Visualizing Object Clouds Through Energy Minimization

by

Omobola Okesanjo

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
October, 2020

Table of Contents

List of Figures

Abstract

When visualizing an object cloud, the pairwise similarity between an object and a central object of interest is used to determine the position of each object within the cloud. This however does not capture the semantic relationship of all the objects and it reduces the expectation of finding an object when performing visual search. To generate a semantic object cloud, we define and subsequently minimize an energy function that captures the pairwise similarity amongst all the objects within the cloud. The energy is minimized using several statistical machine learning techniques and we show that the generated layouts from such techniques outperform those of other object cloud algorithms on a variety of metrics for evaluating word and object cloud layouts.

Acknowledgements

I would like to thank my supervisor Dr. Stephen Brooks for his guidance and direction in writing this thesis. Without his help, this thesis would never have been completed. I would also like to thank my parents and siblings for their love and support during the development of this work. My discussion with Tola were most helpful during this period.

Chapter 1

Introduction

1.1 Background and Motivation

Imagine that in the not-so-distant future, a designer sketches a sports car for an upcoming video game and on his/her see-through Augmented Reality (AR) headset appear several miniature 3D models that look like the car that was just sketched. Focusing on the search results in the headset, the game designer taps on a desired model and proceeds to incorporate it into the video game.

Consider another scenario, a young family just moves into a new house and are deciding what furniture to purchase for the living room. The mother takes out their tablet and mentions a piece of furniture - a collection of 3D furniture that fit her description appears on the tablet screen. The family selects one and through the screen, they can see how a life size version fits within the living room.

Given the recent advances in shape recognition, text-to-speech technology, and headsets that are known as Head-Mounted Displays (HMD), the above scenarios are not far fetched. In the near future, many consumers will be able to browse and interact with virtual 2D or 3D objects in their everyday lives as a result of the ubiquity of smart phones and headsets. However browsing a collection of objects through the view-port of a small device such as the above mentioned requires the compact use of a limited space.

When querying Computer-Aided Design (CAD) models from commercial search engines such as TurboSquid and 3D Warehouse, results are typically presented in a grid of discrete rows and columns. This classic arrangement of search results however can make it tedious to find and compare different sets of result especially if the collection of models cannot

be intuitively arranged in such manner. An alternative to representing such results is as a cloud, where all the models are clustered onto the screen. Such a cluster can take on arbitrary shapes and allow for a more intuitive arrangement of the search results. Tightly packing the cluster also maximizes the use of the limited display space and when there is an ordering within said cluster, it has been shown to facilitate the faster recognition of 3D models on tablets [21].

Object clouds, as their name suggests, are analogous to word clouds in that both are a compact visual summary of various items. However unlike word clouds that are limited to words, object clouds encompass a variety of 2D and 3D items. Like word clouds, the items in an object cloud also have varying sizes which indicates their degree of similarity to a given object of interest. This similarity is based on the how many visual features are shared between a pair of objects. Objects with the highest degree of similarity to an object of interest are larger and objects with lesser degree of similarity are smaller. To facilitate faster recognition of objects, the object of interest or the object with the most degree of similarity is placed at the center of the cloud. This however need not be the case all the time. An example of an object cloud with the object of interest at the center is given in figure 1.1.

Figure 1.1: An object cloud. The object of interest is the deep blue truck at the center of the cloud (Hong and Brooks, 2016) © 2016 IEEE.

Current algorithms for visualizing object clouds attempt to place objects at a distance that also reflects their similarity to the object of interest. Objects that are most similar to the object of interest are placed closer to the center of the cloud and objects that are less similar are placed farther away. This ordering has been shown to facilitate faster recognition of objects than a random or a grid-based ordering [21]. However utilizing the pairwise distance between each object and the object of interest does not create a semantically accurate ordering within the cloud. To create a semantically accurate ordering, the pairwise distance amongst all the objects in the cloud needs to be used instead.

Although semantic object clouds can be generated using semantic word cloud algorithms. We find that algorithms such as the seam carving and context preserving algorithm do not adequately compact items within a cloud nor do they accurately represent the semantic position of the items due to the nature of the graph used or the types of forces applied.

In order to create semantically accurate and more compact clouds, we approach the problem of constructing an object cloud from an optimization perspective. To do so, we formulate an objective function that represents the energy within an object cloud. The energy is defined over a set of latent variables and it encapsulates the aesthetic requirements of a semantic object cloud. These aesthetic requirements are:

- Objects should be compactly placed together without overlaps
- Similar objects should be closer and dissimilar objects should be farther

The objective function is then optimized using several optimization strategies such as gradient descent and majorize-minimize which result in several algorithms for visualizing object clouds. Using the energy function, we can monitor the performance of our algorithms and for some, terminate them when they ceases to minimize the energy within the cloud. This however contrasts with other cloud visualization algorithms that rely on a set number of iterations from the user which may be insufficient to properly minimize the energy.

1.2 Problem Statement

The objective of this thesis is to express semantic object clouds, and by extension their word cloud counterparts, as a real-valued function that can easily be optimized. The function measures a cloud based on the two requirements that:

- Objects in the cloud should be compactly placed without overlaps
- Similar objects should be closer and dissimilar objects should be farther

Optimizing this function should therefore correspond to an algorithm for constructing an aesthetically pleasing object or word cloud within a limited display space such as those of an AR headset.

1.3 Contributions

In order to fulfil the above objective, this thesis proposes a real-valued function to describe and quantify the aesthetic of object clouds. We also propose a few algorithms for the generation of object and word clouds based on the optimization of the proposed function. Furthermore we evaluate and contextualize the performance of our proposed algorithms in relation to existing algorithms for the generation of object and word clouds.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. In **chapter 2**, we review the relevant literature on object and word cloud generation. We also review the literature on dimensionality reduction and graph drawing which form the basis of our intuition for the objective function. In **chapter 3**, we extend the ideas discuss in the previous chapter to object and word clouds. We discuss the formulation of the objective function as well as how it describes an aesthetic object cloud. We also discuss various optimization techniques for the objective function and outline algorithms for generating said clouds. Afterwards we compare the performance of these algorithms to existing algorithms for generating object and word clouds. In **chapter 4**, we extend the discussion of the objective function to include latent variables. We discuss how these latent variables can generate both random or semantic object clouds, which in turn allows us to generate object clouds with varying degrees of randomness and semantic order. Finally in **chapter 5**, we discuss the limitations of our approach, as well as suggest areas for future work and improvement.

Chapter 2

Literature Review

In this chapter we briefly discuss the topics relevant to the development of our objective function and its subsequent optimization. We begin by discussing visual search in Augmented Reality (AR) as it pertains to finding objects within an object cloud. We then discuss graph drawing which is the underlying idea of our work. Next we discuss dimensionality reduction which is important in generating semantic word clouds. We relate the task of graph drawing with dimensionality reduction and discuss a well known optimization technique for dimensionality reduction. Finally we discuss current algorithms for visualizing both object and word clouds.

2.1 Augmented Reality

Augmented Reality (AR) is an interface through which users perceive and interact with Computer Generated Imagery (CGI) that is superimposed on to the real world. The CGI is seamlessly blended with images from the real world so that a user perceives it as though it exists in the real world. Typically the CGI images are used to enhance the user's experience of the real world either by providing more information about items in the real world or by introducing new items into the real world. This is in contrast to Virtual Reality (VR), where the real world is replaced entirely with CGI and the user only perceives a virtual world.

One can think of these two types of realities as existing on different ends of a mixed reality spectrum. On one end of the spectrum, is the real world with little to no virtual content and on the other end of the spectrum is a virtual world with an entirely virtual

content. Virtual reality sits on the virtual end of the spectrum but augmented reality sits anywhere in between the spectrum as long as the user is still immersed in the real world with some virtual content and not entirely immersed in a virtual world.

Formally for an interface to be considered as augmented reality, it needs to meet the following requirements [5]:

1. It must combine both real and virtual content
2. It must allow real time interaction with its contents
3. Its contents must be registered as 3D

2.1.1 AR Display

AR displays typically combine both real and virtual images using one of two techniques: video-based and optical see-through technology. In this section we briefly discuss both kinds of technology.

Video-based Displays

Video-based AR displays, as their name suggests, use video cameras to obtain images of the real world onto which CGI objects are superimposed. The cameras are typically behind the display screen so that a user sees what is in front of them but this is not required as the camera may be elsewhere capturing a different scene. While video-based displays are limited to tablets, smart phones and desktop devices, headsets like the Google Cardboard - which allow for the insertion of mobile phones - have led to the availability of video-based HMD.

To combine both virtual and real world images seamlessly, video-based displays rely on the use of computer vision and image processing algorithms. Powerful Central Processing Units (CPU) or Graphics Processing Units (GPU) are also required in order to quickly update the superimposed CGI contents and allow the user interact with them in real time.

Optical see-through Displays

Optical see-through displays, unlike video-based displays, do not digitize the real world. Instead light from the real world passes through the optical display within the device and

is mixed with virtual images that are generated within the display so that a user ends up seeing both real and virtual images together. Unlike video-based displays that primarily exist on computer devices, optical see-through displays exists on specially built HMD.

These HMD use a variety of optical technology such as: beam splitters, virtual mirrors and transparent projection films. Since this kind of display does not digitize the real world, there is no need to update the real world content which in turn allows for faster real-time interaction compared to video-based AR devices [5]. They however still require image processing algorithms and powerful CPUs to quickly update the CGI contents. Popular HMD that utilize optical see-through technologies include the Microsoft HoloLens and Google Glass.

2.1.2 Visual Search in AR

Perception is an important topic in the field of AR. It is the process by which humans use their senses to extract information from their surroundings. An aspect of perception that is of concern in AR is visual attention. It is the process by which a user filters and attends to a particular stimuli within their visual field. There are two types of visual attention: bottom-up and top-down attention. Bottom-up attention is attention that is triggered by an external stimuli within the visual field. This stimuli directs the user's attention to salient regions within the environment. Whereas top-down attention is triggered by the user's internal factors. Here the user's attention is actively driven by a set of goals, prior knowledge or expectations.

To effectively perform visual search for an object within a visual field, a user needs to utilize both types of attention [43][44]. This is because visual search is a goal oriented process with two components, each of which is driven by either type of attention. These components are: conspicuity and expectation. Conspicuity refers to a set of visual cues that are given to a user in order to make a target object salient. These cues include size, shape or contrast of the object relative to its background. By combining one or more visual cues, one can facilitate a bottom-up attention from the user towards a given target. In military applications of AR, the more conspicuous a target is within the visual field of an HMD, the faster the search time for the object [48].

Expectation on the other hand is a component that requires a top-down attention from the user. It refers to the user's expectation of the properties for a target object. These properties include where the target should be within the visual field and what it should look like. Having some idea of this, a user can drive the search for a particular target object. While most applications of visual search in AR typically focus on the conspicuity

of a given target [25][28], we can also utilize a user's expectation in order to facilitate better visual search in AR. For example, if there are trends within the visual field such as an alphabetic arrangement or a semantic arrangement of objects within the visual field, we can increase a user's expectation of where to find an object within the visual field [43]. When used in combination with the conspicuity of one or more target objects, we can design an AR system that effectively performs visual search and decreases the user search time.

2.2 Graph Drawing

Graph drawing algorithms are algorithms that use the information contained within a graph to generate aesthetic pleasing layouts of said graphs. These graphs are typically undirected graphs with a 2-dimensional (2D) or multi-dimensional layout. A layout can be described as aesthetically pleasing if it has minimal energy or if it both exhibits some symmetry and the pairwise vertex distances are close to some constant [24]. In drawing a graph, graph drawing algorithms start with an initial layout of the graph - this can be random, circular or any other layout. From the initial layout, both the vertices and edges of the graph are moved until a stop criteria is achieved. This criteria includes the number of iterations, net change in forces or net change in the energy of the graph.

In this section, we briefly discuss two different approaches to drawing graphs: force-directed and energy-based approaches. The first applies a set of spring forces to move the edges and vertices of the graph, while the latter optimizes an energy function to move the vertices of the graph into place. We also discuss how these different approaches to graph drawing are related.

2.2.1 Energy-based Graph Drawing

While drawing an undirected graph is thought of as matching the pairwise vertex distances to some constant, the matching process itself can also be thought of as a form of energy minimization whereby the energy corresponds to the discrepancy between the geometric pairwise distances and said constant. Based on this principle, energy-based graph drawing algorithms generate their layout by optimizing a given energy function for a graph.

This energy is defined over the vertices of a graph and the optimization can be done using local methods like gradient descent or global methods like simulated annealing [12][40].

Kamada-Kawai Layout

The Kamada-Kawai algorithm is a well known graph drawing algorithm that optimizes an energy function for a graph using gradient descent. It defines this energy, E, as the squared euclidean difference between the geometric distance and the ideal distance of the graph [23]. This is illustrated as follows:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} (|p_i - p_j| - l_{ij})^2 \tag{2.1}$$

where, k_{ij}, is the spring strength between vertices i and j; p_i and p_j, are their 2D positions respectively; and l_{ij}, is their ideal distance.

The ideal distance is computed from the graph-theoretic distance which is the shortest-path distance between i and j. The shortest-path distance is computed for all pairs of nodes within the graph such that for any pair of nodes, l_{ij} can be expressed as

$$l_{ij} = \frac{l_0}{\underset{i<j}{\text{argmax}}\ d_{ij}} \times d_{ij} \tag{2.2}$$

where, l_0, is the display length of the farthest pair of vertices and $\underset{i<j}{\text{argmax}}\ d_{ij}$ is the longest shortest-path distance between any pair of vertices. For limited displays, l_0, can be set to the length of the display. Similarly the spring strength, k_{ij}, for a pair of nodes can be computed using the shortest-path distance, d_{ij}, and some constant K such that it can be expressed as

$$k_{ij} = \frac{K}{d_{ij}^2} \tag{2.3}$$

The energy formulation in equation 2.1 is optimized using the Newton-Raphson method - an iterative root-finding method - which utilizes both the first and second partial derivatives for fast convergence to a solution. Despite this, it however suffers two drawbacks. The first is that computing the second partial derivatives or hessian, is computationally expensive and in this case, requires solving a system of equations on each iteration of the method [24]. The second is that despite the use of the hessian, it has been reported to get stuck at local minimas where the resulting graph while aesthetically pleasant, is still sub-optimal [12].

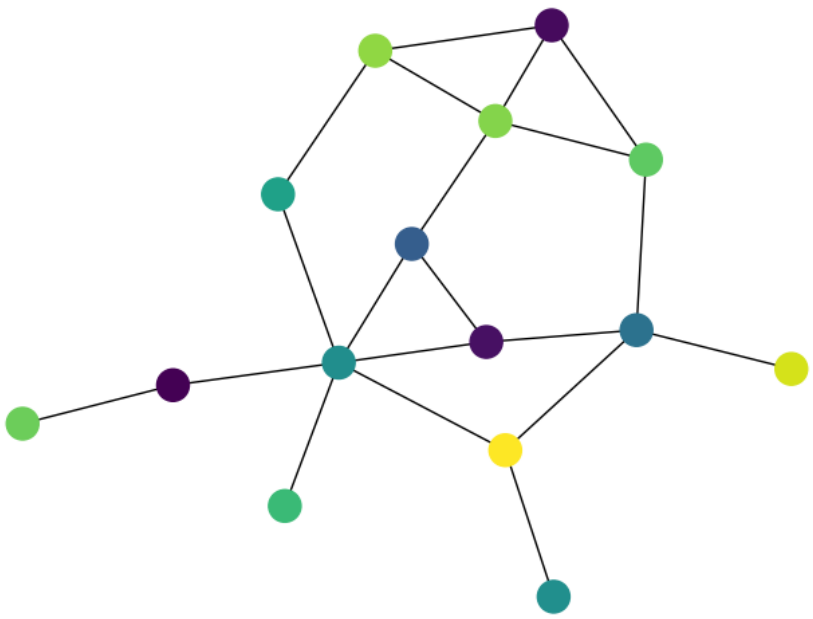


Figure 2.1: The florentine family graph generated using the Kamada-Kawai layout

2.2.2 Force-directed Graph Drawing

Force-directed graph drawing algorithms, as the name suggests, are a class of algorithms that use a set of forces to generate undirected graphs. These forces are typically modelled after spring forces and as such are either attractive or repulsive nature. A force, F, can generally be represented as

$$F \propto kx \tag{2.4}$$

where, k, is some spring constant, and x, is the object that the force is acting on.

Fruchterman-Reingold Layout

The Fruchterman-Reingold method is a well known force-directed algorithm that generates an undirected graph, $G = (\mathcal{V}, \mathcal{E})$, using a set of spring forces that are modelled after Hooke's law. The attractive forces, f_a, act on the edges, $\mathcal{E}$, of the graph while the repulsive forces, f_r, acts on every pair of vertices, $\mathcal{V}^2$, within the graph. These forces can be expressed separately as:

$$f_a(d) = \frac{d^2}{k}, \quad f_r(d) = \frac{-k^2}{d} \tag{2.5}$$

where, d, is an arbitrary input and, k, is a constant that is used for both forces. The constant, k, ideally represents the radius around a vertex and as such it is used to correct the distance between said vertex and other vertices that are either too close or far [15]. It can be calculated as follows using a value, C, that is determined experimentally.

$$k = C \times \sqrt{\frac{\text{area}}{\text{no. vertices}}} \tag{2.6}$$

Although the forces act on pairs of vertices, their input, d, is actually the distance, Δ, between any given pair. The distance between a pair of vertices is scaled by the forces and the resulting distance is used to update the position of each vertex, v, in the pair. In other words

$$v = v + \hat{\Delta} \times f(\Delta), \quad \hat{\Delta} = \frac{\Delta}{\|\Delta\|} \tag{2.7}$$

where $\hat{\Delta}$ is the displacement between the vertices. When updating the position of the vertices with the new distance, the distance is further scaled by a temperature, t, that anneals the magnitude of the forces over time. This idea comes from simulated annealing in which high energy particles within a system are cooled in order to reduce their energy and make them settle into a stable configuration [12]. Similarly the temperature is used to reduce the spring forces so that over the course of several iterations, the vertices settle into a stable position.

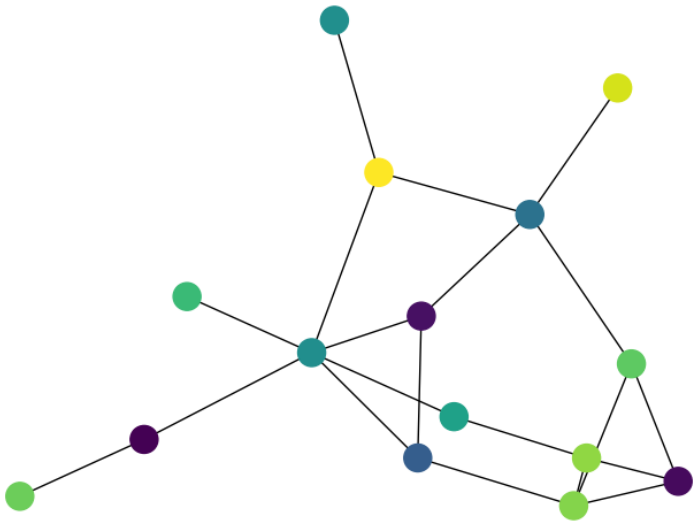


Figure 2.2: The florentine family graph generated using the Fruchterman-Reingold layout

2.2.3 Energy Optimization

It is known in graph drawing that one can derive a pair of force-based algorithms from an energy function [40]. Consider an energy function, E, for a graph $G = (\mathcal{V}, \mathcal{E})$. If the energy is composed of a pair of sub-functions, $f_\theta(.)$ and $g_\theta(.)$, it can be expressed as

$$E = \sum_{\mathcal{E}} f_\theta(d) + \sum_{\mathcal{V}^2} g_\theta(d) \tag{2.8}$$

where d, is an arbitrary input to the sub-functions and θ, is a parameter for the sub-functions. If d represents some distance function of a vertex, v, it follows that

$$\frac{\partial E}{\partial v} = \sum_{\mathcal{E}} \frac{\partial f_\theta}{\partial d}\frac{\partial d}{\partial v} + \sum_{\mathcal{V}^2} \frac{\partial g_\theta}{\partial d}\frac{\partial d}{\partial v} \tag{2.9}$$

and the position of the vertex can be updated as

$$v = v + \frac{\partial E}{\partial v} \tag{2.10}$$

Now consider the following energy function

$$E = \sum_{\mathcal{E}} \frac{d^3}{3k} + \sum_{\mathcal{V}^2} -k^2 \ \log d \tag{2.11}$$

where $d = \|v - u\|$ for vertices v and u. By differentiating this energy function, we can derive the update equation for each of the forces in the Fruchterman-Reingold algorithm [10][32]. Consequently for each energy sub-function, we have

$$\frac{\partial f_\theta}{\partial v} = \frac{d^2}{k}\frac{d}{\|d\|}, \quad \frac{\partial g_\theta}{\partial v} = \frac{-k^2}{d}\frac{d}{\|d\|} \tag{2.12}$$

where $\partial f_\theta / \partial d$ and $\partial g_\theta / \partial d$ represent the attractive and repulsive forces acting on v. While $d/\|d\|$ represents their direction. Similarly we can express the Kamada-Kawai as a force-directed algorithm by differentiating the energy function in equation 2.1.

$$\frac{\partial E}{\partial p} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} k_{ij}(\|p_i - p_j\| - l_{ij}) \frac{(p_i - p_j)}{\|p_i - p_j\|} \tag{2.13}$$

If every pair of vertex is connected by an edge, then the above equation can be expressed in the form of equation 2.8 where,

$$\frac{\partial f_\theta}{\partial d} = \frac{\partial g_\theta}{\partial d} = \frac{k_{ij}}{2}(\|p_i - p_j\| - l_{ij}) \tag{2.14}$$

Hence in the Kamada-Kawai algorithm, both the attractive and repulsive forces are the same [22].

A physics interpretation of this is that for a given vertex to be displaced from its position towards another vertex, a force must act on said vertex. Displacing the vertex is known as *work* and it is defined as a change in the energy of a given system. Therefore applying a set of forces to a graph in order to visualize it is akin to optimizing the energy of the graph. The kind of optimization that is performed, whether it is a minimization or maximization, depends on whether the negative or positive gradient respectively is used to update the position of the vertices.

2.3 Dimensionality Reduction

Dimensionality reduction is the process of representing high dimensional data unto a lower dimension such that important patterns within the data are preserved in the lower dimension. There are typically 2 types of patterns that are preserved: global and local patterns [17]. These patterns are typically found without the aid of human labels in what is known as unsupervised learning and as such it makes dimensionality reduction an indispensable tool in information visualization. Examples of dimensionality reduction techniques include Principal Component Analysis (PCA) and Multi-Dimensional Scaling (MDS) [6].

In this section we discuss MDS, a statistical technique that is used to generate semantic patterns for word clouds. We specifically discuss its variants: classical MDS and metric MDS; as well as majorization, a popular optimization technique for MDS.

2.3.1 Multi-Dimensional Scaling (MDS)

Multi-dimensional scaling is a dimensionality reduction technique that attempts to preserve the distance between all pairs of high dimensional points within their lower dimensional representation. In other words, it attempts to match the pairwise lower-dimensional distances with those of the higher-dimension by minimizing the squared discrepancy between both distances. This can be expressed as

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\|q_i - q_j\|_2 - \|p_i - p_j\|_2)^2 \tag{2.15}$$

where $\{q_i, q_j\}$ are a pair of high-dimensional points and $\{p_i, p_j\}$ are their lower-dimensional representation.

Classical Multi-Dimensional Scaling

Under classical MDS, the aforementioned objective is solved analytically by first double-centering the euclidean distance matrix, D^Q, for the higher dimensional points, Q. This gives us a gram matrix

$$K = -\frac{1}{2} H^\top D^Q H = \hat{Q}^\top \hat{Q} \tag{2.16}$$

that allows us to express equation 2.15 in a matrix form as

$$\mathrm{Tr}(\hat{Q}^\top \hat{Q} - \hat{P}^\top \hat{P})^2 = \mathrm{Tr}(V\Lambda V^\top - U\Sigma U^\top)^2 \tag{2.17}$$

At the minimum of the above objective, it is assumed that both the data points Q, and their lower dimensional representation, P, are equal such that $\hat{Q} = \hat{P}$. Since P is a d-dimensional representation of Q, it follows that the eigenvalues, Σ, of $\hat{P}$ and the eigenvectors, U, of $\hat{P}$ are equal to the top d-eigenvalues, Λ_d, of $\hat{Q}$ and the top d-eigenvectors, V_d, of $\hat{Q}$ respectively. Hence we can express the classical MDS solution, $\hat{P}$, as

$$\hat{P} = \Lambda_d^{1/2} V_d^\top \tag{2.18}$$

which is simply the result of performing PCA on the gram matrix, K.

One problem with classical MDS is that it places too much emphasis on larger pairwise distances at the expense of smaller pairwise distances [17]. This means that larger distances are mostly accurate whereas smaller distances tend to be inaccurate. In fact it does not use as much as a third of all small distances [10][19].

Metric Multi-Dimensional Scaling

In an attempt to solve the shortcomings of classical MDS and place equal emphasis on small pairwise distances, metric MDS adds weights to the MDS objective. These weights are typically the inverse of the original pairwise distances so that as much focus is given to matching smaller distances as is given to matching larger distances. Matching smaller distances allows us to capture patterns between higher dimensional points that are close together. The new objective formed is referred to as *Stress*, S, and it is expressed as

$$S = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij}(d_{ij} - \|p_i - p_j\|_2)^2 \tag{2.19}$$

where $d_{ij} = \|q_i - q_j\|_2$ and $w_{ij} = 1/d_{ij}$.

The pairwise distances, d_{ij}, however need not be represented by euclidean distances. They can be represented by any distance function. In order for MDS to capture local patterns within the neighbourhood of a point q_i, the euclidean distance has to be replaced with the geodesic distance. This is because euclidean distance corresponds to a distance

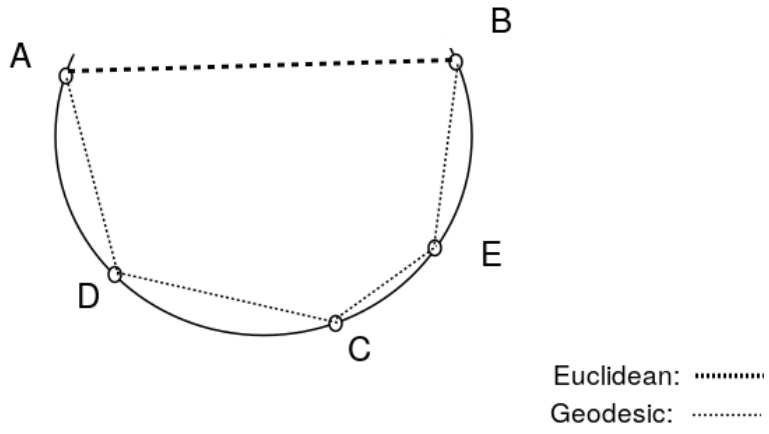


Figure 2.3: The geodesic and euclidean distances between points A and B on a curve

on a straight line and on curved manifolds such as the one illustrated in figure 2.3, this results in smaller pairwise distances for far away points on the manifold.

The geodesic distance on the other hand is the shortest-path distance between any pair of points, q_i and q_j, on a graph. If a manifold is represented as a graph, the geodesic distance can find the appropriate distance between any pair of points on the manifold. This means that on curved manifolds, far away points will always have a larger pairwise distance than neighbouring points. Since the shortest-path distance computation relies on the distance between neighbouring points, MDS is able to capture local patterns when using the geodesic distance [39]. In graph drawing, the geodesic distance is also referred to as the graph-theoretic distance. In fact the Kamada-Kawai algorithm itself is a form of metric MDS where the pairwise euclidean distance is replaced with a scaled geodesic distance and the weight, w_{ij}, is replaced with the inverse squared geodesic distance [16].

Unlike classical MDS where the objective is solved analytically, the objective in metric MDS cannot be solved analytically because the weights perform a non-linear transformation of the points, Q. Instead gradient descent or an iterative optimization process known as majorization is used to optimize the stress objective [17][13].

2.3.2 Scaling by MAjorizing a COmplicated Function (SMACOF)

Smacof is a popular majorization algorithm that is used to optimize the stress objective in equation 2.19. Majorization itself is an optimization technique that is used to solve a complicated function, $f(.)$, by solving a simpler surrogate function, $g(.,.)$. For a given point, y, and a minimizer, x^*, the surrogate function needs to satisfy the following sandwich inequality

$$f(x^*) = g(x^*, x^*) \leq g(x^*, y) \leq g(y, y) = f(y) \tag{2.20}$$

Using the Cauchy-Schwartz inequality, we can create a simple surrogate function for the stress objective. To do this, we begin by rewriting equation 2.19 as

$$S = \sum_{i<j}^{n} w_{ij} d_{ij}^2 + \sum_{i<j}^{n} w_{ij} \rho_{ij}^2 - 2 \sum_{i<j}^{n} w_{ij} d_{ij} \rho_{ij} \tag{2.21}$$

where $\rho_{ij} = \|p_i - p_j\|_2$. Using the definition of,

$$\|p\| = \frac{p^\top p}{\|p\|} \tag{2.22}$$

we further express equation 2.21 in matrix form such that

$$f(p) = \frac{n(n-1)}{2} + \text{tr } P^\top V P - 2 \text{ tr } P^\top B(P) P \tag{2.23}$$

where, P, are the coordinates of the points in the lower dimension and the matrices V and $B(P)$ are weighted laplacian such that,

$$V_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{i \neq j} w_{ij} & \text{if } i = j \end{cases} \quad B(P)_{ij} = \begin{cases} -w_{ij} d_{ij} / \rho_{ij} & \text{if } \rho_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

The last term in equation 2.23 is the what is most important for constructing the surrogate function. Using the Cauchy-Schwartz inequality,

$$\|x\| \geq \frac{x^\top p}{\|p\|} \tag{2.25}$$

for some vectors, p and x [16]. We can construct a surrogate function, $g(x, p)$, that is expressed as

$$g(x, p) = \frac{n(n-1)}{2} + \text{tr } X^\top V X - 2 \text{ tr } X^\top B(P) P \tag{2.26}$$

From equation 2.25, the following inequality holds

$$X^\top B(X) X \geq X^\top B(P) P \tag{2.27}$$

Hence for a minimizer, x, $f(x) \leq g(x, p)$ and the sandwich inequality in equation 2.20 holds [14]. We can find this minimizer by setting the derivative of the $g(.,.)$ with respect to X as zero and then solving for X. This results in the solution

$$X = V^+ B(P) P \tag{2.28}$$

where V^+ is the psuedo-inverse of V.

This solution is also known as the Guttman transform of the points, P [14]. On each iteration of the majorization algorithm, the points P are replaced with X and the above transformation is repeated until $f(x) - f(p)$ is less than some tolerance value, ϵ.

2.4 Word Clouds

A word cloud is a visual presentation of a set of words whereby their font size represents some weighting like their frequency within a given text corpus. Word clouds became popular as a form of artistic visualization when social sites like flickr and del.icio.us started associating web resources like photographs and web articles with various keyword metadata. These words summarized the resource and allowed users to navigate other resources on the sites [36][35]. In recent times word cloud provide a visual statistical summary of a text corpus and are generated using specialized algorithms and well known software like Wordle [20].

There are several types of word cloud generation algorithms. They include random and semantic word cloud algorithms [2]. Random word cloud algorithms generate their layout without any emphasis on the semantic relationship between the words. Semantic word cloud algorithms however generate their layout based on the semantic relationship between the words. Having such structure along with the frequency of the words improves the statistical summary provided by the word cloud.

In this section we briefly discuss well known random algorithms like Wordle and well known semantic algorithms like the Context Preserving Word Cloud (CPWC) and seam carving algorithm [11][45].

2.4.1 Random Word Clouds

Wordle is a popular algorithm that generates random word clouds using an archimedean spiral. After extracting the relevant words that summarize a text, each word is successively placed at a random position on the canvas. During placement, if a word collides with any other word on the canvas, it is moved along an ever increasing spiral until it no longer collides with another word or until it is no longer on the canvas [37]. For any given point, i, on the spiral, the radius, r_i, from the center of the spiral to that point can be described by the following equation

$$r_i = a_0 + b\theta_i \tag{2.29}$$

where a_0 represents the initial radius of the spiral, b represents the distance between successive turns, and θ_i an angle within the spiral [42]. The distance, b, between successive turns can be estimated with a final radius, a_1, as

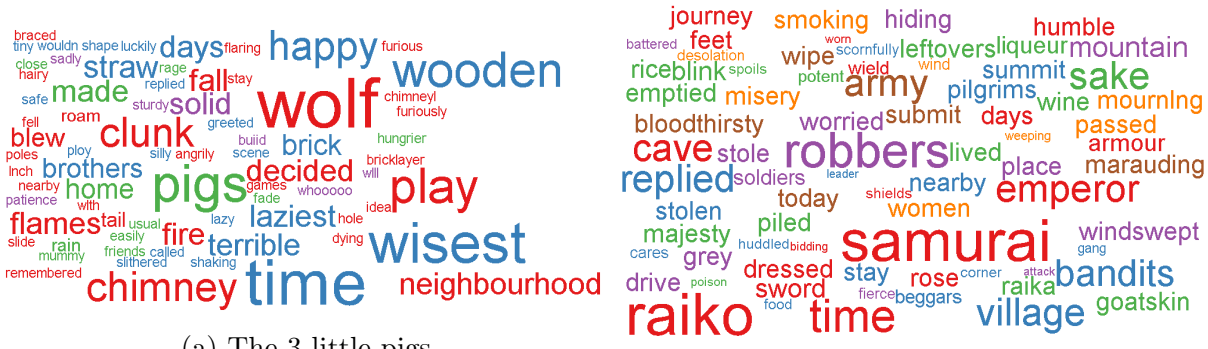
$$b = \frac{a_1 - a_0}{2\pi n} \tag{2.30}$$

where n represent the desired number of turns and $2\pi n$ represents the maximum angle of the spiral. The equation 2.29 can be translated to cartesian coordinates as follows

$$\begin{aligned} x_i &= (a + b\theta_i)cos(\theta_i) \\ y_i &= (a + b\theta_i)sin(\theta_i) \end{aligned} \tag{2.31}$$

To detect collision between words, Wordle uses a combination of techniques such as hierarchical bounding boxes, spatial indexing trees, and caches. Caching is the fastest way

to detect collisions but it does so based on previously collided words and not newly collided ones. Other techniques like spatial indexing trees are more effective but slower [37].



(a) The 3 little pigs

(b) Seven samurai

Figure 2.4: Wordle layout for different texts

2.4.2 Semantic Word Clouds

In order to generate semantic word clouds, most algorithms first utilize some kind of dimensionality reduction to capture the semantic relationship between high level representations for a set of words. Such relationship exists when the text in a corpus can be faithfully represented in some form such as vector embeddings. Most algorithms use classical MDS for dimensionality reduction but other algorithms like t-SNE can be used as well [2][33].

Classical MDS is used to generate an initial semantic layout for most algorithms. However the 2D positions that are generated by MDS does not take into account the size and geometry of the words. As a result, when the words are initially placed, they are not compact and in most cases may overlap. The crux of semantic word cloud algorithms is in how they refine this initial layout. Various algorithms take different approaches to refining the layout of the cloud while preserving its semantic order.

In this section, we will briefly discuss how this is done using the Context Preserving layout, which is a type of force-directed algorithm, and how it is also done using the seam carving algorithm, which is a type of energy minimization algorithm.

Context Preserving Layout

The context preserving algorithm is a semantic word cloud visualization algorithm that uses a set of forces to generate a word cloud. These forces are: attractive, repulsive and planar forces. The algorithm is a force-directed algorithm similar to the Fruchterman-Reingold algorithm.

After an initial layout is generated using MDS, a Delaunay graph is created from the 2D points in order to maintain their semantic relationship during the layout refinement. To refine a layout, the three forces mentioned above are applied to each word represented in the Delaunay graph for a given number of iterations. The repulsive force, f_r, is applied to all pairs of overlapping words within the cloud. It can be expressed as follows

$$f_r(v_i, v_j) = \begin{cases} k_r \min(\Delta x, \Delta y) & \text{if } v_i \cap v_j \\ 0 & \text{otherwise} \end{cases} \tag{2.32}$$

where v_i and v_j represent a pair of words, k_r is a constant and $\Delta x, \Delta y$ are the width and length of the overlapping region between the words. Similarly the attractive force, f_a, is applied to all non-overlapping pairs of words that have adjacent nodes in the graph. It is formulated as

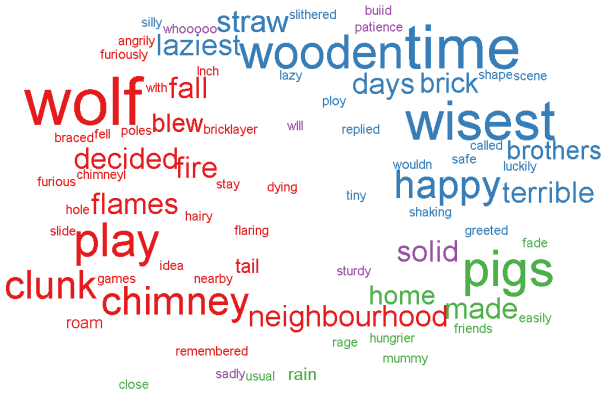
$$f_a(v_i, v_j) = w_i w_j \Delta l \tag{2.33}$$

where Δl is the distance between the pair of words, v_i and v_j; w_i and w_j are their importance respectively. The planar force, f_p, does not exist between a pair of words. Rather it is applied to individual words whose nodes cross an edge in the Delaunay graph. The planar force moves the word over the crossed edge in order to maintain the planarity of the graph [11]. It can be expressed as

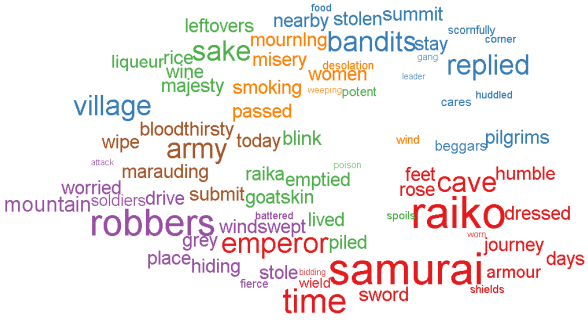
$$f_p(v_i) = \begin{cases} k_p \Delta d & \text{if } v_i \text{ is flipped} \\ 0 & \text{otherwise} \end{cases} \tag{2.34}$$

where k_p is another constant and Δd is the distance of the word from the crossed edge. Similar to the Fruchterman-Reingold algorithm, the context preserving algorithm also utilizes a temperature, t, in order to anneal the effects of the various forces and allow the words to settle into stable configuration.

A major drawback of this method is the use of the Delaunay graph. By flipping the position of the words in order to create each triangle within the graph, some semantic



(a) The 3 little pigs



(b) Seven samurai

Figure 2.5: Context preserving layout for different texts

order is lost. This in turn results in vastly different word clouds that get generated with the insertion or removal of new words.

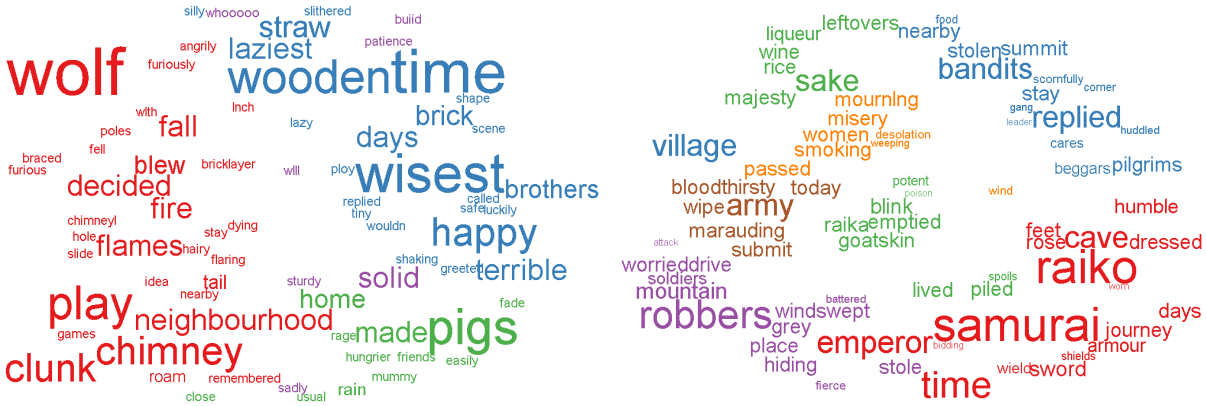
Seam Carving Layout

As the name suggest, this algorithm is inspired by seam carving which is used to resize an image while maintaining the important parts of the image [45]. Seam carving itself is done by removing or adding connected rows or columns of pixels that have very little importance within the image [1]. For word cloud generation, such pixels correspond to those of the empty spaces within the initial layout.

As with the context preserving algorithm, classical MDS is used to generate an initial layout and a repulsive force-directed algorithm is then applied in order to separate overlapping words. After separating the words, the seam carving algorithm strips the empty spaces between them to generate a compact word cloud.

In order to carve out strips of pixels, each pixel, p_i, is given an energy within the word cloud. The energy, $E(p_i)$, for a pixel is computed from the weighted sum of several gaussians. These gaussians are centered at their corresponding words, w_k, and measure how far a given pixel is from the center, μ_k, of a word. The energy can be expressed as follows

$$E(p_i) = \sum_{k=1}^{n} w_k \frac{1}{2\pi\sigma} e^{-\frac{1}{2\sigma^2}\|p_i - \mu_k\|^2} \tag{2.35}$$



(a) The 3 little pigs

(b) Seven samurai

Figure 2.6: Seam carving layout for different texts

where $w_k \in [0, 1]$ and $\sigma = 1$.

Using dynamic programming, the connected set of pixels with the lowest energy within the image is computed. These connected pixels which run from one end of the image to another and are referred to as *seams*, are then removed from the image. The energy of the remaining pixels within the image is recomputed and the seam carving process is repeated until a seam cannot be carved without removing pixels that belong to a word. This process itself is an energy optimization process as with each removal of a seam, the overall energy within the image reduces and only seams with high energy are left within the image [45].

Unfortunately in many cases, there are no connected paths between words that run through one end of the canvas to another. In such cases, the algorithm leaves empty regions between the words and a sub-optimal word cloud is generated.

2.5 Object Clouds

Although object clouds and word clouds are quite similar, there are a few major differences between them. These include their high-level representation and 2D geometry. These differences, although seemingly minor, have a major effect on how both object and word clouds are visualized. In this section, we briefly discuss how 3D models for object clouds are represented and how such representations are used to create random object clouds. In later chapters, we will discuss how such representation affects the generation of semantic word and object clouds.

2.5.1 3D Object Representation

In order to capture the semantic order of various 3D objects within an object cloud, the objects must be represented in some high dimensional form.

To accurately represent such objects, we use a form of vector embedding. Proper vector representation of 3D objects is a challenging task and is actively being researched [9][38]. For object clouds, we represent the 3D object as Multi-View Convolutional (MVC) feature vectors [8]. This kind of vector embedding is generated from a special Convolutional neural network (CNN) architecture and have been very effective for 3D object retrieval tasks.

Multi-view Convolutional Features

Multi-view convolutional (MVC) feature vectors are global shape descriptors that are derived from a pool of parallel convolutional layers. Global shape descriptors are a numerical representations for an entire object. Along with local shape descriptors, which are representations of local points on an object, global shape descriptors form a vector space with which we can represent and compare various 3D objects [8].

Convolutional layers are a type of neural layers that contain multiple shared 2D weights [26]. Each of these shared weights is a 2D kernel that is applied across an image to detect a particular shape or pattern. The weight sharing is what allows the layer to apply a small 2D kernel across the entire image in a single pass. If a pattern for classifying an object is present within an image, there is an image response for that pattern. Otherwise the response is a blank image. The response image for each shared weight is then combined into a single image and passed as the output to the convolutional layer. A hierarchy of convolutional layers and fully-connected layers are what make up the architecture of a convolutional network. Each convolutional layer takes in the output of the bottom convolutional layer and tries to detect patterns within the input. This leads to more complex patterns being detected by higher convolutional layers while lower convolutional layers detect simple shapes and patterns. The output image of the final convolutional layer is flattened into a vector and passed into the fully-connected layers where it is used for classification. This is illustrated in figure 2.7.

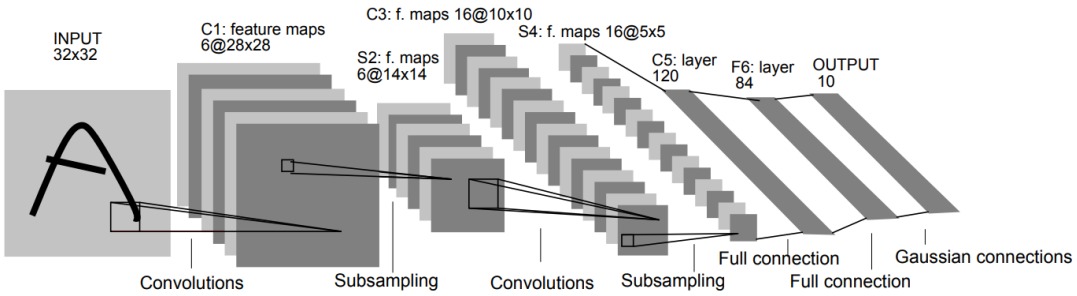


Figure 2.7: Architecture of a convolutional neural network (LeCun et al, 1998) © 1998 IEEE

In order to represent a 3D object, the MVC neural network takes multiple views of a 3D object from 12 cameras that orbit the object at an angle and ground elevation of 30°. This is illustrated in figure 2.8. The image captured by each view is then passed through its own hierarchy of convolutional layers. The convolutional layers for the views are arranged in a series. The output image of each series is pooled into a single image and passed into a convolutional neural network.

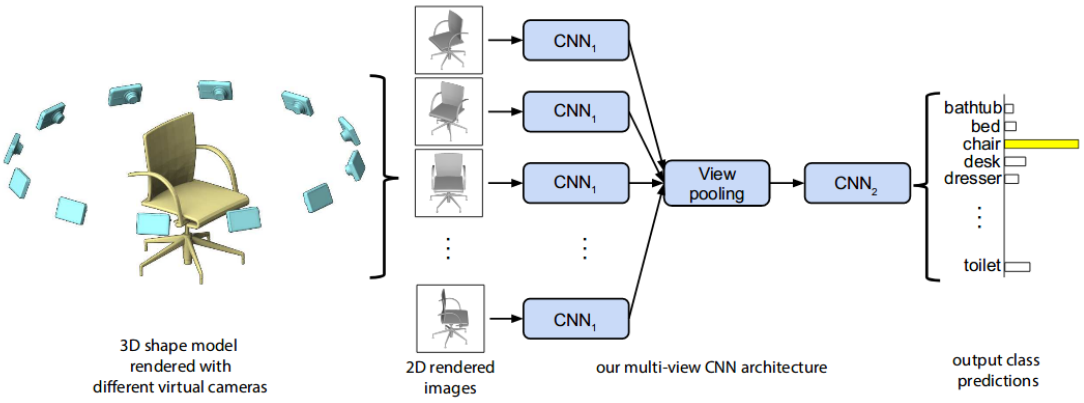


Figure 2.8: Architecture of a multi-view convolutional neural network (Su et al, 2015) © 2015 IEEE

After training the MVC neural network, the output vector from the penultimate layer is used as a shape descriptor. This output vector is rich in classification features because it is what the final layer of the MVC neural network uses for classification. It is therefore used as a vector representation for a given 3D object.

2.5.2 Pseudo-random Object Clouds

Pseudo-random object cloud algorithms are algorithms that generate a seemingly random layout. In most cases the layout is generated from a breadth-first search of discrete positions within the layout [21]. These positions correspond to cells in a row-column grid that overlays the canvas for the object cloud.

Figure 2.9: An object and 3 other objects moving along a spiral from the object of interest (Hong and Brooks, 2016) © 2016 IEEE

In order to generate a cloud, the vector embedding for each object is first generated and its distance from that of the object of interest is computed. Next the 2D layout for the cloud is divided into a grid and the object of interest is placed at the center of the grid. Every other object is then sorted according to their vector distance from the object of interest and placed at the next available square in the grid. The next available square is found by performing a breadth-first search of all the discrete positions with the center square as the root. If an object is placed in a square but it collides with an already placed object, it is rotated along an archimedean spiral until it no longer collides with any object [21]. The new grid position for the object is marked as occupied and the process is repeated until all the objects are placed within the layout.

As discussed in the previous chapter, emphasizing just the pairwise relationship between all the objects within the cloud and the object of interest does not create a proper semantic order within the cloud. In order to generate a semantic order, the pairwise relationship amongst all the objects in the cloud needs to be computed.

Chapter 3

Semantic Object Clouds

3.1 Energy-based Clouds

As discussed in the previous chapter, using dimensionality reduction methods such as MDS to visualize object or word clouds is restricted to projecting high dimensional points unto a 2D canvas without accounting for the geometry and size of the items that said points represent [31]. The Context Preserving algorithm for example uses classical MDS to project the high dimensional points onto the cloud but then has to adjust the position of the points afterwards in order to place the words and create a compact cloud.

We can modify the MDS algorithm so that performing dimensionality reduction simultaneously determines the proper position of various items within the cloud. To do this, we re-express the metric MDS objective for dimensionality reduction as a graph-drawing energy function. As mentioned in the previous chapter, when such functions are minimized, we obtain a resulting force-based graph drawing algorithm.

In this chapter we discuss the construction of this energy function, as well as how it serves as a qualitative measure for an object or word cloud. We also discuss how gradient descent and majorization can be used to optimize it, as well as provide an algorithmic implementation for generating such a cloud on an AR device.

3.1.1 Energy Formulation

In order to express the energy in equation 2.19 as a force-based graph-drawing algorithm, we change the weighting, w_{ij}, into a force-based weighting and replace the original pairwise

distances, d_{ij}, with the sum of the radii for a pair of objects. In order to replace the weight, w_{ij}, in equation 2.19 we decompose it into

$$w_{ij} = (1 - \delta_{ij})w_{ij} + \delta_{ij}w_{ij} = (1 - \delta_{ij})\beta_{ij} + \delta_{ij}\alpha_{ij} \tag{3.1}$$

where β_{ij} is an attractive weighting, α_{ij} is a repulsive weighting and δ_{ij} is an indicator function. We also express the pairwise distances, d_{ij}, in equation 2.19 as

$$d_{ij} = r_i + r_j \tag{3.2}$$

where r_i and r_j are the radii for a pair of objects. The new pairwise distance indicates that we want all the objects to be adjacent each other. Normally this will cause all the objects within the cloud to collapse on each other. However the force-based weighting will prevent this from happening and instead result in a tight packing of the objects within the cloud. From the above, the energy in equation 2.19 becomes

$$E = \sum_{i<j}^{n}((1 - \delta_{ij})\beta_{ij} + \delta_{ij}\alpha_{ij})(d_{ij} - \|p_i - p_j\|_2)^2 \tag{3.3}$$

where the indicator function, δ_{ij}, evaluates to

$$\delta_{ij} = \begin{cases} 1 & \text{if } \|p_i - p_j\| < d_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

The first weighting, β_{ij}, can be thought of as an attractive weighting between points, p_i and p_j while the second weighting, α_{ij}, is the repulsive weighting between the pair of points. Since p_i and p_j represent the center of a pair of objects and d_{ij} represents the distance at which the pair of objects become adjacent without overlapping, the weightings control how fast or slow different pairs of objects become adjacent to one another. When $\|p_i - p_j\| > d_{ij}$, the objects represented by p_i and p_j are far apart and either is attracted to the other at a speed of β_{ij} until $\|p_i - p_j\| = d_{ij}$. Conversely when $\|p_i - p_j\| < d_{ij}$, the objects represented by p_i and p_j are too close and either is repelled away at a speed of α_{ij} until $\|p_i - p_j\| = d_{ij}$. When $\|p_i - p_j\| = d_{ij}$, the weightings have no effect and there is no change in the position of either p_i or p_j.

As mentioned in the previous chapter, under the typical MDS objective both weightings are equal and cancel themselves. In such a scenario the pairwise distances, d_{ij}, must

reflect actual distances between all the objects or the objects will collapse unto themselves. However when $\alpha > \beta$, a given layout has a lot of space with a few overlapping objects and when $\alpha < \beta$, the layout is more compact but with a lot of occluding objects.

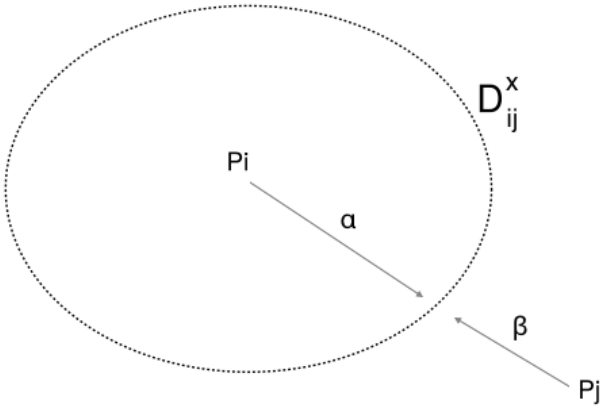


Figure 3.1: Direction of α and β on a given point p_j

After decomposing the weights, we construct a graph over the high dimensional points. This graph allows us to establish a pairwise ordering of the objects which is lost after replacing the original pairwise distances, d_{ij}. In the next section we will discuss what kind of graph is needed to create either a semantic or random object cloud. Using the high dimensional graph, we split the objective into adjacent pairs of points, N_1, and non-adjacent pairs of points, N_2, such that we have

$$E = \sum_{i<j}^{N_1}((1-\delta_{ij})\beta_1+\delta_{ij}\alpha_1)(d_{ij}-\|p_i-p_j\|_2)^2+\sum_{k<l}^{N_2}((1-\delta_{kl})\beta_2+\delta_{kl}\alpha_2)(d_{kl}-\|p_k-p_l\|_2)^2 \quad (3.5)$$

By setting attraction, $\beta_2 = 0$ for the non-adjacent pairs of objects and making the repulsion $\alpha_1 = \alpha_2$ for both adjacent and non-adjacent pairs of objects, we end up with the an energy function that when differentiated, results in a force-based graph-drawing algorithm. The function can succinctly be expressed as

$$E = \sum_{i<j}^{N}(A_{ij}(1-\delta_{ij})\beta + \delta_{ij}\alpha)(d_{ij} - \|p_i - p_j\|_2)^2 \quad (3.6)$$

where A_{ij} is another indicator function that represents the adjacency of any pair of points. In order to create non-overlapping object clouds, especially as the number of objects increases, we find it important to set $\beta < \alpha$. When the number of objects is relatively small, $\beta = \alpha$ produces non-occluding objects but as the number of objects increases, so does the number of occlusions. Setting $\beta > \alpha$ on the other hand results in occluding words regardless of the number of objects.

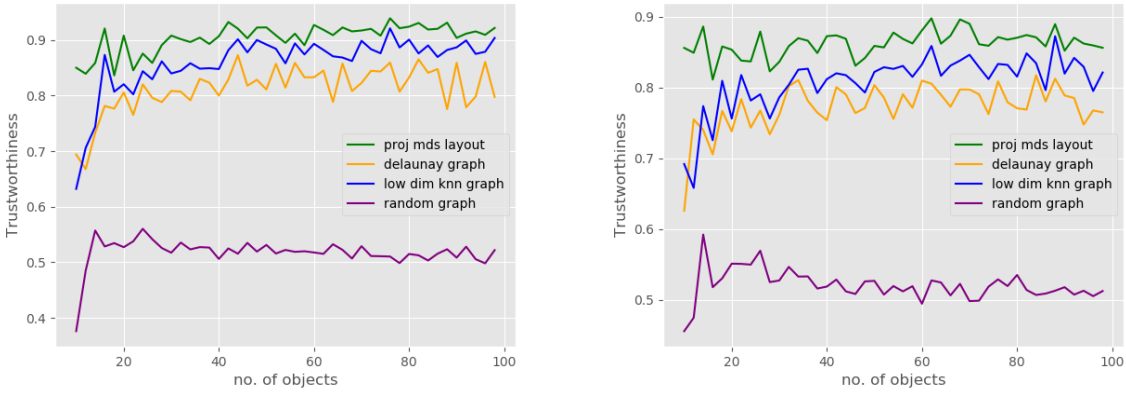
3.1.2 Graphical Structure

In order to create a semantic object cloud, we use a K-Nearest Neighbour (K-NN) graph. For any vector, v_i, in a given space, the K-NN graph finds the k closest vectors, $\{w_1, ..., w_k\}$ and constructs an edge between v_i and each of the vectors. This graph is however directed and unconnected whereas the graph for an object or a word cloud needs to be undirected and connected in order to exert the proper forces amongst the nodes and compact them all. An unconnected graph will have items that are not attracted by any other items nor repelled by all the other items, thus creating excess space within the cloud. To convert a graph into an undirected and connected graph, we simply sum the adjacency matrix of the graph with its transpose and take the non-zero entries as the edges.

To determine the appropriate number of neighbours for the K-NN graph, we choose the smallest number of neighbours, k, that is necessary to form a connected graph. We find that the smaller the number of neighbours, the easier it is to compactly place an object with all its neighbours, thereby reducing the energy for the object. If the number of neighbours is high, an object cannot be placed with all its neighbours because there is a limit to the number of non-overlapping objects that can be placed around a given object on a 2D canvas. Hence the energy of the object in question will increase.

The K-NN graph is typically used in many dimensionality reduction algorithms because it preserves the local ordering amongst high dimensional points [39][4]. To that end, we find that it helps ensure a semantically accurate ordering amongst high dimensional points. However we find that the Delaunay graph which is typically used to construct word clouds does not ensure a semantically accurate ordering due to the flipping of points that may be required when constructing a simplex for the triangulation. Furthermore, the Delaunay graph cannot be used for high dimensional points because in order to construct such a graph, the number of points needs to exceed their dimensionality. However, for both word and object clouds, the dimensionality of the vector embeddings can sometimes exceed the number of items to be visualized.

In figure 3.2, we illustrate the advantage the KNN-graph has over the Delaunay graph



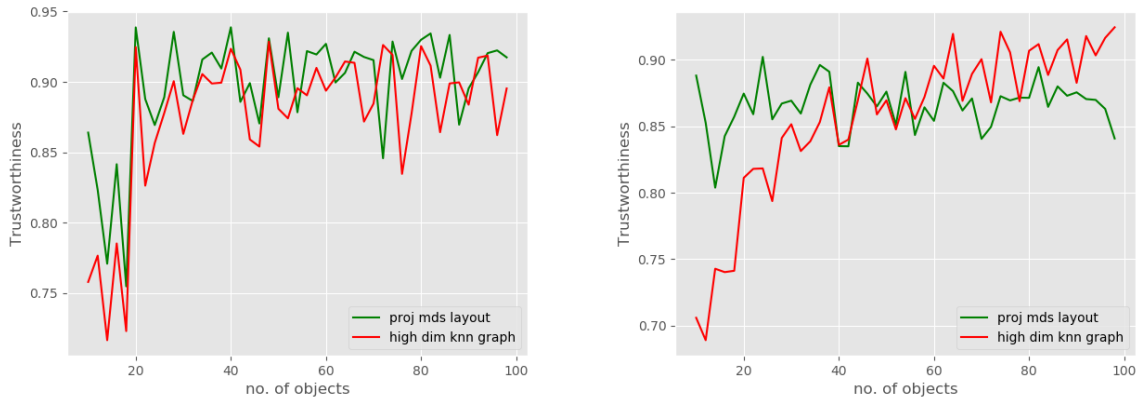
(a) ModelNet 40

(b) Princeton shape benchmark

Figure 3.2: Trustworthiness of 2D graphs and MDS on different datasets with different number of objects.

for both semantic accuracy and dimensionality reduction. To measure the semantic accuracy, we use the *trustworthiness* metric [18][41]. This measures the amount of high dimensional neighbouring points that are present in the neighbourhood of each 2D point. Neighbouring points from the high dimension that are missing from the neighbourhood of a 2D point reduces the trustworthiness of a projection. Conversely, neighbouring points from the high dimension that are present in the neighbourhood of a 2D point increases the trustworthiness. We compare the trustworthiness of a 2-dimensional Delaunay, K-NN and random graphs that are formed from an initial MDS projected layout. We also include the trustworthiness of the MDS layout as a baseline.

From figure 3.2, we can see that the classical MDS projected ordering has a high trustworthiness which indicates that the initial layout is quite similar to the high dimensional ordering of the points. However by applying both the K-NN, Delaunay and random graphs to draw the layout, the trustworthiness decreases. This is due to the fact that some information about the actual closeness of neighbouring points is lost during the projection and graphs like the Delaunay graph change some semantically correct edges during its construction. In figure 3.3 we show how applying a KNN-graph to the high dimensional points themselves simply circumvents these problems and increases the trustworthiness of an initial layout.

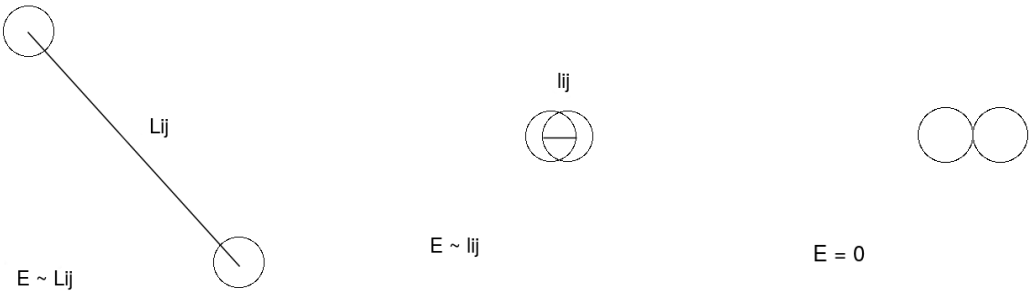


(a) ModelNet 40 (b) Princeton shape benchmark

Figure 3.3: Trustworthiness of MDS and a high-dimensional KNN on different datasets with different number of objects.

3.1.3 Measuring Object Clouds

Formulating an objective function for object clouds allows us to measure the quality of a cloud. By defining a semantic or random graphical ordering over the set of objects, we can measure how well said objects are compactly placed relative to each other within a layout. We will briefly illustrate how the energy of a layout quantifies its aesthetic appeal, as well as what to note when using such qualitative measure.



(a) Distant objects (b) Overlapping objects (c) Compact objects

Figure 3.4: Distance in relation to the energy for a pair of objects

Consider the pair of objects in the figure 3.4a. The farther apart their distance, L_{ij}, the larger the amount of energy, E, within the layout. As their distance approaches infinity, so does the energy. Conversely suppose we have an overlapping pair of objects as in the figure 3.4b. If we assume the energy of this layout is minimal because the distance, $l_{ij} < L_{ij}$, then the objective function will tend towards this configuration for a pair of objects. However since this configuration is as equally undesirable as the previous configuration, we scale l_{ij} - using L_{ij}/l_{ij} - so that the value of the energy is as large as that of the previous configuration. Note that compared to an overlapping configuration, we do not need to scale a distant configuration. That being said, we are left with 2 configurations in which the distance, and subsequently the energy, E, is zero: when the objects are compactly placed side by side and when they completely overlap. The configuration in which the objects completely overlap is a local minimum that can only be escaped by adding noise to their positions. If however we ignore this case, then we see that the configuration with the most aesthetic appeal - compact placement - has no energy.

In order to determine the proximity of a pair of objects, L_{ij} or l_{ij}, the energy in equation 3.6 calculates the position of their centers, $\|p_i - p_j\|$, relative to the size of the objects, $d_{ij} = r_i + r_j$ such that $L_{ij} \triangleq l_{ij} = d_{ij} - \|p_i - p_j\|$. When $\|p_i - p_j\| < d_{ij}$, the pair of objects are overlapping and a force $\alpha > \beta$ is applied to the overlapping distance, l_{ij} in order to compute the energy value. The indicator function in equation 3.4 determines which of the forces to apply and subsequently which distance, L_{ij} or l_{ij} is in effect. We can set the weight β to any positive real value but in order to scale l_{ij} and prevent the energy function from settling into an overlapping configuration, $\alpha > \beta$.

Since computing the energy involves the squared distance between pairs of objects and the weightings are positive real numbers, for an object cloud the energy is always a positive real number. Ideally for every cloud, we would want this number to be as close to 0 as possible without pairs of objects completely overlapping but this is not always possible as the number of objects increases. This is because as the number of objects increases, so does the number of configurations. Many of these configurations are sub-optimal and the objective function may not be able to escape one of such local optima. Hence as the number of objects increases, we may achieve a state of minimal energy rather than a state with no energy.

3.1.4 Extension to Word Clouds

For word clouds specifically, we find it important to start with $\beta = \alpha$ and then slowly decay the magnitude of β. We use $\beta = \gamma^2/K$ where K is some constant and γ is slowly

decayed. Decaying β allows semantically similar words to quickly pair up before being separated by the repulsive weighting. Since such repulsive effect does not appear during the construction of object clouds, we believe it is due to the non-uniform geometry of the words. Due to the non-uniform geometry, we cannot set the pairwise distance between a pair of words to the sum of their radii. Instead we use the sum of the distance from the center of each word to its edge - in the direction of both pairs of words - as illustrated in figure 3.5

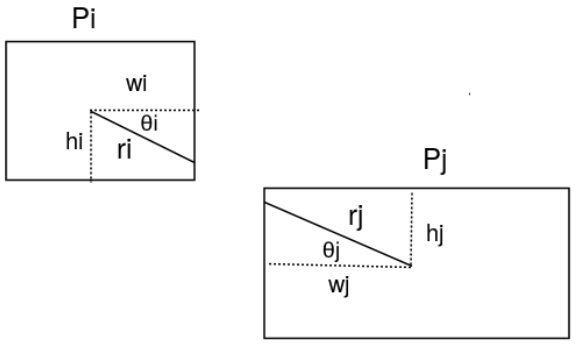


Figure 3.5: Distance from edge to center for a pair of words

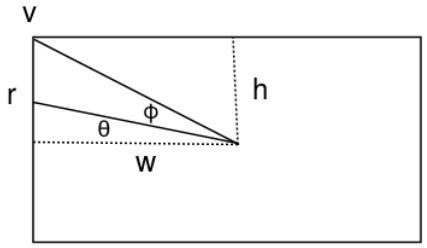


Figure 3.6: Vertex and edge distances from the center of a word

To calculate the distance, r, we use the half-width, w; the half-height, h; and the angle, θ, that the centers $(p_i - p_j)$ make with the horizontal axis. When it is less than the angle, ϕ, made between the vertex, v, and w, r can be calculated as $w/\cos(\theta)$. However when it is greater than ϕ, it can be calculated as $h/\sin(\theta)$. Rather than calculating ϕ we can observe that when $\theta < \phi$, $w/\cos(\theta) < h/\sin(\theta)$ and vice versa. Therefore the distance, r_i, from the center of a word to its edge can be expressed as

$$r_i = \min(\frac{w_i}{\cos(\theta_{ij})}, \frac{h_i}{\sin(\theta_{ij})}) \tag{3.7}$$

Hence with the above radius, we can apply equations 3.4 and 3.6 to the generation of word clouds.

3.2 Energy Minimization

In order to minimize the energy function of an object cloud, we utilize a few optimization methods that have been applied to multi-dimensional scaling and machine learning problems in general.

3.2.1 Gradient Descent

As mentioned earlier, the algorithm represented by equation 3.6 can be converted into a force-based algorithm by taking the negative gradient of the energy, E. Taking the gradient with respect to p_i, we have

$$\frac{\partial S}{\partial p_i} = \sum_j (A_{ij}(1-\delta_{ij})\beta + \delta_{ij}\alpha)(\|p_i - p_j\|_2 - d_{ij})\frac{(p_i - p_j)}{\|p_i - p_j\|_2} \tag{3.8}$$

where $(p_i - p_j)/\|p_i - p_j\|_2$ indicates the direction in which the center, p_i, of an object should move and $(d_{ij} - \|p_i - p_j\|_2)$ indicates by how much the center should be moved for an object in question to be placed compactly amongst its neighbours. Using the gradient, we can update the position of p_i as

$$p_i^{(t+1)} = p_i^{(t)} - \eta\frac{\partial S}{\partial p_i} \tag{3.9}$$

As discussed in the previous chapter, this is the update algorithm 2.7 for force-based graph algorithms. However the problem with this update algorithm is that it converges slowly relative to second-order optimization methods like the Newton-Raphson method and it has a higher likelihood of becoming stuck at a local minimum [23]. Applying the Newton-Raphson method as is done in the Kamada-Kawai algorithm requires the calculation of a Hessian which can be a tedious approach and in some cases, a semi-positive definite

Hessian may not exist. In order to avoid this while guaranteeing faster convergence to a global minimum, we further adopt two optimization strategies to minimize the energy function: random reshuffling and majorization.

Algorithm 1: Gradient Descent

Data: Adjacency matrix, A; Pairwise distance matrix, D^x; α, β and η
Result: A configuration of points, P
Initialize points, P, randomly from $\mathcal{N}(0, 1)$;
for $n = 1$ ***to*** *iterations* **do**
  for $(p_i, p_j) \in P$ **do**
    $u_{ij} \leftarrow p_i - p_j$;
    $\delta_{ij} \leftarrow \|u_{ij}\| < d_{ij}$;
    $\frac{\partial S}{\partial p_i} \leftarrow (A_{ij}(1 - \delta_{ij})\beta + \delta_{ij}\alpha)(\|u_{ij}\| - d_{ij})u_{ij}/\|u_{ij}\|$;
    $p_i \leftarrow p_i - \eta\frac{\partial S}{\partial p_i}$;
  end
end

3.2.2 Random Reshuffling

Random reshuffling is a form of gradient descent in which the training set is shuffled at each iteration before taking the gradient. In our case, the training set, P, consists of pairs of objects whose initial order is permutated at each update iteration. We can express this as

$$p_i^{(t+1)} = p_i^{(t)} - \eta\frac{\partial S_{\sigma(P,t)}}{\partial p_i} \tag{3.10}$$

where $\sigma(P, t)$ represents the permutation of P at iteration t. While the convergence of the random reshuffle is chaotic in nature, its convergence rate is greater than that of the normal gradient descent and can be even close to quadratic in some cases [7]. Additionally, shuffling the order of the training set can allow us to escape local optimas within the energy function.

Algorithm 2: Random Reshuffle

Data: Adjacency matrix, A; Pairwise distance matrix, D^x; α, β and η
Result: A configuration of points, P
Initialize points, P, randomly from $\mathcal{N}(0, 1)$;
for $n = 1$ ***to*** *iterations* **do**
  Permutate P;
  for $(p_i, p_j) \in P$ **do**
    $u_{ij} \leftarrow p_i - p_j$;
    $\delta_{ij} \leftarrow \|u_{ij}\| < D_{ij}$;
    $\frac{\partial S}{\partial p_i} \leftarrow (A_{ij}(1 - \delta_{ij})\beta + \delta_{ij}\alpha)(\|u_{ij}\| - D_{ij})u_{ij}/(\|u_{ij}\| + \epsilon)$;
    $p_i \leftarrow p_i - \eta\frac{\partial S}{\partial p_i}$;
  end
end

3.2.3 Majorization

Given that the energy for object clouds can be expressed is a form of metric MDS, majorization can be used to minimize it. A major advantage of majorization is that it guarantees a set of non-increasing energy values that allows us to stop the minimization process when there is little to no change in the energy values. In other words

$$f(x^t) - f(x^{t+1}) \leq \epsilon \tag{3.11}$$

At which point we can be certain that the energy of the object or word cloud is minimal and has the highest aesthetic value before ending the minimization process. This is particularly useful because most graph drawing algorithms such as the Fructerman-Reingold and the context preserving algorithm as well as the techniques mentioned above minimize the energy in a cloud within a set number of iterations. This is may result in a sub-optimal configuration of objects within the cloud if the number of iterations is inadequate or if the number of iterations is too long that the energy values begin fluctuating at the valley of the energy function.

Algorithm 3: Majorization

Data: Adjacency matrix, A, Pairwise distances, D^x, α, β, η
Result: A configuration of points, P
Initialize P randomly from $\mathcal{N}(0, 1)$;
$D^p = [d_{ij}]$, where $d_{ij} = \|p_i - p_j\|$;
$W \leftarrow A \circ (1 - (D^p < D^x))\beta + (D^p < D^x)\alpha$;
$f_0 \leftarrow \sum_{i<j} w_{ij}(D^x_{ij} - d_{ij})^2$;
$\Delta \leftarrow \infty$;
while $\Delta > \epsilon$ **do**
  $V \leftarrow (\sum_j w_{ij} \circ \mathbb{I}) - W$;
  $S \leftarrow D^x / D^p$;
  $B \leftarrow (\sum_j w_{ij} s_{ij} \circ \mathbb{I}) - (W \circ S)$;
  $V^+ \leftarrow (V + n^{-1}\mathbf{1}\mathbf{1}^T)^{-1} - n^{-1}\mathbf{1}\mathbf{1}^T$;
  $P \leftarrow V^+ BP$;
  $D^p = [d_{ij}]$;
  $W \leftarrow A \circ (1 - (D^p < D^x))\beta + (D^p < D^x)\alpha$;
  $f_1 \leftarrow \sum_{i<j} w_{ij}(D^x_{ij} - d_{ij})^2$;
  $\Delta \leftarrow f_0 - f_1$;
  $f_0 \leftarrow f_1$
end

As mentioned in the previous chapter, in order to majorize a function, $f(.)$, we need to construct and minimize a surrogate function, $g(.)$, such that the sandwich inequality in equation 2.20 holds and the minimum of $g(.)$ is the same as that of $f(.)$. We simply adopt the surrogate function that is used in equation 2.26 with the slight modification that the weight, w_{ij}, is no longer constant and can be expressed as

$$w_{ij}(p) = A_{ij}(1 - \delta_{ij}(p))\beta + \delta_{ij}(p)\alpha \tag{3.12}$$

where $w_{ij}(p)$ are the weights for a particular configuration P such that $\delta_{ij}(p)$ is the activation function for a set of positions p_i and p_j that belong to said configuration. Hence the surrogate $g(x, p)$ can then be expressed as

$$g(x, p) = \sum_{i<j}^{n} w_{ij}(p)d^2_{ij} + \text{tr } X^\top V(W_p)X - 2 \text{ tr } X^\top B(P, W_P)P \tag{3.13}$$

where $V(W_p)$ and $B(P, W_p)$ are the Laplacian matrices computed from the weights $w_{ij}(p)$. Similarly the energy function at a minimizer, x, can be expressed as

$$f(x) = g(x, x) = \sum_{i<j}^{n} w_{ij}(x) d_{ij}^2 + \text{tr } X^\top V(W_x) X - 2 \text{ tr } X^\top B(X, W_x) X \tag{3.14}$$

As mentioned in the previous section, when $x = p$, $A_{ij} = \mathbb{1}$ and $\beta = \alpha$, $w_{ij}(p) = w_{ij}(x) = 1$, then we have the SMACOF equation 2.23. However for object clouds, we recompute both the weight and the Laplacian matrices at every iteration since the value of w_{ij} depends on the amount of attracting and colliding objects at a given iteration. At each iteration, w_{ij} changes because the objects that are being attracted or separated changes. Therefore our weighting is not constant as in the original SMACOF algorithm and as such the Cauchy-Schwartz inequality in equation 2.27 does not always hold true.

Fluctuation in the value of, w_{ij}, however does not affect the convergence of the energy function. We still get a set of non-increasing stress values that obey the sandwich inequality. Under normal MDS, the equation 2.27 is necessary for a set of decreasing stress values because the first two terms in $f(x)$ and $g(x, y)$ are equal and constant when w_{ij} is constant. Therefore the difference between the first two terms and the third term in $f(x)$ is less than difference between the third term in $g(x, p)$.

Under the MDS formulation for object clouds, a similar dynamic comes into play when w_{ij} fluctuates. If tr $X^\top B(X, W_x) X <$ tr $X^\top B(X, W_p) P$ then the first two terms in equation 3.14 are also less than those in equation 3.13 and vice versa. The value of these terms is such that the difference between the first two terms and the third term in equation 3.14 is less than difference first two terms and the third term in equation 3.13. Hence despite fluctuations in the value of w_{ij}, the sandwich inequality still holds for energy-based object clouds and we get a set of non-increasing energy values.

3.3 AR Implementation

To showcase our semantic cloud algorithm, we implemented an augmented reality 3D object cloud with the use of the Unity game engine and the Microsoft HoloLens Development Kit (SDK).

In order to create the object cloud, various 3D models were converted and then imported into Unity as OBJ files. The OBJ file format is a data format that specifies the geometric

properties of 3D objects. These properties include the coordinates of vertices for the object, as well as the texture properties for the object. The 3D models were stored as assets in Unity alongside their distance and adjacency matrices. The distance matrix was pre-computed in Python by extracting the multi-view convolutional feature vectors for the 3D models and calculating their pairwise Euclidean distance. Similarly the adjacency matrix was pre-computed in Python by creating a connected k-nearest neighbour graph over the space of convolutional feature vectors.

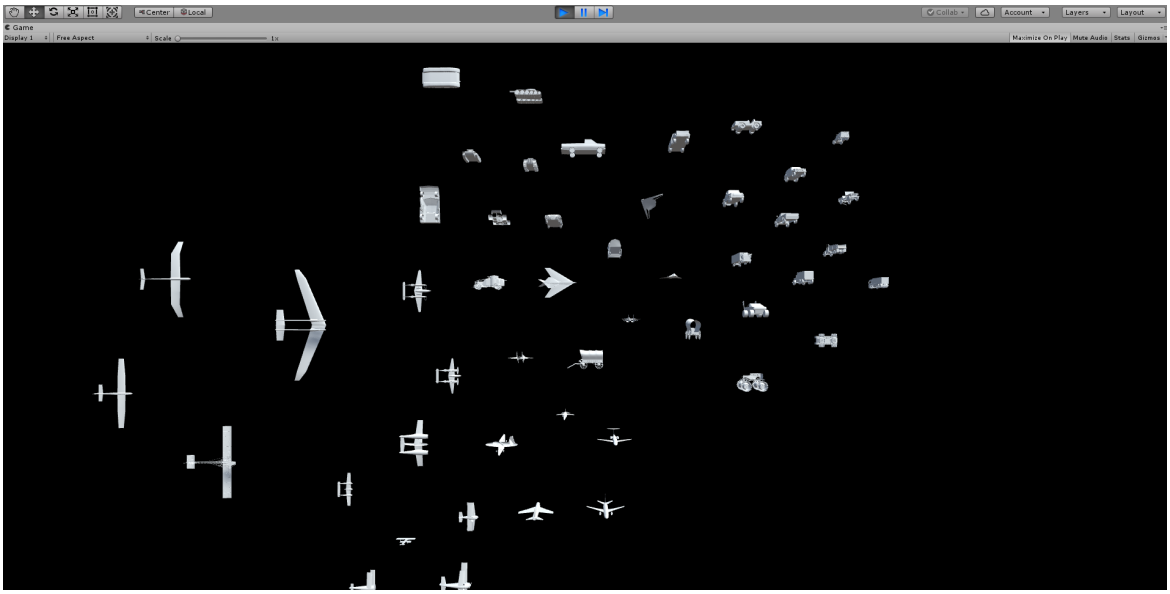


Figure 3.7: An energy based semantic object cloud generated using Unity and Microsoft HoloLens SDK

Using the HoloLens as well as the Microsoft HoloLens SDK allowed us to focus on the algorithmic implementation of the object cloud in Unity while it handled the registration and superimposition of the object cloud within the real world. The Unity game engine displays the 3D models within the cloud based on the camera parameters set by the user. For augmented reality, the position of the camera is taken as the origin of the real world and it is assumed to be the position of the user when wearing the HoloLens headset. To properly view the object cloud, the 3D models were scaled by half and placed at a depth of 2 metres from the main camera. The camera was set to a 60° field-of-view with the near clipping plane set at 0.85 and the far clipping plane at 1000. These settings specify a frustum originating from the camera whereby anything within the frustum can be observed by the user and anything outside it is not. To this end, the 3D models were

placed at random x and y position within the frustum so that the semantic cloud is visible to the user.

To speed up the energy minimization process and guarantee convergence into a stable semantic, we implemented the majorization algorithm in Algorithm 3 for generating object clouds. The algorithm was implemented with the aid of the Math.NET Numerics matrix library. The library provides tools for vector and matrix calculations as well as tools for importing the pre-computed matrices. Despite this we had to create a few operations such as element-wise comparison between a pair of matrices. An example of the generated object cloud is shown in 3.7.

3.4 Experimental Evaluation

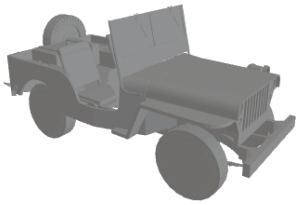
3.4.1 Object Clouds

Dataset

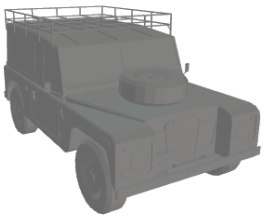
To evaluate the energy formulation for object clouds, we used the Princeton Shape Benchmark (PSB) and the ModelNet40 datasets [34][46].

Princeton Shape Benchmark The PSB dataset is a collection of 1,814 3D CAD models that were retrieved from the internet in order to evaluate the performance shape-based retrieval algorithms. Each of the 3D models is encoded as an Object File Format (.off) file and it contains additional information about its original source. The dataset is divided into an equal number of training and test datasets - 907 models each.

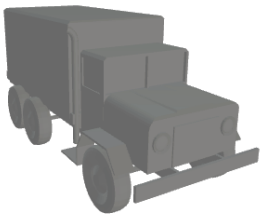
The PSB dataset is unique in that it has several hierarchies of classification. They range from general classes like "musical instruments" and "furniture" to more specific classes like "acoustic guitar" and "desk with hutch" respectively. These hierarchies reflect both the primary and secondary form of each model. In total there are 161 general and specific classes. The lowest classification level contains at least 4 3D models and the largest class (general or specific) contains 100 3D models.



(a) A jeep in the PSB dataset



(b) A jeep in the PSB dataset



(c) A truck in the PSB dataset

(d) A sedan in the PSB dataset

Figure 3.8: Example of 3D vehicles in the PSB dataset. Each vehicle can be sub-classified as truck, jeep or sedan.

ModelNet40 The ModelNet40 is a larger collection of 3D models that was created in order to evaluate the performance of deep learning models on the task of 3D object recognition. There are about 151,128 3D CAD models within the dataset, each of which belong to roughly 660 object categories [46]. The 3D models were retrieved from a variety of 3D search engines like the 3D warehouse and were filtered afterwards by Amazon Mechanical Turkers in order to remove miscategorized objects. Each model is stored as an Object File Format (.off) file. Although there are many object categories containing everyday objects, the ModelNet40 dataset specifies 40 classes.

Models

We evaluate the performance of each of the three energy optimization techniques discuss above. Additionally because our energy-based model is similar to the context preserving algorithm, we also evaluate its performance for generating semantic object clouds. Finally we include the breadth-first search algorithm for pseudo-random object clouds as a baseline for our comparisons.

Metrics

In order to compare the various object cloud algorithms, we utilize 4 different metrics: trustworthiness, realized adjacency, compactness and energy.

Trustworthiness When performing dimensionality reduction, each point in the original manifold has a neighbourhood that contains a set of close points that should ideally be retained in a lower dimensional manifold. Trustworthiness measures how well such neighbouring points from the original dimension are preserved in the lower dimension. It is defined as follows

$$T(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i^k} \max(0, r_{ij} - k) \tag{3.15}$$

where $\mathcal{N}_i^k$ are the K-Nearest neighbours in the lower dimension and r_{ij} is the rank of each neighbour in the original input space. Each neighbour that is unexpected in the lower dimensional space is penalised by its rank in the original dimension and the fractional term helps normalize the output. Trustworthiness has a value from 0 to 1 where 0 indicates that all the points have unexpected neighbours and 1 indicates that the neighbourhood for every point in the higher dimensional space is well preserved in the lower dimension.

Realized Adjacency The realized adjacencies are the set of words that are adjacent each word [2]. A word is adjacent another word if its boundary from that word is within 0.01-0.05% of the size of the smaller word. This metric measures the level of similarity across adjacent words within the cloud. Its value is between $[0, 1]$ with a higher value indicating that there many adjacent words within the cloud are similar and vice versa.

$$A = \frac{\sum_{i,j \in E_r} \text{sim}(p_i, p_j)}{\sum_{k,l \in E} \text{sim}(p_k, p_l)} \tag{3.16}$$

Compactness Compactness measures how tightly packed each of the words are within the given 2D layout of the cloud. It is calculated by dividing the total area by the used area. The total area refers to the tightest rectangular area bounding all the words or objects within the layout. It is measured by multiplying the difference between the X-axis of the leftmost and rightmost object or word with the difference between the Y-axis of the uppermost and lowermost object or word. The used area however is the sum of the area of each individual word within the cloud. The metric can be expressed as follows

$$C = \min(1, \frac{\text{Used Area}}{\text{Total Area}}) \tag{3.17}$$

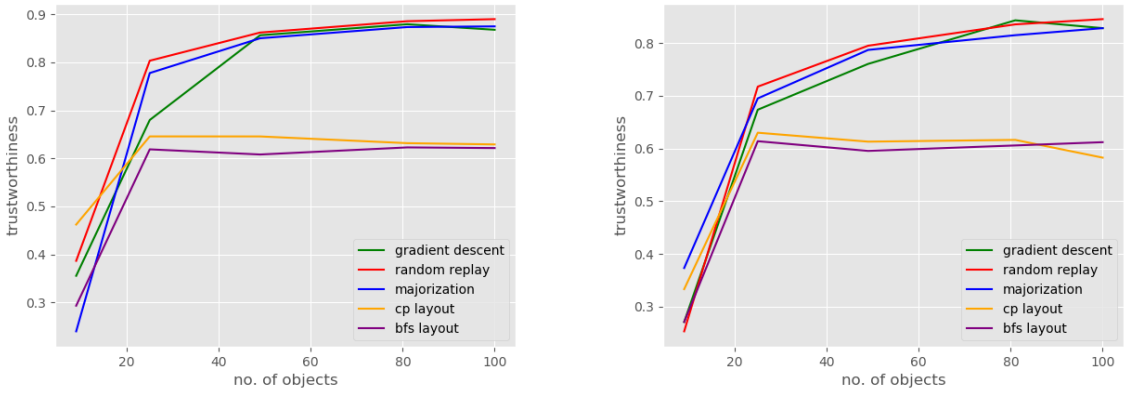
When the rectangular area bounding all the objects or words is larger than the sum of area of all the objects or words, $C < 1$ indicating that there is some space between the words. As this space increases, $C \to 0$. Conversely, when all the object or words are tightly packed, the rectangular area is equal to the sum of the area of all the words or objects and $C = 1$. However if many of the words begin to overlap each other, $C > 1$. Therefore a good value for compactness is $0.5 \leq C \leq 1$.

Data Preprocessing

Each of the 3D models from the PSB and ModelNet40 datasets was passed into a Blender python script in order to extract varying views of the model. As discussed in chapter 2, the various views are gotten from cameras that are positioned at an angle and ground elevation of 30° around the model. The python script generates 12 views which are then passed into an MVCNN that was pre-trained using the Resnet-18 neural network. Rather than use the MVCNN network as it is, we fine-tuned it to each dataset by training it over a dataset for 5 epochs. Each epoch was over 1000 iterations and at the end of training, the MVCNN had an accuracy of over 85% on each of dataset. After training, the dataset was passed into the network and the vector output from the penultimate layer was used as high-dimensional vector embedding for each of the algorithms to be evaluated.

Results

We evaluated how well each of the optimization methods performed on the various shape datasets for 9, 25, 49, 81, and 100 objects. The following results were obtained



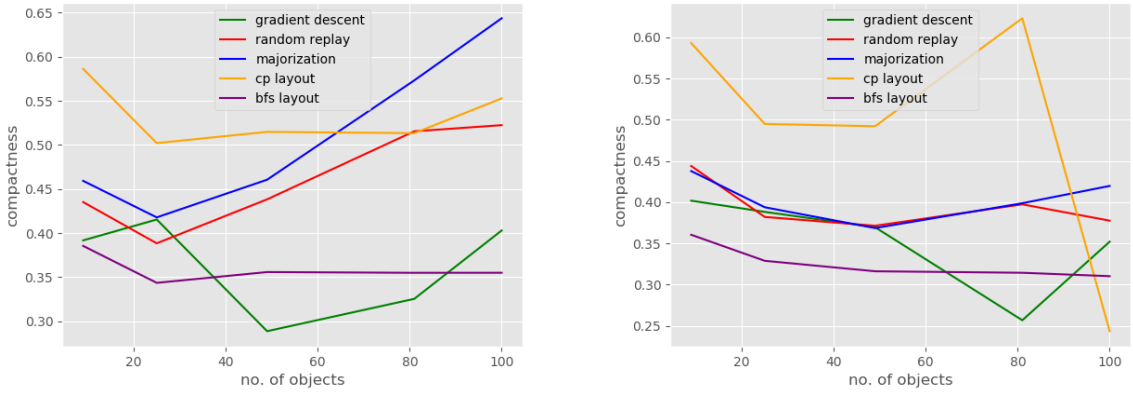
(a) ModelNet 40

(b) Princeton shape benchmark

Figure 3.9: Trustworthiness of the various algorithms on different shape datasets.

In figure 3.9 we see that on both the Princeton Shape Benchmark and the ModelNet 40 dataset, all three optimization layouts outperform the context preserving and breadth-first layouts. While the context preserving and breadth-first layouts manage to preserve some of the high-dimensional semantic structure of the objects, they do not do so to the degree of the optimization layouts. As discussed in the previous sections, this is due to the fact that the optimization layouts utilize a high-dimensional KNN graph in their operation as opposed to the lower dimensional graph. Therefore they lose less information about the semantic structure of the object representations. This is illustrated in figures 3.13, 3.14, and 3.15 where objects like sofas and chairs are grouped separately but placed in such a way that they morph into each other. Whereas in the figures 3.16 and 3.17, there is some semantic order but those object are not clearly grouped within the cloud.

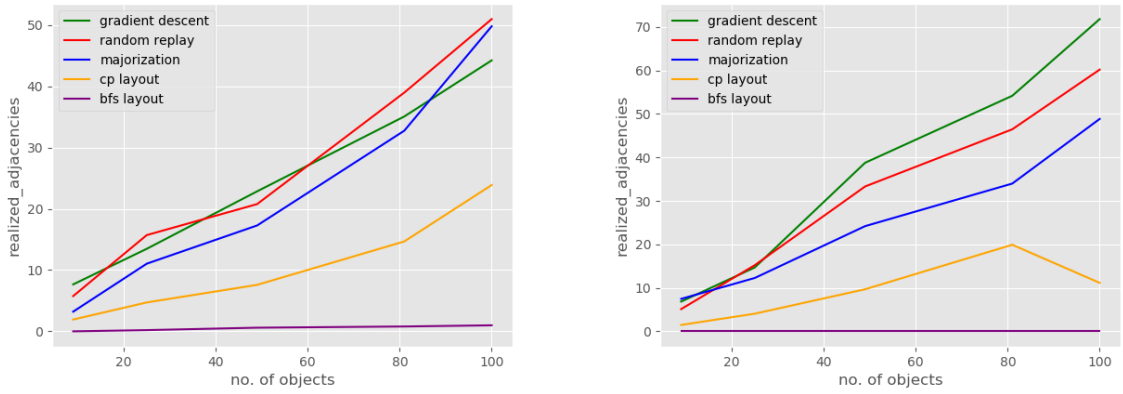
When we observe the compactness of the layouts from figure 3.10, we can see that both the majorization, random replay and context preserving algorithms perform well on both datasets. The relatively poor performance of gradient descent is due to the way compactness is calculated. The figure 3.13 has a wider bounding box for its object cloud than those of figures 3.14 and 3.15. Hence the total area is much larger. The gradient descent layout also has relatively more overlaps which leads to a smaller used area than



(a) ModelNet 40 (b) Princeton shape benchmark

Figure 3.10: Compactness of the various algorithms on different shape datasets.

that of the other two optimization layouts. The smaller used area and larger total area therefore lead to much less compactness for the gradient descent layout. The breadth-first layout on the other hand makes a more uniform use of the layout as illustrated in figure 3.16. However some of the objects are too small for the grid in which they have been placed in, leading to excess space among objects of various grids and thus a relatively low compactness for the layout.

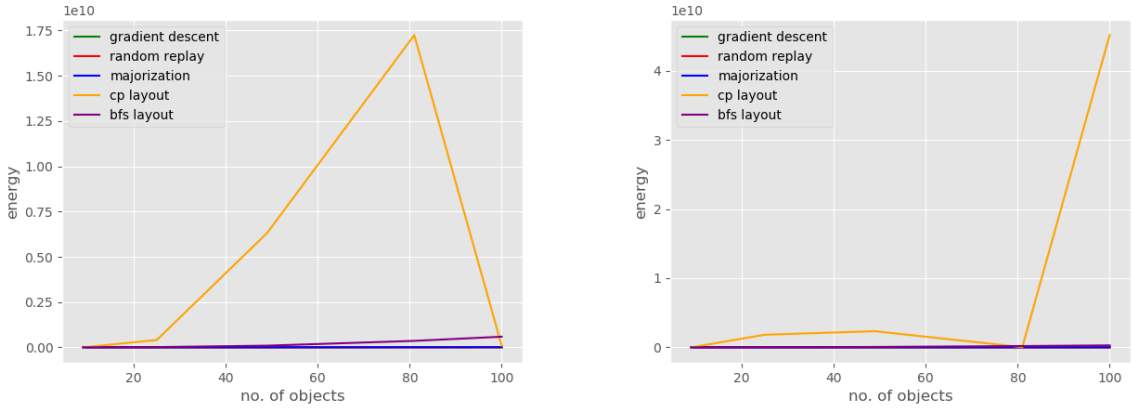


(a) ModelNet 40 (b) Princeton shape benchmark

Figure 3.11: Realized adjacency of the various algorithms on different shape datasets.

From figure 3.11, we once again observe that the optimization algorithms outperformed both the context preserving and breadth-first layouts on the realized adjacency. This is likely due to the fact that the optimization layouts minimize the squared pairwise distance between neighbouring objects while maintaining a high degree of semantic similarity between said objects. By minimizing the distance, similar pairs of objects touch each other which in turn increases the realized adjacency of the layouts. The context preserving layout does the same thing but additionally it tries to maintain the planarity of its underlying Delaunay graph. As discussed earlier, this graph has a lower degree of semantic similarity and so when neighbouring objects do touch each other, they might not be very similar which in turn reduces the realized adjacency of the layout. Furthermore, as can be seen in figure 3.17, the highly compact nature of the context preserving layout means that neighbours that are colliding do not get included in the realized adjacency sum thus leading to a lower value for the layout.

While a highly compact layout in which there are significant collisions may reduce the realized adjacency, a less compact layout such as that of the breadth-first layout also reduces the realized adjacency because fewer object are touching each other. Since many of the objects in the breadth-first layout are smaller than their grid, the space between neighbouring objects was too large for the objects to be considered as touching each other. Hence the very low realized adjacency for the layout.



(a) ModelNet 40

(b) Princeton shape benchmark

Figure 3.12: Energy of the various algorithms on different shape datasets.

In figure 3.12 we observe that except for the context-preserving layout, the layouts for the other algorithms minimize the energy of an object cloud. This may be due to the

the large amount of colliding objects within the context-preserving layout. As discussed in the previous sections, a larger repulsive weighting is needed to repel colliding objects. Therefore when computing the energy of a cloud, colliding objects have higher energy. Since our experiment utilized a repulsive weighting that is 8 times that of the attractive weighting, this may account for the high energy within the context-preserving cloud.

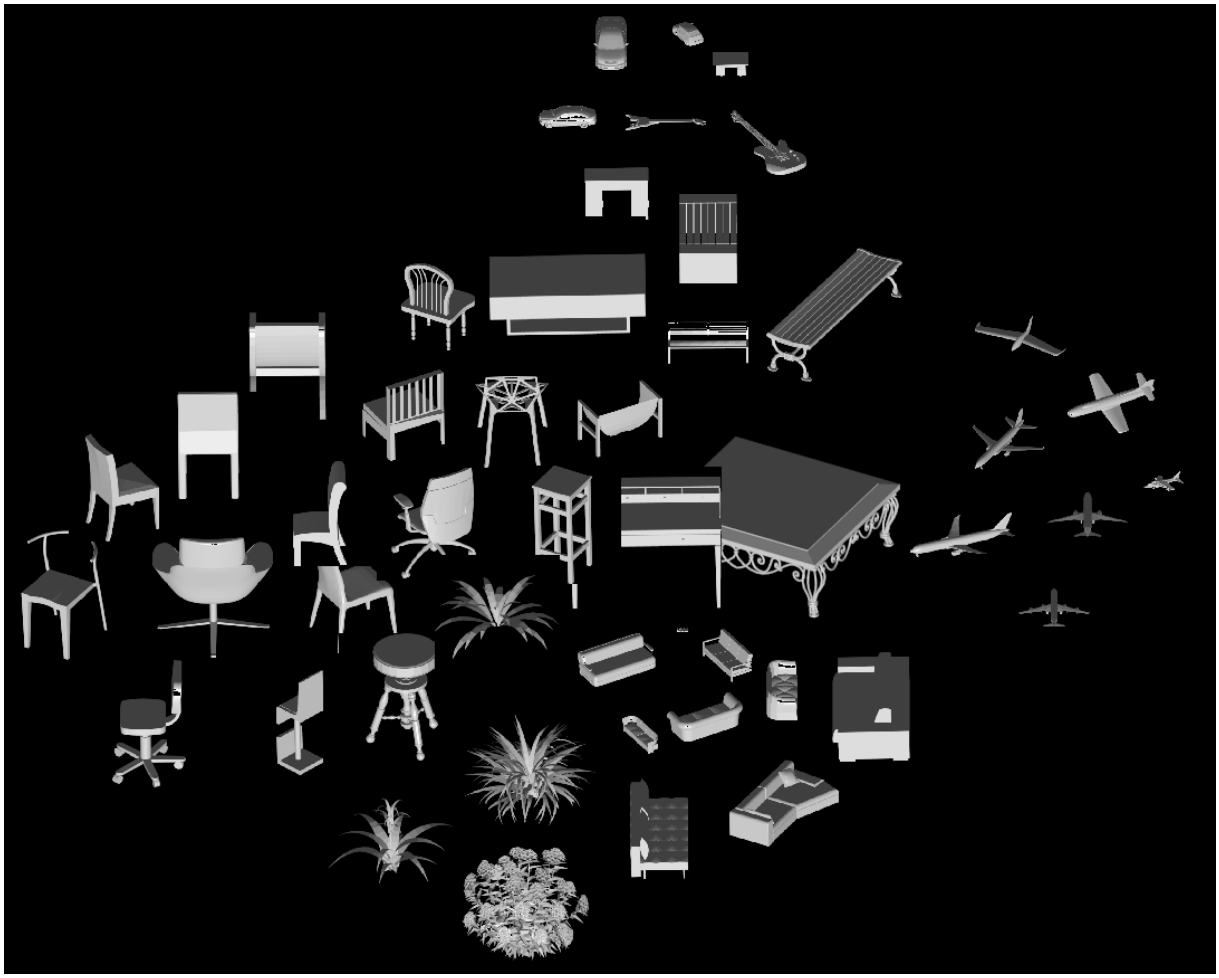


Figure 3.13: Gradient Descent Layout on the ModelNet40 dataset. (no. objects = 49)

Figure 3.14: Random Replay Layout on the ModelNet40 dataset. (no. objects = 49)

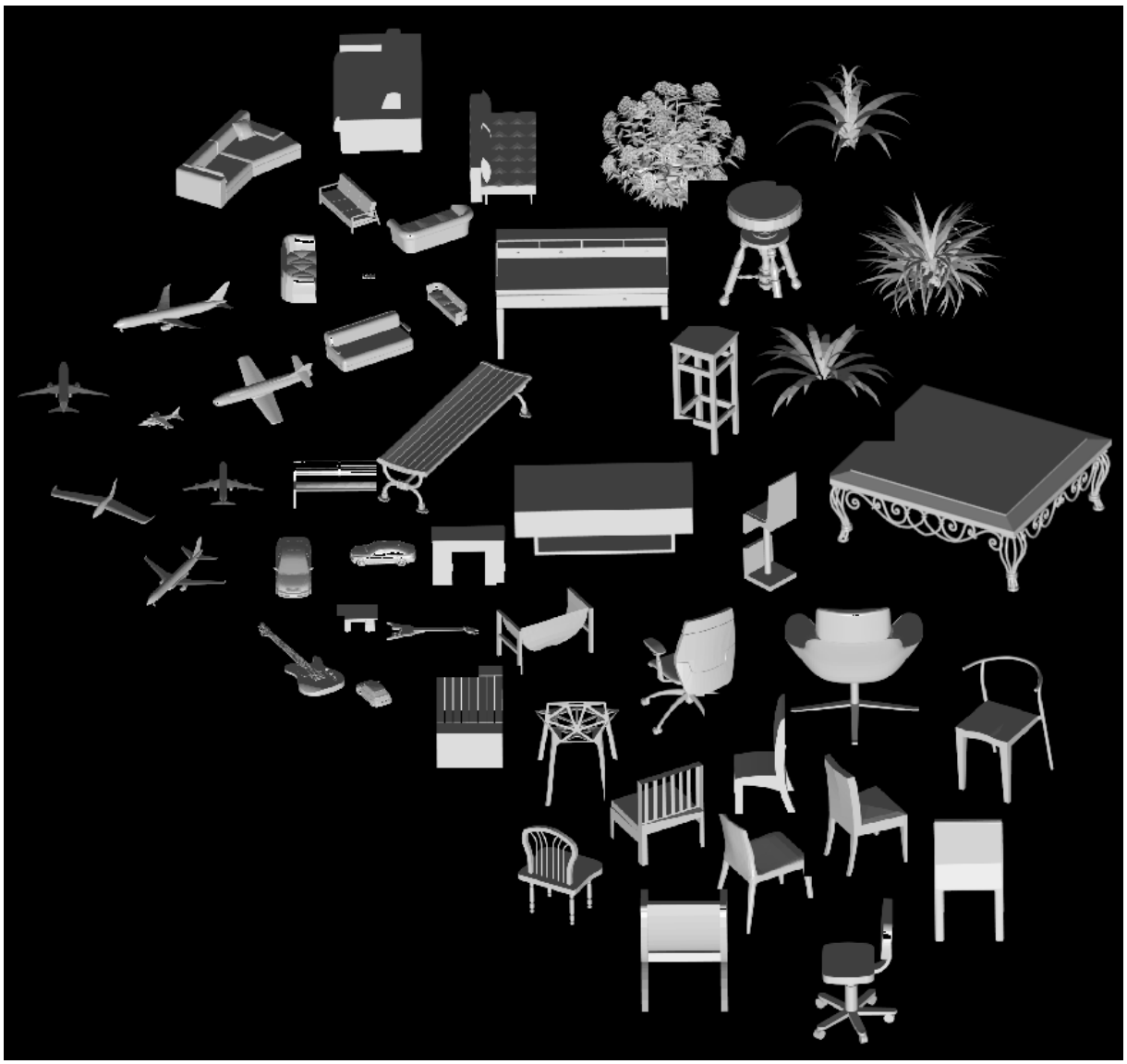


Figure 3.15: Majorization Layout on the ModelNet40 dataset. (no. objects = 49)

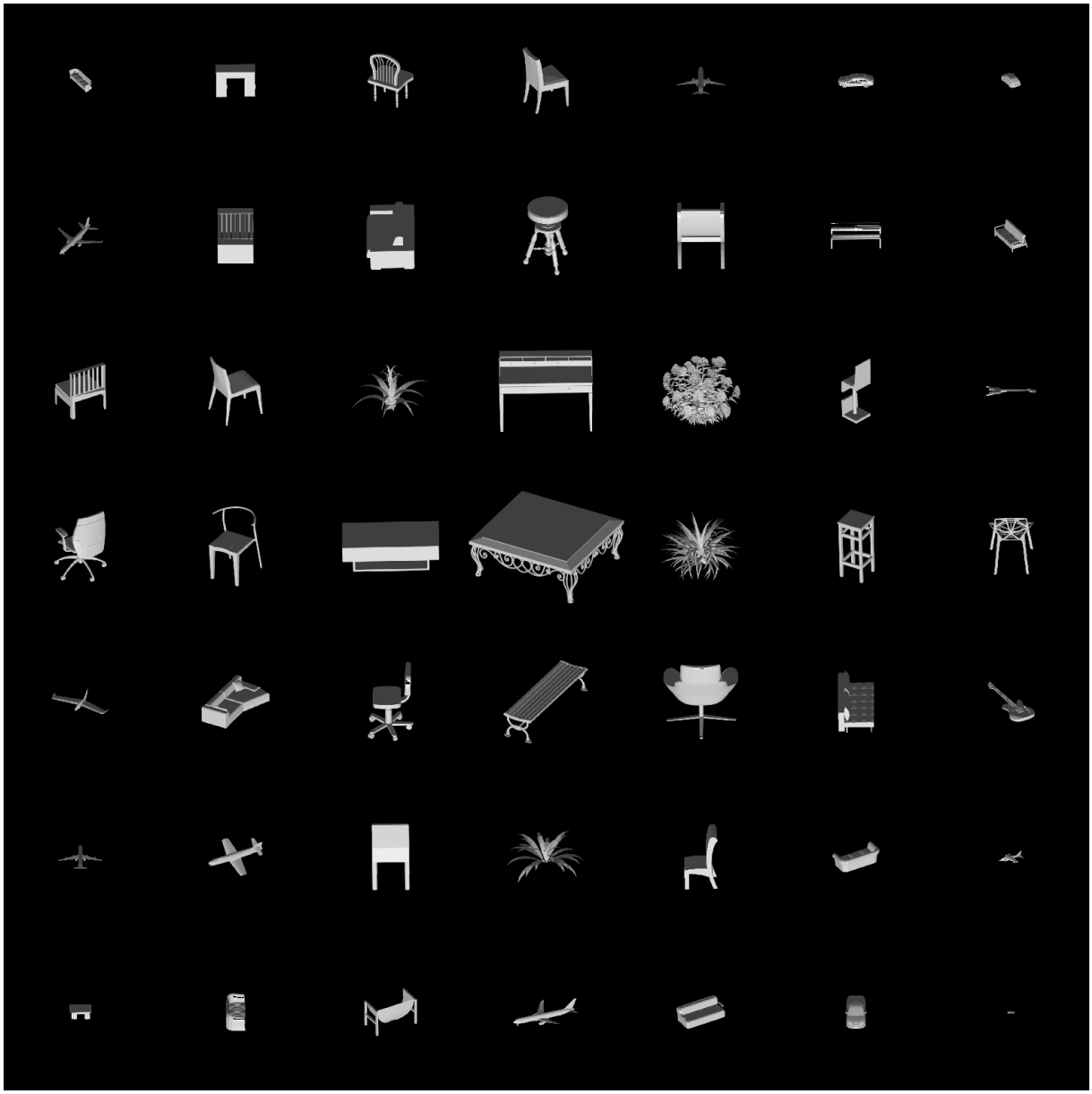


Figure 3.16: Breadth First Search Layout on the ModelNet40 dataset. (no. objects = 49)

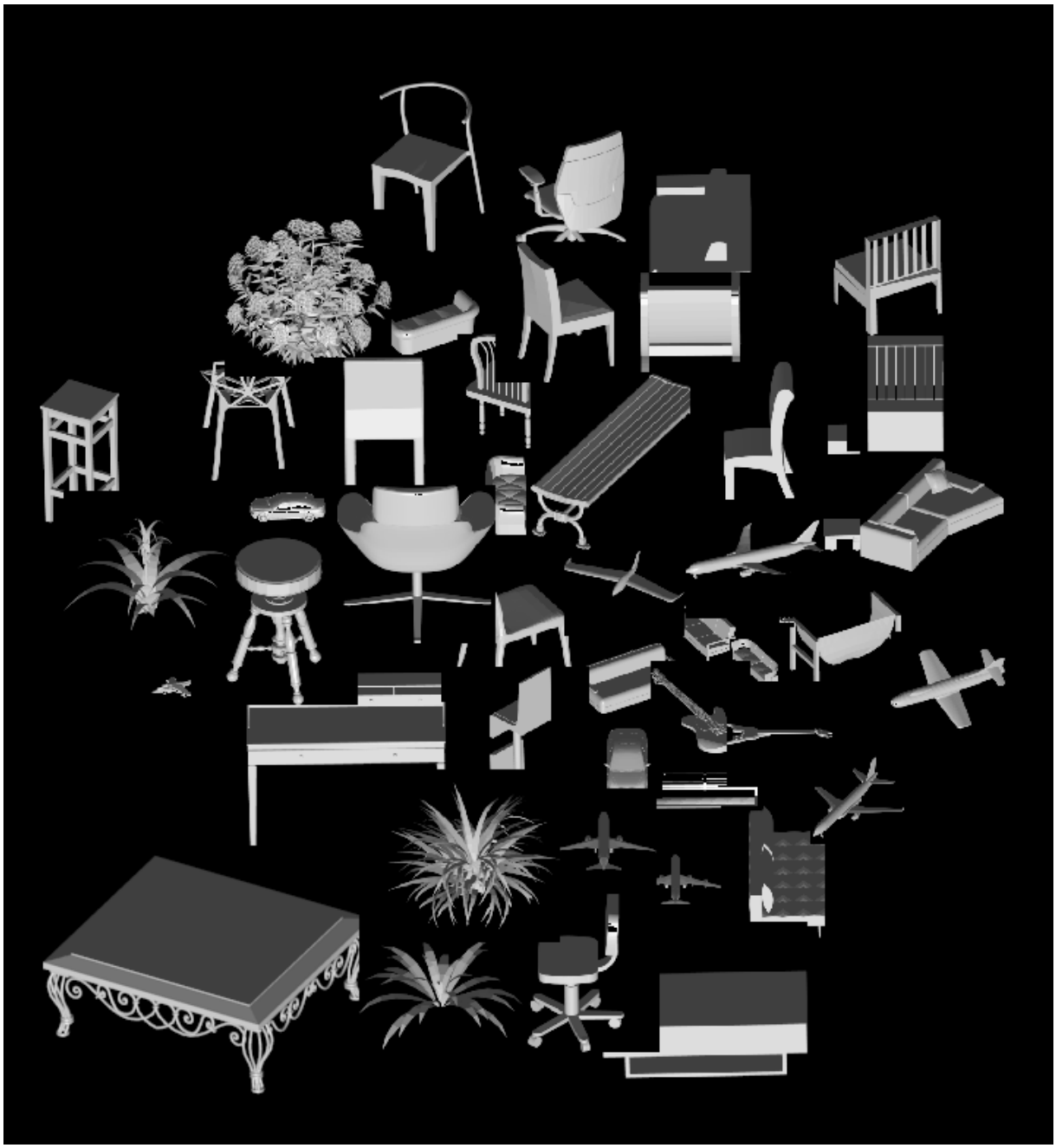


Figure 3.17: Context Preserving Layout on the ModelNet40 dataset. (no. objects = 49)

3.4.2 Word Clouds

Dataset

To evaluate the energy formulation, we use 2 natural language process (NLP) corpora: the Gutenberg and the Reuters corpora. The Gutenberg corpus is a large collection of over 60,000 books that are stored in the plain text format. We use a small subset of this corpus that contains 18 books with over 2 million words. We randomly selected 10 of these books and generated a word cloud for each of them. The performance of a model over each word clouds was then averaged and plotted.

Instead of books, the Reuters corpus on the other hand is a large collection of new articles spanning different categories from sports to entertainment. In total there are around 12 million words in over 10,000 articles that cover 90 topics. The words in these articles are less likely to be formal as opposed to those in the Gutenberg corpus and as such it makes the Reuters corpus a good alternate dataset. Again we randomly selected a set of topics from the corpus and from each topic, randomly pick one news article. We evaluated the performance of a model over the selected news articles and for each model, averaged its performance.

Models

To evaluate the energy-based model on word cloud, we use only the gradient descent optimization. This is because as mentioned earlier, majorization is not easily applied to word clouds and we find that random-replay offers no difference to the gradient descent algorithm on this task. We compare the performance of the gradient descent model to the context preserving algorithm and to the seam carving algorithm which is another energy-based word cloud model. We also include the random word cloud algorithm, Wordle, as a baseline for our comparisons.

Metrics

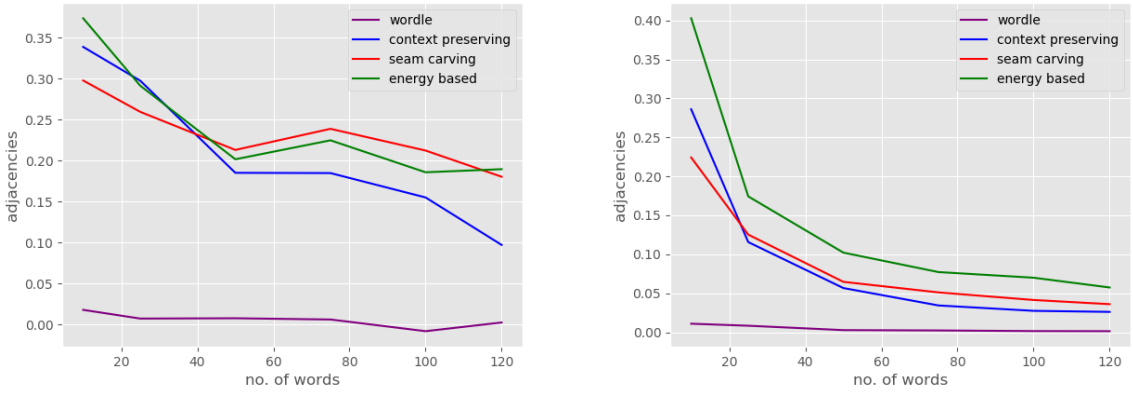
In order to compare the various word visualization algorithms, we evaluate them on the energy metric as well as on the following metrics that are typically used to evaluate semantic word clouds: realized adjacencies and compactness.

Data Preprocessing

Similar to [2] we first extract each word from a given text and filter our common stop words. The unique words are then stemmed using the Porter Stemming Algorithm such that words like "fishing" and "fishes" become "fish". We then generate a vector representation of the remaining words using the Word2Vec algorithm [47][29].

Results

To test the models against the metrics, we repeated each experiment for the following number of words: 10, 25, 50, 100 and 250, in each book or article from both datasets. The following results were observed



(a) Gutenberg dataset.

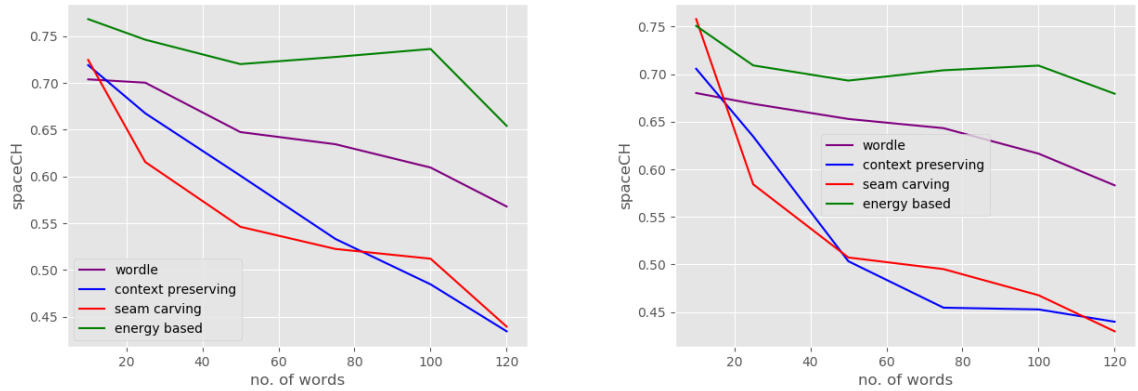
(b) Reuters dataset.

Figure 3.18: Realized adjacency of the various algorithms on different datasets.

In figure 3.18 we observe that the semantic word cloud algorithms all have higher realized adjacencies than Wordle. This is because similar words are placed together and as such the total similarity of the adjacent words to any given word is high. Whereas such total similarity is low for any given word in Wordle because the adjacent words for any given word are dissimilar. This is illustrated in figure 3.24 where words of similar semantic meaning are scattered rather than grouped. Amongst the semantic word clouds, gradient descent outperforms both seam carving and context preserving word clouds. This is because, as shown in figure 3.21, it creates a more compact cloud in which similar words are close enough to have their adjacencies realized. In the other semantic algorithms, there

is more excess space amongst words and as such some words are not close enough to have their adjacencies realized.

We observe the compactness of the algorithms in figure 3.19 where gradient descent is the most compact algorithm followed by Wordle. In Wordle, there is an efficient use of space in that empty spaces are filled with words until there is no space left. Therefore an efficient algorithm should make better use of the layout space than Wordle. Due to the weaker attractive force and the annealed learning rate; words in the context preserving cloud are not being pulled close together to make efficient use of the drawing space. This can be observed in the figure 3.22. Similarly, seam carving does not make efficient use of the drawing space. In figure 3.23, there is are no seams to carve out of the cloud leaving excess space amongst words in the cloud. In fact, we notice that as the number of words increase, the compactness of both seam carving and context preserving algorithms drop drastically whereas those of gradient descent and Wordle are quite stable.



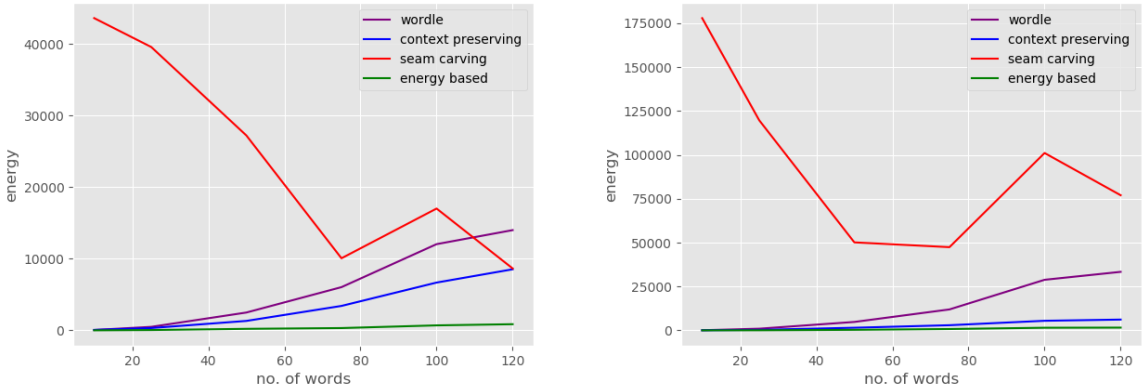
(a) Gutenberg dataset.

(b) Reuters dataset.

Figure 3.19: Compactness of the various algorithms on different datasets.

The effects of compactness and semantic order is captured by the energy metric. Again here we see that the gradient descent algorithm outperforms the other algorithms with a lower energy. Both context preserving cloud and gradient descent have close energy states because semantically similar words are not far from each other in those layouts. The energy metric really captures how far apart similar words are from each other. This is evident as Wordle which has the highest energy. While there are no overlaps in its layout, many similar words are far apart and as such raise the energy of the final layout. This is further seen in the layout of semantic words where the energy does not grow exponentially with

the increase in words. This is because similar words remain close to each other and as such keep the energy from growing too large.



(a) Gutenberg dataset.

(b) Reuters dataset.

Figure 3.20: Energy of the various algorithms on different datasets.

Figure 3.21: Gradient Descent Layout on *Macbeth* from the Gutenberg dataset. (no. words = 75)

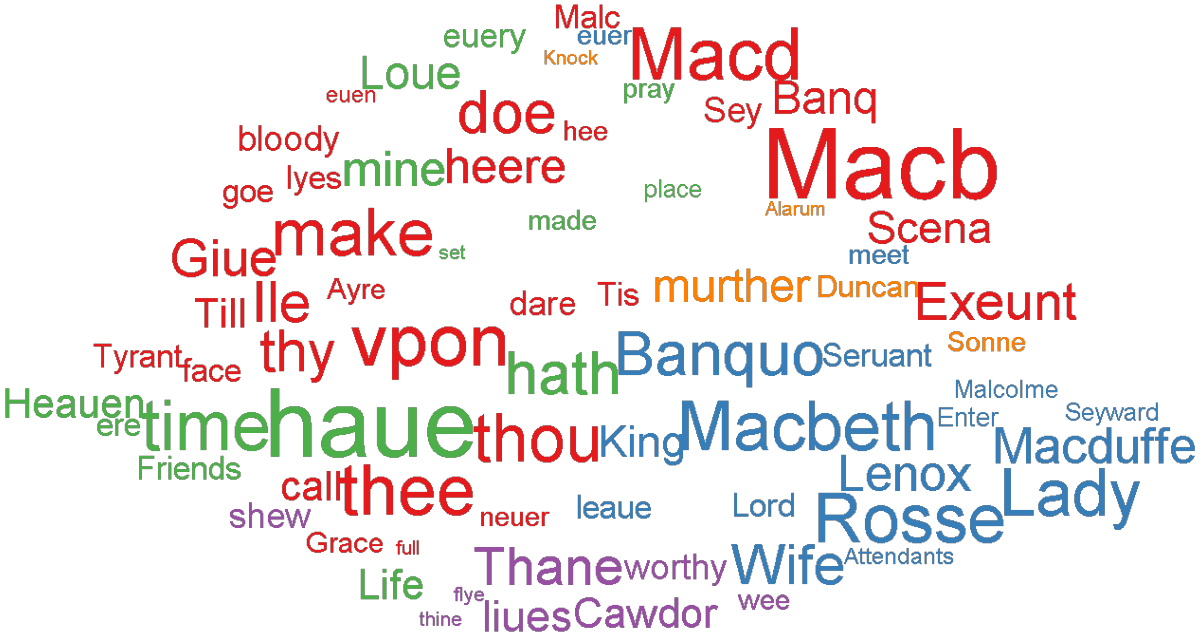


Figure 3.22: Context Preserving Layout on *Macbeth* from the Gutenberg dataset. (no. words = 75)

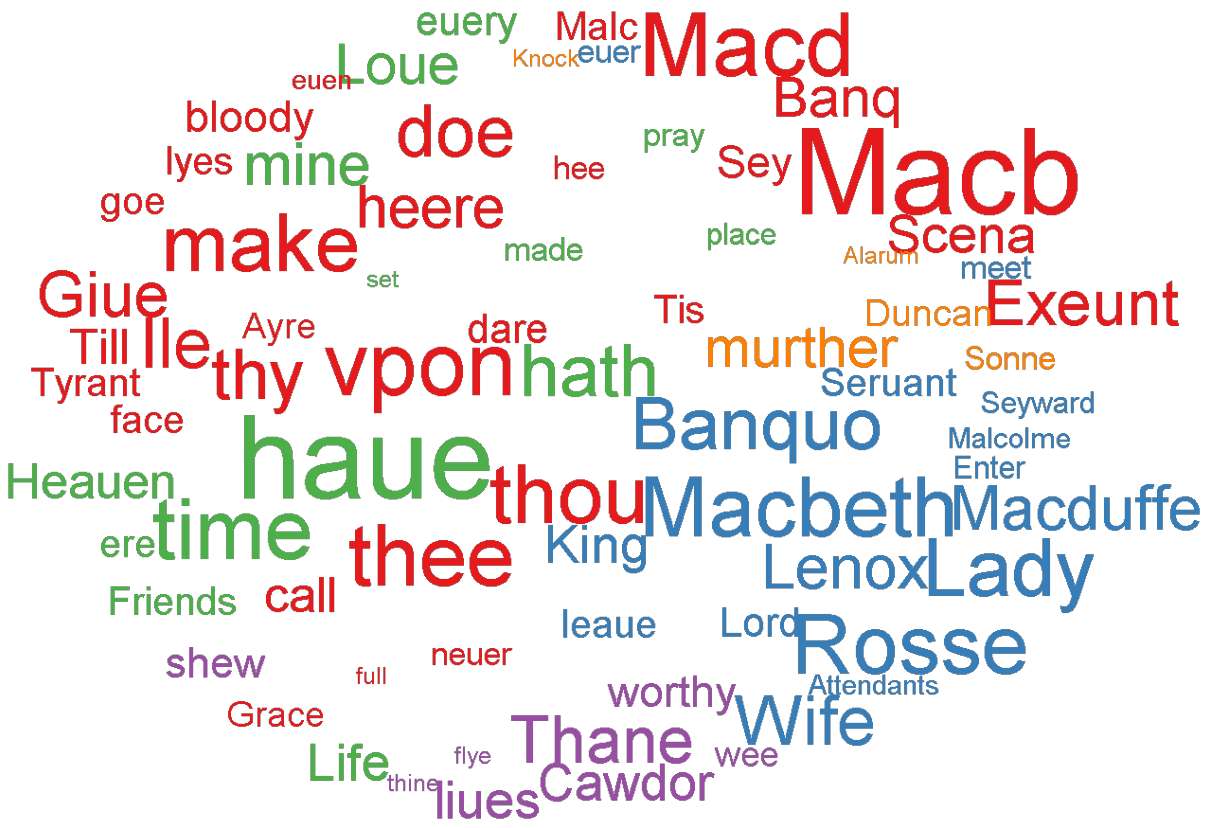


Figure 3.23: Seam Carving Layout on *Macbeth* from the Gutenberg dataset. (no. words = 75)

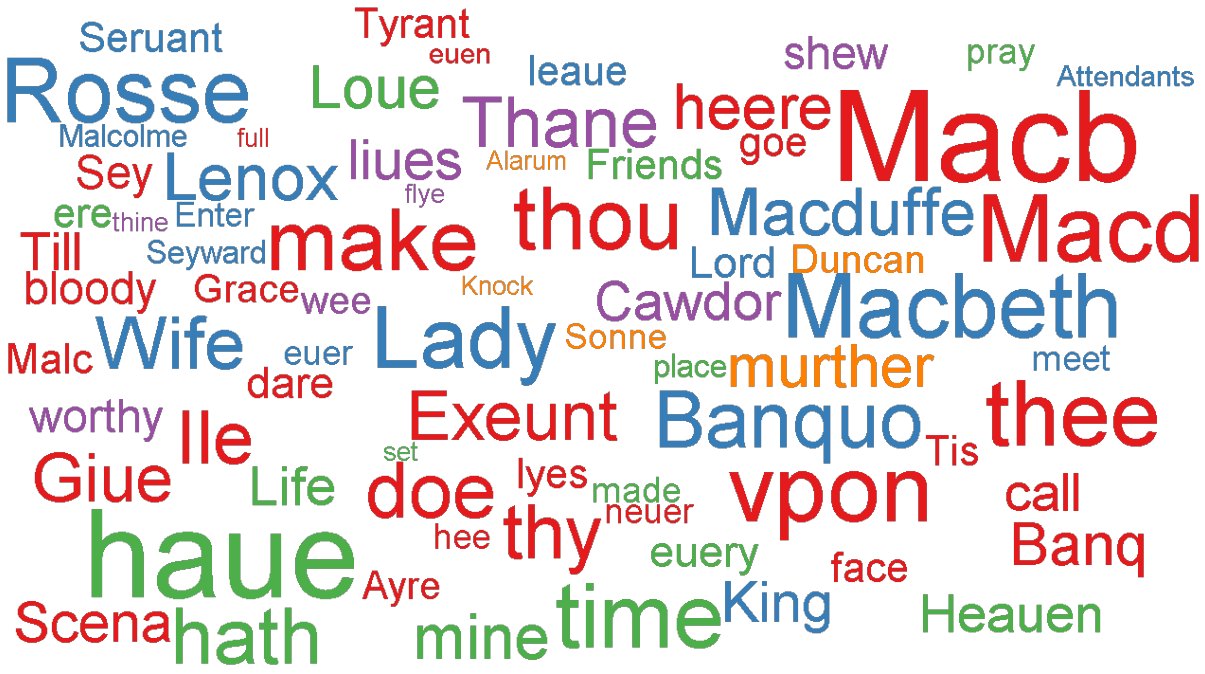


Figure 3.24: Wordle Layout on *Macbeth* from the Gutenberg dataset. (no. words = 75)

Chapter 4

Latent Object Clouds

4.1 Latent Object Cloud

4.1.1 Latent Variables

Semantic object clouds allow us to perform a visual analysis of the relationship between various objects within the cloud. Since objects with similar visual features are placed close together, we can intuitively locate objects within the cloud that have a particular feature.

However there are limitations to the kind of visual analysis that can be performed with semantic object cloud. One such limitation is the ability of the cloud itself to separate groups of objects that share a certain visual characteristics. These characteristics may be coarse visual characteristics that exists between two classes of objects like the "the set of vehicle from the set of non-vehicles" or they may be fine-grained visual characteristics that exist within a particular class like "the set of bi-planes from the set of fighter-jets". Being able to group and identify these sets of characteristics within an object cloud allows us to perform quicker searches and deeper visual analysis of particular classes of objects within the cloud. In order to identify various levels of visual characteristics within a object cloud, we need to be able to cluster the objects into various groups.

Latent object clouds are clustered object clouds in which the clusters are semantically packed together and are each represented by a latent variable. They are similar to word clouds where groups of co-occurring words are clustered [27] [3][31]. There are various techniques for clustering data but for latent object clouds, we adopt K-Means which is an example of hard Expectation-Maximization (EM) applied to a Gaussian Mixture Model (GMM) clustering [6].

The EM algorithm assumes there are a set of unobserved factors that are responsible for an observed data. It then finds the probability of each data point belonging to any of the given factors. There are 2 types of EM: hard and soft EM clustering. In soft EM, each data point can be explained by one or more factors that share the probability of explaining said data point. However in hard EM, each data point is assigned to the factor with the highest probability of explaining it. For latent object clouds, we assume that the factors are characteristics that explain the semantic positions of the objects within the cloud.

$$p(x_i|z_k) = \mathcal{N}(x_i|\mu_k, \sigma_k)^{z_{ki}} \tag{4.1}$$

Each factor, z_k, is one of k latent variables in a latent object cloud and it is a random variable that has a mean, μ_k, and a co-variance, σ_k, that are used to characterize a probability distribution of images or 3D-objects. This probability can be expressed by the Gaussian probability in equation 4.1. Using these parameters, we can ideally generate images and 3D-objects that share a given characteristics in question [30]. For example, if z_k represents the set of cars within an object cloud, μ_k represents what the average car looks like and, σ_k represents variations in what the cars look like. Another latent variable z_{k+1}, may represent the set of airplanes and its parameters would function the same way. Generating objects using this approach is however not without its challenges, one of which is aligning the views of the 3D-objects or utilizing their canonical view in order for the average object, μ_k to be properly visualized.

The primary goal of latent variables is not to generate objects within the cloud but instead to group them based on certain visual features. Using the average object, μ_i, for a factor, z_k, we can determine if z_k explains the characteristics of an object, x_i, by comparing the euclidean distance between x_i and μ_k with those of other factors. If μ_k is the closest to x_i in comparison to the average object of other factors, then z_k is most likely the factor that explains x_i and x_i is grouped with all other objects that are explained by z_k. Subsequently μ_k is updated by taking the mean of the objects assigned to z_k. This is illustrated by the equation below.

$$z_{ki} = \begin{cases} 1 & \text{if } k = \underset{m}{\operatorname{argmax}} \|x_i - \mu_m\|_2^2 \\ 0 & \text{otherwise} \end{cases} \quad , \mu_k = \frac{\sum^{N_k} x_i}{N_k} \tag{4.2}$$

where N_k is the number of objects assigned to factor z_k.

It follows that for every object in a latent object cloud, there is only one latent variable that is assigned to it. This means that we can have one latent variable that explains all

the objects within an object cloud or many latent variables - one per object - that explains the semantic position of all the objects.

4.1.2 From random to semantic object clouds

Using the concept of latent variables, we can transform a random object cloud such as the one generated by Wordle to an object cloud with a lesser degree of randomness.

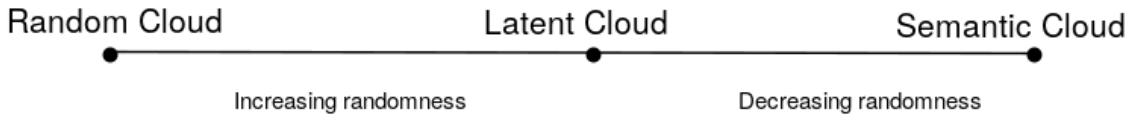
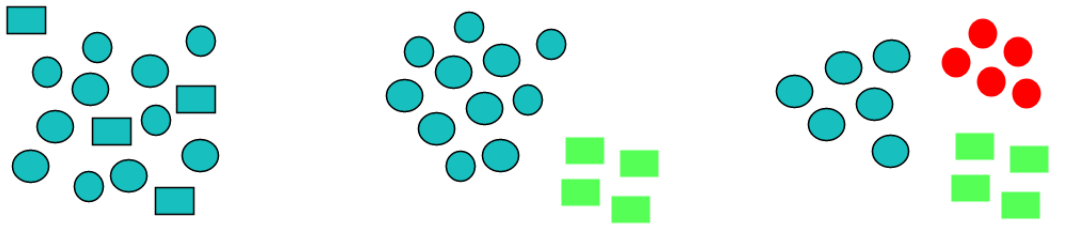


Figure 4.1: Relationship between random and semantic clouds

To add some semantic orderliness to a random word cloud, we first cluster an object cloud using a set of k latent variables so that each object within the cloud is assigned to one of k variable. All objects assigned to the same latent variable are then clustered together within the cloud. In order to do this, two types of clustering occur: high-dimensional and low-dimensional clustering. The high-dimensional clustering involves performing K-Means on the image vector for the objects so as to decide which latent variable an object belongs to. The low-dimensional clustering involves specifying a graphical structure over all objects in the cloud that belong to a latent variable and then minimizing the energy amongst those objects so that they are compactly placed close to each other. Since we are converting a random cloud to a semantic cloud, this graphical structure is typically a connected random graph between all the objects of a particular latent variable.

After clustering the objects for each latent variable, the variables themselves are then semantically ordered relative to each other by using their mean object, μ_k. The mean object, μ_k, for each latent variable acts as an actual object within the cloud even though it is not actually present in the cloud. This way, the energy-based algorithm of 3.6 can be used to minimize the energy within the latent variables and semantically organize them. For the mean object, μ_k, to be treated as an actual object, it needs to have a position and size. Its position is the average position of the clustered objects for the latent variable and its size is the radius gotten from the total area of the clustered objects. After semantically organizing the latent variables, each cluster of objects is moved to the position of the mean

object, μ_k. Since each mean object is the size of its cluster and all the mean objects are compactly placed, the clusters are also compactly placed along with the objects within them and we therefore end up with a compact object cloud.



(a) 1 latent variable (b) 2 latent variables (c) 3 latent variables

Figure 4.2: A set of unobserved latent variables within an object cloud

Energy Minimization

In order to create a compact object cloud, we express the energy within a cloud as the sum of the inter-cluster energy, E_s, and the intra-cluster energy, E_r. This is expressed in equation 4.3. The inter-cluster energy is the energy between the mean objects, μ, for the latent variables while the intra-cluster energy is the energy between the objects in each cluster.

$$E = E_s + E_r \tag{4.3}$$

The intra-cluster energy is the sum of the energies within each cluster. We can minimize this energy by simply applying the energy minimization algorithms described in the previous chapter to each cluster. Similarly, we can apply the energy minimization algorithms to the mean objects, μ, for the latent variables so as to minimize the inter-cluster energy. Since the position of a given mean object μ_k is the center of the cluster for a latent variable, z_k, and its radius is the radius of the cluster, we can treat the mean objects as actual objects themselves and minimize the semantic energy between them. Once the inter-cluster energy has been minimized, we can move each cluster to its new position.

It is not necessary however to generate each cluster using energy minimization. After assigning objects to a latent variable, the objects for each cluster can simply be generated using other algorithms like Wordle etc. In that case, we can ignore the intra-cluster energy, E_r, and minimize the energy within the object cloud by simply minimizing the inter-cluster energy, E_s, so that the clusters are compactly arranged to form an object cloud.

If the intra-cluster energy is to be minimized using an energy minimization algorithm like majorization, so that we can do things like display an animation of the objects being clustered for a user's visual coherence [21], we can do so by expressing the weight, distance and laplacian matrices as block diagonal matrices. Doing so requires sorting the original configuration of points so that all objects that belong to the first latent variable are in the first set of rows for the new configuration followed by those of the second latent variable and so on. This new configuration is then used to create block matrices where the first block corresponds to objects for the first latent variable and so on. A mapping of the old configuration to the new configuration is kept so that afterwards the objects are placed in their correct positions for visualization.

Algorithm 4: Latent Clouds using Majorization

Data: Adjacency matrix, A, Pairwise distances, D^x, α, β, η
Result: A configuration of points, P
Initialize P randomly from $\mathcal{N}(0, 1)$;
$D^p = [d_{ij}]$, where $d_{ij} = \|p_i - p_j\|$;
$W \leftarrow A \circ (1 - (D^p < D^x))\beta + (D^p < D^x)\alpha$;
$f_0 \leftarrow \sum_{i<j} w_{ij}(D^x_{ij} - d_{ij})^2$;
$\Delta \leftarrow \infty$;
while $\Delta > \epsilon$ **do**
  $V \leftarrow (\sum_j w_{ij} \circ \mathbb{I}) - W$;
  $S \leftarrow D^x / D^p$;
  $B \leftarrow (\sum_j w_{ij} s_{ij} \circ \mathbb{I}) - (W \circ S)$;
  $V^+ \leftarrow (V + n^{-1}\mathbf{1}\mathbf{1}^T)^{-1} - n^{-1}\mathbf{1}\mathbf{1}^T$;
  $P \leftarrow V^+ BP$;
  $D^p = [d_{ij}]$;
  $W \leftarrow A \circ (1 - (D^p < D^x))\beta + (D^p < D^x)\alpha$;
  $f_1 \leftarrow \sum_{i<j} w_{ij}(D^x_{ij} - d_{ij})^2$;
  $\Delta \leftarrow f_0 - f_1$;
  $f_0 \leftarrow f_1$
end

If block diagonal matrices are used, minimizing the intra-cluster energy using majorization is similar in nature to minimizing the inter-cluster energy. The only difference between both is finding the pseudo-inverse, V^+, of the block laplacian matrix V. For the block laplacian matrix, we can express its pseudo inverse as

$$V^+ = \text{diag}(V_1^+, ..., V_k^+) \tag{4.4}$$

where, V_i^+, is the pseudo-inverse of an individual cluster which can be computed using

$$V_i^+ = (V_i + n_i^{-1}\mathbf{1}\mathbf{1}^T)^{-1} - n_i^{-1}\mathbf{1}\mathbf{1}^T \tag{4.5}$$

4.2 Experimental Evaluation

In this section, we analyze the performance of an object cloud that is generated with different numbers of latent variables. The performance of the cloud is measured using the metrics used for energy-based object clouds. These metrics include: energy, trustworthiness, realized adjacency and compactness. For our evaluation, we utilize the the Princeton Shape Benchmark and ModelNet40 which were used to evaluate energy-based object clouds. Our goal is to observe and quantify the effect that different latent variables have on an object cloud.

Data Preprocessing

For the experiment, we drew a 100 random objects from each of the dataset. Accordingly object clouds for 1 - 100 latent variables were generated and their performance on the respective metrics were recorded. This process was repeated for 10 rounds and the average results are reported.

Results

The first metric we discuss is the semantic energy of the object cloud. As we can see in figure 4.3, the semantic energy decreases as the number of latent variables within the cloud increases.

This is because the semantic graphical ordering of the objects is used to compute the energy within the cloud. Since the cloud however has a random order and most objects are

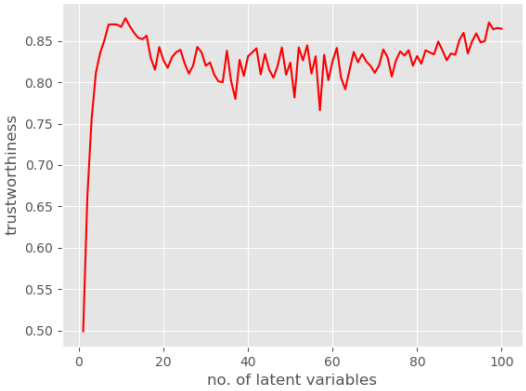
(a) Modelnet 40 (b) Princeton shape benchmark

Figure 4.3: Energy for different latent variables in a latent object cloud

not close to similar objects, this energy is high. This is best illustrated in figure 4.7 where similar objects like planes are scattered across the cloud. However as the random position of various objects decreases and similar objects get closer to each other, as is illustrated in figure 4.13, the final energy within each cloud of latent variables decreases.

Following the energy, we look at the trustworthiness of each object cloud that was generated by the latent variables. Since the ordering of the object cloud is random when there is a single latent variable, from figure 4.4 we can observe that the trustworthiness starts at a random value of 0.5. This value is equivalent to tossing a random coin. It however increases as the number of latent variables approaches 5 latent variables. This might be because, as illustrated in layouts of figures 4.8, 4.9, and 4.10, similar objects begin to adjoin each other in a way that reflects the local neighbourhood of their high-dimensional vectors.

Beyond 5 latent variables however, the trustworthiness reduces before once again increasing until we have a fully semantic object cloud. This is likely because as the number of latent variables increases, the number of objects within each cluster decreases and similar objects may be assigned to different clusters which are farther away, thereby reducing the trustworthiness. This is illustrated in figure 4.11 where there seem to be excess space amongst similar objects. However as the number of latent variables increases, the cluster sizes are reduced and are closer therefore similar objects that belong to different clusters become closer to each other and therefore increase the computed trustworthiness of the object cloud. This is illustrated in figure 4.12 where there is less space amongst similar



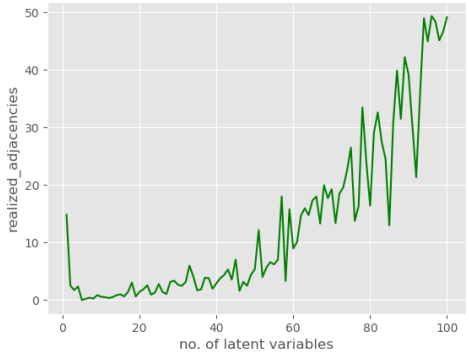
(a) Modelnet 40

(b) Princeton shape benchmark

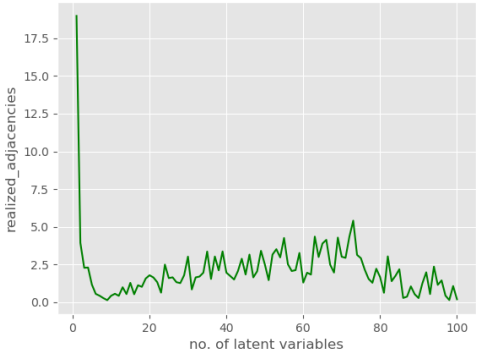
Figure 4.4: Trustworthiness for different latent variables in a latent object cloud

objects and better grouping.

Next we observe the realized adjacencies of each cloud generated by the various latent variables. As illustrated in the figure 4.5, as the number of latent variables increases, so does the realized adjacencies. This is because similar objects are increasingly placed next to each other as the random ordering decreases, thereby increasing the total similarity value of their adjacent objects.

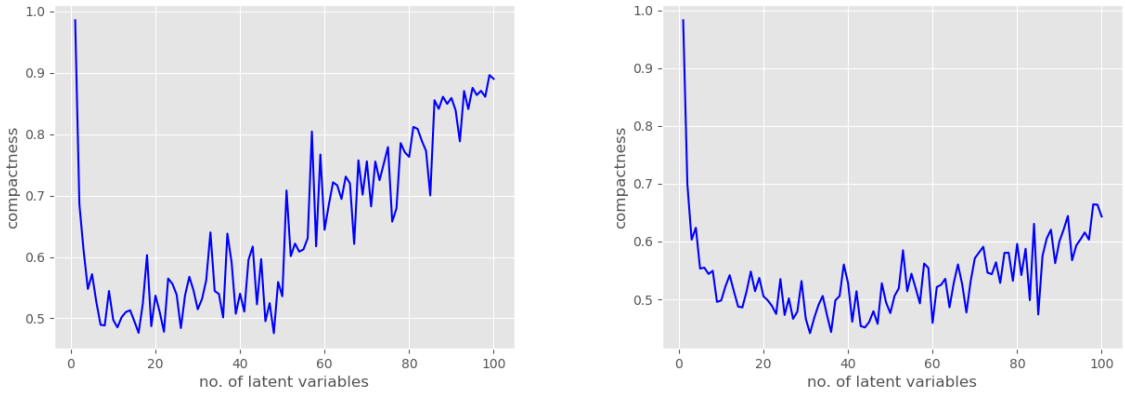


(a) Modelnet 40



(b) Princeton shape benchmark

Figure 4.5: Realized adjacencies for different latent variables in a latent object cloud



(a) Modelnet 40 (b) Princeton shape benchmark

Figure 4.6: Compactness for different latent variables in a latent object cloud

Finally we look at the compactness of each object cloud generated by the various latent variables. When there is a single latent variable, the objects within the cloud are compact, albeit randomly ordered. However once we introduce a few latent variables, the object cloud contains large clusters of objects that are quite separated and have some space between them. These large clusters cause the compactness to drop. As the number of latent variables decreases, the clusters become closer to each other and the compactness of the object cloud increases until we have roughly a single object per cluster. At this point the cloud is as compact as a semantic object cloud.

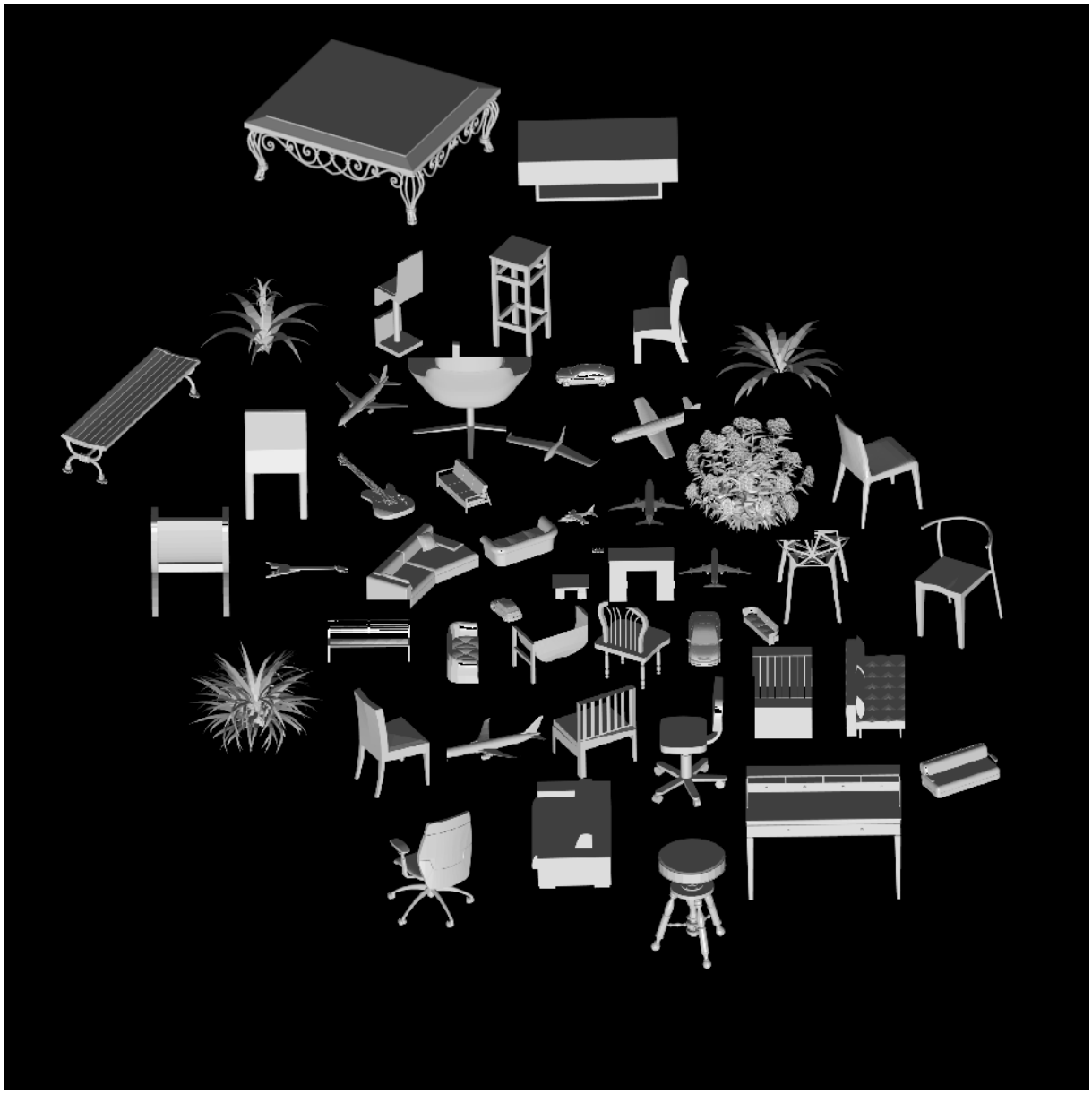


Figure 4.7: 1 latent variable on the ModelNet40 dataset. (no. objects = 49)

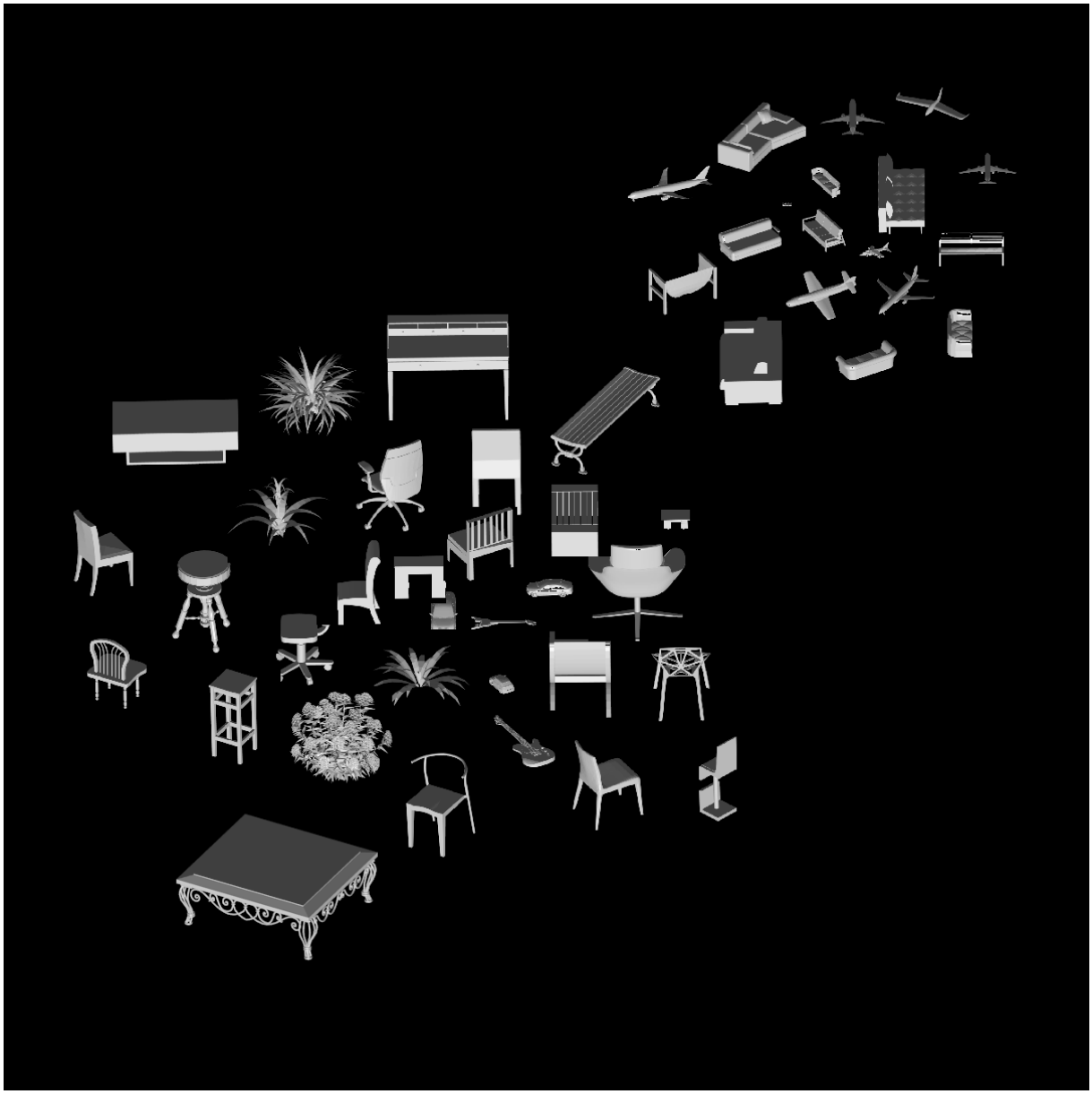


Figure 4.8: 2 latent variables on the ModelNet40 dataset. (no. objects = 49)

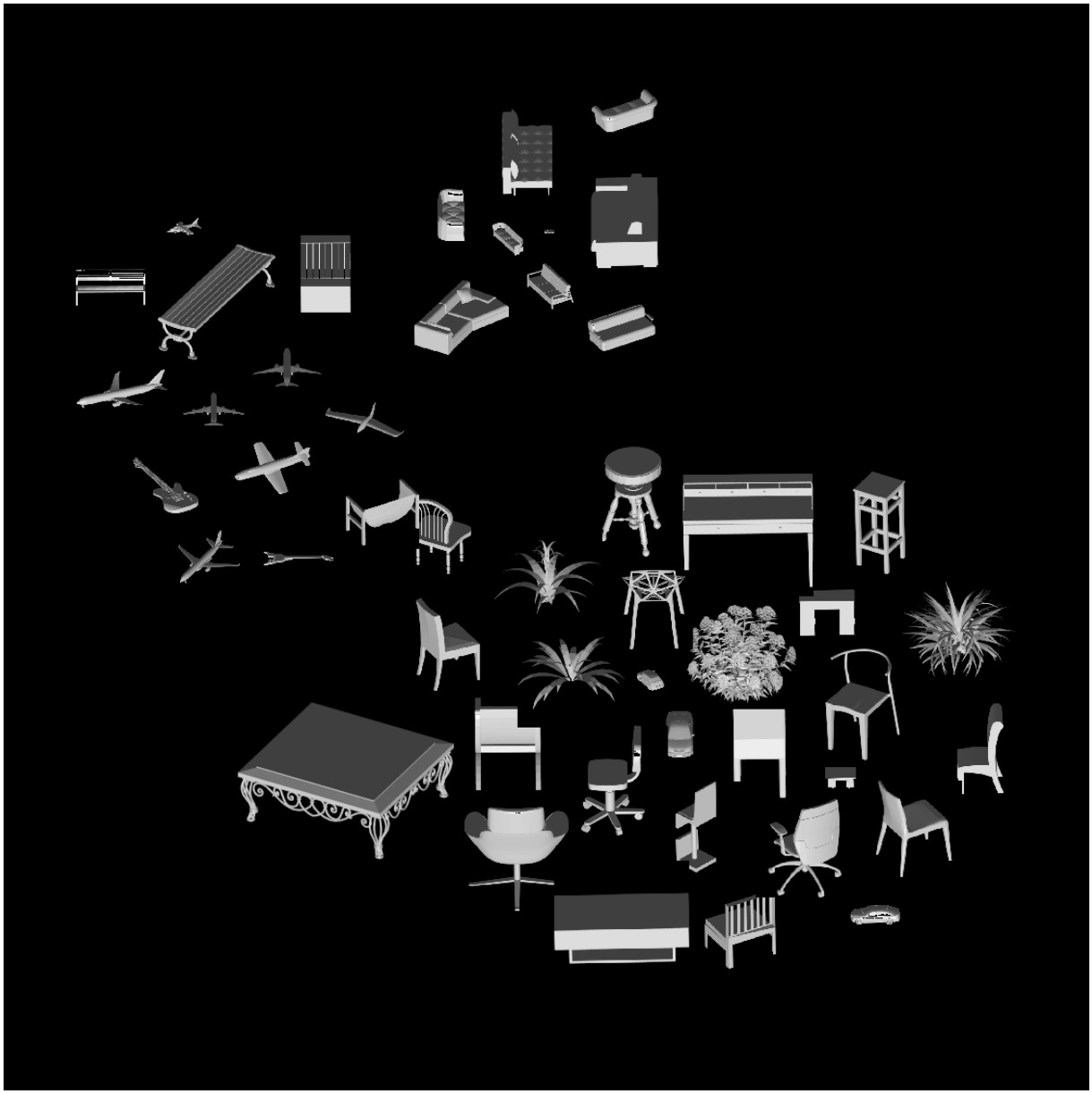


Figure 4.9: 3 latent variables on the ModelNet40 dataset. (no. objects = 49)

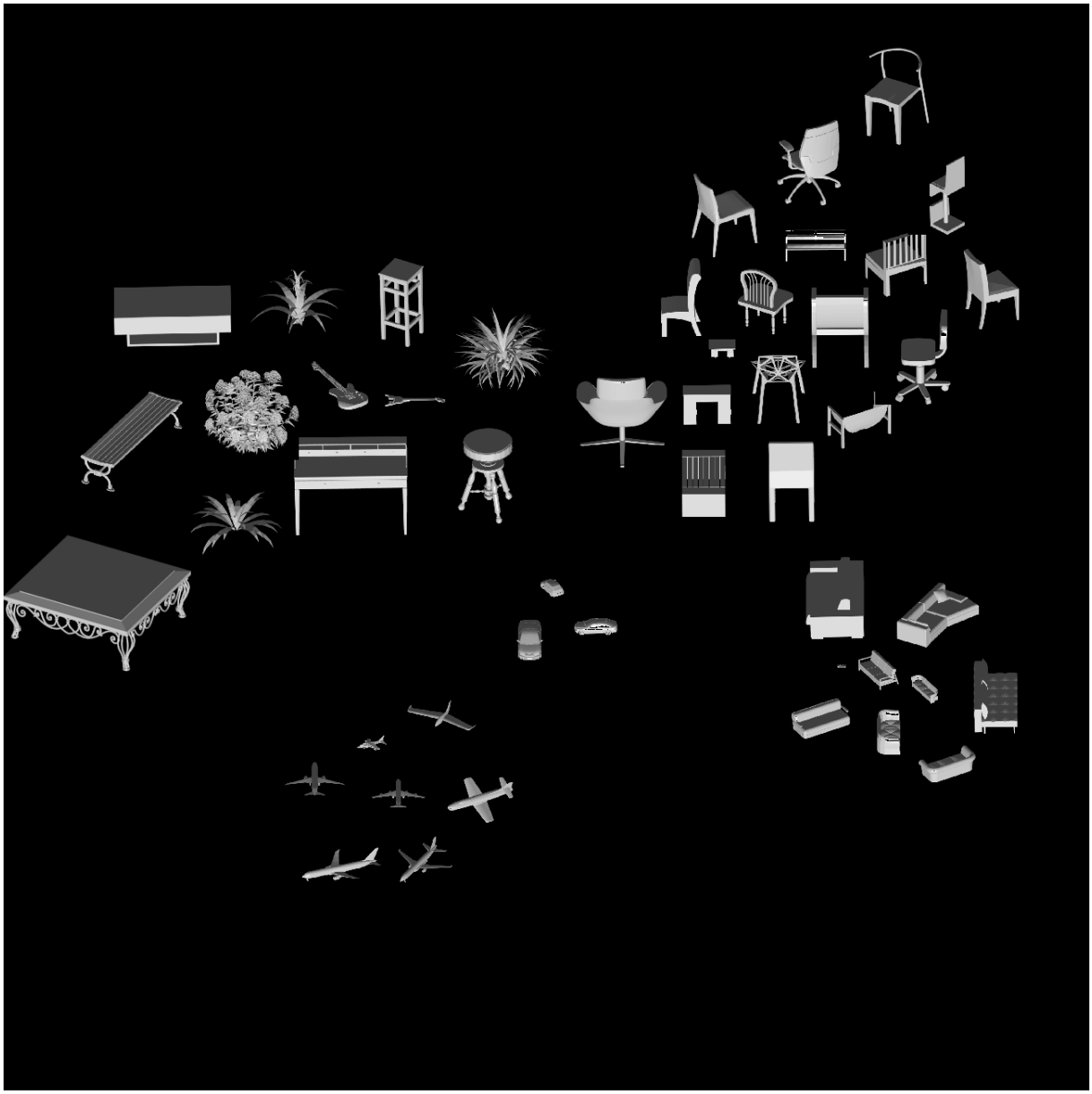


Figure 4.10: 5 latent variables on the ModelNet40 dataset. (no. objects = 49)

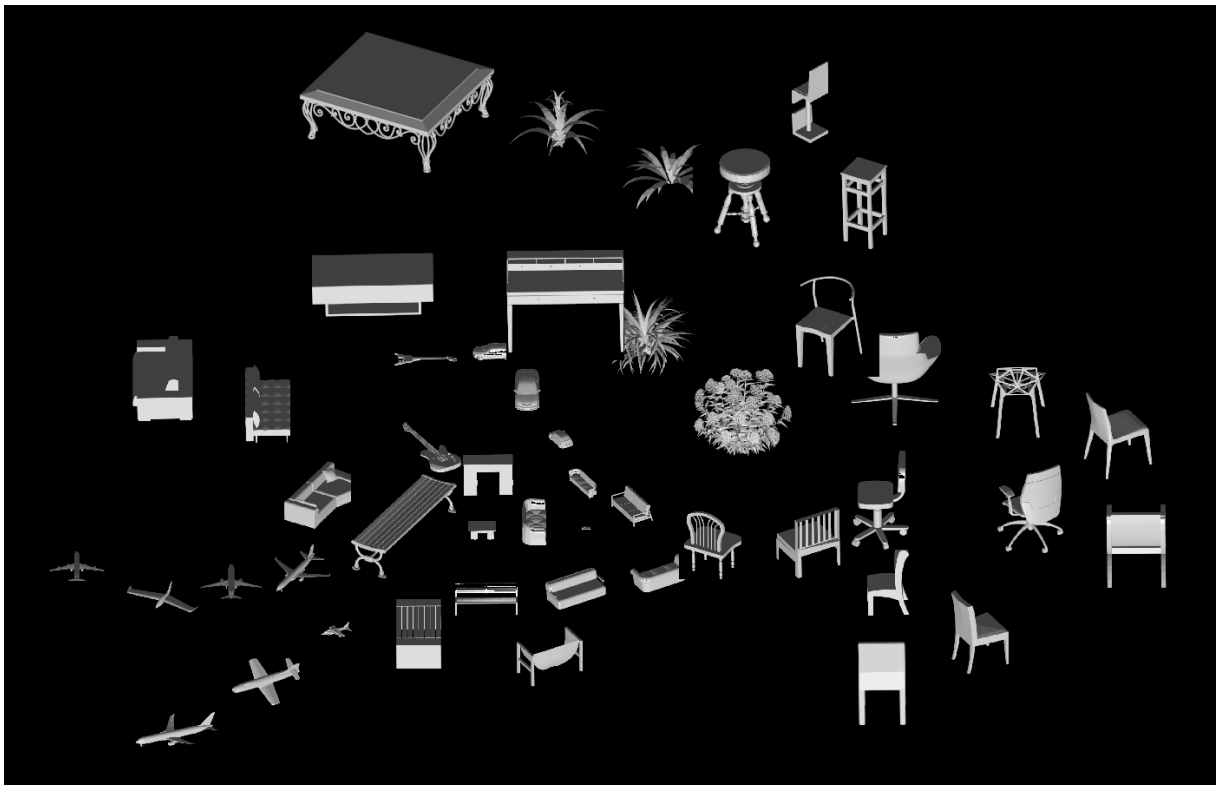


Figure 4.11: 25 latent variables on the ModelNet40 dataset. (no. objects = 49)

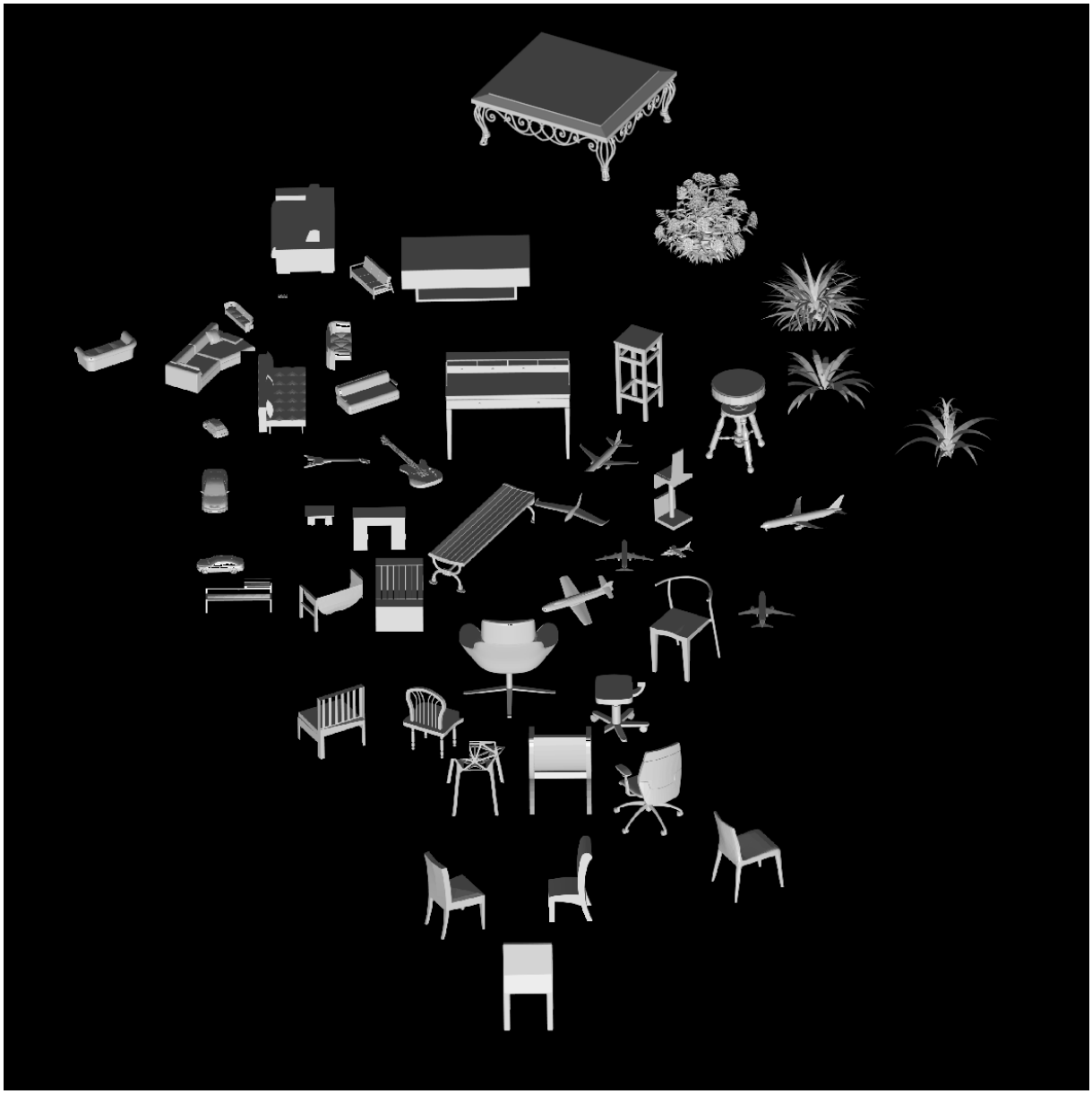


Figure 4.12: 35 latent variables on the ModelNet40 dataset. (no. objects = 49)

Figure 4.13: 49 latent variables on the ModelNet40 dataset. (no. objects = 49)

Chapter 5

Conclusion

In this chapter, we summarize the thesis, briefly discuss its limitations, and discuss potential directions for future work.

5.1 Summary

We proposed an energy function for object clouds. This energy function can be defined over a set of latent variables or over the set of objects within an object cloud. The number of latent variables indicate the degree of randomness within the cloud. If there is a single latent variable in the cloud, then the cloud will have a random layout. As the number of latent variables increases, so does the semantic order within the cloud. When the number of latent variables is equal to the number of number of objects in the cloud, then the cloud will have a fully semantic layout. Using the weighted squared distance between a set of objects or latent variables, the energy function encapsulates the compactness requirement for a cloud. It also encapsulates the semantic order by using a K-NN graph that is constructed from the high dimensional representation of the objects or latent variables.

By optimizing the energy function, we showed that we can create a semantic or random object cloud. This is due to the fact that we minimized the squared distance between similar objects in the K-NN graph while maximizing the squared distance between dissimilar objects in the graph. To that end, we proposed 3 optimization strategies: gradient descent, random replay and majorization. We discussed each of these strategies and their advantages over each other. Random replay is like gradient descent but it randomizes the order in which the objects are adjusted thereby allowing it to escape local minima that gradient

descent may get stuck in. Majorization however is a very different strategy in that all the objects are adjusted at the same time and the adjustment is based on the minimization of a surrogate for the proposed energy function. This allows it to construct object clouds faster and it decreases the energy function monotonically. Using the decreasing monotonicity, we can stop the construction of an object cloud whenever the energy function ceases to decrease rather than specifying a set number of iterations within which to minimize it.

From the minimization strategies, we proposed a set of algorithms for constructing object clouds. We then compared the layout from these algorithms against other the layout of algorithms that include the breadth-first search and context preserving algorithms. We used metrics such as trustworthiness, compactness, and realized adjacency to facilitate our evaluation. The metrics measure how faithful a layout is to the semantic order of the object representations, how compact a layout is and how close similar objects are to each other respectively. The optimization algorithms outperformed the other algorithms on trustworthiness due to the fact that less semantic information is lost when using a high dimensional graph as opposed to a lower dimensional graph like the Delaunay graph. In terms of compactness, both the context preserving layout and the optimization layouts were compact. The breadth-first search layout was not as compact because of its uniform use of layout space. Finally in terms of realized adjacency, the optimized layouts did outperform the other algorithms due to a combination of high semantic order and compactness. The context preserving algorithm did not perform as well because the semantic order of the layout was not as high and some objects were too close to be considered as touching each other. The breadth-first search in a similar vein did not perform as well because the space between the objects was too large for them to be considered as touching.

Finally we extended the energy function to word clouds and evaluated the algorithms against other algorithms like seam carving, Wordle and the context preserving algorithm. For the evaluation, only gradient descent was used because randomizing the order of the objects yielded no difference in result and majorization could not be sufficiently applied to word clouds to facilitate a monotonically decreasing energy. We evaluated the algorithms on metrics such as realized adjacency, compactness and energy. The gradient descent algorithm once again outperformed the other algorithms due to its high degree of semantic accuracy and optimized pairwise distance between neighbouring nodes. It also outperformed the other algorithm on the compactness metric because of the highly optimized pairwise distance.

5.2 Limitations and Future Work

Following this thesis, we have proposed an objective function that describes and quantifies the aesthetics of object clouds. We have also proposed a few algorithms for the generation of object and word clouds based on the optimization of the proposed function. Furthermore, we have evaluated and contextualized the performance of our algorithms in relation to other algorithms that are used to generate object and word clouds.

Despite this, there are some limitations to the proposed function and subsequent algorithms. One major problem is that a few objects still get occluded in the layouts for our proposed algorithms. We are uncertain as to why this happens but it may be because some objects have too many neighbours that end up colliding with each other in an attempt to minimize the energy. Finding a way to further reduce or limit the number of neighbours for each node in the high-dimensional K-NN graph may help alleviate this problem. The overlaps themselves could also represent a local minimum in the energy function. Currently our optimization methods still get stuck at some minimum in the energy function and so the energy value never reaches zero especially as the number of objects increases. A better understanding of the energy landscape or how to escape such local minima may also solve the problem of overlapping objects within a layout.

Although we use a variety of optimization methods to minimize the energy function, only gradient descent can be applied to word clouds. Techniques like majorization that monotonically decrease the energy value cannot be applied to word clouds. The energy value fluctuates and often ends up not converging. This is in part due to the non-uniform rectangular geometry of the words in a word cloud. The geometry makes it difficult to specify a fixed distance between any pair of words and so the distance between a given pair changes with their orientation from each other. This results in energy values that fluctuate and do not monotonically decrease. A surrogate function constructed with a more appropriate inequality may be able to decreasing the energy values monotonically, allowing for the technique to be adapted to word clouds.

Finally, a user study carried out on an AR device or on other devices with limited view ports, can help evaluate the efficacy of our algorithm on the recognition and exploratory search for 3D objects in a cloud on such displays. The study will likely highlight the usability of our approach and also provide both weaknesses as well as areas for improvement in our proposed visualization.

References

[1] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM Trans. Graph*, page 10. SIGGRAPH, 2007.

[2] Lukas Barth, Stephen G Kobourov, and Sergey Pupyrev. Experimental comparison of semantic word clouds. In *International Symposium on Experimental Algorithms*, pages 247–258. Springer, 2014.

[3] Grigory Begelman, Philipp Keller, Frank Smadja, et al. Automated tag clustering: Improving search and exploration in the tag space. In *collaborative web tagging workshop at WWW2006, Edinburgh, Scotland*, pages 15–33, 2006.

[4] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[5] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. *Foundations and Trends in Human-Computer Interaction*, pages 73–272, 2015.

[6] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[7] Léon Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the symposium on learning and data science, Paris*, 2009.

[8] Siddhartha Chaudhuri. Shape descriptors - iii. URL: `https://www.cse.iitb.ac.in/~cs749/spr2016/lecs/17_features.pdf`. Last accessed on 8 Sept 2020", Indian Institute of Technology Bombay, May 2016. CS749: Digital Geometry Processing.

[9] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003.

[10] Lisha Chen and Andreas Buja. Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *Journal of the American Statistical Association*, 104(485):209–219, 2009.

[11] Weiwei Cui, Yingcai Wu, Shixia Liu, Furu Wei, Michelle X Zhou, and Huamin Qu. Context preserving dynamic word cloud visualization. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pages 121–128. IEEE, 2010.

[12] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics (TOG)*, 15(4):301–331, 1996.

[13] Jan De Leeuw. Convergence of the majorization method for multidimensional scaling. *Journal of classification*, 5(2):163–180, 1988.

[14] Jan De Leeuw and Patrick Mair. Multidimensional scaling using majorization: Smacof in r. *Journal of Statistical Software*, 31(3), 2009.

[15] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[16] Emden R Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing*, pages 239–250. Springer, 2004.

[17] Hinton Geoffrey. Non-linear dimensionality reduction. URL: `https://www.cs.toronto.edu/~hinton/csc2535/notes/lec11new.pdf`. Last accessed on 8 Sept 2020", University of Toronto Dept of Computer Science, May 2013. CSC2535: Advanced Machine Learning.

[18] Antonio Gracia, Santiago González, Victor Robles, and Ernestina Menasalvas. A methodology to compare dimensionality reduction algorithms in terms of loss of quality. *Information Sciences*, 270:1–27, 2014.

[19] Jed Graef and Ian Spence. Using distance information in the design of large multidimensional scaling experiments. *Psychological Bulletin*, 86(1):60, 1979.

[20] Florian Heimerl, Steffen Lohmann, Simon Lange, and Thomas Ertl. Word cloud explorer: Text analytics based on word clouds. In *2014 47th Hawaii International Conference on System Sciences*, pages 1833–1842. IEEE, 2014.

[21] Xiaoting Hong and Stephen Brooks. 3d objects clouds: Viewing virtual objects in interactive clouds. *IEEE transactions on visualization and computer graphics*, 2018.

[22] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[23] Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[24] Stephen G Kobourov. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*, 2012.

[25] Alan J Lambie. *Directing Attention in an Augmented Reality Environment: An Attentional Tunneling Evaluation*. PhD thesis, Rochester Institute of Technology, 2015.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Martin Leginus, Peter Dolog, Ricardo Lage, and Frederico Durao. Methodologies for improved tag cloud generation with clustering. In *International Conference on Web Engineering*, pages 61–75. Springer, 2012.

[28] Weiquan Lu, Henry Been-Lirn Duh, Steven Feiner, and Qi Zhao. Attributes of subtle cues for facilitating visual search in augmented reality. *IEEE transactions on visualization and computer graphics*, 20(3):404–412, 2013.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] Kevin P Murphy. *A Probabilistic Perspective*. The MIT Press, 1 edition, 2012.

[31] Fernando V Paulovich, Franklina MB Toledo, Guilherme P Telles, Rosane Minghim, and Luis Gustavo Nonato. Semantic wordification of document collections. In *Computer Graphics Forum*, volume 31, pages 1145–1153. Wiley Online Library, 2012.

[32] KB Petersen, MS Pedersen, et al. The matrix cookbook, vol. 7. *Technical University of Denmark*, 15, 2008.

[33] Erich Schubert, Andreas Spitz, Michael Weiler, Johanna Geiß, and Michael Gertz. Semantic word clouds with background corpus normalization and t-distributed stochastic neighbor embedding. *arXiv preprint arXiv:1708.03569*, 2017.

[34] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The princeton shape benchmark. In *Proceedings Shape Modeling Applications*, pages 167–178. IEEE, 2004.

[35] James Sinclair and Michael Cardew-Hall. The folksonomy tag cloud: when is it useful? *Journal of Information Science*, 34(1):15–29, 2008.

[36] Gene Smith. *Tagging: people-powered metadata for the social web, safari*. New Riders, 2007.

[37] Julie Steele and Noah Iliinsky. *Beautiful visualization: Looking at data through the eyes of experts*, chapter 3, pages 37–58. " O'Reilly Media, Inc.", 2010.

[38] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

[39] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[40] Daniel Tunkelang, Daniel Sleator, Paul Heckbert, and Bruce Maggs. A numerical optimization approach to general graph drawing. Technical report, Carnegie-Mellon Univ Pittsburgh PA Dept Of Computer Science, 1999.

[41] Jarkko Venna and Samuel Kaski. Neighborhood preservation in nonlinear projection methods: An experimental study. In *International Conference on Artificial Neural Networks*, pages 485–491. Springer, 2001.

[42] Kateryna Vyshenska. How to build a parameterized archimedean spiral geometry, Jul 2016.

[43] Christopher D Wickens, Sallie E Gordon, Yili Liu, et al. *An introduction to human factors engineering*. Longman New York, 1 edition, 1998.

[44] Jeremy M Wolfe. Visual search. *The handbook of attention*, pages 27–56, 2015.

[45] Yingcai Wu, Thomas Provan, Furu Wei, Shixia Liu, and Kwan-Liu Ma. Semantic-preserving word clouds by seam carving. In *Computer Graphics Forum*, volume 30, pages 741–750. Wiley Online Library, 2011.

[46] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[47] Jin Xu, Yubo Tao, and Hai Lin. Semantic word cloud generation based on word embeddings. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 239–243. IEEE, 2016.

[48] Michelle Yeh and Christopher D Wickens. Visual search and target cueing: A comparison of head-mounted versus hand-held displays on the allocation of visual attention. Technical report, Army Research Lab Aberdeen Proving Ground MD, 1998.