually discussed:
• Time Complexity: The amount of time an algorithm takes to run as a function of the size of the input.
• Space Complexity: The amount of memory space that the algorithm needs to run.

Time Complexity

When measuring time complexity, instead of measuring the "real time" the algorithm takes for the program to run, but rather the number of operations required. This is because the actual time can vary based on the particular hardware or other factors. Therefore, when measuring time complexity, we need to look at each line of the algorithm.

Function Growth

When we want to approximate the complexity, we are looking at what happends with the algorithm when the input size becomes large. When the input size gets large, the leading term in the function dominates the end result.

Function	Example
constant	f (n) = 2
logarithmic	f (n) = log(n) + 2
fractional	f (n) = √n = $n^{0.5}$
linear	f (n) = 3n + 2
linearithmic	f (n) = n · log(n)
quadratic	f (n) = n^2 + n
polynomial	f (n) = n^4 + n^3 + 1
exponential	f (n) = 2^n + 2n
factorial	f (n) = n!

This concept of function growth is integral to Big O notation, which is used to classify algorithms based on how their running time or space requirements grow as the input size increases.

Asymptotic ('Big O') Notation

'Big O' notation is a mathematical notation used to describe the performance or complexity of an algorithm in computer science. In 'big O' notation, all the lower terms, such as constants was got rid of, then the categories of complexity got left is what we are looking at. 'Big O' notation helps understanding the efficiency of algorithms in a way that is independent of hardware or software variations. It provides an upper bound on the time or space complexity, particularly for large input sizes. It is also known as asymptotic notation and Landau notation.

$$f(n) = O(g(n))$$

There is a c > 0, N > 0 such that f(n) ≤ c · g(n) for every n > N

In the above notation, c and N are representing some kind of constant. The growth of the function f(n) is "bounded above" by the function g(n) for sufficiently large n. In this notation, for example, if we want to test whether n = O(n^2) is true, as we think c = 1, N = 1, n = 2, f(n) ≤ c · g(n) can be seen as 2 ≤ 1 · 2^2, then return true. This concept helps us to understand how quickly f(n) grows in comparison to g(n) as n gets larger and larger.

```
maxX = 30
maxY = 100

n = np.linspace(0, maxX)
n1 = np.linspace(0.1, maxX)

f1 = n
f2 = np.log(np.log(n))
f3 = 2**n
f4 = n**2
f5 = n * np.log(n)
f6 = np.sqrt(n)
f7 = math.e**n

fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(n, f1, label = "n")
ax.plot(n, f2, label = "ln(ln(n))")
ax.plot(n, f3, label = "2**n")
ax.plot(n, f4, label = "n**2")
ax.plot(n, f5, label = "n*ln(n)")
ax.plot(n, f6, label = "sqrt(n)")
ax.plot(n, f7, label = "e**n")

ax.set_ylim([-1, maxY])
ax.grid()
plt.legend(loc='upper right')
plt.show()
```

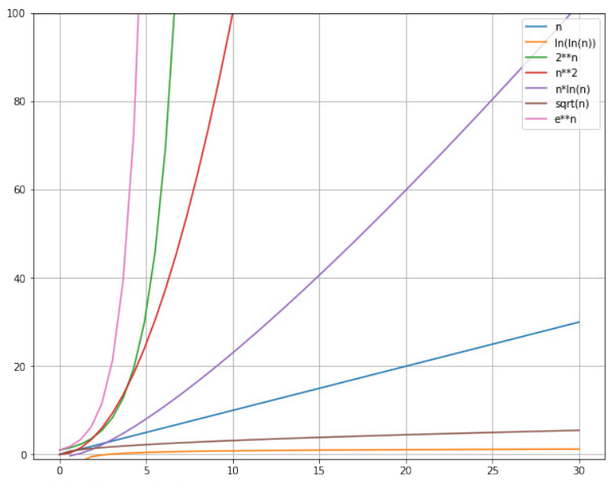


Fig 8. Complexity Visualization
Source: Author

I used the code on the right to visualise the order of time complexity on different algorithms, from the diagram, it's clear to see that e**n grows the fastest, ln(ln(n)) grows the slowest. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Ceel_assignment/ConceptExplanation/bigO.ipynb)*

Insertion Sort Algorithm

Insertion sort is a simple sorting algorithm that builds the final sorted array one item at a time. It places an unsorted element in the array to sort its suitable place in each iteration.

```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        i = j - 1
        while i > 0 and A[i] < key
            A[i + 1] = A[i]
            i = i - 1

        A[i + 1] = key
```

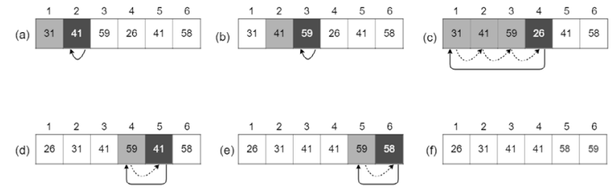


Fig 9. Insertion sort
Source: https://walkccc.me/CLRS/Chap02/2.1/

When analyzing the time complexity of an algorithm, we often consider the best case, average case, and worst case scenarios to get a complete picture of the algorithm's performance. The average case scenario often requires more sophisticated analysis and may not always be easy to determine. So here we only consider the best case and worst case scenarios for the insertion sort algorithm. The best case scenario shows the minimum time the algorithm needs to complete its task, which here would be the list is already sorted. The worst case scenario indicates the maximum time an algorithm can take, which here would be the list is reverse sorted.

For the time complexity of the algorithm, the outer loop runs n times. In the worst case, the inner loop runs j times for each j in the outer loop. Then we multiply their run times, then the time complexity for the worst case would be O(n^2). As the inner loop never runs in the best case, so the time complexity is O(n).

Divide-And-Conquer Paradigm

Divide and Conquer is a powerful algorithm design paradigm of dividing problems into smaller problems recursively. Then conquer these small problems and combine the smaller solutions back into the original solution of the original problem. It has three stages:

- Divide: the problem into a number of subproblems that are smaller instances of the same problem.
- Conquer: the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Combine: the solutions to the subproblems into the solution for the original problem.

Divide and conquer is a foundational concept used in many algorithms, including merge sort, quick sort, etc.

Merge Sort Algorithm

Divide the array into two halves, divide the rest in half, until it cannot be divided anymore, then sort then and merge the resultes. The technical process is to divide the unsorted list into n sublists, each containing one element that is considered sorted, and repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining.

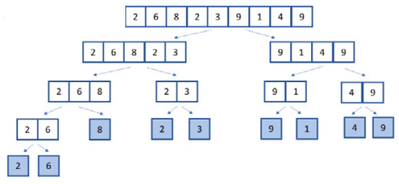


Fig 10. Divide component
Source: https://developer.nvidia.com/blog/merge-sort-explained-a-data-scientists-algorithm-guide/

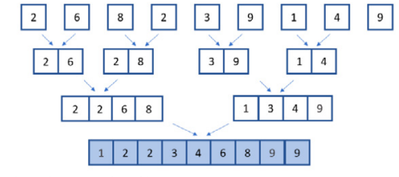


Fig 11. Conquer and combine components
Source: https://developer.nvidia.com/blog/merge-sort-explained-a-data-scientists-algorithm-guide/

To calculate the complexity of the algorithm, we assume that we are working with lists that have a length that is a power of 2. As we set up recurrence equation for the recursive algorithm, the time complexity will also be recursive.

Divide: The divide step just computes the middle of the subarray, which takes constant time.

D(n) = O(1)

Conquer: We recursively solve two subproblems, each of size n/2, which contributes 2 · T(n/2) to the running time.

2T(n/2)

Combine: We have already noted that the Merge procedure on an n-element subarray takes O(n) time.

C(n) = O(n)

Total

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \quad \text{base case} \\ 2T(n/2) + O(n) & \text{if } n > 1 \quad \text{recursive step} \end{cases}$$

Matrices

a matrix is a way to structure data into a two-dimensional array or vector that is arranged in rows and columns. The numbers in the matrix are often called elements or entries. As the matrice on the left has 3 rows and 2 columns, it can be called a 3 x 2 matrix. More generally, if we have a matrix that has m rows and n columns, we can all it m x n matrix. As below shown, the addition between matrices are pretty straightforward.

$$\begin{bmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{bmatrix}$$

Fig 12. Matrices
Source: https://study.com/academy/lesson/matrix-notation-equal-matrices-math-operations-with-matrices.html

$$\begin{bmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1+2 & 3+3 \\ 5+0 & 7+1 \\ 9+1 & 11+0 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 5 & 8 \\ 10 & 11 \end{bmatrix}$$

Fig 13. Matrices
Source: https://study.com/academy/lesson/matrix-notation-equal-matrices-math-operations-with-matrices.html

Matrix Multiplication

Comparing to matrix addition, matrix multiplication is a little complex. Matrix multiplication is not just multiplying elements of two matrices. To multiply two matrices, the number of columns in the first matrix should be equal to the number of rows in the second matrix.

$$AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

Fig 14. Matrix Multiply
Source: https://www.i-programmer.info/news/112-theory/3453-breakthrough-faster-matrix-multiply.html

Matrix multiplication can also be implemented with a recursive version of the Strassen's Matrix Multiplication algorithm. This algorithm is used to multiply two matrices in a more efficient way than the standard method. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Ceel_assignment/ConceptExplanation)*

```
def SMMRec(A, B):
    n = len(A)
    C = np.array([[0 for col in range(n)] for row in range(n)])
    if n == 1:
        C[0][0] = A[0][0] * B[0][0]
    else:
        A11 = [A[i][:n // 2] for i in range(n // 2)]
        A12 = [A[i][n // 2:] for i in range(n // 2)]
        A21 = [A[i][:n // 2] for i in range(n // 2, n)]
        A22 = [A[i][n // 2:] for i in range(n // 2, n)]
        B11 = [B[i][:n // 2] for i in range(n // 2)]
        B12 = [B[i][n // 2:] for i in range(n // 2)]
        B21 = [B[i][:n // 2] for i in range(n // 2, n)]
        B22 = [B[i][n // 2:] for i in range(n // 2, n)]
        C11 = SMMRec(A11, B11) + SMMRec(A12, B21)
        C12 = SMMRec(A11, B12) + SMMRec(A12, B22)
        C21 = SMMRec(A21, B11) + SMMRec(A22, B21)
        C22 = SMMRec(A21, B12) + SMMRec(A22, B22)
    return C
```


Algorithm Design and Optimization

Strassen's optimization is over 50 years old. Yet it is quick enough so that we can use it for, for example 100x100x100 matrices, according to which programming language or implementation is using, etc.

Other than Strassen, lots of other practices has been made which are theoretically interesting, yet They are only more efficient on the tasks of searching for enormous size of matrices than the simpler algorithms, therefore nobody uses them in practice.

What we can do for optimization:

- Try to lower the 'hidden' constant;
- Design or adapt more problem specific algorithms
- Design or adapt more PC-architecture specific algorithms;
- Parallelization of algorithms;
- Pre or post processing the dataset;
- Use relevant data structures.

Space Complexity

Space complexity works as the same principle as time complexity. When measuring time complexity, we need to look at each line of the algorithm, whereas measuring space complexity is easier since we only need to look at where our variables are declared, how many do use and how big are each variable. We measure space complexity by counting the amount of memory space used by the algorithm on a given input size. We use asymptotic ('big O') notation, same as the time.

The complexity is always 'n', because when we are looking at a given input 'n', the size of the input is always the least amount of what we need, we always need the input. However, as we are not interested in that complexity, sometimes we split between input space and auxiliary space. What's important is simultaneous space used.

However, time is 'infinite', space is finite. We need to consider the demanding applications and our primary memory (RAM). Time complexity is generally bigger than the space complexity. However, there's exceptions, for example, for insertion sort algorithms, space complexity is smaller than time complexity.

```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        i = j - 1
        while i > 0 and A[i] < key
            A[i + 1] = A[i]
            i = i - 1

        A[i + 1] = key
```

Time complexity: In the worst case, the input array is sorted in reverse order, the inner loop goes i steps back. Therefore, time complexity is $O(n^2)$. In the best case, the array is already sorted, the time complexity would be O(n).

Space complexity: O(1) As this algorithm only uses a fixed amount of additional space to store the 'key' variable and loop counters, the sorting is done in-place. Therefore, it doesn't require any extra storage proportional to the input size.

Complexity Classes

complexity classes are used to classify problems based on the amount of resources they require when solved by an algorithm. The resources can be time, space, or other computational resources. For certain problems, we can prove that any conceivable solution has a certain minimum complexity. Here are a few important complexity classes:

- P or PTIME is the collection of decision problems that can be solved by a program in polynomial time. Which means the time required to solve the problem can be bounded by a polynomial function of the size of the input. Problems in P are considered feasible/tractable/efficiently solvable. Often decision problems are polynomial time solvable.

- NP is nondeterministic polynomial. It is basically the same as P, but it is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time. NP is also the collection of decision problems that can be verified by a deterministic machine in polynomial time.

1st Algorithm

(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Ceel_assignment/ApplyingKnowledge/FeatureHomeworkVideoClassification.ipynb)

This code is searching for the best matching video clips in a set of films I scraped given a key film. It does this by computing the Euclidean distance between the features of the clips in the key film and those in the other films.

```
def searchForMatch(keyFilmName, featureDictionary, frame = None, fps = 25, nBestMatches = 1):

    fragments = []

    keyClips = featureDictionary[keyFilmName]['clips']
    keyFeatures = []
    for c in keyClips:
        keyFeatures.append(np.array(c['features']))

    if frame:
        ft = 16/fps

        sf = int(frame[0]/ft)
        ef = int(min(frame[1]/ft, len(keyFeatures)-1))

        keyFeatures = keyFeatures[sf:ef]

    for film in featureDictionary.keys():
        if not film == keyFilmName:
            filmClips = featureDictionary[film]['clips']
            filmFeatures = []
            for c in filmClips:
                filmFeatures.append(np.array(c['features']))

            for i in range(len(filmFeatures)-len(keyFeatures)):
                distance = 0
                for j in range(len(keyFeatures)):
                    d = np.linalg.norm(filmFeatures[i+j]-keyFeatures[j])
                    distance += d
                distance = distance/len(keyFeatures)

                a = fragmentsExists(fragments, [film, i, len(keyFeatures), distance])

                if a:
                    fragments = a
                elif len(fragments) < nBestMatches:
                    fragments.append([film, i, len(keyFeatures), distance])
                    fragments.sort(key = lambda x : x[-1])
                elif distance < fragments[-1][-1]:
                    fragments.pop()
                    fragments.append([film, i, len(keyFeatures), distance])
                    fragments.sort(key = lambda x : x[-1])

    return fragments
```

O(1) — This is looking up the dictionary, therefore it is a constant time operation, O(1).

O(k) — This is a loop that runs once for each clip in the key film. If the key film has k clips, this operation has time complexity O(k) as the time it takes is proportional to the number of clips.

O(k) — Slicing a list takes time proportional to the size of the slice, so in the worst case, sf is 0 and ef is (k-1), this operation would have time complexity O(k).

O(nm(m-k)) — The outer for loop iterates over all films in the dictionary. If there are n films in the dictionary, the time complexity of this loop is proportional to n times the time complexity of the code inside the loop.

Inside the outer for loop, a filmFeatures list is constructed for each film. If each film has m clips on average, this operation has time complexity O(m).

This inner loop runs (m-k) times, where m is the number of clips in the current film and k is the number of clips in the key film.

Innermost for loop runs k times. It time complexity is constant (O(k)).

The time complexity of the code inside the innermost loop is constant (O(1)), because it involves only a fixed number of operations

When considering Big O notation, we generally drop constants and lower-order terms because they become insignificant as the input size grows. Therefore, the time complexity is O(1) + O(k) + O(k) + O(nm(m-k)) = O(k + nm(m-k)).

Meanwhile, for the space complexity, the keyFeatures and filmFeatures lists each store the features of m clips, so their space complexity is O(m). The fragments list stores up to p best matches, so its space complexity is O(p). Therefore, the total space complexity of the function is O(m + p).

For optimizing the algorithm, we can use more efficient data structures mentioned in the previous section to store the best matches which then can improve the time complexity of insertions and deletions.

2nd Algorithm

(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Ceel_assignment/ApplyingKnowledge/Skill-Coordinate.ipynb)

This code is a recursive function for spatial partitioning of a region into uniform areas based on some criteria. Different from 1st Algorithm, this code involves recursion. The function evaluate is calling itself within its definition. Each recursive call is working with a smaller or simpler version of the problem, which in this case is a smaller range of latitudes and longitudes.

```
def evaluate(latrange, lonrange, coordinates, statThresh = 0.7, pThresh = 0.05):
    uniform, inrange = testIfUniform(latrange, lonrange, coordinates, statThresh, pThresh)
    latdiff = latrange[1]-latrange[0]
    londiff = lonrange[1]-lonrange[0]

    if uniform:
        return [(latrange[0], lonrange[0]), (latdiff, londiff), inrange]

    else:
        a = evaluate([latrange[0],latrange[0]+latdiff/2], [lonrange[0],lonrange[0]+londiff/2], inrange, statThresh, pThresh)
        b = evaluate([latrange[0]+latdiff/2,latrange[1]], [lonrange[0],lonrange[0]+londiff/2], inrange, statThresh, pThresh)
        c = evaluate([latrange[0],latrange[0]+latdiff/2], [lonrange[0]+londiff/2,lonrange[1]], inrange, statThresh, pThresh)
        d = evaluate([latrange[0]+latdiff/2,latrange[1]], [lonrange[0]+londiff/2,lonrange[1]], inrange, statThresh, pThresh)

        return[a,b,c,d]
```

O(n) — The time complexity of this line depends on the implementation of testIfUniform, which we are not going to look at for this analysis. However, since it's processing all the coordinates, it's likely O(n), where n is the number of coordinates.

O(1) — These lines have a time complexity of O(1), as they are simple arithmetic operations.

O(1) — This line has a time complexity of O(1), as it's a simple conditional check.

This line also has a time complexity of O(1), as it's simply returning a list of values.

$O(4^k)$ — This is where the majority of the time complexity comes from. Since there are four recursive calls, and each call processes a subset of the problem that is half the size of the current problem, this results in a time complexity of $O(4^k)$, where k is the number of times the latitudinal and longitudinal ranges can be divided by 2.

To add everything up, therefore, the time complexity of this algorithm is O(n) + O(1) + O(1) + $O(4^k)$ = $O(4^k)$.

The space complexity of a recursive algorithm is proportional to the maximum depth of recursion, because each recursive call requires additional space on the call stack. In this case, the space complexity is O(k).

As we are only analyzing this short part of whole algorithm. These are approximate complexities, as the actual time and space required will depend on a variety of factors, including the specifics of how the 'testIfUniform' function works, as well as the specific distribution of coordinates.

In terms of improving this algorithm, even though the 'big O' time complexity might necessarily be reduced. There are potential ways to improve performance or reduce complexity. As mentioned in the previous section, we can use parallelization of algorithms for optimization. Since each of the four recursive calls to 'evaluate' is independent of the others, this algorithm could potentially be parallelized. This could significantly speed up the execution in practice on a multi-core machine without reducing the 'big O' time complexity.

In addition, this algorithm could be generalized in to solve a broader set of problems. This algorithm is a specific instance of Divide and Conquer algorithms mentioned in the previous section. For generalization, for example, as this algorithm uses 'testIfUniform' to decide whether to continue dividing the space. A more general algorithm could accept any function as a parameter, allowing the user to define their own criteria for whether a space should be divided. However, as the algorithm becomes more generalized, it also tends to become more complex and potentially less efficient for any specific problem.

DATA VECTORISATION APPLICATION

taught by Julian Besems

Vectorise Images and Film

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Julian_assignment/VectoriseImagesAndFilm

Design of Algorithms to Discern Structures

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Julian_assignment/DesignOfAlgorithmsToDiscernStructures

Vectorisation of Texts

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Julian_assignment/VectorisationOfTexts

Representation and Visualise Data in GODOT

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Julian_assignment/RepresentAndVisualiseDataInGODOT

Github link (entire folder) : https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Julian_assignment/VectoriseImagesAndFilm

Neural Networks In General

Neural networks are a class of machine learning algorithms inspired by the structure and function of the human brain. They consist of interconnected layers of nodes or neurons, which are organized into input, hidden, and output layers. Each neuron receives input from other neurons, processes it, and transmits the result to other neurons in the network. Neural networks are essential directed graphs. They are particularly effective at learning complex patterns and representations from large datasets, which makes them suitable for tasks such as image recognition, video recognition, etc.

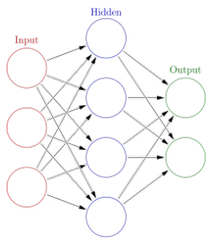


Fig 15. Artificial Neural Network
Source: https://en.wikipedia.org/wiki/Artificial_neural_network

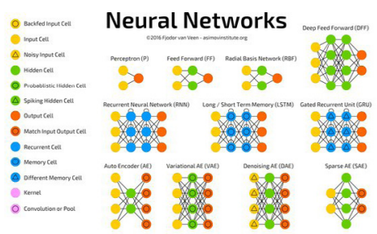


Fig 16. Various types of neural networks
Source: https://www.quora.com/What-are-the-various-types-of-neural-networks

CNN

The neural network that we used to vectorise the images and films is Convolutional Neural Network (CNN). The CNN model we used for images and for videos is based on different deep learning frameworks. The model used for image classification is based on TensorFlow: https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf.h5; the model used for videos is from PyTorch: https://github.com/kenshohara/video-classification-3d-cnn-pytorch.git. Image classification model is better at looking for similar images in terms of mood, quality, or object, whereas video classification model is looking for the motion activities in clips of frames. Both models have the same principle behind: CNN is breaking up input grid-like data, shifting focus and then rebuilding some form of new features based on that, then connect them through the interconnected layer to the next one, and breaking up these inputs into abstract understanding of it. The key part is that it's sampling the local patterns like shapes and textures, by a small focus and it keeps shifting that focus. There's a lot of overlap between the focus, therefore the classification doesn't rely on the composition of the entire image.

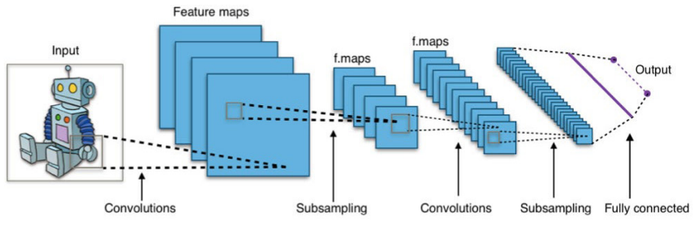


Fig 17. Convolutional neural network
Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

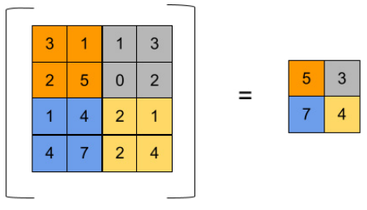


Fig 18. Explaining pooling layer
Source: https://androidkt.com/explain-pooling-layers-max-pooling-average-pooling-global-average-pooling-and-global-max-pooling/

The Difference Between The Last Layer And Second To Last Layer In CNN

Second to last layer

The second to last layer is the fully connected layer that summarizes the spatial information in the feature maps. It might be different pooling layers depending on the models. For the image and video classification model that we used, the second to last layer is the global average pooling layer, which is fully connected, it reduces the spatial dimensions and summarizes the spatial information in the feature maps by computing the average value of each feature map across all spatial locations, resulting in a fixed-size feature vector that can be used as input to a classifier or further processing.

For example, in the image classification model we used (https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf.h5), the global average pooling layer gives a 1-Dimentional output, it computes the spatial average of the feature map of size 7x7x1024 generated by the previous convolutional layers and produces a feature vector of size 1024. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/IstanbulImageClassification.ipynb)*

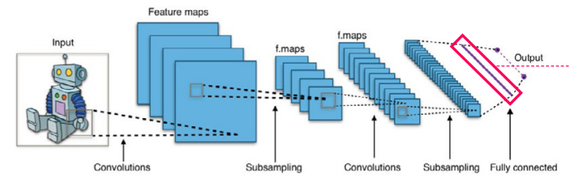


Fig 19. Convolutional neural network
Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

(layers in the front)

...		
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1, 1, 1024)	0
dropout (Dropout)	(None, 1, 1, 1024)	0
conv_preds (Conv2D)	(None, 1, 1, 1000)	1025000
reshape_2 (Reshape)	(None, 1000)	0
predictions (Activation)	(None, 1000)	0

Second to last layer

Total params: 4,253,864
Trainable params: 4,231,976
Non-trainable params: 21,888

Fig 20. original moddel layers
Source: author

In the video classification model we used (https://github.com/kenshohara/video-classification-3d-cnn-pytorch.git), same principle as the image classification model applied.

Last layer

The last layer in CNN is the output layer. If we use a pre-trained CNN model, we can adjust the last layer to get the outputs that we want to fit in.

In the original image classification model that we used, last layer gives 1000 output values, that are correspond to the labels that trains the original model, therefore 1000 different categories. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/IstanbulImageClassification.ipynb)*

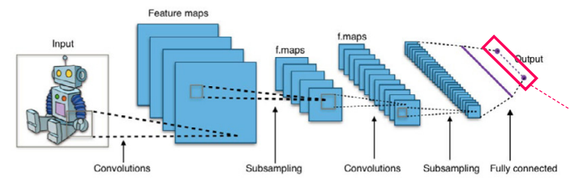


Fig 21. Convolutional neural network
Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

(layers in the front)

...		
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1, 1, 1024)	0
dropout (Dropout)	(None, 1, 1, 1024)	0
conv_preds (Conv2D)	(None, 1, 1, 1000)	1025000
reshape_2 (Reshape)	(None, 1000)	0
predictions (Activation)	(None, 1000)	0

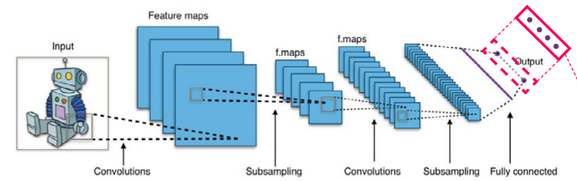
Original last layer

Total params: 4,253,864
Trainable params: 4,231,976
Non-trainable params: 21,888

Fig 22. original moddel layers
Source: author

To train the image classification model according to the different categories that we wanted, we chopped down the original model until the second to last layer (GlobalAveragePooling2D layer). Then we replaced the last layer with 3 extra layers of neurons to get the results that we want to present, which here is to alter the dropout rate, then change the number of outputs, which in this case is the categories, to 4, and use Softmax activation layer to get the possibility distribution of each category.

(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/IstanbulImageClassification.ipynb)




```
x = Dropout(rate=0.4)(model.output)
x = Dense(4)(x)
x = Softmax()(x)
model = Model(model.inputs, x)
```

3 new layers to adjust the number of categories and show possibilities

Fig 23. Convolutional neural network
Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

DATA VECTORISATION APPLICATION | Vectorise Images and Film

As an example, I scraped 300 images each for the four categories from Google: Istanbul kebab stall, Istanbul heritage tram, Istanbul mosque, then filtered the the scraped images a little for better dataset. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/DataScraping/GoogleImageScraper.ipynb)* I used one image from each category to see their possibilities of being the four categories, the result came out great. As I personally think that the simitci bread cart in Istanbul always look a bit like the heritage tram, therefore I input an image of simitci bread cart to see its possibility of being the four categories. What I expect is for the possibility of the simit bread cart image to match the tram category would be the highest, however, interestly the result came out to be the one for stall category is the highest. Then I noticed that the bread inside the cart would actually look like the kebab stall. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/IstanbulImageClassification.ipynb)*

Images in original dataset

stall	tram	mosque	taxi
[[0.997	0.001	0.	0.002]]
[[0.	1.	0.	0.]]
[[0.	0.	0.999	0.001]]
[[0.003	0.003	0.	0.993]]

Istanbul simit bread cart image

stall	tram	mosque	taxi
[[0.997	0.002	0.	0.]]

In video classification model that we used, both for score mode and feature mode, the output is the json file which contains all the data and the classification vectors.

In the score mode, the last layer outputs the class names and predicted class scores for each 16-frame clip. In the code that we use, the size is 5, as one clip contains 16 frames, the code averages the scores of 5 clips which can also be seen as 5x16 frames. Therefore, the outputs of score mode get the actual nature of each time point, rather than predicting the class scores frame by frame. The output json file contains lists the entire classification of the video, in that the classification is the dictionary, which contains 'video' and 'clips'. In addition, 'video' contains the video name and 'clips' contains a list of dictionaries, where each of those dictionaries contains a timestamp, the label, and the scores. For the example below, The model think that the activity of this short clip might be 'busking'. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/ScoreHomeworkVideoClassification.ipynb)*

Output layer of score mode

```
segments = json.load(file)
```

Information within the output layer

```
segments[0].keys()
dict_keys(['video', 'clips'])

segments[0]['clips'][0].keys()
dict_keys(['segment', 'label', 'scores'])
```

For example

```
'SorryToBoderYou.mp4', 5 x 16-frame clip, 'busking', '0.7490741610527039'
```

(representation here)

In the feature mode, the output layer took a step back from the last layer of the score mode. Instead of giving the scores that corresponding to the activities, the last layer of feature mode gives the features of 512 dimensional vectors for each 16 frames. Which then we can used to comparing the activities of the clips in multiple videos and search for the clips with similar activities. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/FeatureHomeworkVideoClassification.ipynb)*

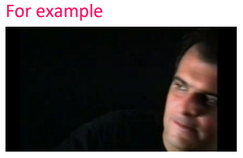
Output layer of feature mode

```
segments = json.load(file)
```

Information within the output layer

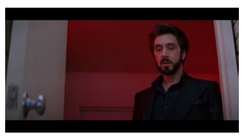
```
segments[0].keys()
dict_keys(['video', 'clips'])

segments[0]['clips'][0].keys()
dict_keys(['segment', 'features'])
```

For example



```
['frame-099599.mp4', 21, 5, 2.951256350375716]
```

other video with matching feature



Application

Image classification

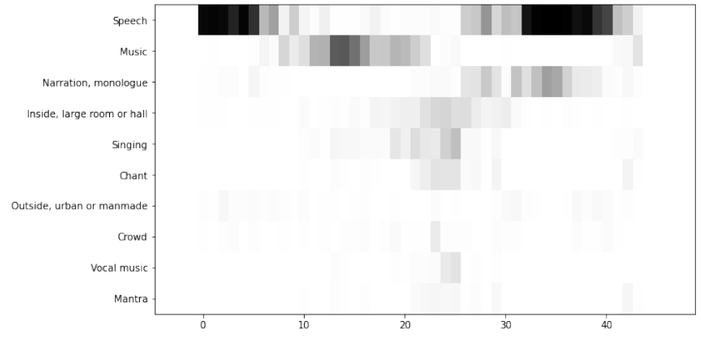
Even though CNN is primarily used as supervised learning models. It can also be adapted for unsupervised learning tasks by combining them with other unsupervised techniques like autoencoders. In these cases, the CNN learns to identify patterns or structures in the input data without explicit labels. In my group design project 'Reactivate City Memories', we try to simulate how human recall a memory. As memory is something very similar to the original thing, but ultimately different from the original. Therefore we use Autoencoder to train the model on different categories on things in Istanbul, including 'biscuit', 'cat', 'chair', 'cup', 'face', 'fish', 'flower', 'roof', 'shoes', 'transportation'. Encode the input through different layers of neurons to a latent space, then decode the latent space to get the output.

Video classification

I used video classification - feature mode to make a video collage for my individual project. The small video on the top left is the video that I used for training the feature, the video on the right is the video that is searched for matching the feature. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/VideoLink)*

Audio classification

Other than images and videos, CNN can also be applied on audio files to classify the possible content within the audio. For our group project, we found a pretrained model (AudioSet: An ontology and human-labelled dataset for audio events). Using this model, we can classify and extract the content within the audio and model the process of remembering and forgeting for our design. For example, this is the original audio input *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/01-speech.wav)*, the top 10 contents within the audio file can be classified as below. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectoriseImagesAndFilm/CNNApplication-AudioClassification.ipynb)*



DATA VECTORISATION APPLICATION | Design of Algorithms to Discern Structures

Github link (entire folder) : https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Julian_assignment/DesignOfAlgorithmsToDiscernStructures

Nearest Neighbor Algorithm

In the group design project, in the second attempt of modeling the forgetting and reactivating process through 'touch', each texture image is one result that is randomly picked out of the 16 images that are generated from Stable Diffusion using the same prompts. Therefore, the results from forgetting and reactivating process lack consistency. In that case, we decided to improve the consistency of the each stage in the process using Nearest Neighbor algorithm. The nearest neighbor algorithm is used here to find the images that are most similar to each other based on the features extracted by the MobileNet model. It is from scikit-learn, which is sklearn.neighbors. It fits the model on the extracted features of the images and then finds the number of nearest neighbors for each image, which in this case is set to 5. The nbrs.kneighbors(features) function call returns two arrays: distances, which contains the distances to the nearest neighbors for each image, and indices, which contains the indices of these neighbors in the features array.

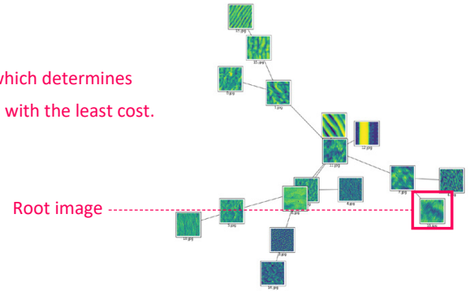
```
nbrs = NearestNeighbors(n_neighbors=5, algorithm='auto').fit(features)
```
This code fits the algorithm to features

We set the number of neighbors of each element to 5

```
distances, indices = nbrs.kneighbors(features)
```
This code extracts information

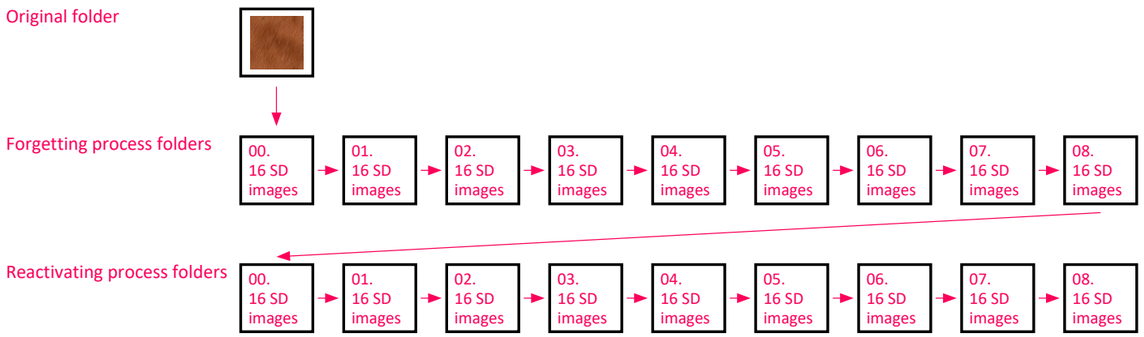
For example: the output of `indices` is `[0, 7, 11, 14, 1]` means that the first image has itself as the nearest neighbor and the next one is in place 7, and the next to that is in place 11, etc; the output of `distances` is `[0, 14.08751, 14.542966, 16.046225, 17.319742]` means that distance between the nearest neighbor to the first image is 0, because its itself, then the distance to the next one is 14.08751, etc. We can remove the first distance which is to the image itself. Then the rest of the distances can then be used as the weighted edges for generating the graph or the minimum spanning tree, so that we can visualising the connection between the images.

For example, I create a minimum spanning tree between the images, which determines the edges that are necessary to get from the node to every other node with the least cost.

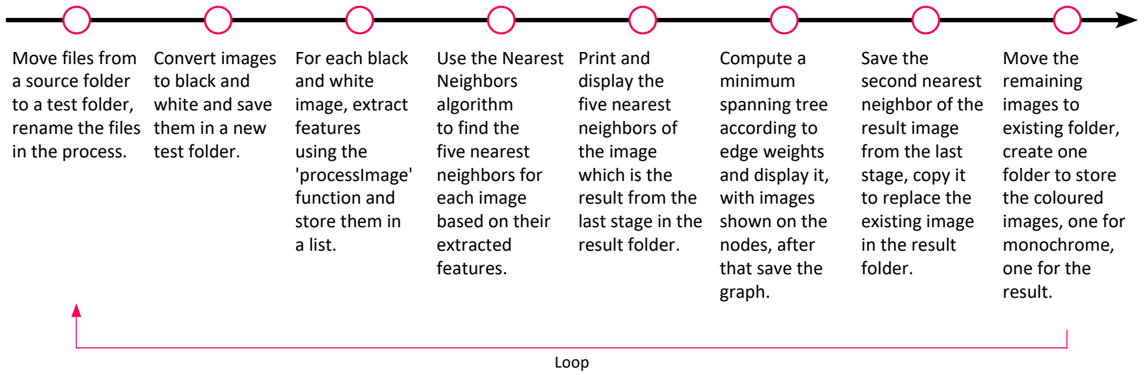


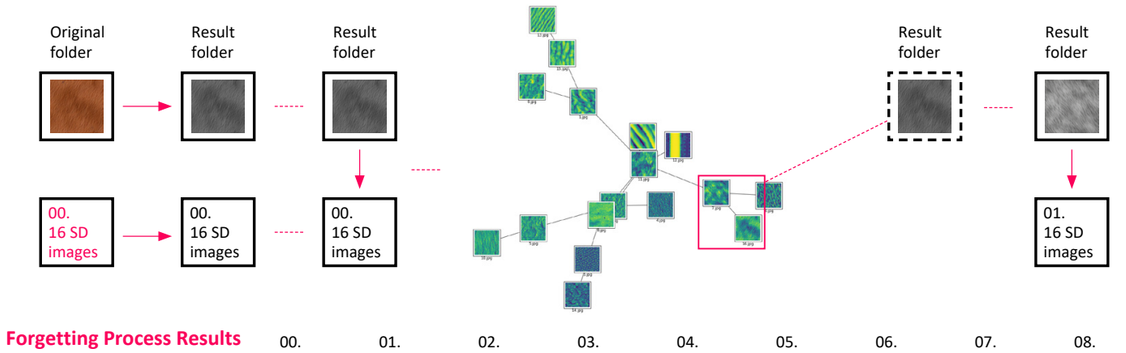
Design of Algorithms

In the group design project, we try to improve the consistency of the each stage in the process. For each texture, we need to input an original texture image to the algorith, then find its nearest neighbor in the first stage folder of forgetting process (00. folder) which contains 16 images generated from Stable Diffusion, next we need to input that result to the second stage folder of forgetting process (01. folder) which also contains 16 images generated from Stable Diffusion, etc. In the meantime, we want to store all the records we need properly for every step, including the original version of the images, the monochrome version of the images, the nearest neighbor result in the next stage in each stage, and the minimum spanning tree graph that shows the connection of the result image with the next stage images. Therefore, I defined a function that can automatically loop through all the existing folders of the texture and store the information properly. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/DesignOfAlgorithmsToDiscernStructures/04-Texture-NearestNeighbour-Auto-Graph.ipynb)*

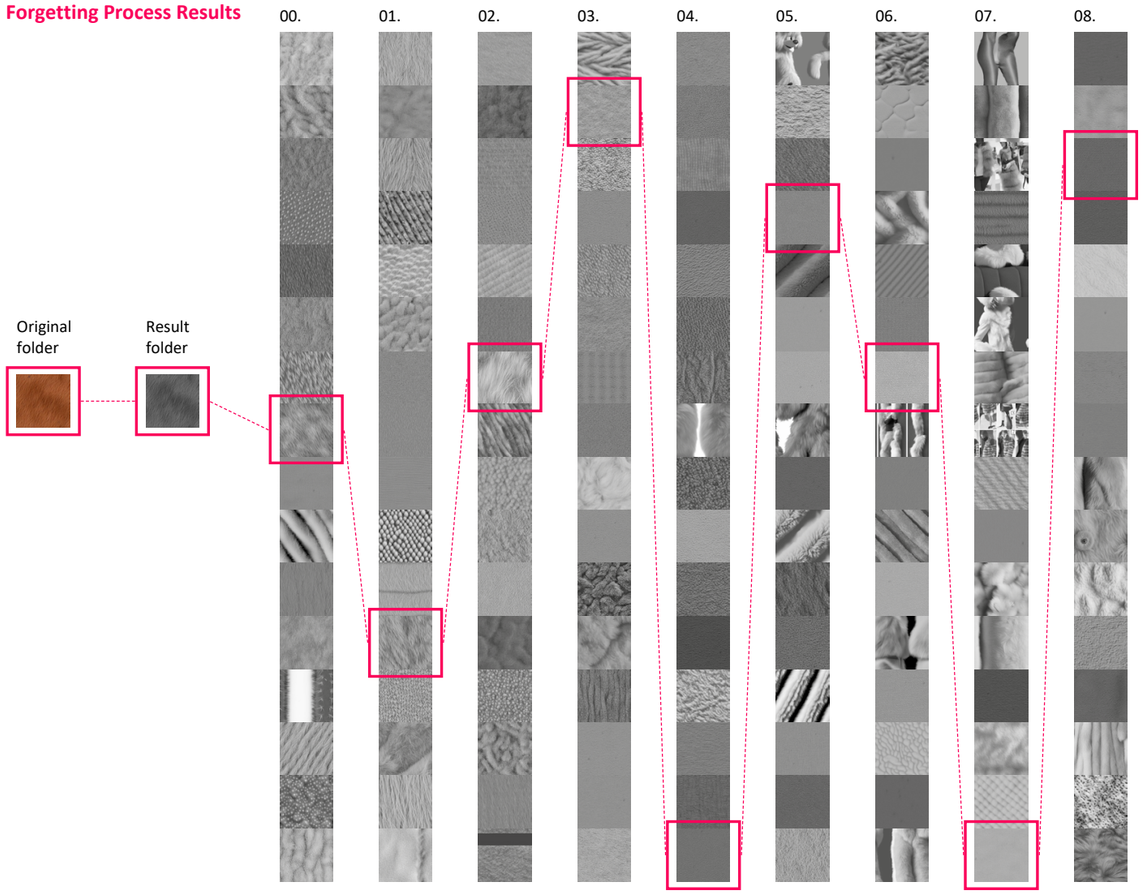


Process explanation





Forgetting Process Results



Github link (entire folder) : https://github.com/RC11-SkillsClass2022-23/XingFeng/tree/main/Julian_assignment/VectorisationOfTexts

Vectorisation Of Texts

Other than vectorizing images, videos, and audios, we can also vectorize texts and use it in natural language processing (NLP). Similar as other vectorisation I mentioned before, we use vectorisation to extract features. First we convert text to numerical vectors, then we get some distinct features out of the text for the model to train on. There are plenty vectorization techniques, including Bag of Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), and Word2Vec. (Jha, 2023)

Data Scraping

For better text vectorisation performance, we can scrape the books that we are interested in online and extract texts as our own dataset, then we can filter out where things are relevant or not. I used the Selenium library to navigate the archive.org site, then scraped 520 books which 500 of them are in English. Afterwards, I extracted information about each book, including its title, subtitle, date, and URLs for the .txt, .gif, .pdf versions of the book, then store the information in a CSV file, which in my case is tactile_books.csv *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectorisationOfTexts/tactile_books.csv)*. After checking for English books, I split the text into paragraphs. The paragraphs are then further preprocessed by removing punctuation, converting to lowercase, and removing common stopwords. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/DataScraping/SkillsClassNLP1Dataset.ipynb)*

In the data scraping and data cleaning process, removing the frequent yet meaningless 'stop words' is quite important for the later vectorisation process. As these words don't provide useful distinguishing information about the content of the document. By removing them, the noise in our data is reduced, and the vectorization methods can focus on the more meaningful words that provide more useful information about the content of the text. Otherwise, for example, if two documents have too many stop words, despite their real meaning, these documents will be considered to be similar. In addition, as I scraped 500 books for my dataset, if I keep all the stopwords, my program would crash.

Bag of Words (BoW)

BoW represents each document or sentence as a bag (multiset) of its words, it does not consider the grammar, semantics, or the order of words, but keeps track of the frequency. The value at each position in the vector can be a binary indicator of whether the word appears in the document or the frequency of the word in the document. The idea to use BoW here is to look at the paragraphs in the books as seperate entities, look at how they occur within the overall body of text, then create a vector based on how often words occur in that piece of the paragraph in respect to the whole thing (Jha, 2023). *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectorisationOfTexts/NLP1-Tactile_books.ipynb)*

After processing my dataset, I tokenized the input text. Then, these words left fit in the requirements and occured more than once in different paragraphs. Therefore, this dictionary below can be considered as the feature vector, which is what we are going to base the vectorisation of different paragraph on.

```
Dictionary(30810 unique tokens: ['acrylic', 'are', 'been', 'constant', 'determine']...)
```

For example, the text is now 30810 unique tokens.

Create the document in a link to the indexes of where they are, then the vector gets made.

```
new_doc = "Cat fur is very soft"
new_vec = dictionary.doc2bow(new_doc.lower().split())
print(new_vec)

Output: [(388, 1), (4943, 1), (6936, 1), (16384, 1)]
```

For example, 388 is the index of 'cat' in the 30810 unique tokens and only occured once in the new document.

Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF is another frequency-based method. Different from BoW, TF-IDF not only considers the frequency of a word in a particular document, but also is a numerical statistic that's intended to reflect how important a word is to a document. The result is a vector representation for each document, with each element representing a TF-IDF score for a specific word in the document. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/VectorisationOfTexts/NLP1-Tactile_books.ipynb)*

After calculating the BoW vectors, we used models.TfidfModel which is fit on the vector representation of all of the paragraphs in respect to this frequency counting of the entire collection of books, which makes the other parts of text searchable. As TF-IDF model looks at all of the words in all of the pieces of text. It is looking at proportionally how often does a word occur in that piece of text, in how much that in proportion to how much that word occurs in all of the text. Instead of showing counts, the relevance is shown. The output below shows which word is most relevant and make the sentence most unique in respect to the entire corpus, while the other words are also contributing (Jha, 2023).

```
words = "Cat fur is very soft".lower().split()
print(tfidf[dictionary.doc2bow(words)])

Output: [(388, 0.16971666412434216), (4943, 0.34703842720071204), (6936, 0.5951543363542922), (16384, 0.7046643881190842)]
```

For example, different from before, now we can see that the word 'soft' is what makes this sentence most unique.

Then, we can calculate the cosine similarity between a query document, which is represented as a TF-IDF weighted BoW vector, and each paragraph in the corpus. According to the cosine similarity values, we can get the paragraphs that are most similar to the query.

For example, we can get paragraph that is most similar to "walking on stone road" in the corpus.

```
Output: which the Oribatidce possess; when the creature is walking they are in continual movement, their points constantly
touching the ground or surface on which the animal is walking. The articulations appear to be ginglymous except the terminal
one.
```

Word2Vec (W2V)

Different from BoW and TF-IDF, W2V attempts to capture the context and semantics. This is a predictive deep learning based model, which aims to provide vector representations of words that capture the meanings and semantic relationships between words. Word2Vec doesn't work at the document level, but rather it provides a vector representation for each word. These vectors are generated in such a way that words with similar meanings or that appear in similar contexts will have similar vectors. (Vatsal, 2023)

For example, words like "King" and "Queen" would be very similar to one another. The 2 dimensional embedding vector of "king" - the 2 dimensional embedding vector of "man" + the 2 dimensional embedding vector of "woman" yielded a vector which is very close to the embedding vector of "queen".

```
King    -    Man    +    Woman    =    Queen
[5,3]   -    [2,1]  +    [3, 2]   =    [6,4]
```

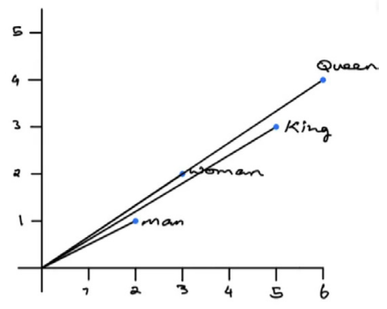


Fig 24. Word2Vec Explained
Source: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71

DATA VECTORISATION APPLICATION | Representation and Visualise Data in GODOT

Github link (entire folder) : https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Julian_assignment/RepresentAndVisualiseDataInGODOT

Godot

Godot is a gaming engine that obtains powerful 3D and 2D graphics capabilities allow us to visualize data in various ways. When we try to code the action in Godot, we can use GDScript, Godot's built-in scripting language, to dynamically load, manipulate, and visualize data. This could be useful if we want to create a more interactive visualization.

Further Iteration

In the 2D exploration in Godot, we realise that it has a lots of connections to the image analysis we operated using python. Even though I haven't succeed the exploration, I have a general idea on how I might achieve it.

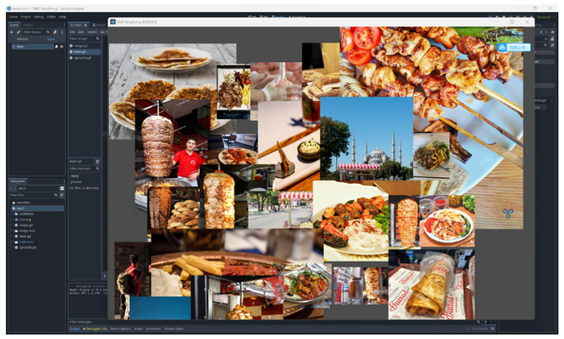


Fig 25. Exploration in Godot
Source: author

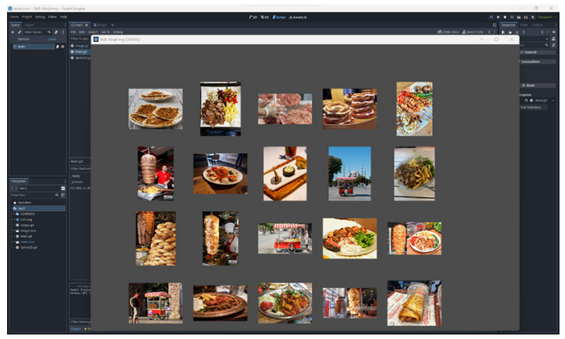


Fig 26. Exploration in Godot
Source: author

Self-Organizing Map (SOM)

Take SOM as an example. We used a pretrained deep learning model, MobileNet. To look at the process that we are using SOM. *(Github link: https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Julian_assignment/DesignOfAlgorithmsToDiscernStructures/SOM.ipynb)*. After Initialize SOM and training data, we train SOM. We first load a set of images from a directory, uses the MobileNet model to extract features from each image, and stores these features in a list. Each image is then represented as a high-dimensional vector of features.

We extract the using this function along with the MobileNet model.

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

Then we add the features to the feature list.

```
features = []
for m in momaFiles:
    path = os.path.join('C:/Users/itsfr/Desktop/Skill Class/SkillsClasses-main (1)/SkillsClasses-main/06_PythonRecap_SOM/
Stallimages', m)
    f = processImage(path, model)
    features.append(f)
```

Then the SOM is trained using the image feature vectors. The size of the SOM and the learning parameters are different this time. Each image is assigned to the BMU in the SOM for its feature vector. The images assigned to each neuron are stored in a list. For each neuron in the SOM, the image whose feature vector is closest to the neuron's weights is selected. These selected images are arranged in a grid that corresponds to the layout of the SOM. Then we display them and store the close ones in the same folder.

Fig 27. SOM output
Source: author

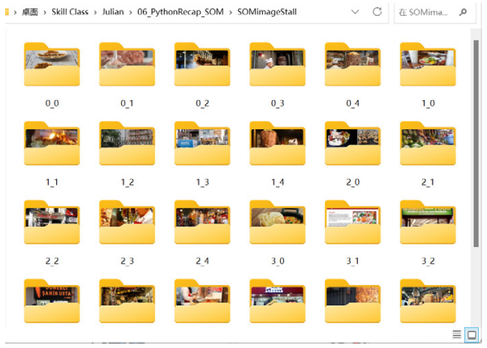


Fig 28. SOM output
Source: author

In order to make association with Godot, we need to export the data in a format so that Godot can read them. Then we can write script in godot to open and read the data. Later we can place sprites based on the SOM grid positions. Then we can make interaction with them, for example, hide certain images according to its feature vectors.

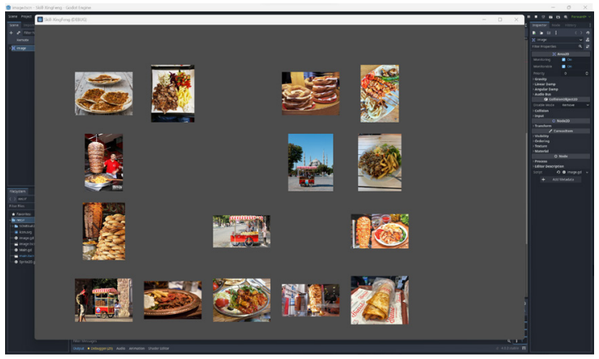


Fig 29. Interaction in Godot
Source: author

Instead of just create 2D visualization for SOM, we can also try 3D visualization. As each folder contains close images, we can represent each image as a sprite and position them in a way that reflects its position in the SOM grid and also its layer based on the folder it belongs to.

Other Ideas

Other than SOM, maybe we can visualize data for Nearest Neighbour as well. In the previous pages, I mentioned how I need to use the nearest neighbour result for the original image in one set of images, than use that result to find the next result in the next set of images. We may can create the nodes for each image in Godot, then interact within the images. For example, when I click on the original image, the nearest neighbour in the next set of images would appear, while the rest of the images would spread in a tree, etc.

3D MODELING

taught by Joris Putteneers

Scraping and Mapping

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Joris_assignment/blender-py_assignment

Mapping the Earthquake

Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Joris_assignment/blender-mapping%20the%20earthquake

Stable Diffusion in Blender

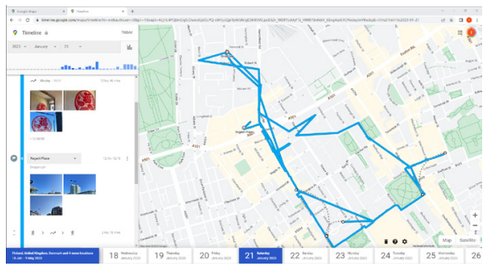
Github link: https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Joris_assignment/blender-ai

3D MODELING | Scraping and Mapping

Github link (entire folder) : https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Joris_assignment/blender-py_assignment

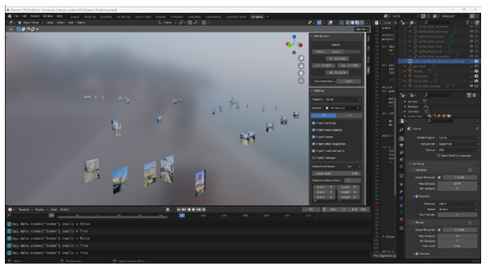
For this section, we are using our own gpx data together with the photo we took to model the path and view along the way.
First, I took a walk around the university with google recording my path and took a lot of photos along the way.

Workflow

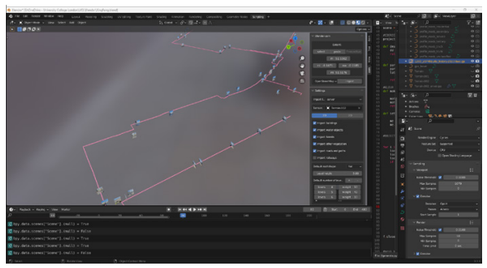


STEP 01 Use Timeline in google maps to get the trace of the location that took the photos. In this step, I need to clean up the recorded path a little, as it is not always very accurate.

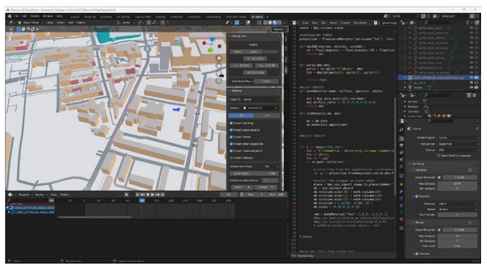
STEP 02 Use google earth to export our trace into a path/gpx file for us to map our path in blender.



STEP 03 Import photos and link them together with the gpx file.



STEP 04 Visualise the path and set camera parent to the path.



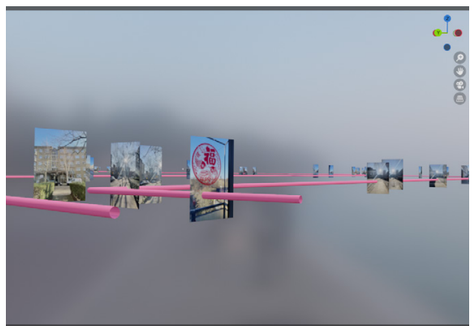
STEP 05 Import osm mapping for a better context. Render the image frams and use them to generate an animation.

In the future, if we want to record the information about the site that we design, we can just simply walk around and take photos. Then we can import osm to get the model for the context, and simulate the path in blender to get an overview.

Result

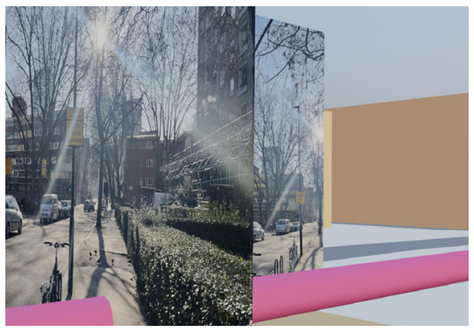
Github link (video) : https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Joris_assignment/blender-py_assignment/FINAL-OutputVideo

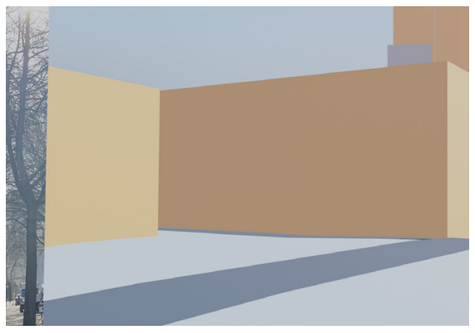
Overview



Phoetos will set on the path according to its gpx data.

Walk



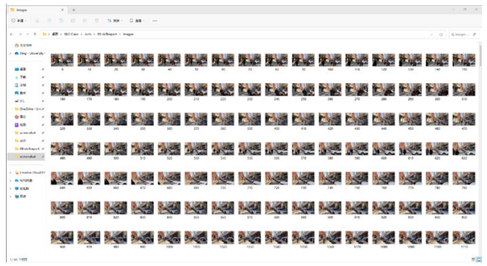


3D MODELING | Mapping the Earthquake

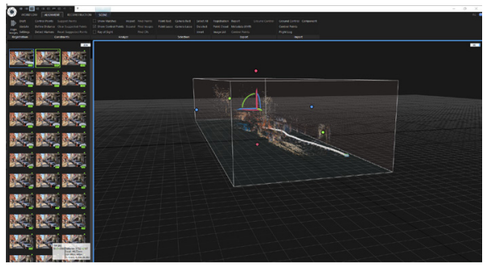
Github link (entire folder) : https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Joris_assignment/blender-mapping%20the%20earthquake

For this section, I scraped a walking tour video from our design project site, Istanbul, then I used RealityCapture, Houdini together with blender, to reconstruct the scene and add a bit of flavour to it. This technique is more powerful than the previous one, it can reconstruct the whole context model with a walk tour video.

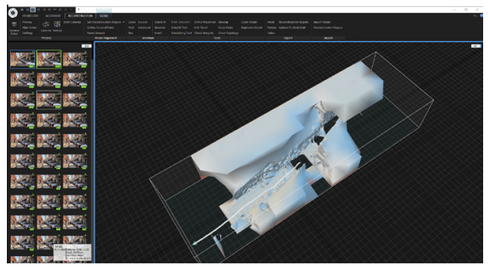
Workflow



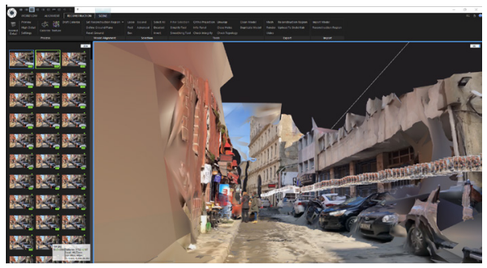
STEP 01 Video scraping on youtube using python, then extract the image frames for reconstructing the mesh.



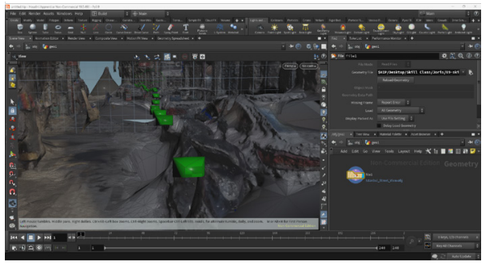
STEP 02 Select a group of images and align them in RealityCapture. As the camera that came out a bit loose, so I repeated the last step and extract the frames more fequently than the last time. After aligning, pull the boundry of the box to cover the whole model and cameras.



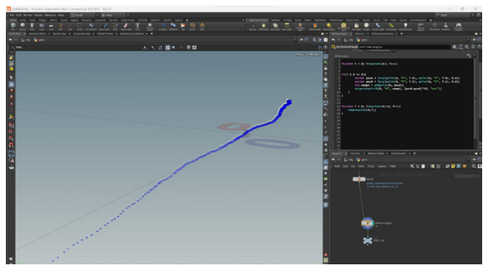
STEP 03 Generate mesh.



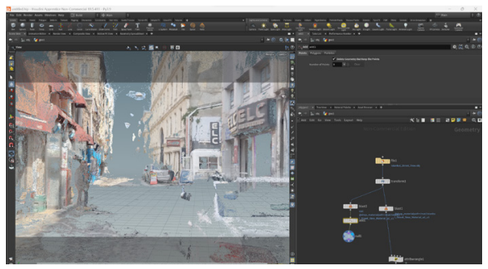
STEP 04 Applying texture on the mesh.



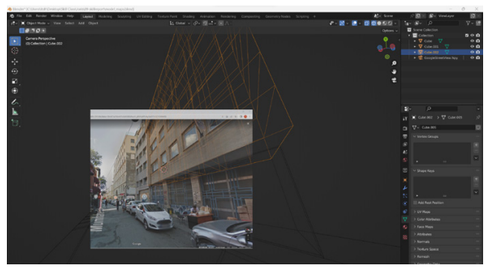
STEP 05 Import the model to Houdini, now deal with the cameras.



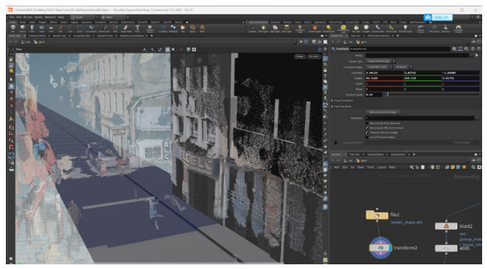
STEP 06 Vectorise the central points of each camera to be the coordinates. Use the camera to reconstruct and simulate the original camera path.



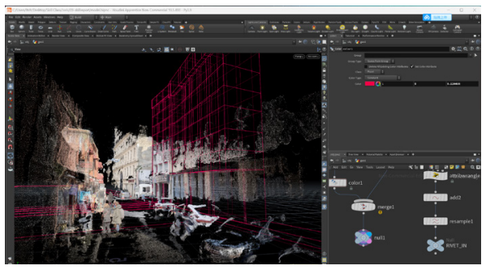
STEP 07 Adjust the camera scene for getting better angle for render. Delete the geometry of the model but keep the points of it to create this point cloud.



STEP 08 Find one obvious building in the video on google map, adjust the angle of the street view and screen shot and use fspy to set coordinates on the screens shot. Import the fspy file in blender to reconstruct the building and ground according to the actual context.



STEP 09 Import the building geometry made in blender to Houdini, and find its corresponding position.

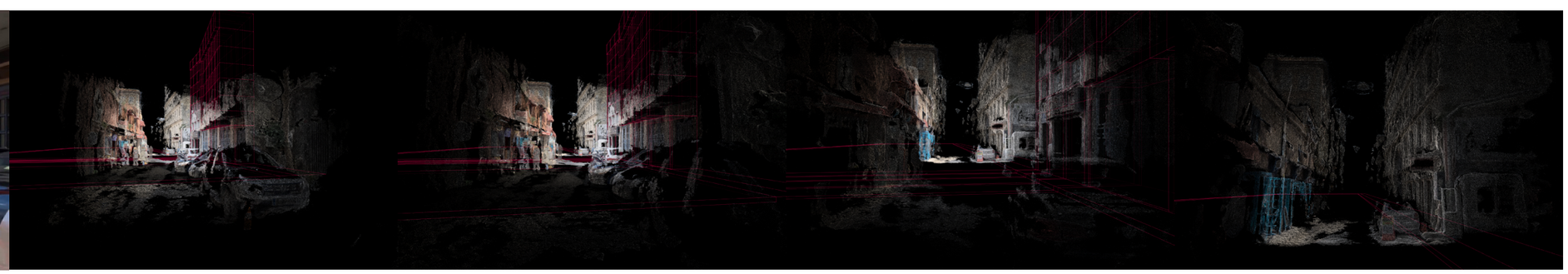


STEP 10 Set the building geometry to wireframe mode and adjust it to a preferable style. Afterwards, render the frames using the original camera path. In cmd combine the frames and create an animation.

Original Video

Result

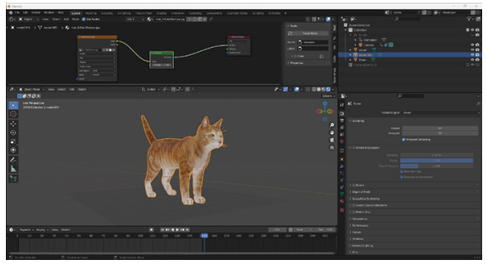
Github link (video) : https://github.com/RC11-SkillsClass2022-23/XingFeng/tree/main/Joris_assignment/blender-mapping%20the%20earthquake/FINAL-OutputAnimation



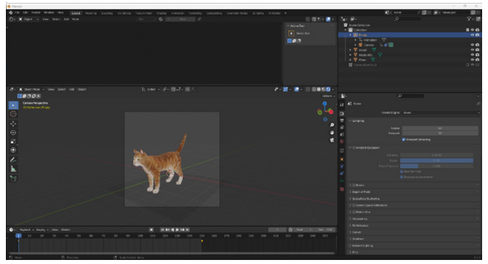
Github link (entire folder) : https://github.com/RC11-SkillsClass2022-23/22136945/tree/main/Joris_assignment/blender-ai

For our design project: 'Reactivate City Memories', we are trying to use sensory data as a tool to collect and reactivate the forgotten space through the process of remembering and forgetting. As I'm modelling the process of remembering, forgetting, and reactivating the memory through 'touch', in this section, I'm trying to model the reactivation of cat fur using the tactile adjectives I get through the process as prompt: textured, woven, embossed, fluffy, fuzzy, shaggy, seek, grooved, bumpy.

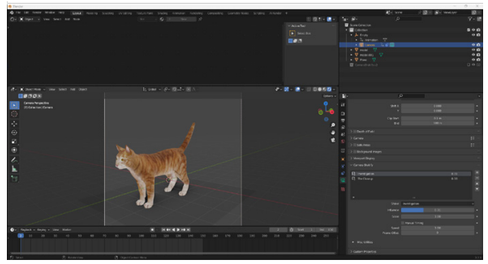
Workflow



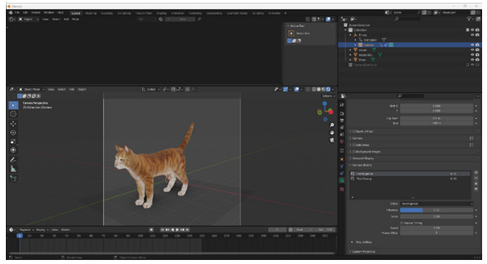
STEP 01 Import original cat geometry with texture and adjust the model by pressing 'G' to move its position and 'S' to scale it. Then in the shader editor to replace 'BSDF' tab with 'Emission' tab, so the model would not have shadow on its own.



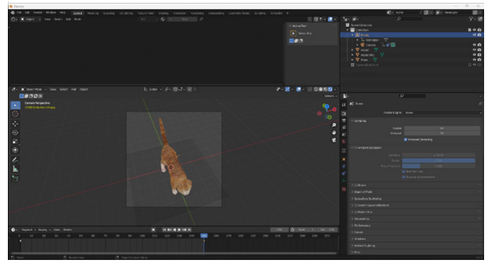
STEP 02 Create an 'Empty', place it in the center of the model, then set parent to 'camera' to allow the camera viewport can rotate or move around the 'Empty' or the model.



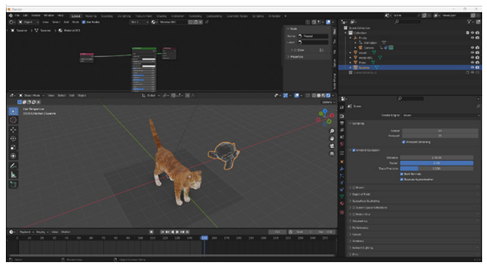
STEP 03 Add camera shakify 'Investigation' and 'The Closeup' to add a sense of reality to the animation.



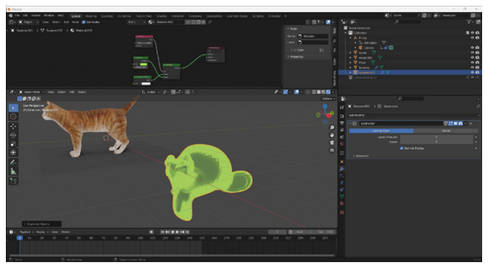
STEP 04 Adjust the length of the frame to a preferable length. Here I set the end frame to 150. After that, set the frame to the first frame, then set an initial view for the first frame of the animation. To achieve that, adjust the position and angle of the camera for a preferable first frame. Then insert a keyframe for frame one by pressing 'I'.



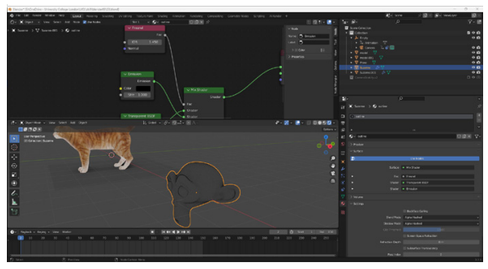
STEP 05 The set the frame to the end frame, which is 150 here, then set an end view for the end frame of the animation. Same as the last step, adjust the position and angle of the camera for a preferable end frame. Then insert a keyframe for frame one by pressing 'I'.



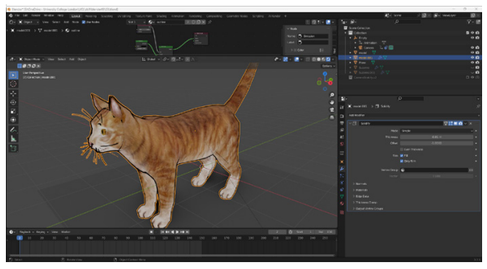
STEP 06 To create a cartoon-look for the model, first add a default mesh from blender, which here I use the monkey.



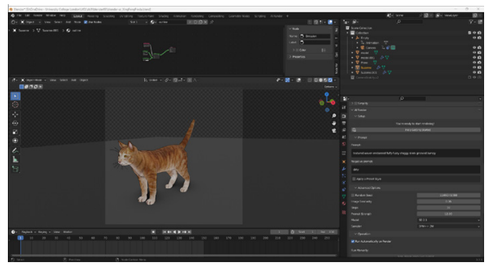
STEP 07 Then in the shade editor, connect 'Fresnel', 'Emission', 'Transparent BSDF' to 'Mix Shader', then connect 'Mix Shader' to 'Material Output'. Copy the monkey model in the exact same position, then add 'Subdivision Surface' and 'Solidify' modifier to it to add a cartoon-look filter and an edge to it.



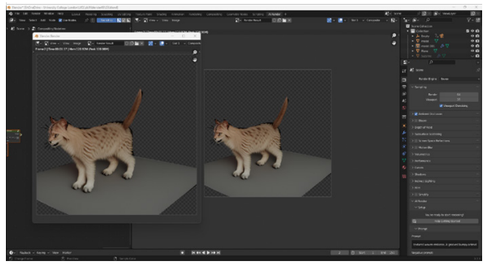
STEP 08 Rename the material of the monkey model to 'outline' in the shader editor.



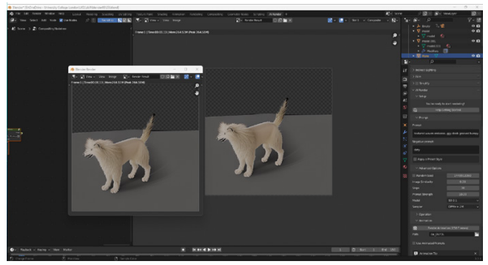
STEP 09 Copy the cat model in the exact same position and change its material to 'outline'.



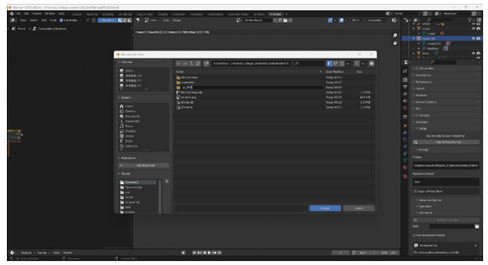
STEP 10 Under the render properties, set the image size to 512x512, fill in the prompt and negative prompt for Stable Diffusion and adjust the image similarity and prompt length.



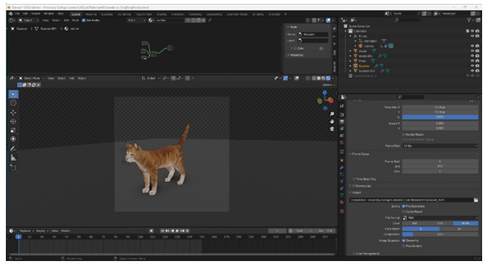
STEP 11 Press 'F12' to see the render result generated by Stable Diffusion.



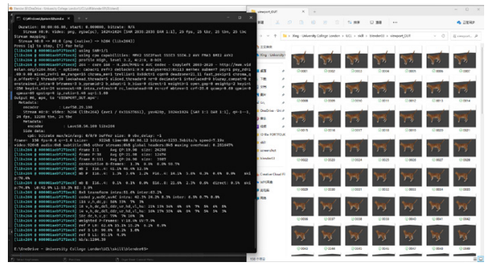
STEP 12 Play with the parameters in the last step, get a preferable result.



STEP 13 Once choose the preferable result, choose a folder as the output path, then uncheck 'Random Seed' to render animation.



STEP 14 To create an animation of the original cat model with the original texture, set the output folder, adjust the output resolution, then press 'Viewport render animation' under the 'view' tab.



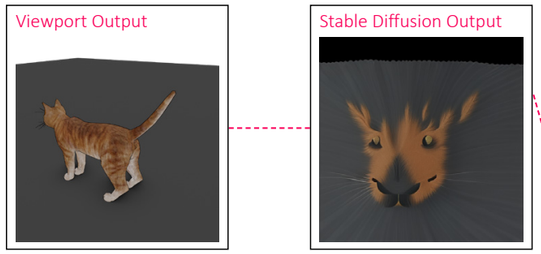
STEP 15 In cmd, use ffmpeg.exe to combine the frame images generated by Stable Diffusion and create a video. Same as the last step, create a video using cmd for the viewport render images.

Github link (video) : https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Joris_assignment/blender-ai/FINAL-OutputAnimation/AI_OUT2.mp4
https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Joris_assignment/blender-ai/FINAL-OutputAnimation/VIEWPORT_OUT.mp4

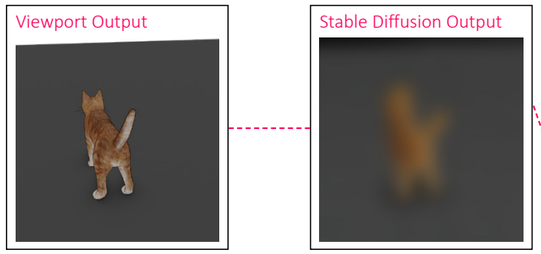
Result 01 (AI_OUT2.mp4)

For the first attempt of generating images using Stable Diffusion, I noticed that the colour of the original texture largely affected the results. In the meantime, a few unexpected results showed up:

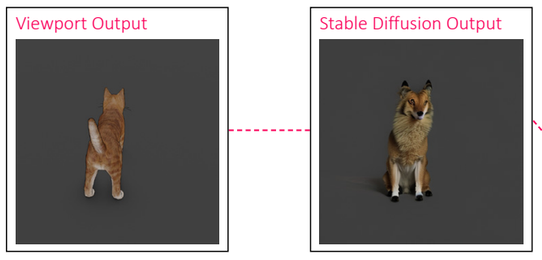
1. It's unexpected to see that, Stable Diffusion gave an output of animal face when the cat turns around.



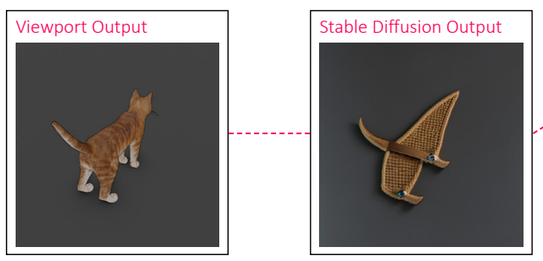
2. The blurred result suddenly came up.

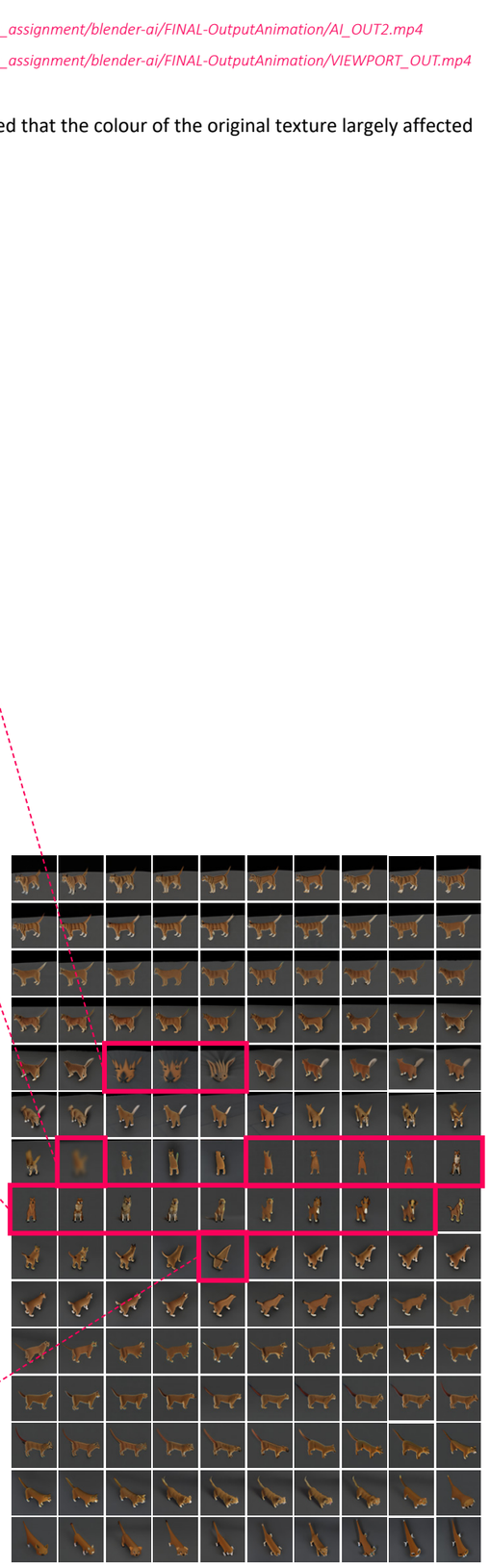


3. It's quite funny to see that, when the cat turns around, Stable Diffusion cannot recognize its head and tail.



4. Sometimes the result would suddenly recognize the cat as an object.



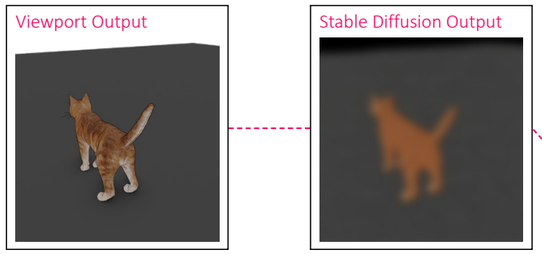


Github link (video) : https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Joris_assignment/blender-ai/FINAL-OutputAnimation/AI_OUT4.mp4

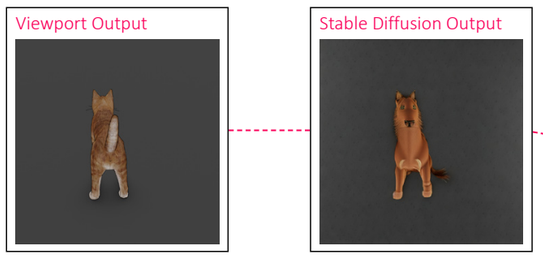
Result 03 (AI_OUT4.mp4)

For the third attempt, I change the output image size to 1024x1024 for higher resolution. However, even the random seed remained unchecked and all the other values stayed the same, the result became different with the similar issues as the last time and did not meet my expectations. Instead the texture gets plain.

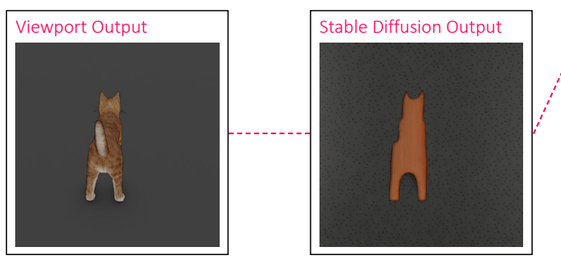
1. The blurred result suddenly came up.

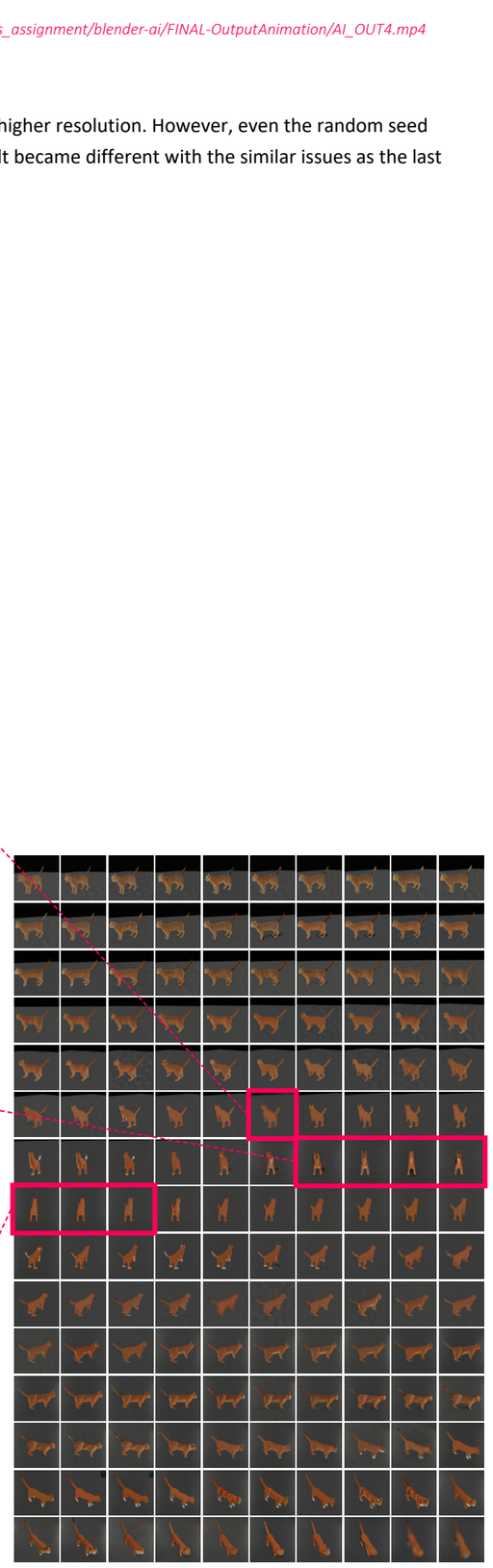


2. When the cat turns around, Stable Diffusion cannot recognize its head and tail.



3. The result would get a bit more plain on the texture, even the prompt I gave was all tactile adjectives.





Github link (video) : https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Joris_assignment/blender-ai/FINAL-OutputAnimation/AI_OUT3.mp4
https://github.com/RC11-SkillsClass2022-23/22136945/blob/main/Joris_assignment/blender-ai/FINAL-OutputAnimation/VIEWPORT_OUT.mp4

Result 02 (AI_OUT3.mp4)

For the second attempt, I decreased the image similarity and prompt strength. The result turned to be my favourite one. As the colour slightly shifted from brown to white, I think the result came a bit better than the last time, because it matched my purpose of modeling the process of remembering and forgetting of cat fur. Since I'm trying to model the process of remembering, forgetting and reactivating through 'touch', color should not be taken into the consideration, I should focus on the alternation of the texture. However, a few same issues appeared like the last time.

Future Development

For the future development, I might need to find a way to optimise the prompts to avoid the issues remained in all of the attempts.

1. The blurred result suddenly came up.

2. When the cat turns around, Stable Diffusion cannot recognize its head and tail.

3. The result would recognize the cat as an object, however this time, there's a smooth change from the cat to the object.

Reference

Jacquard machine (2023) Wikipedia. Available at: https://en.wikipedia.org/wiki/Jacquard_machine (Accessed: 5 May 2023).

Rhodes, B. (2020) Computing is born: The sumerian abacus, Medium. Available at: https://medium.com/tech-is-a-tool/the-dawn-of-computing-sumerian-abacus-83bdefb697ba (Accessed: 6 May 2023).

Jha, A. (2023) Vectorization techniques in NLP [guide], neptune.ai. Available at: https://neptune.ai/blog/vectorization-techniques-in-nlp-guide (Accessed: 8 May 2023).

Vatsal, V. (2023) Word2Vec explained, Medium. Available at: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71 (Accessed: 8 May 2023).