Automatic Hierarchical Arrangement of Vector Designs

M. Fisher[1], V. Agarwal[2] and T. Beri[2]

[1]Adobe Research, USA [2]Adobe Inc., India

Abstract

We present a method that transforms an unstructured vector design into a logical hierarchy of groups of objects. Each group is a meaningful collection, formed by proximity in visual characteristics (like size, shape, color, etc.) and spatial location of objects and models the grouping principles designers use. We first simplify the input design by partially or completely flattening it and isolate duplicate geometries in the design (for example, repeating patterns due to copy and paste operations). Next we build the object containment hierarchy *by assigning objects that are wholly enclosed inside the geometry of other objects as children of the enclosing parent. In the final clustering phase, we use agglomerative clustering to obtain a bottom-up hierarchical grouping of all objects by comparing and ranking all pairs of objects according to visual and spatial characteristics. Spatial proximity segregates far apart objects, but when they are identical (or near identical) designers generally prefer to keep (and edit) them together. To accommodate this, we detect near identical objects and group them together during clustering despite their spatial separation. We further restrict group formation so that z-order disturbances in the design keep the visual appearance unaffected for tightly-overlapping geometry. The generated organization is equivalent to the original design and the organization results are used to facilitate abstract navigation (hierarchical, lateral or near similar) and selections in the design. Our technique works well with a variety of input designs with commonly identifiable objects and structural patterns.*

CCS Concepts

• ***Applied computing*** → ***Document analysis;*** • ***Information systems*** → *Clustering;*

1 Introduction and Related Work

Modern day vector designs are complex and are built by putting together several smaller artwork objects. The constituent artwork objects are sometimes created from scratch (in vector editing tools like Adobe Illustrator), while at other times they are imported from external sources (e.g. SVG, EPS, PDF, etc. exported from tools like PowerPoint). These often lack a structural arrangement of individual vector objects that comprise the imported graphic or artwork components. Collaborative environments offered by modern vector graphic tools allow several designers to work together to produce various pieces of an artwork that are assembled later. This can further complicate the organization and sense in a design which makes it hard to select or manipulate objects and object collections.

Several hierarchical clustering mechanisms have been proposed in the literature. The two widely adopted ones are *agglomerative* [ZMRA13] and *divisive* [Rou15]. While the former performs a bottom-up merger of objects, the latter starts from the top and recursively splits down the hierarchy. However, these methods have been mostly used for statistical inferences or data mining. Several challenges prevent their direct application to vector graphics. First, artists can structure the same vector graphics document in many different ways. The objects are free-form and largely a reflection of designer's creativity. This requires specialized similarity analysis for objects. Second, the definition of similarity is different for different designers - some prefer closely placed objects to be grouped while others like objects with similar color to get higher emphasis while clustering. Third, free movement of vector objects up and down the hierarchy must be restricted or it may visually alter the artwork. Re-arrangement of an object is only possible if that does not impact its visual z-order with respect to others in the design.

Structural, regularity and similarity analysis of graphical patterns has been extensively studied and used for clustering. Color contrast has been used to detect salient regions in images [CMH*14]. Symmetry detection has been proposed to segment images into similar local regions [BSGF10, LSS*17, CMZP14, KR91]. Affinity based methods work with boundaries and edges in natural images [AMFM11, ZD14]. In [LZH*17], the authors propose a deep learning based method, where two tiers of the network respectively focus on local neighborhoods and global structures appearing in an image. The network requires a decomposition of input image into known atomic elements, and a post-processing phase uses the encoded descriptors to create clusters of these elements. Results show reasonably accurate groupings of non-organic atomic elements. In vector graphic designs, however, there is inherent knowledge of objects and object boundaries. There also are designer created visual characteristics like stroke weight (and color), and peripheral dis-

DOI: 10.2312/egs.20211016

tance between objects. This data often provides a more precise insight into designer's perception of inter-object co-relation as compared to deducing the same information from their derived rasters.

Thus, we present an orchestration of *agglomerative* clustering that makes it suitable for vector graphics. Instead of inferring structural similarities, we work with the inherent visual and spatial characteristics in the design. We also introduce *containment* and *chain reaction*, and augment clustering with *constrained movement* of objects (owing to z-order restrictions). *Containment* builds logical segregation in the design allowing different similarity metrics (color, shape, size, etc.) to dominate in different parts of the same design. While spatially far apart objects are generally meant to be kept in different clusters (they have a low score on placement metric), but nearly identical objects like keys of a keyboard should be placed together despite the distance between extreme keys. So, we build logical chains of near identical objects and any object of the chain while being placed inside a cluster pulls the entire chain along (*chain reaction*). Finally, visual bounds of objects are cross-checked with others above or below it and clustering is disallowed if an overlapping object at the front would go back or vice-versa.

2 Our Solution

We present an agglomerative clustering technique that determines designer's intent in a given vector design and re-organizes artwork objects into a meaningful hierarchy of groups. Our solution operates in three phases: *pre-processing*, *containment* and *clustering*.

2.1 Pre-processing

Vector designs exist in many forms. Some are partially organized while others need complete rearrangement. We provide three options (via a user interface) to the designers: *retain existing groups*, *retain named groups* and *no retention*. The first option does not disturb any existing groups, i.e. no clustering is performed inside existing groups (and their internal organization is preserved verbatim), but they participate normally in cluster formation up the hierarchy. For the rest of this text, we refer to these as *complete groups*. The second option treats explicitly named user groups as *complete groups*, but the unnamed ones are flattened and their content is considered for clustering. The last option expands the entire design and clustering is executed on a flat initial arrangement of vector arts.

Some content may frequently recur in a design and be a natural candidate for grouping. We process such content by detecting duplicate geometries in the design and placing them into organized *complete groups*. Thus, all duplicates can reuse the same internal organization. Note that duplicates are not just exact replicas but also affine transformations (i.e., rotation, scaling and translation) of replicas. For our solution, we use the method from [DBPD20].

2.2 Containment

Often, a designer visually places objects within others. An example is a face containing eyes and mouth. Even though eyes do not have the same color or shape as mouth, the designer expects to put them all under the group for face. In these cases, visual similarity like color and shape may take a back seat to the shape hierarchy. To make this goal concrete, we introduce the concept of *containment*, where we find objects which are confined within visual bounds of larger objects. We create a special group (called a *containment group*), which is an aggregation of the larger object and the smaller ones it contains. For example, in the artwork of a face, the *containment group* contains the face, both eyes and the mouth.

In the *containment* phase, we make a single bottom-up pass on the entire design (in the initial designer arrangement) and create *containment groups* for the objects fully contained inside others. These groups can be hierarchical, in case a designer creates multiple levels of objects that are visually confined within the bounds of others. We also create a root *containment group*, sitting at the top of the design, that encloses all other *containment groups* and independent un-contained objects. *Containment groups* differ from *complete groups* as we look inside them and perform clustering (as defined in section 2.3). However, when an organized *containment group* is considered in its parent, it is treated as a *complete group*.

Each containment group is isolated from others and thus different basis of internal organization can be used for them. One *containment group* can be organized on color similarity while the other on the basis of shape similarity. This gives local influence to our system and is synonymous to how people observe similarity.

2.3 Clustering

Agglomerative clustering [ZMRA13] is the final phase of our solution. At the start of this phase, all objects lie inside *containment groups* (*root containment group* or a nested one). As this phase progresses, more groups are formed on the basis of object similarity.

Usually, *agglomerative clustering* is performed bottom-up among all available objects. However, that approach does not generate desired results for vector graphics. This is because vector designs widely vary and no single clustering criterion works for all. Even within one design, different areas focus on different aspects. Some areas exhibit excessive co-relation (among objects) in color, while others have high co-relation in shape and size. Thus, we run multiple parallel instances of bottom-up agglomerative clustering (each limited to a *containment group*) and allow each instance to figure out its own criterion for clustering.

For better understanding, we classify the objects as following -

1. **Complete groups**: These are the groups generated by the *pre-processing* phase. They are either a result of *duplicates isolation* or are chosen to be so by the *designer*. No clustering is performed inside them. But they are treated as pre-defined *clusters*.
2. **Containment groups**: These groups are produced by the *containment* phase. They contain other objects (all four types) that lie within their visual bounds. Clustering is performed within these groups only and a hierarchy of sub-groups (called *clusters*) may be produced.
3. **Independents**: These are non-group objects that lie within *containment groups*. Examples of such objects are images, Bezier paths and text.
4. **Clusters**: These are the groups that are produced during clustering. These are essentially an organization of the four object types described here.

Within each *containment group*, we quantify and compare every pair of objects (*groups* and *independents*) on a common feature space of important vector descriptors (*color*, *stroke*, *size*, *shape* and *spatial placement*). Two objects are considered at a time and a normalized comparison score (in the range $[0-1]$) is computed for each of these descriptors. A weighted average of these descriptor

scores gives the final comparison score of the object-pair (section 2.3.1). Once the comparison data of all object-pairs is produced, this similarity information is used to form clusters (section 2.3.2).

2.3.1 Object Similarity Evaluation

Object similarity is an important criterion for designers to group objects together. Similarity can be perceived from color, or size, or any other visual attribute. In our study of vector designs, we find that most designers like to organize content with one or more of the following vector descriptors - *color*, *stroke*, *size*, *shape*, and *spatial placement*. While *color* refers to the fill of a vector object, *stroke* is multi-dimensional and encapsulates parameters like *stroke-width*, *line-cap*, *line-join*, *dash-array* and *stroke-color*. *Size* is the area of a two-dimensional vector object, *shape* is an impression of the outer periphery of a vector object and *spatial placement* refers to how close two objects are (in a design) relative to other object-pairs. For brevity, we limit the discussion on choice of these descriptors and their evaluation formulae, but those can be found in the accompanying supplemental material.

The scores produced by all our descriptors lie in $[0-1]$. However, the range of these descriptors depends upon object properties and thus could differ i.e. scores from one descriptor may be limited to the range $[0.5-0.7]$ while another may vary from $[0.1-0.9]$. This imbalance has the potential to influence rest of the algorithm and bias the output towards descriptors with higher range. Thus, we bring all descriptors to the same scale by normalizing their scores and rescaling their ranges to $[0-1]$. We use *Min-Max Normalization* for all our descriptors. It is given by the following formula -

$$S_k = \frac{U_k - \min_{\forall k} U_k}{\max_{\forall k} U_k - \min_{\forall k} U_k}$$

where, U and S respectively represent un-normalized and normalized score. k indexes all scores of a descriptor.

Our score computation is symmetric. In other words, all descriptors return the same result for object-pair (o_i, o_j) and its inverse object-pair (o_j, o_i). Thus, we use an upper diagonal matrix to keep scores (for all object-pairs), and save memory and evaluation time by computing just one permutation for every pair of objects. The final comparison score (G_{o_i,o_j}), of a pair of objects (o_i, o_j) is computed as a weighted average of their normalized descriptor scores:

$$G_{o_i,o_j} = \frac{\sum_{n=1}^{5} W_n * S_{n_{o_i,o_j}}}{\sum_{n=1}^{5} W_n}$$

where W_n is the weight of n^{th} descriptor and $S_{n_{o_i,o_j}}$ is the normalized score of object-pair (o_i, o_j) on that descriptor. n indexes the array of descriptors *[Color, Stroke, Size, Shape, Spatial Placement]*. In practice, we also optionally allow artists to adjust or disable the per-parameter weight for each category, allowing them to control the types of groups that are created by our method.

For computing descriptor weights, we statistically analyze un-normalized scores (i.e., before min-max normalization) for each descriptor (over all pairs of objects) and find the *variance* within each descriptor. Recall that these scores lie in $[0-1]$. Weights are influenced relative to this variance – for example, if every object (within a *containment group*) is colored blue, then we discard the color when determining object similarity. To account for this, we keep a threshold and if a descriptor's variance is less than the threshold, its observed weight is set to *zero*. In our implementation, we suppress any descriptor with less than 15% variance. The weight of a descriptor (W) is given by the following formula -

$$W = \begin{cases} 0, & \text{if } \max_{\forall k} \mathrm{U_k} - \min_{\forall k} \mathrm{U_k} \leq 0.15 \\ 1, & \text{otherwise} \end{cases}$$

The above text describes how descriptors are evaluated for a pair of *independent* objects. In case, one or both of the objects in the pair are *groups*, we do the following -

1. for *complete groups*: these groups are treated as images and thus rasterized. All descriptors are evaluated by assuming these groups to be *independent* raster objects.
2. for *containment groups*: *stroke*, *size*, *shape*, *spatial placement* of the group are computed from the containing object of the group. On the other hand, *color* is computed from the group as a whole by rasterizing it into an image.
3. for *clusters*: a *single linkage* paradigm is used, which means that the score of the group is same as the score of the child that yields maximum similarity. We experimented with both single and complete linkage and empirically found that single linkage tends to produce more salient clusters.

2.3.2 Grouping Similar Objects

Once the scores are computed, the next step is to insert all object-pairs in decreasing order of their scores into a priority queue. At the start of clustering, the priority queue contains *independents*, *complete groups* or *containment groups*. As we make progress with this algorithm, we form *clusters* of these objects and they also enter as object-pairs in this priority queue. The complete clustering algorithm is described below -

1. Pop the top element off the queue, which corresponds to the pair of objects that have maximum similarity.
2. Two possibilities exist depending upon the type of objects in the object-pair popped off the queue -
   a. both are *independents/groups* or both are *clusters*: If the objects appear next to each other (in the z-order defined by the designer), a new *cluster* is created and both the objects are put inside that (maintaining their relative z-order). Otherwise, we can not do this clustering as this could alter designer's z-order and change appearance of the design. Thus, we try to re-arrange the objects such that they appear next to each other in z-order and the design remains visually unchanged. If the re-arrangement is feasible, both objects are *clustered*. Otherwise, this clustering is denied and the algorithm moves to step 5 to find the next available object-pair. The re-arrangement process is described after this algorithm.
   b. one is a *cluster* and the other is an *independent/group*: A recursive lookup is performed inside the *cluster* to find the object with maximum similarity (i.e., *single linkage*) with the input *independent/group* object. The objective is to move the *independent/group* object next to the maximum similarity object. However, re-arrangement restrictions apply in the same way as described for the previous case. Thus, the maximum similarity object that can become a sibling without disturbing design's visual perception is found and used.
3. In case a new *cluster* is formed (or an object is moved inside an existing *cluster* or clustering is denied) in the last step, we need

to update the upper diagonal matrix (storing similarity scores for object-pairs) and reflect the update in the priority queue by removing the rows/columns of all objects from the matrix and inserting the new *cluster* in the matrix. Scores of all other objects are computed with respect to this new *cluster* and the corresponding matrix entries updated.

4. If an object is moved inside a cluster, we perform *chain reaction* by finding other nearly similar objects (i.e. more than 95% similar on all metrics except distance) and moving them into the same cluster. The motivation here is that designers prefer far away alike objects (that otherwise may have low similarity score owing to distance) to be clustered together. The process is repeated for newly found objects till no more are near similar.
5. Repeatedly pop object-pairs off the queue until it is empty or all remaining object-pairs are denied *clustering*. This can happen due to re-arrangement limitations or if the similarity score between top object-pair is too low (defined as 30% of the maximum similarity score at the start of this algorithm).

Objects that are not placed next to each other need to be re-arranged before they are allowed to form a *cluster* (step 2 above). However, objects can't freely move up and down the hierarchy as that would bring backward objects up-front or vice-versa, thereby changing the visual appearance of the design. Thus, while re-arranging objects A and B, we use the following approach -

1. Move object A towards B: Hop over one intermediate (between A and B) object C at a time. If there is no overlap between A and C, swap them. Continue doing so, till we reach B. In this case, re-arrangement is successful and complete. Otherwise, keeping A at its new location (i.e., just before C) go to the next step.
2. Move object B towards A: Start moving B towards C (from the last step), hopping over one intermediate object at a time. If B is able to swap itself with all intermediates and reaches next to A, re-arrangement is declared successful. Otherwise, it fails.

We evaluate our solution on a variety of vector designs created by different designers. An example is shown in figure 1, where we start with a flat unorganized design and form meaningful clusters for paper bags, bowls, hats, plates, and other objects. Further details on this and additional example designs with corresponding generated arrangements can be found in the supplemental material.

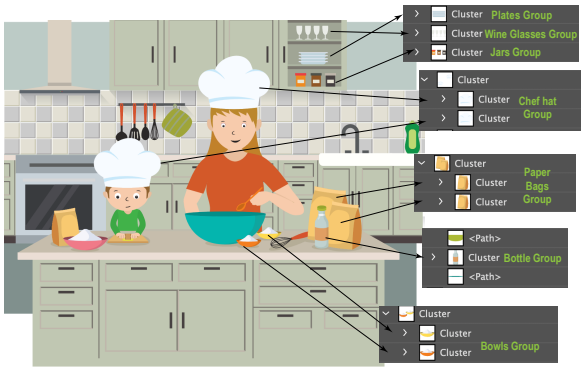

Figure 1: *Clusters created by our algorithm (for the shown artwork) are in dark grey boxes with green labels and black pointers.*

In addition to automatically creating object arrangements, our method can be used to build advanced navigation through designs. We can execute clustering in the background and allow designers to use left/right arrow keys to iterate through sibling objects (keys in the keyboard in figure 1), or up/down arrow keys to traverse through parent/children (keys, keyboard, desk, etc. in figure 1). The ability to enable such simple navigation even when it is not present in the original structure of a vector graphics file is a key application of our automatic clustering approach. Additionally, our method can be used for similarity detection and collectively editing similar objects post selection. A designer can select (and subsequently edit) similar objects through a proximity slider (for color, shape, size, etc.) that specifies permissible variation in a metric.

3 Conclusions

We present an orchestration of *hierarchical agglomerative clustering*, adapting the standard algorithm to vector graphics clustering. We introduce *containment* and *chain reaction* to make our algorithm more robust and sensitive to the needs of vector graphics. Our algorithm dynamically adapts the clustering criterion, allowing different portions of the vector design to cluster on different aspects of similarity, i.e. one part of the design may group objects on the basis of their color proximity while the other part reflects emphasis on shape or size. Our method can also be used for proximate selection, and advanced navigation through objects in the design.

References

[AMFM11] ARBELAEZ P., MAIRE M., FOWLKES C., MALIK J.: Contour detection and hierarchical image segmentation. URL: https://doi.org/10.1109/TPAMI.2010.161. 1

[BSGF10] BARNES C., SHECHTMAN E., GOLDMAN D. B., FINKELSTEIN A.: The generalized patchmatch correspondence algorithm. In *Proceedings of ECCV 2010: Part III* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, p. 29–43. 1

[CMH*14] CHENG M.-M., MITRA N. J., HUANG X., TORR P. H., HU S.-M.: Global contrast based salient region detection. *IEEE transactions on pattern analysis and machine intelligence 37*, 3 (2014), 569–582. 1

[CMZP14] CEYLAN D., MITRA N. J., ZHENG Y., PAULY M.: Coupled structure-from-motion and 3d symmetry detection for urban facades. URL: https://doi.org/10.1145/2517348. 1

[DBPD20] DHANUKA P. K., BATRA V., PHOGAT A., DHINGRA S.: Automatic vector graphic organization and asset extraction. In *ACM SIGGRAPH 2020 Posters* (2020), SIGGRAPH '20, Association for Computing Machinery. URL: https://doi.org/10.1145/3388770.3407427, doi:10.1145/3388770.3407427. 2

[KR91] KAHNG A. B., ROBINS G.: Optimal algorithms for extracting spatial regularity in images. *Pattern Recognition Letters 12*, 12 (1991), 757–764. 1

[LSS*17] LUKÁČ M., SÝKORA D., SUNKAVALLI K., SHECHTMAN E., JAMRIŠKA O., CARR N., PAJDLA T.: Nautilus: Recovering regional symmetry transformations for image editing. URL: https://doi.org/10.1145/3072959.3073661. 1

[LZH*17] LUN Z., ZOU C., HUANG H., KALOGERAKIS E., TAN P., CANI M.-P., ZHANG H.: Learning to group discrete graphical patterns. URL: https://doi.org/10.1145/3130800.3130841. 1

[Rou15] ROUX M.: A comparative study of divisive hierarchical clustering algorithms. *CoRR abs/1506.08977* (2015). URL: http://arxiv.org/abs/1506.08977, arXiv:1506.08977. 1

[ZD14] ZITNICK C. L., DOLLÁR P.: Edge boxes: Locating object proposals from edges. In *Computer Vision – ECCV 2014* (Cham, 2014), Fleet D., Pajdla T., Schiele B., Tuytelaars T., (Eds.), Springer International Publishing, pp. 391–405. 1

[ZMRA13] ZEPEDA-MENDOZA M. L., RESENDIS-ANTONIO O.: *Hierarchical Agglomerative Clustering*. Springer New York, New York, NY, 2013, pp. 886–887. URL: https://doi.org/10.1007/978-1-4419-9863-7_1371. 1, 2