SIMILARITY-BASED IMAGE ORGANIZATION AND BROWSING

GRANT STRONG

Similarity-Based Image Organization and Browsing

by

© Grant Strong

A thesis submitted to the

School of Graduate Studies

in partial fulfillment of the
requirements for the degree of

Master of Science

Department of Computer Science
Memorial University of Newfoundland

September 28, 2009

St. John's Newfoundland

Abstract

Users do not always know what they want, in which case traditional query-based image retrieval approaches fail. This thesis serves to amend this shortcoming with a novel approach to organize and browse large image collections based on visual similarities in a way that extends the user's natural search sense. Starting with an unordered set of images, salient color and gradient information are extracted into feature vectors. A self organizing map (SOM) then projects these high-dimensional vectors onto a 2D canvas so that similar ones are grouped together. When browsing around on the canvas through intuitive operations like pan and zoom, a dynamic collage is generated that shows the most pertinent images. To make organizing larger image collections practical, a parallel SOM training algorithm is designed that runs on graphics processing units. The results of using a variety feature vectors are also evaluated.

Acknowledgements

First and foremost I would like to thank my supervisor Dr. Minglun Gong for taking me under his wing and giving me his unwavering support, patient help, and sagacious advice. NSERC deserves some credit for his financial funding during my time at work on this project. I extend my gratitude to Dr. Wolfgang Banzhaf for loaning his graphics hardware in the early stages of this research. I must give a nod to Memorial University, its School of Graduate Studies, and its Faculty of Science for their tireless commitment to the graduate students of this institution, not to mention my trip funding. Last but not least, no acknowledgement would be complete without some praise for Memorial's Department of Computer Science and each and every faculty, student, and staff member in it for handling my pedantic concerns and network crises with the utmost professionalism and their genuine interest in and appreciation of my work; and for accepting me to come and be a part of them as a graduate student in the first place. To everyone I have had the pleasure of meeting and getting to know over the past two years as a result of this undertaking, I salute you.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

2D	Two Dimensional
BMU	Best Match Unit
CBIR	Content-Based Image Retrieval
CPU	Central Processing Unit
GPU	Graphics Processing Unit
QBE	Query by Example
QBIB	Query by Image Content
SBIB	Similarity-Based Image Browsing
SOM	Self Organizing Map

Chapter 1 Introduction

Since the dawn of the digital camera photos have gotten easier and cheaper to take, store, and share. One of the primary reasons many consumers own a computer is to manage the photos they take. As a result of this, the amount of visual data circulating cyberspace is too at an all time high. Massive online databases of community photos are cropping up constantly.

Due to the proliferation of all of this visual content in the past decade or so there became a need to be able search that content not with text, but rather by submitting the content the user had to get more of it; as a result content-based image retrieval was born. Up until that point users would query against metadata they or others had created describing the available images, be it filenames, directories, and/or keywords, and then cycle through the results in sequence. Accompanying thumbnails are provided in most cases but this visual aid is usually a secondary means of narrowing the search. People in the field were tired of trading apples for oranges and started to design new ways of capturing the essence of images discretely so they could then use that captured data in systems that would enable users to find what they wanted, when they wanted, the way they wanted. Some systems were completely automated, some required search-time user feedback, but on the whole they all addressed the issue as a query-to-results problem. The people behind such systems posited that their users, like their text searching counterparts, knew what they were in search of. Though sound, this postulate overlooks one indelible nuance of human behavior, impulse. When we go to the supermarket on an empty stomach and leave with the cart twice as full as we should, this is impulse. When we decide we are going to lose 50 pounds, likely due to our supermarket buying habits, and subsequently spend a small fortune on a yearlong gym membership only to trade the rowing

machine for the remote in a week, this is impulse. It should not come as a surprise then, that when users jump into their favorite collection of photos or onto their favorite social networking site that they may not have a method to their madness, it may simply be impulse. Maybe we could be presenting all of this visual content in an equally impulsive, but deceptively useful, way. The human mind is capable of processing quite a bit of visual stimulus extremely quickly; irritating case in point: channel surfing. Instead of forcing it to explore a collection one image at a time, why not give users as much as we can and let them survey it in an instinctual way that will open their mind to what they might find? This philosophy is at the core of similarity-based image organization and browsing. This thesis is the consolidation and continuation of the work on that subject first presented in [1, 2].

The pages that follow describe in detail how to compare images, how to organize images in 2D based on those comparisons, and how to facilitate effective browsing of the organized images, notably large numbers of such images. Chapter 2 reviews related work. Chapter 3 shows how color-based and/or gradient-based feature vectors are extracted from the images. Chapter 4 solves the following 2D image organization problem:

> Given a set of images, T, assign a 2D coordinate (x_I, y_I) to each image I, where $(I \in T)$, such that:
>
> 1. The distance between two visually similar images is as small as possible
> 2. Images are evenly distributed to make full use of the available space

It does so using the aforementioned feature vectors to train a self organizing map, which, by its topology preserving nature, prompts similar vectors to congregate and dissimilar ones to be

spread out. To make the organizing of large sets practical, a training algorithm is designed that runs in parallel on graphics processing units. The organizing process maps images with distinct feature vectors to unique positions on a 2D virtual canvas. Displaying all images at their mapped locations groups images by visual similarities but may overwhelm the users when the number of images is large. Chapter 5 addresses this by taking the image organization results as input and dynamically generating an image collage that always displays the most pertinent set of images based on which portion of the 2D canvas is currently being viewed. The interface to manipulate the collage allows for the use of simple well known operations like pan and zoom. The effects of the interface operations tailor to the natural inclinations users have when they browse and search. After presenting this organization and browsing solution, Chapter 6 goes over different organizational results obtained using a variety of types of feature vectors by evaluating them using new metrics specifically designed to target the image organization problem presented above.

Solving the 2D image organization problem described above has important and practical applications. For instance, operating systems can adopt this technique as an option to arrange images spatially in a folder in 2D by their visual contents. Online photo sharing sites and image search engines can also organize images in a similar interface by their content similarities so that the users can easily locate the images they have in mind in a more tactile environment. It has been found that users find such an environment enjoyable and useful [3]. In summary, the following lists the contributions of this thesis that facilitate a solution of the described problem:

1. A way to organize images in 2D based on similarity using a SOM
2. A survey and comparison of old and original feature vectors that can be used for

computing similarity between images

3. A fast SOM training algorithm for the GPU is proposed and shown to run tens of times faster than the standard training method carried out on a CPU
4. A flexible way of browsing large amounts of images within the confines of limited screen space

Chapter 2 Related Work

The previous works in the following areas are related to this thesis; how the works are similar to, different from, or have influenced this thesis is discussed.

2.1 Content-Based Image Retrieval

Content-based image retrieval (CBIR) has been an active research area in the past decade. Based on the user interaction required, existing "pure" CBIR approaches rely only on image content and can be classified into two categories: query by example and relevance feedback. Today however, researchers are no longer ignoring the already wide array of search tools available for and textual metadata attached to images found on the web. Emergent so-called "composite" CBIR systems [4] are popping up that piggyback off image search results of well established search engines like Google, enhancing their results by re-ranking or filtering them based on established CBIR techniques.

2.1.1 Query by Example

Query by example (QBE) approaches require the user to describe the desired image content of the results in some way. Over the years there have been quite a few such systems proposed. Depending on the system users might have to specify image properties or give a visual query such as an example image [5, 6]. IBM's venerable Query by Image Content (QBIC) system for example ranks the most likely matches based on how well the images in the database adhere to the given color properties or layout. Color properties would often be specified directly as numeric percentages of selected colors while the layout would be

sketched with colored regions by hand. The color properties and regions would then be analyzed and then results would be found and ranked accordingly.

Virage [7] was one of the first open systems to break free of the old mold of manual annotation and start making use of visual features primarily. It established a framework whereby different primitive descriptors that characterized things like color, shape, and texture could be plugged in and made use of. In order for a plug-in primitive to be useful it needed to strike a balance between time complexity and search accuracy. Virage seemed to promote a paradigm of simply swapping images for text in terms of search and results, no longer requiring anything from the user but a simple query image. These were some of the most influential prototype QBE systems of the early years [6]. In most cases the systems that followed extracted low-level features from the query image, sometimes with and sometimes without spatial information, and searched in clever ways for images with similar features. Some of the later systems started extracting features via segmentation so that image similarity was based more on the high level objects and structures than on low-level global distributions. The Blobworld [8] framework, for instance, recognizes the nature of images as combinations of objects. Though past systems of the time strayed away from segmentation because of the performance constraints those types of algorithms imposed, the people behind Blobworld felt that image description at the level of objects is crucial for CBIR progress and so they developed a segmentation algorithm that, while imperfect, was fast enough to be used for retrieval. The feature extraction stage of this system consists of the segmentation and refinement of regions, called blobs, followed by feature extraction techniques being performed on those blobs that were similar to the color, shape, and texture ones used globally on the whole image in other systems. In effect, this approach is a way to limit the similarity

checking to salient regions of the image. The results were generally a little better than simple histogram-based retrieval when there is a clearly defined object of focus in the query image, but they tended to be a little worst when the whole scene was important.

Though different QBE approaches have used different types of features and search methods, the optimal setup often depends on the application and the user's intention, whether it be browsing aimlessly, surfing with a slight agenda, or searching with a goal [9].

(a) Color properties (b) "Hand" drawn sketch (c) Real image

Figure 2.1: Some common query styles. In (a) a series of colors can be selected and their relative percentage in the image set by sizing the respective box. (b) is an example image drawn in a paint program that would be valid input to certain CBIR systems. (c) is the most common type of query used in CBIR systems, which is the best and easiest option given that the user has an image similar to the one that they want.

2.1.2 Relevance Feedback

Relevance feedback is a technique adopted from 1960s document retrieval literature that refines search results to what a user is looking for based on feedback given from the user through multiple rounds of interaction [10]. At every round, the search algorithm selects several images on which the user is required to provide feedback (relevant or irrelevant). The feedback is then used for refining the results so that better ones can be provided in the next

round. The goal of different relevance feedback approaches is to minimize the amount of user interaction needed before the candidate set has been whittled down to good results.

2.1.3 Composite CBIR

Composite CBIR systems tend to build on the back of what is already available. Usually their target domain is the web since it contains the most images that would already have metadata attached to them, either implicitly, from their being on a certain page, or explicitly, being tagged by users from a community photo site. The recent VisualRank [4] is one such composite CBIR system; it is touted as an image focused complement to Google's PageRank. Its purpose is to find the visual themes in a set of images and then determine their relative strength after which the set can be filtered so that only the "best" images relative to the strongest themes remain. The set of images is the results returned from Google's image search which is based predominately on textual page data. Sometimes there are redundancies in such search results, like when a product is the query and multiple images of it come back but they all look the same. In these cases VisualRank can be used to select one of these images to represent the product.

VisualRank starts by generating a similarity graph where the original image results are the vertices. The vertices are connected by weighted edges based on their similarity which is determined by the amount of shared local features (Scale Invarient Feature Transform interest points). The centrality (connectedness) of the vertices is then used to determine things like theme and importance which are later used to filter spam and unrelated images out.

Figure 2.2: Here is a good example of the structure of a similarity graph from the VisualRank literature [4]. Notice that the enlarged ones are the most connected and relevant.

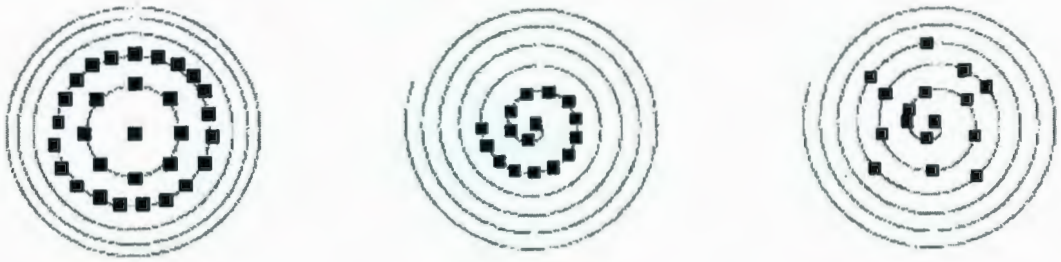
This thesis organizes a large number of images based on the similarity of extracted features and visualizes them in a dynamically generated image collage. Organizing is based only on the pixel content in the image as the work strives to make the organization procedure as free from outside interaction as possible so as to be robust and friendly to users. While both the proposed approach and CBIR techniques rely on extracted feature vectors to perform their tasks, their methods of user interaction and potential uses are different. CBIR is query driven whereas the proposed approach can constitute a form of interactive query or simply be a means of casually surfing a collection for pleasure or with the agenda of finding something of value in a diverse database.

2.2 Similarity-Based Image Browsing

While CBIR approaches are for users with clear goal about what they are searching for, similarity-based image browsing (SBIB) [11] aims to please users who just want to surf/browse an image collection, possibly in search of something and possibly not. The challenge here is how to arrange images based on visual similarities in such a way as to

optimize the experience.

Several approaches have been proposed for browsing images. Torres et al. prescribe ways to enhance CBIR results by browsing them in visual structures [12]. They use either a spiral or a concentric representation to display images. The concentric representation consists of a series of rings sharing a common center. The rings get closer to each other as their radii enlarge. In a similar way, images placed on the outer rings are displayed at a shrunken size. The other representation is a spiral one. Images, starting with the query one, are laid out along a classical Archimedes spiral that gets tighter with succesive turnings, similar to its concentric counterpart. The layout of images on the spiral comes in two flavors. The first flavor lays out the images in their ranked order around the spiral with an equal spacing based on their similarity to the query image (which is at the center). The image sizes shrink the further they are from the center. The other type of layout is the same as the first with the only difference being that instead of placing the images around the spiral equally based on rank, the positions of the images are determined with respect to the similarity measure.



(a) Concentric rings (b) Ranked spiral (c) Similarity spiral

Figure 2.3: Visual structures used by Torres et al. obtained from [12]. Note that the changing image sizes are not reflected.

In [13] Chen et al. describe a way to visualize the contents of image databases by using a structured modeling technique, called the pathfinder network. It is a branched clustering

method that was originally proposed for analysis of psychological proximity data that they have retrofitted for the purposes of SBIB. The strength of the pathfinder network is that based on the similarity between the images (i.e., histogram or texture in this case) it links them in the least redundant most intuitive way by making sure the links adhere to the triangular inequality condition. If the number of target links is specified to be one less than the number of images in the database then the paths that result between images in the network are minimum-cost paths. Such is the way their networks were constructed. The results depend highly on the way in which image similarity is determined, be it by color, texture, or shape. Color produces the most intuitive networks when viewed from a distance. The final goal of the network is a situation where the distance between any two images is proportional to their similarity.

Figure 2.4: A pathfinder network created by Chen et al. in [13] based on the images' color histograms.

Snavely et al. showed us another very interesting way to arrange and browse large sets of photos of a scene taken by a community of photographers from different viewpoints in [14]. Their proposed method exploits the common underlying 3D geometry in the scene. They first

map out the viewpoint of each photograph with respect to the scene at large and then generate a sparse 3D model of that scene. The images are then placed where they were taken from with respect to the model, effectively filling in as much of the visual detail as they can by having these images overlap based on their perspectives. The result is a stunning three dimensional rendition of a scene generated and composed entirely of photos casually taken by independent photographers without restrictions.

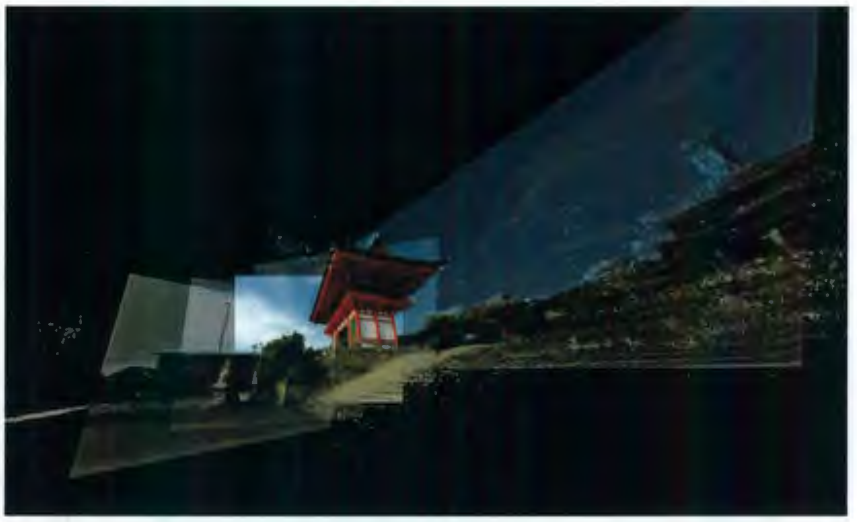


Figure 2.5: A snapshot obtained from Microsoft's Photosynth [15] which is built upon the work of Snavely et al. in [14].

Differing from the previous work, this thesis organizes images on a 2D virtual canvas and presents them in the form of a dynamically generated collage that is composed of images selected automatically according to the panning and zooming performed by a user as they search interactively for images they want to see. Unlike [14], the purpose of this work is not to find a common spatial relationship between images of a common scene but rather to measure the amount of similarity between images (which often includes their common spatial relationship) and organize the images based on those similarities to facilitate browsing. There is no stipulation on where the images have to come from, they can be completely dissimilar

and the result will reflect that.

2.3 Self Organizing Map

A self organizing map (SOM) is a special type of artificial neural network which consists of a set of interconnected units each of which has its own weight vector [16]. Once trained using unsupervised learning, a SOM can represent a set of high-dimensional samples in a topology preserving way, meaning similar samples will be placed together whereas dissimilar ones will be pushed away from each other. As a result, SOMs are useful for clustering and visualizing high-dimensional data.

SOM-based algorithms have been proposed for both image retrieval [17] and image browsing [18]. Both approaches use tree-structured SOMs [19] to organize images. Due to the high computational cost for training, the SOMs used in these two approaches are quite small in size. The proposed approach trains the SOM on the GPU and hence can afford using large SOMs that have a lot more units than the number of images in the dataset. This makes it possible to assign unique 2D coordinates for images with distinct features. These coordinates are then used for visualizing the dataset, as well as for clustering images into groups when only a subset of images can be displayed.

2.4 General Computing on the GPU

Since the introduction of the programmable rendering pipeline, the graphics processing unit (GPU) has become a very capable coprocessor for general-purpose computing. Existing research has shown that many image processing and computer vision problems can be solved on GPUs much faster than on CPUs. How to implement the SOM on a GPU has been

previously described. For example, a simple GPU-based SOM algorithm is discussed in [20] which can handle up to 4-dimensional vectors, such as RGBA colors. The implementation of a parameter-less SOM on the GPU is proposed in [21].

The GPU-based SOM training approach used in this work is similar to the one described in [20] however, we extend the algorithm and use the SOM as a means to organize image feature vectors of arbitrary dimensions. The GPU is also used to create the image collage for interactive image browsing and searching.

Chapter 3 Feature Vectors

In order to organize images based on similarity, we first need to define a way of measuring the similarity between two images. If we were to compute the similarity based on a pair-wise pixel comparison the computation costs would be unreasonably high and oftentimes small shifting and/or rotation would result in large differences. A better approach is to define and utilize image feature vectors, also called image descriptors [22] or signatures [9], which represent the salient information of images effectively at a low dimension. The ideal image feature vector is unique enough to correctly describe the image it was generated from, but close enough to vectors of related images to link them upon feature vector comparison. CBIR approaches sometimes leverage advanced feature extraction techniques to produce precise results for queries but such techniques require more processing. When the goal is to interactively organize a dynamic set of images computationally simple feature vectors are preferred so that hundreds of images can be analyzed at interactive speeds. In this chapter the feature vectors are chosen on the basis of their being fast to compute and also their robustness to superficial image manipulations like translation and scaling.

Here are the notations used in the following sections:

- I denotes an image or more formally a set of pixels p that each have a color (r_p, g_p, b_p).
- $C(p) = \lfloor N_R \times r_p/256 \rfloor \times N_G \times N_B + \lfloor N_G \times g_p/256 \rfloor \times N_B + \lfloor N_B \times b_p/256 \rfloor$ is the quantized color of pixel p, where N_R, N_G, N_B are the total number of bins used for quantizing the RGB color channels (assuming 8-bits per channel), respectively.

- $\|S\|$ denotes cardinality of a set S.
- $\Omega^h(p)$ is the set of pixels within the $h \times h$ window centered at pixel p, not including p.
- G is the gradient map of image I and $G(p) = (\theta_p, l_p)$ keeps the gradient direction and magnitude for pixel p.
- $\Theta(p) = \lfloor N_\Theta \times \theta_p / 2\pi \rfloor$ is the quantized gradient direction for pixel p, where N_Θ is the number of bins used for quantizing directions.
- $L(p) = \lfloor N_L \times l_p / L_{\max} \rfloor$ is the quantized gradient magnitude, where $L_{\max}$ is the maximum possible gradient magnitude and N_L is the number of bins used for quantizing gradient magnitudes.

3.1 Color-based

Color-based features are the dominating ones when images are small and shape is unperceivable or are displayed in a large group and details are less focused upon.

3.1.1 Color Histogram

The baseline of all image feature vectors is the color histogram. Each value in the final vector represents the number of times that a certain color occurs in the image. By simply summing up the occurrences of colors in the image and normalizing those sums by the total number of pixels we have a feature vector that is robust to rotation and scaling transformations. It should be noted that the main weakness of the color histogram is caused by the same property that makes it robust in the first place — its lack of spatial information. An

example of this issue is that you can randomly shuffle the pixels of an image and get the same histogram, which is undesirable. The N-dimensional color histogram feature vector can be calculated as follows:

$$\mathbf{F}_{\text{clr\_hist}}(I) = \frac{1}{\|I\|}[n_1, \cdots, n_N]$$
$$n_c = \|\{p | p \in I \wedge C(p) = c\}\| \tag{1}$$

where the vector's dimension is decided by $N = N_R \times N_G \times N_B$.

3.1.2 Color Autocorrelogram

The color autocorrelogram, originally proposed in [23], extends the color histogram by introducing spatial information in the form of neighboring color probability. The value of each dimension of such a vector represents the probability of finding two pixels of a certain color in the image that neighbor each other. It is computed by counting the number of pairs of neighboring pixels that have the same color and then dividing by the total number of pairs that were checked. Since the spatial information in this type of vector is relative, it is robust to rotation and scaling transformations. The problem with this type of vector is it does little to account for shape in the image.

The original color autocorrelogram considers multiple neighborhood size samples [23]. Our experiments have shown that a single neighborhood size sample is often more effective and hence the autocorrelogram calculation is simplified to:

$$\mathbf{F}_{\text{clr}_{\text{corr}}}(I) = \left[\frac{n_1}{m_1}, \cdots, \frac{n_N}{m_N}\right]$$
$$m_c = \|\{(p, q) | C(p) = c \wedge q \in \Omega^3(p)\}\| \tag{2}$$

$$n_c = \|\{(p,q)|C(p) = C(q) = c \wedge q \in \Omega^3(p)\}\|$$

where the dimension of the vector is also decided by $N = N_R \times N_G \times N_B$.

3.2 Gradient-based

Gradient-based feature vectors are more adept at modeling general shape than their color-based counterparts. Organization by a gradient-based vector is useful but considering it alone can lead to unobvious results when an organization is viewed from afar and changes in images are less apparent. The this section is inspired by previous work that uses orientation histograms for image classification [24]. It extends the orientation histogram from the mentioned work by describing variants based on those vectors found in section 3.1 above.

In all cases the initial gradient information is extracted from a grayscale version of the image by first convolving it with the Sobel filters. The horizontal and vertical gradient values are then converted into direction (angle) and magnitude for every pixel in the image. Since gradient direction is linked to image rotation, the resulting feature vectors are not rotationally invariant. The feature vector of a rotated image contains the same set of values in the same order but these values are translated in the vector with respect to the amount of rotation applied to the image [24]. Vectors based on gradients are however robust to scaling until the resolution of the image becomes too low to preserve the original shape, like when a circle becomes a dot.

3.2.1 Gradient Histogram

This is a direct adaptation of the color histogram to gradient data from an image. The directions and magnitudes of possible gradients are quantized into a number of bins. Each

dimension in the feature vector represents the number of occurrences of gradient values belonging to the corresponding bin. Similar to the color histogram, the occurrences are normalized based on the total number of values available.

$$\mathbf{F}_{\text{grad\_hist}}(I) = \frac{1}{\|G\|}[n_1, \cdots, n_N]$$
$$n_i = \|\{p | L(p) \times N_\Theta + \Theta(p) = i\}\| \tag{3}$$

where the dimension of the vector is decided by $N = N_\Theta \times N_L$.

3.2.2 Gradient Autocorrelogram

This vector extends the probabilistic nature of the color autocorrelogram to the gradients. The gradient information is first quantized, as seen in the previous case, and then the neighboring occurrences of gradients are counted, summed into a final vector position, and divided by the total pairs of the neighbors checked. The feature vector extracted measures how likely a certain gradient in the image is to remain constant within its neighborhood.

$$\mathbf{F}_{\text{grad\_corr}}(I) = \left[\frac{n_1}{m_1}, \cdots, \frac{n_N}{m_N}\right]$$
$$m_i = \|\{(p,q) | L(p) \times N_\Theta + \Theta(p) = i \wedge q \in \Omega^3(p)\}\| \tag{4}$$
$$n_i = \left\|\left\{(p,q)\middle|\begin{matrix} L(p) = L(q) \wedge \Theta(p) = \Theta(q) \wedge \\ L(p) \times N_\Theta + \Theta(p) = i \wedge q \in \Omega^3(p) \end{matrix}\right\}\right\|$$

where the vector's dimension is also decided by $N = N_\Theta \times N_L$.

3.2.3 Gradient Direction Histogram

Like the standard gradient histogram, this version models the global distribution of the gradient vectors but where it differs is it eliminates the quantization of the gradient

magnitude. Each dimension of the final vector represents the sum of all gradient magnitudes along a certain direction. The final values are normalized by the sum of all of the magnitudes. When the total feature vector dimension is fixed, this histogram allows the gradient direction to be quantized in finer resolution than the gradient histogram does.

$$\mathbf{F}_{\text{dir\_hist}}(I) = \frac{[m_1, \cdots, m_N]}{\sum_{i=1}^{N} m_i}$$
$$m_i = \sum_{p \in G, \Theta(p)=i} l_p \tag{5}$$

where the dimension of the vector $N = N_\Theta$. Please note that the above equation sums the actual (non-discretized) magnitude.

3.2.4 Gradient Direction Autocorrelogram

Similar to gradient direction histogram, an autocorrelogram can also be defined using the gradient direction only, without considering the gradient magnitude. The corresponding feature vectors depict how edges in a given image change orientations within the local windows.

$$\mathbf{F}_{\text{dir\_corr}}(I) = \left[\frac{n_1}{m_1}, \cdots, \frac{n_N}{m_N}\right]$$
$$m_i = \|\{(p,q)|\Theta(p) = i \wedge q \in \Omega^3(p)\}\| \tag{6}$$
$$n_i = \|\{(p,q)|\Theta(p) = \Theta(q) = i \wedge q \in \Omega^3(p)\}\|$$

where we also have $N = N_\Theta$.

3.3 Hybrid Approaches

The visual content of an image is a combination of both color and shape, for this reason feature vectors based on either color or gradient only may not provide the best results. Two hybrid approaches are proposed below which combine color- and gradient-based approaches in two different ways.

3.3.1 Color Histogram + Gradient Direction Autocorrelogram Aggregation

This feature vector simply joins two separately generated parts together. The first part is generated using the color histogram and the second is obtained using gradient direction autocorrelogram.

3.3.2 Color-Gradient Correlation Histogram

This is a novel feature vector that was originally devised in [2] that measures the correlation between color and gradient direction in an image. It is computed by assigning a bin to every possible color and gradient direction pair and then summing into those bins the gradient magnitudes of pixels that have the corresponding color-gradient pair. The vector can be represented as follows:

$$\mathbf{F}_{\text{clr\_grad\_corr}}(I) = \frac{[m_1, \cdots, m_N]}{\sum_{i=1}^{N} m_i}$$
$$m_i = \sum_{\substack{p \in G \wedge \\ C(p) \times N_\Theta + \Theta(p) = i}} l_p \qquad (7)$$

where the dimension of the vector is decided by $N = N_R \times N_G \times N_B \times N_\Theta$. Please note that the above equation sums the actual (non-discretized) magnitude.

Chapter 4 Organization of Images

Now that we have a compact description of our images in feature vectors we want to organize them using those feature vectors in 2D. We use an SOM to put visually similar images (ones with similar feature vectors) together and spread dissimilar ones apart.

4.1 SOM Construction

In order to organize a set of images T based on their N-dimensional feature vectors the SOM needs to consist of $M \times M$ units, where $M \times M \gg \|T\|$. This makes it possible to map distinct feature vectors to unique locations in the SOM because each image has enough space to occupy its own region of the map. Each SOM unit $\mathbf{x}$ has its own N-dimensional weight vector $\mathbf{W}(\mathbf{x})$. All of the values in the weight vectors are selected at random from within ranges that correspond to those derived across the space of the feature vectors in T. The range of the ith dimension of the N possible dimensions, for instance, is defined as $[\min_{I \in N} \mathbf{F}_i(I), \max_{I \in N} \mathbf{F}_i(I)]$, where $\mathbf{F}(I)$ is the feature vector of image I, specifically one of those described in Chapter 3, and $\mathbf{F}_i(I)$ is the ith value in that vector. Ranging the initialization minimizes the number of training iterations needed for meaningless random data to be phased out.

4.2 SOM Training

The SOM is trained with an unsupervised learning process that works to minimize the mean squared Euclidean distance of $\mathbf{W}(\mathbf{x})$ for all units $\mathbf{x}$ with respect to their closeness in the map. The process requires multiple iterations in which all images are shown to the SOM in a

random order. When a particular image I is shown to the SOM for training, the goal is to find the best match unit (BMU) and then update the weight vectors in the BMU's neighborhood. The BMU for a given image I is the unit $\mathbf{x}$ for which the Euclidean distance between the feature vector $\mathbf{F}(I)$ and the weight vector $\mathbf{W}(\mathbf{x})$ is minimum. Once the location of the BMU for image I, $\mathbf{B}(I)$, is known, the weight vectors of the BMU itself and those of the nearby units are adjusted to match $\mathbf{F}(I)$ more closely. The new weight vector is a linear interpolation between $\mathbf{F}(I)$ and $\mathbf{W}(\mathbf{x})$. The parameter $\lambda(\mathbf{x})$, called the influence, is used for the interpolation at unit $\mathbf{x}$ and is computed using the Gaussian function:

$$\lambda(\mathbf{x}) = r \cdot e^{\frac{-|\mathbf{x}-\mathbf{B}(I)|^2}{s^2}} \tag{8}$$

where s is the neighborhood size, r is the learning rate, both of which decay exponentially over time. The number of total iterations affects the decay rate of the neighborhood size and learning rate. Generally the more iterations the SOM is put through the slower the decay and the better the final convergence. In this thesis the preference is to consider the coordinate system as wrapping so that images placed on the edges of the SOM will be similar. This means that the $|\mathbf{x} - \mathbf{B}(I)|$ quantity in equation (8), the Manhattan distance in the SOM, becomes the minimum that can be computed amongst the true and wrapped versions of $\mathbf{x}$ and $\mathbf{B}(I)$. See section 4 in appendix A on the influence shader for more details.

```
for a given number of iterations
  compute the new learning rate r and neighborhood size s
  for each randomly selected and unprocessed image I
    find the BMU for I
    for each unit x within s of the BMU in the SOM
      compute influence λ(x)
      set W(x) to interpolation between F(I) and W(x) based on λ(x)
```

Figure 4.1: Pseudocode for a standard SOM training algorithm.

Once the SOM has converged the final BMU for each image in T is found just as it was during training. The mapping from an image I to its final BMU's coordinate $\mathbf{B}(I)$ is therefore obtained. By design the SOM preserves topology, which ensures that images having similar feature vectors are mapped to locations that are closer to each other, and vice versa.

4.2.1 Implementation on the CPU

The SOM training algorithm is iterative in nature and can be implemented in a straight forward fashion by a direct implementation of the algorithm in Figure 4.1. The problem with the direct approach is that it will quickly become a computational burden since time complexity is $O(M \times M \times \|T\| \times iterations)$, and as stated above $M \times M \gg \|T\|$. Luckily SOM training is very parallel, a feature which will be exploited in the following section.

4.2.2 Implementation on the GPU

As shown in Figure 4.2, the weight vectors of all units in the SOM are stored using a single color texture. Since each pixel can hold four values in its four color channels, to handle weight vectors of N-dimensions, the texture needs to contain $N/4$ times more pixels than the number of SOM units. These pixels are grouped into $N/4$ tiles, with pixels from the same

position of different tiles holding the values of the same unit's weight vector.

During initialization the weight vector values are pre-generating as described in the SOM Construction section and placed in a texture that is loaded into the GPU. Although not as fast as generating random values on the GPU directly [25], the difference is negligible as the initialization is performed only once.

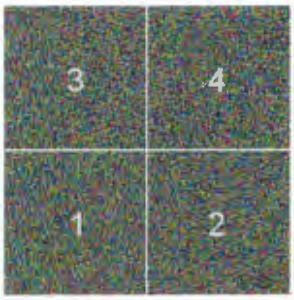


Figure 4.2: The texture that represents the initial SOM with random weight vectors. The SOM in this case consists of 256×256 units, each holding a 16-dimensional weight vector. To store the weight vectors for all units, the texture contains 512×512 pixels (each containing 4 values), which are grouped into 4 tiles.

After initialization, the SOM is trained on the GPU using the unsupervised learning process outlined in Figure 4.3 which is a version of the algorithm in Figure 4.1 that has been modified for parallelism.

```
for a given number of iterations
  compute new learning rate r and neighborhood size s
  for each randomly selected and unprocessed image I
    render distance texture based on F(I) and current SOM texture
    repeat
      render quarter-sized texture that keeps smallest distances
    until the size of the texture becomes 1 × 1
    render influence texture using r and s
    render new SOM texture with updated weight vectors
```

Figure 4.3: Pseudocode for the GPU-based SOM training algorithm. Please note that the `render` lines are carried out on the GPU and Appendix A contains the code that performs them.

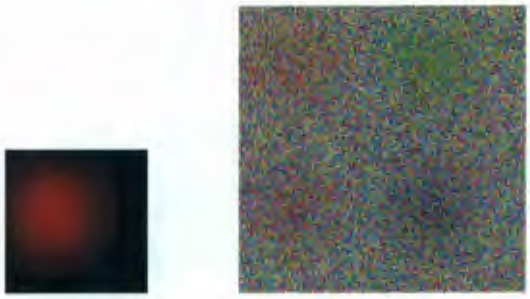
The first task is to find the BMU for a given image I. We start with one rendering pass to compute the Euclidean distance between the feature vector $\mathbf{F}(I)$ and the weight vector $\mathbf{W}(\mathbf{x})$ for each unit $\mathbf{x}$. The distance obtained is stored in a distance texture along with the coordinates of the unit as shown in Figure 4.4(a). Parallel reduction is then used to locate the pixel in the distance texture that has the smallest value in the blue channel, which is the one computed from the BMU. Parallel reduction works by finding and keeping the pixel containing the minimum blue channel value of four corresponding pixels from the previous pass. Each time the process reduces the number of pixels by a factor of four. For a SOM with $M \times M$ units, the parallel reduction process requires $\log M$ rendering passes.

(a) Distance texture (b) Parallel reduction

Figure 4.4: The distance texture and parallel reduction results obtained during the search for the BMU. Images in (b) are scaled for better illustration. In all textures the blue

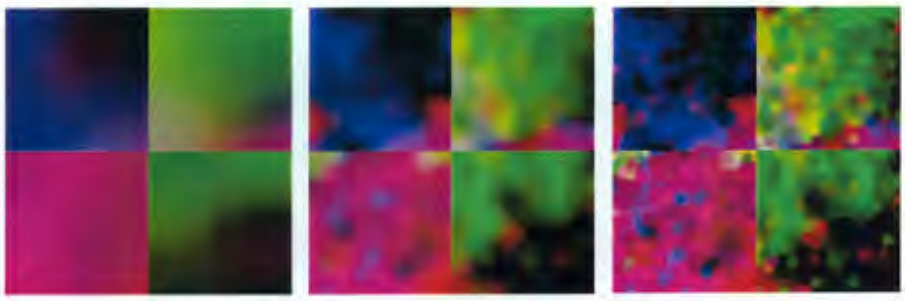
channel keeps the distance between the image feature vector and the weight vector of the corresponding unit in SOM. The red and green channels keep the x- and y-coordinates of that unit. The final result of the parallel reduction is a single pixel whose blue channel has the smallest value and red and green channels keep the coordinates of the BMU.

Next we update the SOM texture in two rendering passes. The first pass computes the interpolation parameter $\lambda(\mathbf{x})$ for all units using and stored the result in an influence texture (shown in Figure 4.5(a)). The second rendering pass takes the influence texture and the current SOM texture as input, performs linear interpolation for all of the tiles at the same time, and then stores the results into an updated SOM texture (shown in Figure 4.5(b)). Figure 4.6 displays the intermediate and the final stages of training for a sample SOM.



(a) Influence texture (b) Updated SOM texture

Figure 4.5: The influence texture and the SOM texture with updated weight vectors. The parameters used for interpolation are stored in the red channel of the influence texture.



(a) After 1 iteration (b) After 30 iterations (c) After 100 iterations

Figure 4.6: The SOM textures obtained during a training process that uses 100 iterations. As the neighborhood size decays over time, the SOM texture becomes more and more detailed.

4.3 Constructing a Multi-Level SOM

The single level SOM is fine if all images are to be considered equal. This is not the case since there are often images in populated areas that better represent those images that neighbor them. Those representative images are the ones that should be selected when there are limitations on the number that can be chosen. For this reason we delve into the idea of the multi-level SOM that gives information on how important each image is. This importance information translates into display order priority when it becomes time to browse the organized result.

4.3.1 Bottom-Up Approach

The bottom SOM, which is the largest, is obtained using SOM training procedure described above. The upper level SOMs are then generated from the lower level SOMs directly without training. The actual size of each generated level is a factor of the one directly below it, this way each unit in that level corresponds to an equal sized region in next lower level. The generation itself is carried out by assigning each unit in the level being generated a weight vector that is equal to the average of the weight vectors of those units in its child region in the next lowest level to it. The average weight vector is then used to find the best matching image from the child region for each unit in the upper level, as shown in Figure 4.7. Since the best images have feature vectors that are closer to the average feature vector of their

neighborhoods, we consider those images to be more representative and hence more important. The set of images available for the training of the next higher level is the set of all of the best matching images found during the most recent level generation (or the whole set when starting from the bottom). This approach produces what is dubbed the multi-resolution SOM since the level building process is akin to scaling an image down.

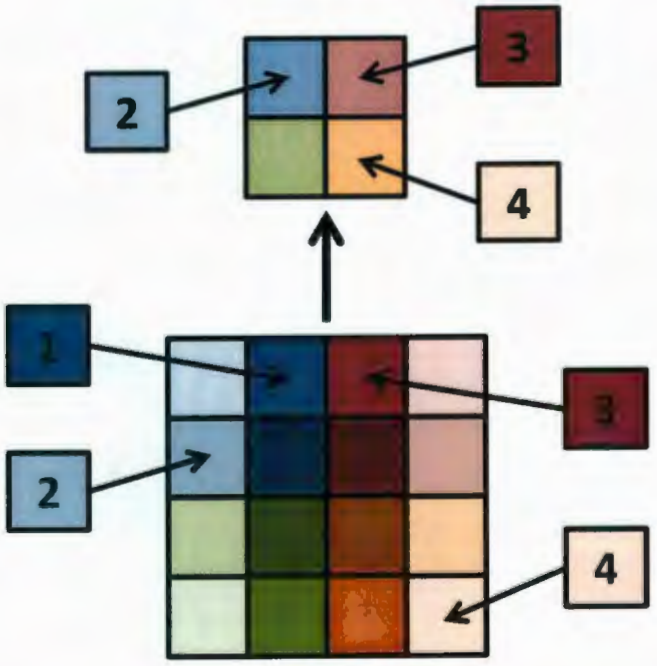


Figure 4.7: An example of a level generation taking place in a multi-resolution SOM. The images are the numbered squares and as you can see only the closest one in the corresponding child region to the averaged weight vector in the upper level (based on color) is selected and moves up.

4.3.2 Top-Down Approach

Top-down SOM level generation consists of fully training multiple SOMs of increasing size at each level. The training process is modified slightly such that the search for $\mathbf{B}_i(I)$, the BMU position at level i, is limited to the region that corresponds to $\mathbf{B}_{i-1}(I)$ of the previous level for each image I. While the center of the influence injected into the map at $\mathbf{B}_i(I)$

because of I is limited based on this region, the influence itself is allowed to permeate past region's borders. As a result any links between neighboring images in neighboring regions are realized by their close proximity at their region's shared border. The purpose of this so-called Tree-Structured SOM [19] is to limit the potential area for like images to sprawl out resulting in tighter clusters and quicker searches, $O(\log(M \times M))$ as opposed to the $O(M \times M)$ of the original SOM. Figure 4.8 and Figure 4.9 show a training example and comparison of results highlighting the shortcomings of this approach.

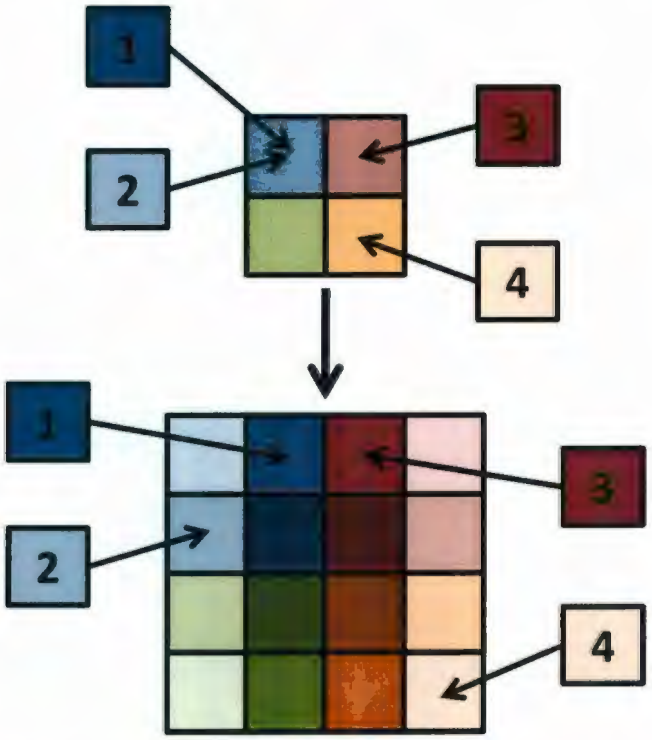
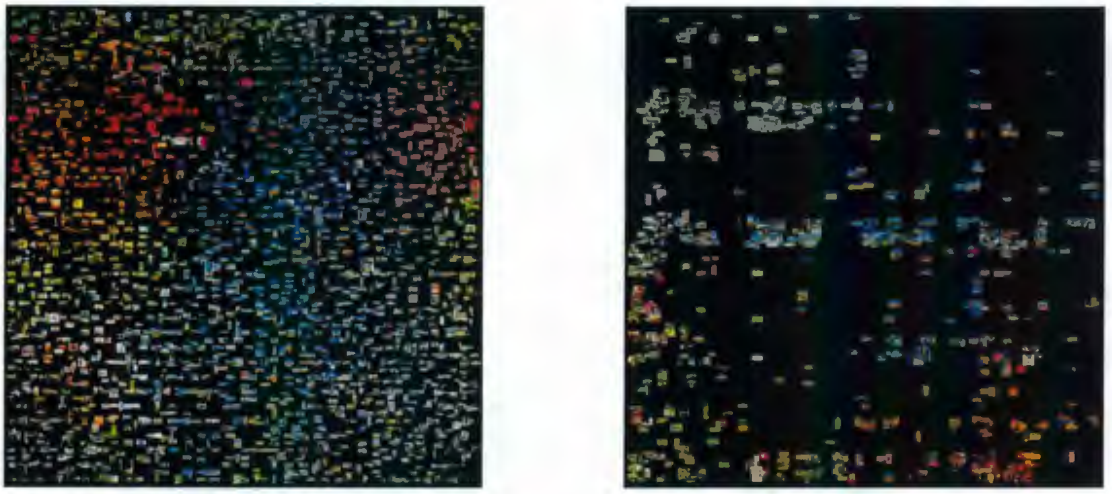


Figure 4.8: An example of a multi-level SOM trained from the top down where the numbered squares represent images. This approach starts by fully training a relatively small top level SOM. Once the training is complete there will most likely be shared BMUs, e.g. 1 and 2. The next level is then created with a size that is a product of the higher level's size, as a result each unit of the higher level has an equally sized region below corresponding to it in the new level; in this case the multiplicative factor was 4. The new level is then trained with limitations as 4.3.2 describes, after which the images have new BMUs at that level inside the regions

corresponding to their BMUs in the level above.



(a) Multi-Resolution SOM (b) Tree-Structured SOM

Figure 4.9: The results of the two different style SOMs trained under the same settings tell vastly different stories. On the one hand, the multi-resolution SOM in (a) maintains the nice, dispersive property of the original SOM on which it is based whilst in (b) half or more of the space is not used and most of the clusters have heavy overlapping. This is because even though the total size of the levels is enlarging, the images are still being locked into what are effectively tiny SOMs all the way down the tree, which means there is not enough room for the images in those regions to spread out (a problem eluded to in 4.1 SOM Construction above). If they spread out across the regions more evenly we might see a similar, perhaps better clustered, version of the bottom up multi-resolution SOM, but in this form it is not the case regardless of the settings.

Chapter 5 Browsing of Images

At this point we have a 2D virtual canvas onto which each image I has been mapped to a point I_{canvas} based on its final BMU position in the SOM. If there are only a small number of images we can simply display the canvas as is, however, when the image set gets larger, displaying the canvas and all of its images within limited screen area so that they will not overlap too much will force each image to be too small for the user to recognize its content. To address this problem an image collage is dynamically generated based on the portion of the 2D virtual canvas that is currently being viewed. The user adjusts the view of the canvas through three simple navigating operations: panning, zooming, and changing the individual image display size. All of these controls work independently; notably, zoom does not change the image size, only the size of the canvas. The specifics of this collage generation under different user interactions are described in the following sections.

5.1 Panning and Zooming

The user can use simple panning and zooming operations to select which portion of the 2D virtual canvas is being viewed. In this approach the viewed area is kept track of by mapping the screen coordinates of the browsing window onto the coordinates of the 2D canvas which effectively establishes a viewport. Both the panning and zooming operations are implemented by adjusting the mapping relationship, as shown in Figure 5.1.

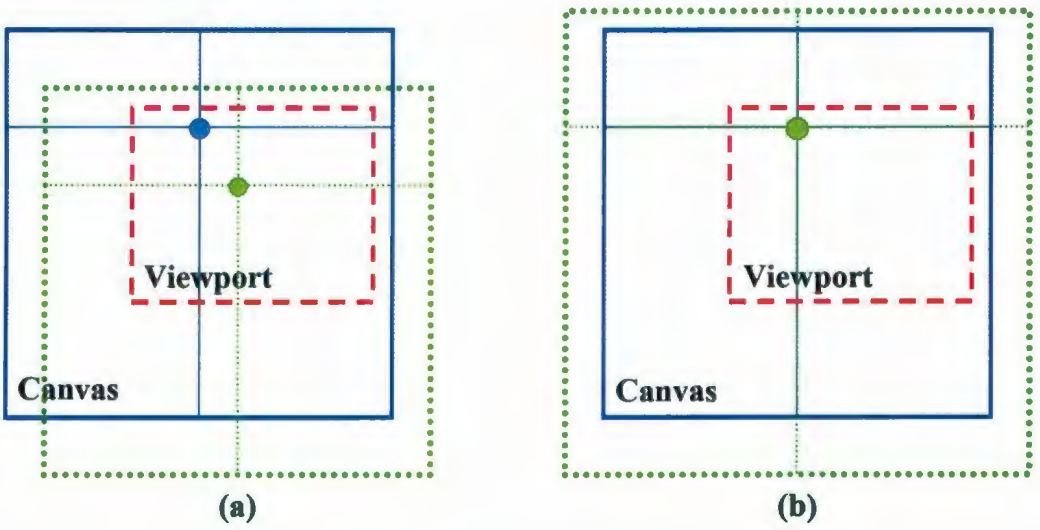


Figure 5.1: Adjusting the area of the 2D canvas being viewed through (a) panning and (b) zooming. The solid and dotted squares represent the original and the modified positions of the 2D canvas, respectively. The smaller dashed square represents the viewport area that is seen on screen. The dots in the crosshairs represent the mouse location before and after the panning.

Panning is merely a translation of the 2D canvas so that a different portion of the canvas will show up under the fixed viewport. Panning is implemented using mouse drag, when the user clicks point A on screen and drags it to point B, we ensure that the point on canvas that originally maps to point A now maps to point B.

The zooming operation is implemented using mouse wheel, i.e., scrolling the wheel up makes the virtual canvas bigger and scrolling the wheel down makes it smaller. It is noteworthy to reiterate that the zooming operation does not affect the display size of each individual image, which is controlled separately. As a result, when the users zoom in on a portion of the 2D canvas, they will observe more images in this region rather than seeing the same set of images at a bigger size. To provide a better user experience the center of the zooming is set based on the current mouse location. In the first scenario, where the mouse is

not inside any images being displayed, scrolling the mouse wheel up or down will scale the canvas bigger or smaller about the current mouse location. In the second instance, when the mouse is inside a displayed image, the center of the scaling is set to the center of the image to ensure that the image under focus does not move during the zoom.

Having these points of interest not move is tricky though because after zooming in or out the canvas shifts. In fact, the only point on the canvas that will move under scaling is the point at the origin of the viewing area. For instance, $origin \times 2 = (0,0) \times 2 = (0,0)$ whereas $(0.1, 0.2) \times 2 = (0.2, 0.4)$. To undo this zoom shift we can apply a translation that is the inverse of the amount of shift that occurs at the point of interest. For instance, in the previous example no translation is necessary for $(0,0)$, but if $(0.1, 0.2)$ is the point of interest then an inverse translation of $(-0.1, -0.2)$ is required after scaling to keep that point stationary.

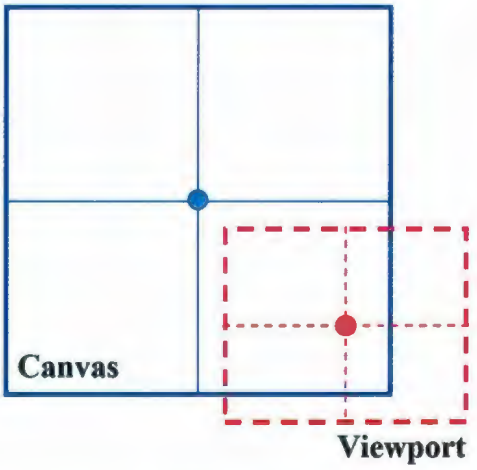


Figure 5.2: The point in the canvas (the solid square) at the center of the viewport (the dot at the center of the dashed crosshair) determines its translation.

Here, translation (panning) is managed by remembering which point in the canvas is at the center of the viewport (see Figure 5.2). The viewport position of an image I under this model is:

$$I_{viewport} = (I_{canvas} - center) * scale \tag{9}$$

where $I_{viewport}$ and I_{canvas} are the positions of the image in the viewport and canvas, respectively, $center$ is the point described in Figure 5.2, and $scale$ is the factor by which the canvas has been zoomed. To undo zoom shift on a mouse location we compute and apply the inverse translation to the model like so:

$$center_{new} = center_{old} + \frac{mouse_{viewport}}{scale_{old}} - \frac{mouse_{viewport}}{scale_{new}} \tag{10}$$

where $mouse_{viewport}$ is, as you would expect, the mouse position in the viewport, and the $scale$ factors are determined based on the amount of zoom being done. Notice that the sum of the first two terms is actually the position of the mouse in the canvas. In a special case, if the mouse is over an image the standard zoom shift correction still results in the image moving under the mouse slightly. In this situation we can simply consider the mouse to be at the center of that image over which it hovers and fully compute the center like so:

$$center_{new} = I_{canvas} - \frac{I_{viewport}}{scale_{new}} \tag{11}$$

thus achieving the effect of the image under the mouse remaining still under zoom. Panning the canvas is done by translating the center point based on mouse drag like so:

$$center_{new} = center_{old} + \frac{mouse_{new} - mouse_{old}}{scale} \tag{12}$$

where the $mouse$ positions are with respect to the viewport.

5.2 Image Display Size

The image display size is independent of panning and zooming. The upshot of this is the ability to quickly move toward images of interest by zooming, making similar ones appear as undesired ones slide off out of the viewport without having to perform repetitive changes to the image display size. The size is changed by holding down a mouse button and scrolling the mouse wheel up or down to increase or decrease the image display radius. The radius change is a straightforward one:

$$radius_{new} = radius_{old} \times \gamma^{wheel} \tag{13}$$

where $\gamma > 1$ is the scale change per scroll click and $wheel$ is how many wheel clicks occurred; it is positive for clicks up and negative for clicks down.

5.3 Selection

Once the viewing area is set by pan and zoom, all of the images inside it from the canvas are candidates for generating the image collage. The number of images actually used for creating the collage depends on the screen size of the browsing window, as well as the user specified image display size. If the number of images to display is based only on the ratio of screen area to image size when panning the canvas, images with higher priority entering/leaving the viewing area may cause currently displayed images disappear/reappear. The flicking that would result as the user pans across the canvas is undesirable and does not take full advantage of the static order. To address this problem, the images that might be displayed over the whole canvas at the current scale are pre-selected and then only the ones within the current viewing area when the user pans across the canvas are displayed. The total

number of images selected is computed as:

$$H = \delta \times \frac{A_{\text{canvas}}}{A_{\text{viewport}}} \times \frac{R_{\text{screen}}}{R_{\text{image}}} \tag{14}$$

where A_{canvas} and A_{viewport} is are the areas of the 2D canvas and the currently viewed portion, R_{screen} and R_{image} are the resolution of the browsing window which the viewport is in and the image display size, and the parameter δ controls the margin between images, for example setting it to 0.85 would leave approximately 15% of the area open to margin. The value H is reevaluated every time the user performs a zoom operation.

In order for the selection of H images to be clear and consistent a static order over all images is predetermined. Visible images selected using the order will be used to compose the image collage. The ordering is defined based on the following two criteria:

- Images that are more representative should have higher priorities to allow them to be selected first.
- Images with similar priorities should be ordered in a dispersive way so that the images selected for display are spread across the screen evenly.

The priority of each image I is a value z_I that equals to the highest level in the multi-resolution SOM (described in 4.3 above) that the image is mapped to.

To ensure the subset of images selected spreads uniformly across the screen, the images are also disperse ordered. Here an idea is borrowed from image halftoning and a dispersed-dot dithering matrix of the same dimensions as the bottom level SOM is calculated. Dithering matrices are recursively defined as:

$$D^{(2)} = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \quad \text{and} \quad U^{(m)} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

$$D^{(2m)} = \begin{bmatrix} 4D^{(m)} & 4D^{(m)} + 2U^{(m)} \\ 4D^{(m)} + 3U^{(m)} & 4D^{(m)} + U^{(m)} \end{bmatrix} \tag{15}$$

For each image I, its disperse order d_I is the value in the matrix where I_{canvas} maps to. The mapping is performed by simply mapping the image position in canvas coordinate space into the coordinate space defined by the dimensions of the dither matrix. The uniformity property of the dispersed-dot dithering matrix ensures that the first H images selected for display will be spread evenly across the screen.

At the end of the organizing process, each image I has a 4D vector (x_I, y_I, z_I, d_I) associated with it. When selecting among visible images to composite the image collage, the ones with smaller z values are selected first. If images have the same z value, the ones with smaller d values are then selected first. Once selected, each image is texture mapped to a rectangle centered at the location (x_I, y_I) in the draw window. The selected images are drawn in reverse precedence so that, in case of overlapping, ones with higher priority overlap ones with lower priority.

5.4 Experience Enhancement

At this point the interface described is robust at managing large numbers of images under a variety of useful user interactions. The user has been given the ability to pan and zoom the canvas and resize the images as they see fit. To further improve the browsing experience, a number of techniques are also applied, which include mouse hover, mouse flicking, and smooth zooming/resizing.

Since users usually focus on one main image at a time it is of benefit to enlarge and bring the image of interest to the front. If something is not done with the focus image the user has to resize all of the images at once, which causes the total number of images display H to change, which in turn may cause the image of interest disappear requiring the user use more zoom to relocate it. For this reason a hover behavior is implemented to enlarge the image which is currently in focus, deemed to be the one the mouse is over. The enlargement ratio is calculated based on both the distance of the mouse location to the image center and the image's original dimensions. This feature allows the user to quickly enlarge an image by moving the mouse over it, as well as easily cancel the enlargement effect by moving the mouse outside the image's original area. Images are sized based on a bounding square whose radius (half of its dimension) is the display size described in 5.2. They are scaled so that the maximum of their width and height is equal to the square's dimension. For any image I its current radius $\mathbf{R}(I)$ is calculated as:

$$\begin{aligned} \mathbf{R}(I) &= (1-\alpha)radius + (\alpha)radius_{focus} \\ \alpha &= \min\left(\frac{|mouse_x - I_x|}{radius \times I_{width}}, \frac{|mouse_y - I_y|}{radius \times I_{height}}\right) \end{aligned} \tag{16}$$

where $radius_{focus} > radius$ is the radius of a focus image, α is the enlargement ratio, $(mouse_x, mouse_y)$ is the mouse position in the canvas, is the center, $I_{width}: I_{height}$ is the aspect ratio of I and I_{width} or I_{height} or both are 1.

If the collage contains many images users will have to zoom in quite a ways to see the bottom level ones. As a result the scale of the 2D virtual canvas can be significantly larger than the viewing area. If the user would like to move around the canvas at this scale, a lot

panning operations are required. To reduce the panning operations needed, a way to throw the canvas with a mouse flick is implemented. The velocity of user's mouse drag operations is kept track of by computing the difference between mouse's coordinates over the past few drag events. When the user releases the mouse, instead of stopping the movement of the canvas immediately, we allow it continue to move in the same direction it was travelling on release but dampen the velocity gradually using a pre-defined deceleration factor. This enhancement allows the user to control how far away the canvas is thrown through the speed of mouse drag, rather than having to repeatedly drag it.

The final enhancement we made is to dampen discrete changes in the interface, like image size and canvas zoom, with animation based on exponential decay between states for a less ridge feel. Upon such a change we simply interpolate between the old and new states using enough steps in between to make the transition smooth. In a generic sense:

$$state_i = state_0 \alpha_i + state_N (1 - \alpha_i)$$
$$\alpha_i = \exp\left(\rho\left(1 - \frac{\mathrm{i}}{\mathrm{N}}\right) - \rho\right) \tag{17}$$

where N is the total number of steps to be taken, $state_0$ is the old state, $state_N$ is the new state, and parameter ρ controls the shape of the exponential curve and is experimentally set to 5. Notice that α_i decays exponentially so that changes start quick but finish smoothly. This type of animation was found to improve the aesthetic appeal without sacrificing productivity to slow visual effects.

5.5 Demonstration

Figure 5.3 gives a small demonstration of the browsing interface through screenshots.

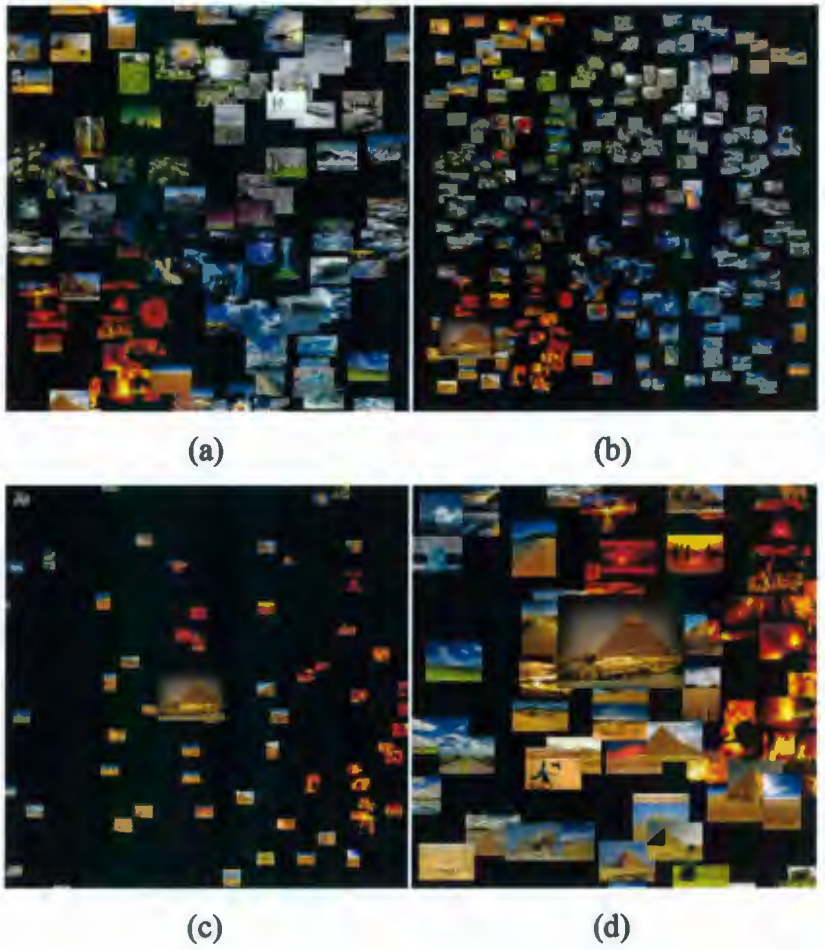


Figure 5.3: After the SOM has been trained with 256 images the image collage looks like (a). In (b) the image display size is decreased resulting in more images being shown, the canvas is also zoomed out a little. An image of interest is also found in the bottom left corner, its enlargement is due to the mouse cursor hovering over it. In (c) the canvas is panned so that the image of interest is at the center and then the canvas is zoomed in on. Notice that the image display size is not affected by the zoom and that we are seeing all of the images in this area since new images have stopped populating the vacant regions. We can see that most of the small images surrounding the one that is focused upon bare similar characteristics and are also of interest. To get a better look at these, the image size is increased in (d) beyond that of what it was in (a).

Chapter 6 Experiments and Evaluations

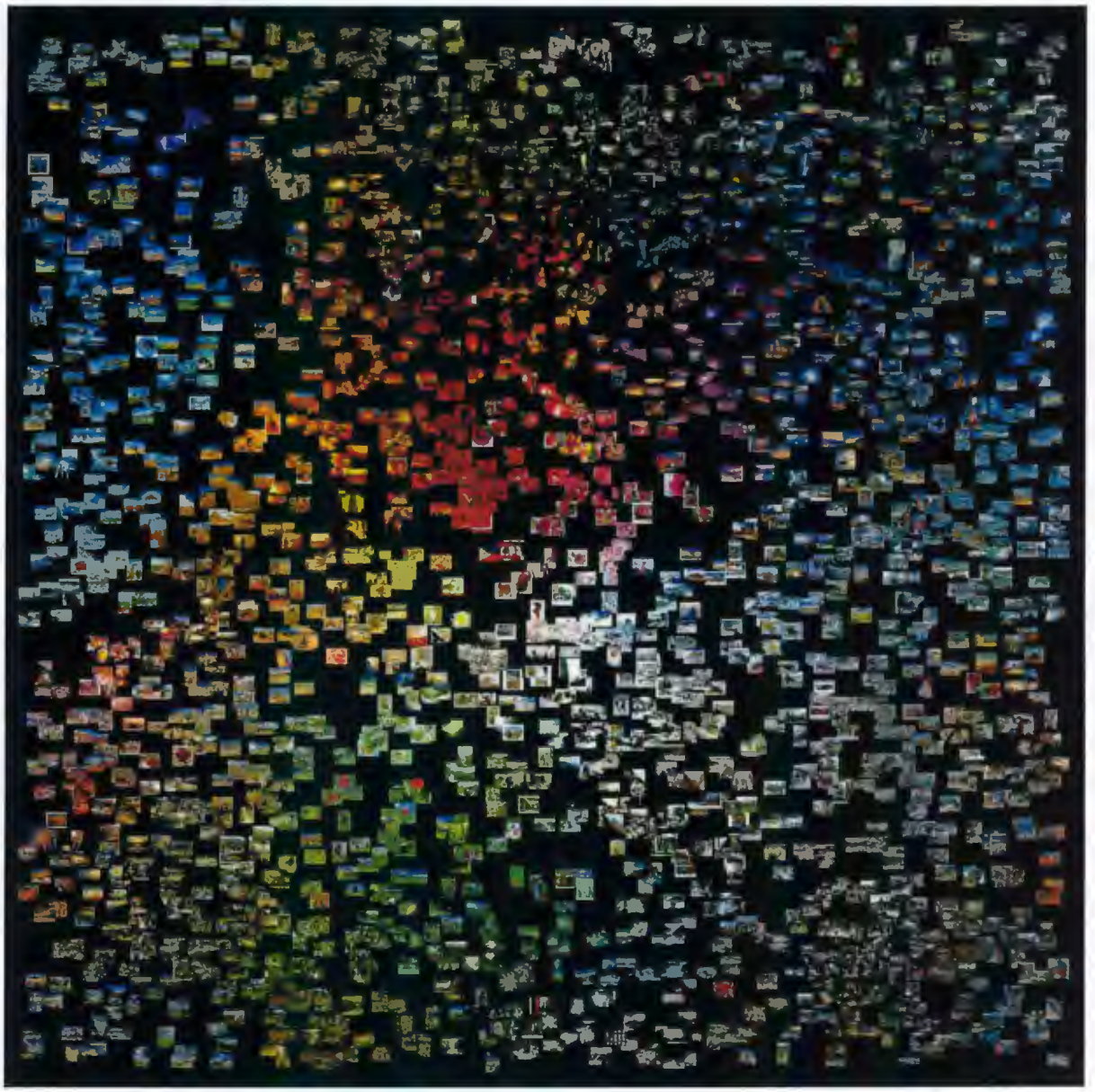


Figure 6.1: The results of organizing a collection of 2200+ images. Close inspection suggests that similar images are grouped together. Remember that the organization wraps from each edge to its opposing one.

The proposed algorithm is implemented using Java. The JOGL API is used for accessing

OpenGL commands and all the shaders that run on the GPU are written in GLSL (included in Appendix A). The implementation is tested on a desktop computer with a 3 GHz Intel Core 2 Duo CPU and a NVIDIA Quadro FX 1700 GPU. The experiment is performed using a collection of images downloaded from the Flickr photo sharing site. A dataset of 2200+ images are put together by performing 22 different searches based on keywords. It is worth noting that some of the keywords used, such as lily and daisy, return visually similar images which make it challenging for clustering. One of the organization results is shown in Figure 6.1.

6.1 Organizing Speed

The most time-consuming part of approach presented is the SOM training for image organization. To verify whether the approach is capable of handing a large number of images at interactive speed the SOM training time needed under different settings are presented in Table 1. To measure the speedup of the GPU implementation, the SOM training algorithm is also implemented on the CPU.

Table 1: The time needed for training the SOM on the GPU under different settings.

Number of images	**256**	**256**	**2220**	**2220**
SOM size	128×128	128×128	256×256	256×256
Feature vector dimension	16	64	16	64
CPU time needed	1m 50s	5m 41s	47m 27s	3h 8m 52s
GPU time needed	19s	21s	2m 2s	5m 10s
Speedup	6x	16x	23x	37x

The testing shows that the CPU implementation is not even in the ballpark of the GPU when the number of images is high. In the CPU's defense this was a simple single-threaded

implementation. Since dual core CPUs are commonplace at the time of writing the numbers shown could be halved to mimic the optimal use of two processing cores but still the high end load would require an unreasonable amount of time.

6.2 Organizing Quality

As explained in Chapter 1, the goal of 2D image organization is to group similar images as close as possible and at the same time evenly distribute images across the canvas. This objective differs significantly from the one under which feature extraction techniques for content-based image retrieval (CBIR) have been evaluated in the past [5]. For this reason, rather than using precision and recall as is customary for CBIR, new performance metrics are proposed and used in this section that properly characterize the image organization results obtained under different feature vectors.

6.2.1 Metrics

To allow quantitative evaluation we assume that a ground truth classification of images is available. That is, the set of images T is manually divided into N subsets S_k such that $\bigcup_{1 \le k \le N} S_k = T$ and so that images within each subset are considered to be visually similar. The quality of image organizing result can then be evaluated using the span of the area occupied by the images in each subset.

To evaluate a given subset S_k, we first need to find the centroid image for that subset. That is, find an image $c_k, (c_k \in S_k)$ that satisfies the following condition:

$$\forall j (j \in S_k \wedge j \neq c_k), \sum_{i \in S_k} D(i, c_k) < \sum_{i \in S_k} D(i, j) \tag{18}$$

where $D(i,j)$ is the Euclidean distance between the coordinates of images i and j.

Then, the span of the area occupied by images in subset S_k is measured using the average distance between each image in S_k and its centroid c_k:

$$R_k(S_k) = \frac{1}{\|S_k\|} \sum_{i \in S_k} D(i, c_k) \tag{19}$$

This metric gives us a measure of the overall closeness of the positioned images from the subset S_k. It is worth noting that if the maximum distance is used it will be too prone to outliers and if the median is used it will not penalize enough for outliers; for instance, up to 50% of the images in S_k could be scattered far away from the rest without a negative impact on $R_k(S_k)$ if it was median-based. For these reasons the average distance is chosen as the foundation of this metric.

On the other hand, we can calculate the average distance for all images not in the subset k using:

$$R_k(T - S_k) = \frac{1}{\|T - S_k\|} \sum_{i \notin S_k} D(i, c_k) \tag{20}$$

This metric tells us how far away that the images not belonging to the subset S_k are.

The ratio of the above two measures is a dimensionless number, which is a good evaluator of the effectiveness of the image organization for the subset S_k:

$$E_k = \frac{R_k(T - S_k)}{R_k(S_k)} \tag{21}$$

where E_k is referred to as the effectiveness of the feature vector being used on the image

subset S_k. An average effectiveness measure across the entire set of images is then simply:

$$E = \frac{1}{N}\sum_{k=1}^{N} E_k \tag{22}$$

where the higher the E value the better the overall organization is as it indicates that the images from the same subset are close to each other whereas the images not belonging to the same subset are far away.

The above effectiveness measure evaluates how well the first objective for image organizing is satisfied. A similar approach can also be used for evaluating the second objective, that being, whether images from all of the different subsets are distributed across the available space. To evaluate this we can calculate the average span of all images in the collection using:

$$F = \frac{1}{\|T\|}\sum_{i \in T} D(i, c) \tag{23}$$

where c is the centroid image for the whole set T. Since the coordinate system we use has the range [-1, 1] and is wrapped, evenly distributed images will give a F measure of about 0.75.

6.2.2 Results

In all of the experiments the number of units in the SOM is set to 256×256, which is large enough to ensure images with distinct feature vectors in our dataset are mapped to unique locations. The number of iterations used in the SOM training is fixed to 30, which is determined by experiment to be sufficient for a reasonable convergence without sacrificing the quality of the result. In all cases one organization is performed with random vectors for

each image to serve as a basis of any observed improvement, or lack thereof. These vectors can be represented like so:

$$\mathbf{F}_{\text{rand}}(I) = [r_1, \cdots, r_d]$$
$$r_i = rand(0, 1) \tag{24}$$

where $rand(0, 1)$ is simply a function that returns a random real number from the given range.

We first evaluate whether the SOM training approach can evenly distribute images under different feature vectors settings. Figure 6.2 plots the measure F, the average span, for all image locations in the organization obtained from the SOM alongside the average feature vector distance in the vector space. It confirms that, when different types of feature vectors are used, the average distances among feature vectors differ dramatically. However, the average span of the images' locations stays at about 0.71, close to the average span of randomly positioned images, which is the most evenly distributed. This suggests the SOM training algorithm can effectively map images across the final coordinate plane regardless of the original feature vector distribution.

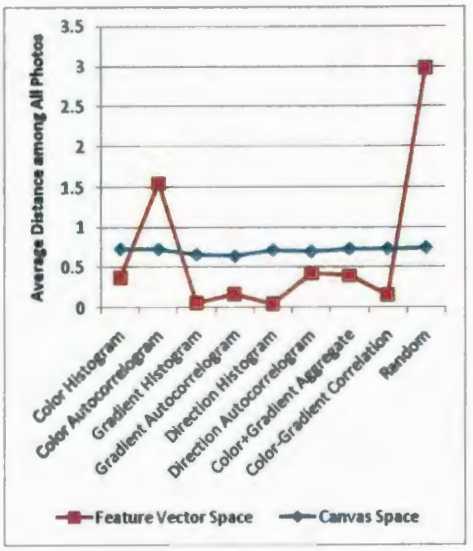


Figure 6.2: The average span of all images in the SOM and the average feature vector distance in the feature space.

Since not all of the images found by Flickr's searches visually depict the keywords used, to test the performance of different feature vectors we manually select 10~15 images for each category and use them as the ground truth classification for evaluation. Figure 6.3 shows the average effectiveness measure, E, for different feature vectors under different size settings.

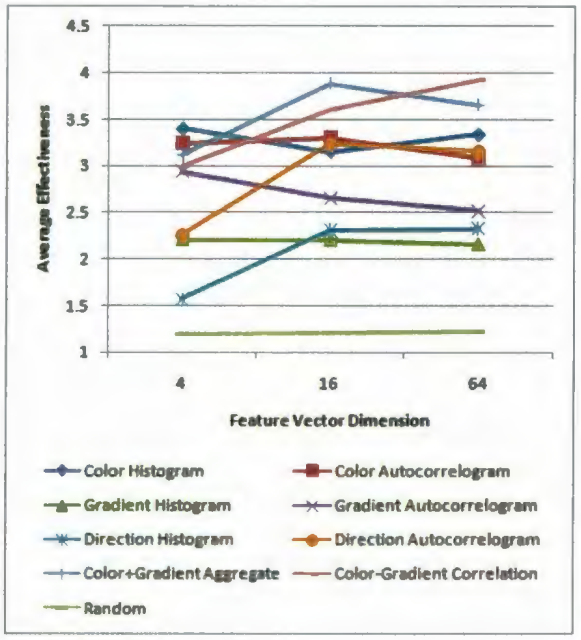


Figure 6.3: The effectiveness of different feature vectors under different vector dimension settings.

The following can be observed from the evaluation:

- When the size of the feature vector is limited to four dimensions, the two color-based approaches offer the best results. However, increasing the feature vector size does not necessarily improve the clustering.
- The gradient-based approaches generally perform better with larger feature vector settings.
- When the feature vector size is larger than four, the gradient histogram is outperformed by the gradient direction histogram and the gradient autocorrelogram is outperformed by the gradient direction autocorrelogram. This suggests that the gradient magnitude information does not play as significant a role in depicting the visual contents of images.

- Throughout our experiments we found that to achieve the optimal performance, the gradient-based approach generally requires feature vectors with higher dimensions than the color-based approach does. Hence, in the aggregate vector we allocate roughly 1/4 of the feature vector to store the color histogram information and 3/4 for the gradient direction autocorrelogram information.

- As expected, the best performances are obtained using the two hybrid approaches. Among the two approaches, the color-gradient correlation feature vector performs better at higher dimensions and the simple aggregated vector performs better at lower dimensions, most likely due to the standout performance of the low dimensional color histogram.

Figure 6.4 shows the effectiveness obtained for different image categories using four selected feature vectors.

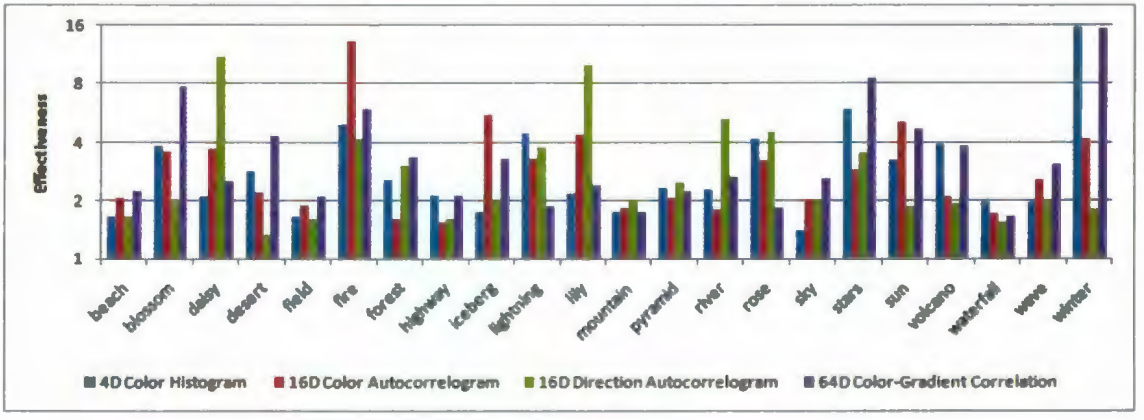


Figure 6.4: Effectiveness of four different feature vectors across different categories.

The results show that:

- Using only four numbers, the color histogram can effectively group together images of fire, lightning, rose, star, and winter. This can be attributed to the distinct color

distributions of these images.

- The color autocorrelogram performs the best on the fire, iceberg, and sun categories. This is due in no small part to the unique neighbor color pattern existing in images of these categories.
- The gradient direction autocorrelogram is very effective at grouping daisy and lily images. These images contain textures from which the gradient direction autocorrelogram production procedure can extract a distinct gradient direction pattern. However, it does poorly on desert and winter images where image gradient is not as prevalent.
- With combined color and gradient direction information, the hybrid color-gradient correlation approach performs the best for blossom, desert, forest, sky, stars, and wave images and also performs well in categories such as fire, sun and winter.
- Some categories are hard to distinguish from others since they are visually similar. For example waterfall images may appear similar to wave or river images. Beach and field images contain large portion of sky, which can influence the feature vectors generated.

Figure 6.5 shows the image placements generated using three different feature vectors. As expected, images are placed at random locations in a fairly uniform distribution when the random feature vector is used for training. The result obtained using the color-gradient correlation feature vector shows better clustering based on category than the one generated with the color histogram, which is consistent with the effectiveness values measured. (b) and (c) also visually confirm the results in Figure 6.3 and Figure 6.4 and the conclusion that the SOM evenly distributes the images over the entire space (Figure 6.2).

(a) 64 dimensional random vector

(b) 64 dimensional color histogram vector

(c) 64 dimensional color-gradient correlation vector

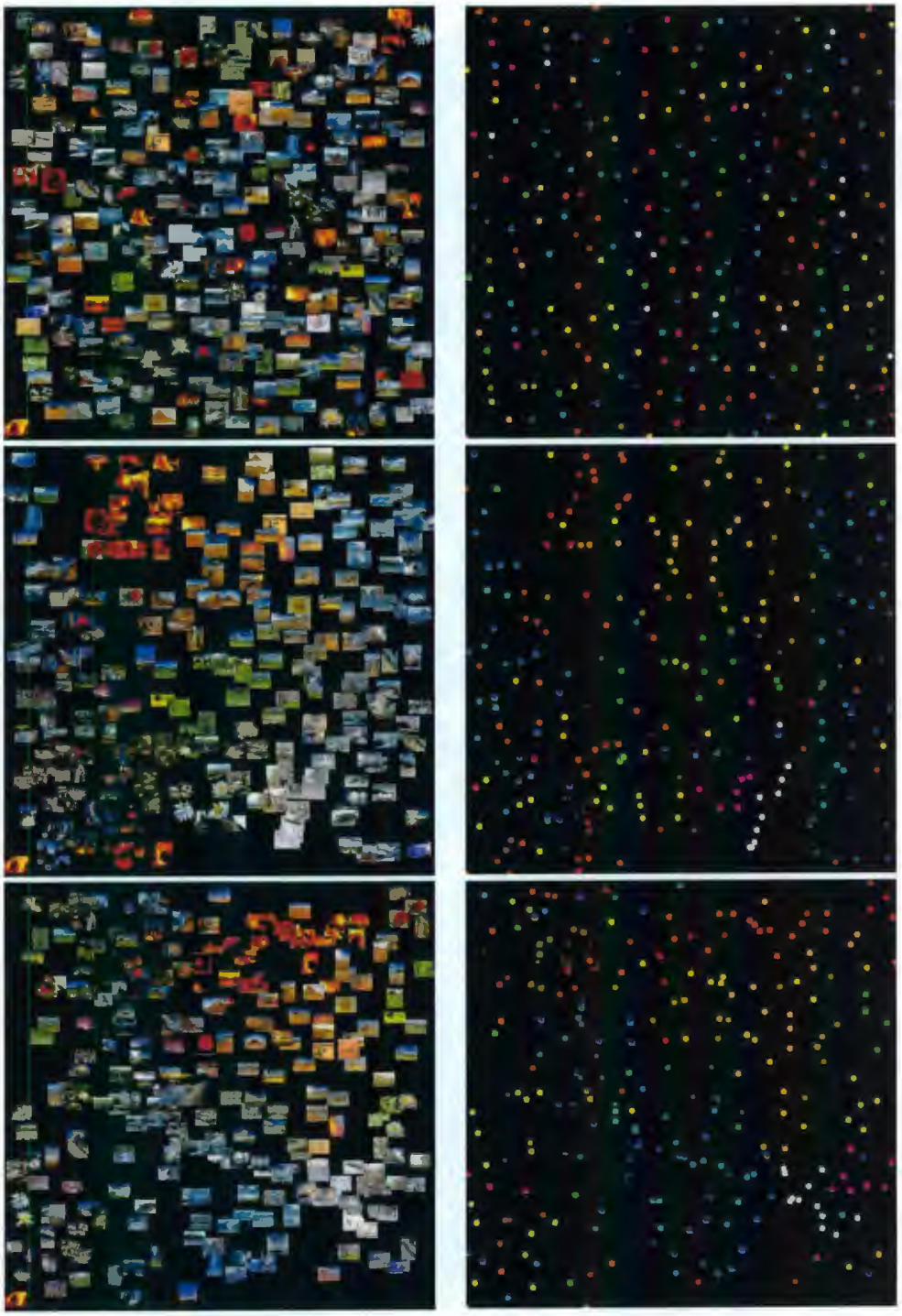


Figure 6.5: Image organizing results obtained. Left column shows all the images and right column shows the distribution of different categories using different colors. Please note that the SOM is trained with a wrapped influence so the image positions will be better for browsing and as a result similar images may be placed near boundaries of the opposite sides.

Chapter 7 Conclusion

This thesis details an algorithm for the automated organization of a large collection of images based on the content of those images. The process of mapping multi-dimensional feature vectors extracted from the images onto a 2D coordinate system through the use of a Self Organizing Map (SOM) is described along with a way to offload the majority of the computationally heavy SOM training onto the GPU. The evaluation of the proposed GPU training algorithm yields remarkable speedups over the CPU based implementation. A method of transforming the similarity-based image organization data into a user driven browsing interface is presented. Intuitive ways of handling zooming, panning, and image sizing are explained with the ultimate goal of helping the user find what they want, whether they know it or not, quicker. Ways to maximize screen real estate are discussed for when the collection to show contains more images than can feasibly be displayed at once. The validity of this design as an interactive browsing solution that improves upon the status quo is evaluated along with the effectiveness of the organizations obtained from a host of different types of feature vectors. This is done both qualitatively, by way of visualizations of the organizations, and quantitatively, through metrics tailored specifically to the image organization problem. From the experimental data, conclusions about the pros and cons of each type are drawn.

Overall the content of this document has many practical applications, not the least of which is a useful method of surfing on community image sites and as an alternative to the standard thumbnail/filename layout currently employed by operating systems. The feature vectors studied are useful general purpose ones that offer a wide array of image processing and computer vision related merit. The GPU implementation of the SOM for an arbitrary

dimension of feature vector is a powerful tool for anyone looking to leverage the SOM's unique abilities but is worried about the computational constraints imposed by its training. Finally, the concepts backing the browsing interface are of value under any circumstances that require the visualization of a large number of overlapping objects with limited screen resolution.

References

1. Strong, G. and M. Gong, *Browsing a Large Collection of Community Photos Based on Similarity on GPU*, in *Proceedings of the 4th International Symposium on Advances in Visual Computing, Part II*. 2008, Springer-Verlag: Las Vegas, NV.
2. Strong, G. and M. Gong. *Organizing and Browsing Photos Using Different Feature Vectors and Their Evaluations*. in *International Conference on Image and Video Retrieval*. 2009. Santorini, Greece.
3. Rodden, K., et al., *Does organisation by similarity assist image browsing?*, in *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2001, ACM: Seattle, Washington, United States.
4. Jing, Y. and S. Baluja, *VisualRank: Applying PageRank to Large-Scale Image Search.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2008. **30**(11): p. 1877-1890.
5. Tan, K.-L., B.C. Ooi, and C.Y. Yee, *An Evaluation of Color-Spatial Retrieval Techniques for Large Image Databases.* Multimedia Tools and Applications, 2001. **14**(1): p. 55-78.
6. Smeulders, A.W.M., et al., *Content-based image retrieval at the end of the early years.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000. **22**(12): p. 1349-1380.
7. Bach, J., et al. *Virage image search engine: an open framework for image management.* in *Storage and Retrieval for Still Image and Video Databases IV*. 1996. San Jose, CA, USA: SPIE.
8. Chad, C., et al., *Blobworld: a System for Region-based Image Indexing and Retrieval.* 1999, University of California at Berkeley.
9. Datta, R., et al., *Image Retrieval: Ideas, Influences, and Trends of the New Age.* ACM Computing Surveys, 2008. **40**(2).
10. Zhou, X.S. and T.S. Huang, *Relevance feedback in image retrieval: A comprehensive review.* Multimedia Systems, 2003. **8**(6): p. 1432-1882.
11. Heesch, D., *A survey of browsing models for content based image retrieval.* Multimedia Tools and Applications, 2008.
12. Torres, R.S., et al. *Visual structures for image browsing.* in *Conference on Information and Knowledge Management*. 2003.
13. Chen, C., G. Gagaudakis, and P. Rosin. *Similarity-Based Image Browsing.* in *IFIP International Conference on Intelligent Information Processing*. 2000. Beijing, China.
14. Snavely, N., S.M. Seitz, and R. Szeliski. *Photo tourism: Exploring photo collections in 3D.* in *SIGGRAPH*. 2006.

15. Microsoft and Schn828. *Photosynth: Kiyomizu-dera "Pure Water Temple" Kyoto Japan.* 2009 [cited 2009 June 17]; Available from: http://photosynth.net/view.aspx?cid=bc104172-4a02-4e64-9c97-c209ddda8a71.

16. Kohonen, T., *Self-Organization Maps*. 1995, Berlin: Springer-Verlag.

17. Laaksonen, J., M. Koskela, and E. Oja, *PicSOM – content-based image retrieval with self-organizing maps.* Pattern Recognition Letters, 2000. **21**(13-14): p. 1199-1207.

18. Prentis, P. *Galsom - colour-based image browsing and retrieval with tree-structured self-organising maps.* in *International Workshop on Self-Organizing Maps*. 2007. Bielefeld, Germany.

19. Koikkalainen, P. and E. Oja. *Self-Organizing Hierarchical Feature Maps.* in *International Joint Conference on Neural Networks*. 1990. San Deigo, CA, USA: IEEE Press.

20. Luo, Z., et al. *Self-Organizing Maps Computing on Graphic Process Unit.* in *European Symposium on Artificial Neural Networks*. 2005. Bruges, Belgium.

21. Campbell, A., E. Berglund, and A. Streit. *Graphics Hardware Implementation of the Parameter-Less Self-organising Map.* in *International Conference on Intelligent Data Engineering and Automated Learning*. 2005.

22. Manjunath, B.S., et al., *Color and texture descriptors.* Circuits and Systems for Video Technology, IEEE Transactions on, 2001. **11**(6): p. 703-715.

23. Huang, J., et al. *Image Indexing Using Color Correlograms.* in *IEEE Conference on Computer Vision and Pattern Recognition*. 1997.

24. Alemán-Flores, M. and L. Álvarez-León, *Texture Classification through Multiscale Orientation Histogram Analysis*, in *Scale Space Methods in Computer Vision*. 2003. p. 1077.

25. Tzeng, S. and L.-Y. Wei. *Parallel white noise generation on a GPU via cryptographic hash.* in *Symposium on Interactive 3D Graphics*. 2008. Redwood City, California.

Appendix A Shaders

These shaders are written in GLSL. Most of what follows is comments that have been inserted to aid understanding; the code itself is very minimal. Each shader should be thought of as running independently on every pixel fragment being processed by the GPU in parallel.

A.1 Distance

```
/* all uniform variables are set by external GL code */

/* used to access the som texture data */
uniform sampler2D somTexUnit; /* = 0 */

/* used to access the sample (image) texture data */
uniform sampler2D sampleTexUnit; /* = 1 */

/* blocks is the number of som tiles due to the feature vector size,
   it is used to keep the output coordinates correct */
uniform vec2 blocks; /* = (columns, rows) */

/* constant for normalization because shader output through gl_FragColor
   (at the end) is clamped from [0.0f, 1.0f] */
uniform float maxNormDistance; // = sqrt(number feature vector dimensions)

void main()
{
      /* the relative position of this unit in the som (somTexUnit couldn't
         be used with gl_TexCoord unfortunately but remember it's 0) */
      vec2 coord = gl_TexCoord[0].st;

      /* initially a zero vector to be added into */
      vec4 dVec = vec4(0); // = (0, 0, 0, 0)

      /* euclidean distance code is generated dynamically at runtime
         externally based on the number of feature vector dimensions and
         injected before the shader is loaded because loops in shaders can
         be slow */
      /* BEGINNING OF GENERATED */
      vec2 blockCoord = vec2(coord);
      vec4 dif;
      dif = texture2D(sampleTexUnit, blockCoord)
                  - texture2D(somTexUnit, blockCoord);
      dVec += dif * dif;
      blockCoord = vec2(coord.x + 0.5, coord.y + 0.0);
      dif = texture2D(sampleTexUnit, blockCoord)
                  - texture2D(somTexUnit, blockCoord);
```

```
        dVec += dif * dif;
        blockCoord = vec2(coord.x + 0.0, coord.y + 0.5);
        dif = texture2D(sampleTexUnit, blockCoord)
                    - texture2D(somTexUnit, blockCoord);
        dVec += dif * dif;
        blockCoord = vec2(coord.x + 0.5, coord.y + 0.5);
        dif = texture2D(sampleTexUnit, blockCoord)
                    - texture2D(somTexUnit, blockCoord);
        dVec += dif * dif;
        /* END OF GENERATED */

        /* add up the individual sums and perform the final square root */
        float d = sqrt(dVec.x + dVec.y + dVec.z + dVec.w);

        /* the return is ((som x, som y), normalized distance, opaque) and is
           written to the distance texture which is set up externally */
        gl_FragColor = vec4(coord * blocks, d / maxNormDistance, 1);
}
```

A.2 Minimum

```
/* all uniform variables are set by external GL code */

/* used to access the distance texture data written by the distance
   shader */
uniform sampler2D distanceTexUnit; // = 0

/* used to find reduction partners, it depends on the
   current reduction being performed */
uniform vec2 offset; // = (0.5, 0.5) on first reduction

void main()
{
        /* coordinate of the bottom left pixel can be gotten directly */
        vec2 coord0 = gl_TexCoord[0].st;
        /* the other three are computed based on the offset */
        vec2 coord1 = vec2(coord0.s + offset.x, coord0.t);
        vec2 coord2 = vec2(coord0.s, coord0.t + offset.y);
        vec2 coord3 = vec2(coord1.s, coord2.t);

        /* get distance data out of the distance texture */
        vec4 unit0 = texture2D(distanceTexUnit, coord0);
        vec4 unit1 = texture2D(distanceTexUnit, coord1);
        vec4 unit2 = texture2D(distanceTexUnit, coord2);
        vec4 unit3 = texture2D(distanceTexUnit, coord3);

        /* find the one with the minimum blue channel, i.e., the distance */
        vec4 minimum = unit0;
        minimum = unit1.b < minimum.b ? unit1 : minimum;
        minimum = unit2.b < minimum.b ? unit2 : minimum;
        minimum = unit3.b < minimum.b ? unit3 : minimum;

        /* return the minimum unit directly, it is written to a duplicate of
           the distance texture (because read/write is not possible, only
```

```
        read _or_ write) whose address is swapped externally, called ping
        pong */
    gl_FragColor = minimum;
}
```

A.3 Influence

```
/* all uniform variables are set by external GL code */

/* used to access the distance texture data which actually just contains
   the best match unit coordinates in the first pixel after the parallel
   reduction performed by multiple renderings with the minimum shader */
uniform sampler2D distanceTexUnit; // = 0

/* the position of the pixel containing the bmu data in the reduced
   distance texture, which is the bottom left pixel and is accessed on the
   half pixel */
uniform vec2 bmuCoord; // (0.5 / distanceTexWidth, 0.5 / distanceTexHeight)

/* the decaying neighborhood radius that is updated on iteration, note the
   normalization by somWidth, the max possible neighborhood diameter, so
   radiusSq will be relative to the distances calculated from texture
   coordinates, which are in the range [0, 1], below. the biggest radiusSq
   can be after normalization is 0.5 */
uniform float radiusSq; // = initialRadius * exp(-time / lamda) / somWidth

/* the decaying learning rate that is updated after on iteration */
uniform float learning; // = learningRate * exp(-time / iterations)

void main()
{
    /* best match unit's coordinates from distance texture */
    vec2 bmu = texture2D(distanceTexUnit, bmuCoord).xy;

    /* the coordinates of this pixel fragment, that is the unit in the
       som, but note that they come from distanceTexUnit's texture
       mapping because it is the same size as the _original_ som (not the
       tiled som texture with blocks)) */
    vec2 coord = gl_TexCoord[0].st;

    /* computes the closest position taking into account wrapping */
    vec2 dif = abs(bmu - coord);
    vec2 best = min(dif, vec2(1) - dif);

    /* physical euclidean distance in the som from this unit to the best
       matching unit */
    float d = length(best);

    /*  neighborhood influence function using this node's physical
        distance from the bmu */
    float influence = learning * exp(-(d * d) / radiusSq);

    /* returned influence for this position will be written to the
       influence texture (set up externally) in the red channel of this
       pixel fragment */
```

```
        gl_FragColor = vec4(influence, 0, 0, 1);
}
```

A.4 Update

```
/* all uniform variables are set by external GL code */

/* used to access the som texture data */
uniform sampler2D somTexUnit; // = 0


/* used to access the sample (image) texture data, note that the
   GL_TEXTURE_MAG_FILTER _must be_ GL_NEAREST for the texture to use the
   same coordinates as somTexUnit without having the data
   filtered/changed */
uniform sampler2D sampleTexUnit; // = 1

/* used to access the influence texture data, note that the
   GL_TEXTURE_WRAP_S and GL_TEXTURE_WRAP_T _must be_ GL_REPEAT for the
   texture with the mapping adjusted externally so a duplicate of the
   texture is mapped exactly over each of the tiles containing the
   different parts of the unit's weight vectors in the som */
uniform sampler2D influenceTexUnit; // = 2

void main()
{
        /* the relative position in the som _and_ sample texture because the
           sample one is scaled via nearest neightbor to the same size */
        vec2 coord = gl_TexCoord[0].st;

        /* from the som unit width vector */
        vec4 unitPart = texture2D(somTexUnit, coord);

        /* from the image feature vector */
        vec4 samplePart = texture2D(sampleTexUnit, coord);

        /* from the influence texture */
        float influence = texture2D(influenceTexUnit, gl_TexCoord[2].st).r;

        /* linear interpolation, note that this shader runs on different
           parts of the weight vectors in the som independently in
           parallel */
        gl_FragColor = unitPart + influence * (samplePart - unitPart);
}
```