Organisation of Image Collections

Martina Rasch
Student of Visual Computing
Vienna University of Technology
e0925447@student.tuwien.ac.at

1 Introduction

On the internet there are a lot of image collections, for example on Flickr [2] or Imgur [3]. Many of them let any user upload photographs or images. As a result, they have a large number of images, for instance Flickr had more than 10 billion image uploads in 2014 alone [6]. This makes it difficult to find specific images or images with certain attributes. Therefore, we propose a new method for searching pictures in image collections.

Existing image collections usually place the images next to each other over several rows. Often there is a sidebar or a menu bar on top to sort the images, or to filter some out by certain parameters. Later in this paper we will call this grid view, like Morrish did in his application [4]. An example of this can be seen in Figure 1, where the site TinyPic [7] is shown.

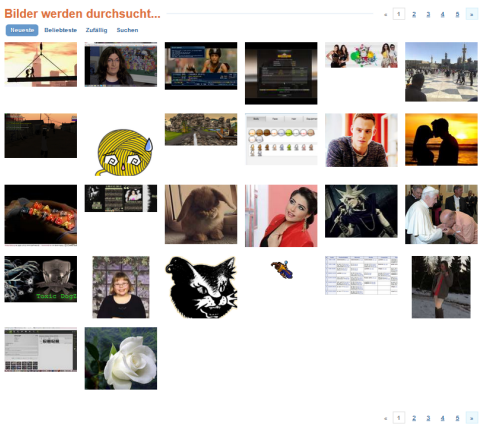

Figure 1: The image portal site TinyPic [7].

Our method is based on the approach by Morrish [4], called Internet Sites Pivot, which is an application for comparing websites. The websites are represented as small images having attributes like country of origin, industry, rating and date of creation. When sorted, they are shown in different columns depending on one of their attributes. From now on we will call this graph view, like it is called in the application. In our work, we used the concept introduced by Morrish, and applied it to image collections. Our application allows to sort a set of images, and to select groups of images according to certain attributes (e.g., name, width, height, hue).

In this paper we will first describe what we wanted to achieve with this application and afterwards the implementation is discussed in detail. At the end we will show that our method has several advantages over the grid view. If the user filters out some pictures, it is easier for him/her to relocate a particular image, since he/she can find it in the same column. As images in the same range of the attribute are put in the same column, groups of similar images often appear in the same column, making it simpler to find groups. Furthermore, because the range of the attributes is written beneath the columns, it is easier to find links between the images and the attribute ranges and to find, and use, patterns in the image collection.

2 Task Description

This section provides deeper insight into the goals and features of our program. As already mentioned in the introduction, many image collec-

tions display the images one after another over several rows, with a bar on top or on the side to filter the images.

In constrast, our approach is to sort the images by a specific attribute and to divide them into several columns. It supports two views: The grid view, where all the images are lined up one after another, and the graph view, which is shown in Figure 2. On the right side of the interface miniature versions of the websites are plotted. They are sorted into columns by an attribute which can be set in the bar on top. In the sidebar on the left the images can additionally be filtered by attributes. In January 2015 [4] the attributes were industry, rating and date of creation, in October 2014 the images could also be sorted by country of origin.

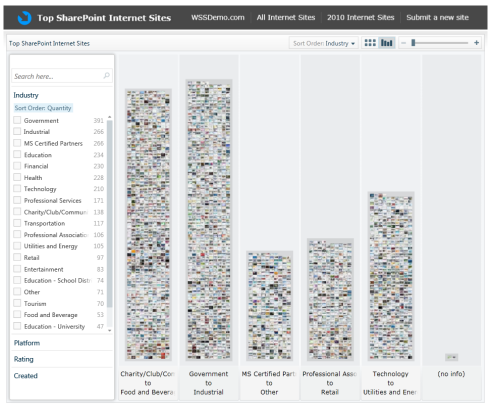

Figure 2: The graph view of the Internet Sites Pivot [4] by Ian Morrish, which our approach is based on.

In our program, the attribute used for sorting and the range of the attribute can be chosen on the left side, while the images can be filtered on the right side. The attributes are the same for both sorting and filtering. Nevertheless, filtering is independent of sorting and all attributes can be used for filtering whether they are already used for sorting or not. Figure 3 shows our image collection, where the images are sorted by the attribute height.

Figure 3: Our implementation of an image collection.

3 Implementation

In this section we describe our implementation in detail. We implemented our image collection using JavaScript and the KineticJS [5] library. The image meta data is stored in a JSON file. When our application is opened, it shows the images in the order of the data in the JSON file. The user can later return to this view by resetting all the parameters using the reset button at the bottom of the page or both the clear buttons on the left and the right.

To divide the images into columns by attributes, they are first sorted using selection sort. Selection sort is used because of its simplicity. Then the range of the selected attribute is estimated using the minimum and the maximum value of this attribute in the pictures, and split into equal ranges for each column. The maximum number of columns can be set in the application, while the actual number of columns depends on the number of images. If all the images have the same value for this specific attribute, there is only one column. This can also happen if the number of images to display is below a threshold, which can also be set in the application.

The images can be sorted by the attributes *name*, *width*, *height*, *size*, *ratio*, *description*, *tag*, *brightness* or *hue*. The name, description and tag are taken from the JSON file, the width, height, size and ratio are directly taken from the image, or calculated using width and height. For the brightness and the hue the library ColorFinder [1] is used to get the most often occurring color from the pixels in the picture. The library offers different functions for getting the dominant color, for example one excluding black and white. This is useful since many images return black or white as the dominant color. Depending on the application, it can be more helpful to use the second most common color and get more diverse results. For instance, in our case most images would end up in two of the columns, if black and white where not excluded. Furthermore, the most frequent color in the image might not be the color the user perceives as the most common color, depending on where in the image the color is. For example, if the object in the most common color is not an object of interest to the user, or if the color is only slightly more frequent than the second most common color, the user might not judge the most common color as the right choice. Therefore, we use the function that promises to exclude both black and white, although it still sometimes returns black or white as the most prominent color. This leads to the photographs being more evenly distributed over the columns, as well as to a better user experience, since fewer images that do

not seem to be of the same color are in the same column. One downside of this approach is that images that appear black or white are classified as different colors, but we find that less problematic than the other way round. To get the hue, the RGB value is converted to HSV. However, the HSV model does not include black and white - although these are still needed, since black and white have not been completely excluded by the color finding library. Furthermore, colors which are almost black/white also look black/white to the human eye. For this reason, the range of the hues is extended to contain black as the hue value -1 and white as 361. The brightness is calculated as the value of the HSV model. In Figure 4 the images in the range from black over red to yellow are sorted by the attribute hue.

Figure 4: The image collection sorted by the attribute hue.

Besides sorting, the number of images can be reduced by filtering the images using the aforementioned attributes. The filtering attribute does not have to be the sorting attribute. Figure 5 shows how the number of images is reduced from 100 to 12 by sorting by height and additionally filtering the images by width. To assist the user in keeping track of the steps he/she has done and to go back to previous states, breadcrumbs are placed above the canvas. These breadcrumbs are buttons which return the application to the step written on the button by clicking on it. To distinguish the sorting and the filtering attributes, the buttons and breadcrumbs for sorting are kept in blue, and those for filtering in red.

For the details of an image, the user can click on an image to open an enlarged view. The attributes of the image are shown to the right of the image. After closing the detail view, the website returns to the previous view. Figure 6 shows an example of the detail view.

4 Results

Our application allows to visualise an image collection in a way where the user can interactively look for and view photographs. In contrast to

Figure 5: The collection sorted by height and filtered by width to reduce the images to a small number.

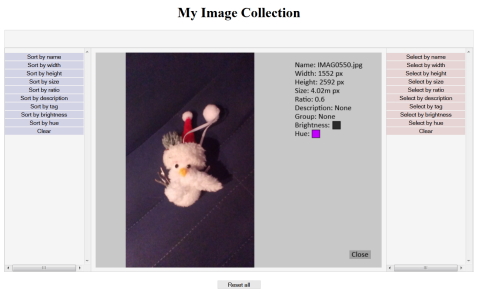

Figure 6: One of the images in detail view.

other image collections, our application supports a new view, the graph view, and a new way to search images. If the pictures are sorted, they are divided into multiple columns using the sorting attribute. Images can also be filtered to decrease the number of images.

One of the advantages over the grid view is that if the filtering is modified, the user has a reference of where each image will go, since he/she knows it will remain in the same column. In the grid view, the image could land anywhere depending on how many images are filtered away. Therefore, our approach allows to easily retrieve pictures with some specific parameters. Figure 7 shows the image collection before and after a filtering step in grid view and graph view. In the grid view an image that remains on the screen can land anywhere on the canvas, while in the graph view it stays in the same column. In the grid view of our program the image even moves to a location below its previous spot, because the images are scaled to fit on the canvas. In other approaches the image sizes will stay the same, and the images will only move to the left, or to rows above the original row. The area where the image could end up is still very large if the picture was not in one of the first rows to begin with, or if the image collection or the number of the remaining images is large.

Another benefit of our application is that the splitting by attribute makes it more convenient

Figure 7: The first and the second image are in grid view, the other two in graph view. The first and the third image were taken before, the others after the filtering step. In the grid view, the image marked in red moves from the middle to the outer left and to a spot below the original location, in the graph view, it nearly stays at the same spot and is therefore easier to relocate.

to find similar images, or images belonging to the same group, as they end up in the same or in neighbouring columns. In Figure 8 it can be seen that the sorting by hue places most of the pictures of the medieval pageant in the second column, because they were taken in the same environment under similar lighting conditions.

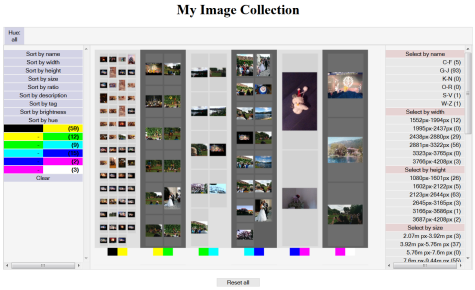

Figure 8: The image collection sorted by hue. The second column includes many photographs of the medieval pageant.

Since the range of the attributes is written beneath the columns, it is also easier to find patterns in the collection. The user might find combinations of sorts and filters which lead to images taken by a specific camera, at a specific location or under specific conditions displayed on the screen or in the same column. For instance, as shown in Figure 9, sorting the pictures by name alone reveals that one camera names pictures starting with c and another starting with i. The user can investigate the cause and use it for later searches.

In conclusion, our method has several advantages over the grid view: Orientation is easier for the user, since he/she has a reference where images will end up, even if he/she changes the settings. Moreover, certain splits show groups of images in the same column, simplifying the search for groups. Furthermore, the link between the images and the attributes is more obvious, since the images are sorted and split up by the attributes, which can also be conceived better, since the attribute ranges are written beneath the columns. Of course, all of these advantages are not present in every search, but they can lead to a better user experience overall.

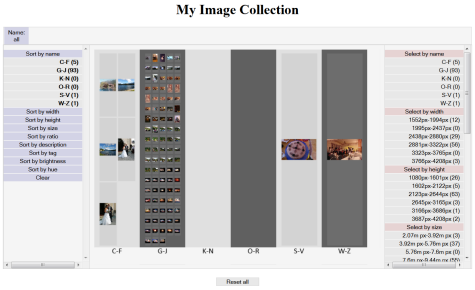

Figure 9: Sorting by name reveals that there are at least a camera who names images starting with c and another naming images starting with i.

5 Future Work

In future work additional attributes for sorting and filtering could be added. Especially the hue attribute could be altered. In our work we used the hue of the HSV model to compare the hue values, which does not include black and white. Therefore we extended the model by adding -1 as black and changing 361 to white. Other models could be used to sort by hue, or a completely other way to sort by color might be tested. It would also be interesting to divide the images into black-and-white images, grayscale images and color images. This would need a consideration of the colors of all the pixels in the image, not only the most frequent one. In addition, the problem, that the most often occurring color is not necessarily the color perceived as the most common one by the user, could be solved. Other sorting methods based on the distribution of pixels in the images could also be investigated.

Another possibility would be to allow the user to add and delete images and to change their meta data. Finally, adding pictures from image uploading sites would also be interesting.

References

[1] **ColorFinder**, http://pieroxy.net/blog/pages/color-

finder/index.html

[2] **Flickr**, https://www.flickr.com/

[3] **Imgur**, http://imgur.com/

[4] **I. Morrish**, Internet Site Pivot, http://www.wssdemo.com/livepivot/, 10.01.2015 18:54

[5] **E. Rowell**, KineticJS, http://kineticjs.com/, 10.01.2015 19:49

[6] **thenextweb.com**, thenextweb.com, http://thenextweb.com/apps/2015/01/10/canon-dslrs-and-iphone-5-dominated-photo-posting-on-flickr-in-2014/, 15.1.2015 18:25

[7] **TinyPic**, TinyPic, http://de.tinypic.com/images.php, 11.1.2015 16:26