FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Illustrative Thumbnails

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Rebeka Koszticsak
Matrikelnummer 1325492

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Dr. techn. Manuela Waldner, Msc.

Wien, 1. Jänner 2001

Rebeka Koszticsak	Manuela Waldner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Illustrative Thumbnails

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Rebeka Koszticsak
Registration Number 1325492

to the Faculty of Informatics
at the TU Wien
Advisor: Dr. techn. Manuela Waldner, Msc.

Vienna, 1st January, 2001

Rebeka Koszticsak	Manuela Waldner

Erklärung zur Verfassung der Arbeit

Rebeka Koszticsak
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Jänner 2001

Rebeka Koszticsak

Danksagung

Ihr Text hier.

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

Thumbnails are used to display the screens when switching between them on the computer and on mobile devices. These images make it easier to recognize the opened applications, and help to find the needed window quicker. Thumbnails display however only a screenshot of the windows, so they get potentially confusing if there are more opened windows or if the same application is opened multiple times. Depending on the resolution of the display, the screenshot size decreases as the number of opened windows increases. Furthermore, within the same application (like MS Office World) the screenshots are similar in appearance (eg.: white paper and tool bar), but the important text is not readable. There are several approaches that filter the important areas of the images to make editting less obvious or enhance the main region. In this bachelor thesis an application is implemented that uses these methods on the screencaptured images. The less important areas of the screenshots are cut off, and the thumbnails show only important information, which makes them more illustrative and easier to fulfill their purpose.

Contents

CHAPTER 1

Introduction

With the increase of the usage of mobile devices and the need of multi tasking thumbnails became more and more important. Thumbnails are showed while switching between tasks and have the duty to represent the different windows and so the running applications. To keep the work uninterrupted it is essential to make the switching fast and as smooth as possible. The user needs to find the desired window quickly, the choice between the applications should not take more time than absolutely necessary. But having more tabs and the same application several times running the common thumbnail system is not able to fulfill its actual purpose. Usual thumbnails take a simple screenshot of every application and they show it in smaller size. Because of the resampling process some areas and intresting parts of the window are no more recognizable and in addition the UI widgets of the same application take also the valuable place from the content elements. Furthermore with the usages of tablets, smart phones and smart watches the display, thus the thumbnail size needs to shrink even more, but the calculation algorithm does not take this fact into account.

This project introduces a new approach using seam carving to make thumbnails more illustrative even on small displays. Seam carving is a special resampling method, where related, but not necessarily straight lines are determined, and the least important of them will be eliminated. The definition of importance depends on the implementation, but it usually evaluates at least the color changes and the sequences of the edges on the source image. Since the the algorithm is used for calculation of thumbnails the usual seam carving algorithm is customized for this purpose. It takes into account that the input is a screenshot, so probably some UI elements are located on the picture. Additionally, considering that letters are more common on computer screens then on usual images, to calculate the salient regions the text content is also investigated.

1.1 Definition of the word illustrative

To provide the most satisfactory results a detailed and clear definition of the word illustrative is needed. It influences not only the measurement of pixel and region saliency, but it helps to evaluate the results and the future development. For the most fitting definition the fact needs to be taken into account, that the seam carving algorithm is used for screenshots and not for usual natural images.

To make a thumbnail more illustrative means that it is able to represent more information on a same sized picture than the other not illustrative thumbnail. In the case of a computer screen not only the usual visual features observed on real word images, like color changes and the location of edges, are crucial, but it has some special properties, which needs to be paid attention for. Most of the screenshots have UI widgets on it, which is classified as not illustrative. There are other ways to get information about the actual application, for example giving a title for the thumbnail, and it is rarely the case that a window is opened for an application itself and not for its content processing abilities. Furthermore text data has on the computer screen more important role then usual pictures. To transmit as much information as possible text data is not allowed to get damaged, even after the size reduction it needs to stay readable.

To conclude, in this project a thumbnail is called as illustrative, if its information content stays saved. To measure that property the contrast values, the color changes, the location of edegs, the text data and the presence of UI elements is evaluated and considered.

CHAPTER 2

Related Work

There are several image processing algorithms, which can be helpful by creating illustrative thumbnails. The main difference between them is, if they consider the fact that the input images always are screenshots. Therefore, this section is divided into two parts. The first one discusses algorithms with UI processing segments. Algorithms, invented for retrieve visual data from common images, are examinated in the second section. According to the information visualization method, like combining the most important parts in form of collages or simple resizing, two classes are divided. In the following some of these methods are compared.

2.1 Processing as UI

Considering that the input is a screenshot, there is a high chance that common UI elements are shown on the screen. Exceptions are only the cases where graphics, images, videos or application containing these, e.g. video games, gallery program, video player are captured. The applications like that tend to hide all UI elements, "fullscreen" mode, or redefine them, e.g. game menu.

Labeling the image parts as content and non-content, the metadata about the UI elements can be helpful. Forras uses already existing accessibility APIs to segment UI and non-UI data. Matching the metadata with the screen content provides a fast and robust result about the location of any kind of UI content. There are howerer several disadvantages, that need to be take into account when using such APIs. The range and the granularity of the support is often not wide enough. The use of an accessibilty API does not make sure, that every UI element will be recognized, because some metadata is not reachable or it will be ignored by the application.

Because of that Forras Prefab works with its own prototypes. In the database models and prototypes of common UI elements are saved. The (components of) UI elements

has to be matched with the prototypes in the data base, and after that the predefined metadata can be accessed. Since the Prefab system is able to split complex widgets into their base elements, the database does not need to be unnecessary big, but it is still able to cover the most common UI elements. In the case of special or rare UI widgets, like in a video game application or elements of a not not widely used software, the system fails. If a widget is not saved in the database, there is no way, that it will be recognized.

Forras Sikuli offers a solution not for the incompleteness of the database, but for the granularity issues too. It uses its own templates just like the Prefab system, but only in case of small icons and widgets. Since in case of larger objects template matching would be too expensive, after the processing of a training pattern, the Sikuli system is able to create new object models too. Although this feature is used in the application for other purpose, i.e. reduce the matching cost, it can solve the problem of database and of the granularity. Sikuli makes it possible to expand the database and to make the database entry afterwards more detailed.

But in case of accessibility APIs not only the availability of metadata causes possible problems. It has also no information about the actual visibilty of the UI elements. One window or rather one widget can hide an other one, some content can be out of the range of the borders of the screen etc. Forras is based on the Prefab system, but it builds a hierarchical tree from the widgets. The content is found at the leafs, and the parents are the widget, where the child is built in. With the help of this tree the order of the UI elements gets clear, and so the misleading information can be eliminated.

After the labeling of the UI elements correctly, they can be processed as needed. They can cut off as whole or processed according to the information content. Forras invented an algorithm to eliminate objects of a picture and fill their place with the texture of the object behind them using the z-buffer information. The tree mentioned above works as a z-buffer in this case. With this cut off algorithm unnecessary widgets can be easily eliminated and more important elements of the UI can so get better visible.

According to the definition of "illustrative" of this thesis the UI elements of a screen has to get cut so the segmentation of the UI elements is not needed. Although the Prefab and Sikuli systems can be helpful for labeling the images as UI and content. But theses approaches has the drawback of the usage of the database and the slow template matching. Since there is no need to know, which widgets are on the screen, and processing the actual content can already cause performance issues, the use these methods would overcomplicate the application without providing noteworthy advantages.

2.2 Processing as common picture

There are several information retrieving methods for processing images with any content. Furthermore in case of making any kind of image more illustrative, there are important tasks like interesting point recognition, Region of Interest (ROI) selection, image or feature composition etc. which they can handle easily. According to the manner of the

input data there are two different groups of these algorithms, which need to be discussed in the following sections. The first category works with more than one picture in the same time. Its strength is to choose singe features which represents the most the whole input data. In exchange it is likely that no input image will be recognizable on the result. On the other hand there are the methods in the second group, which take only one picture for input and process it as one unit. So the result is similar to the input data, but therefore the chance is high that not only the unimportant but the areas with high information ratio are damaged At the very and of this section some approaches are presented, which can help the work of both of the above mentioned groups.

2.2.1 Collage creating methods

A collage is an assembled image, containing parts of a bunch of input images and being representative for the whole input data. There are two cases by creating thumbnails more illustrative where such methods can be helpful. On the one hand the actual information of a screenshot image concentrates only in few regions of the picture. Many parts, for example UI elements, space between the content etc, can be ignored. The other way around the content can be retrieved in form of ROIs, and afterwards they can be combined arbitrary. On the other hand using collage creator algorithms thumbnails for desktop switching can be easily generated, where the input are screenshots of the open application of the desktop instead of some ROIs of one screen.

For a representative collage the most important task is to choose the best images, which information content covers the whole input data. Forras takes the parameters representativeness, importance costs, transition cost and object sensitivity into account. Representativeness means being interesting in this case. A picture tends to have high representativeness value, if there are many special textures on it, and if it is not similar to the rest of the data (so that no image is chosen twice). Importance cost evaluates and collect the ROIs of the input. While transition cost stands for the smooth transition between every two images. At last, the parameter object sensitivity holds the results of object recognition, and it helps in the arrangement, that the object has a reasonable placement.

Forras concentrates however only the first two parameters above. It clusters the images, according their source and the time, and measures the quality. Thank for the clustering by the choice of the final images it is clear, which images are the same or have similar content. This feature, accordingly modified, can be useful by sorting ROIs of a screenshot, i.e.: text content, image content etc. or of the running applications of the desktop, i.e.: textprocessing, gallery application etc. The parameter quality summarizes the value of the results of the following calculations: blurriness, compression, contrast and color balance. Since in this case only screenshots, thus computer generated pictures, can be the input, these measurements invented for camera data would provide less meaningful results than the algorithm above.

Having the best ROIs for the collage the last task is to merge them into one output

picture. For this purpose Forras uses a method called ROI packing. First the central point of every ROI needs to get chosen and every pixel on the canvas needs to get assigned to one of the using the k-Means algorithm. After that the ROIs can be placed on the area calculated for them. To fill the place between the ROIs they have to grow, keeping the aspect ratio, until they overlap. Then every ROI is shifted to the middle of its area. This method can be repeated until no ROI grows anymore. To fill the white areas, neighboring ROIs are allowed to cover them, and so fill the whole image.

Collage methods are very useful to represent a large image dataset only in a small place. They work with numerous input data, and taking the most important parts of them create a new image, that is not similar to any before. That is why they are more useful for making a thumbnail for a desktop but not for one application. Even though taking only the most important ROIs of one screen it would be possible to create an image more illustrative that any other, since the important content is the largest and the best readable. Cutting one screen apart and arranging some parts of them willingly would confuse the user, and even more time would be needed to recognize the screen.

2.2.2 Resampling methods

Treating the input image as one unit, has a big advantage against the approaches mentioned before. Even after modification the result will be highly similar to the input. Resampling means, that some parts of the image will be eliminated, but all remaining areas will have the same proportion to each other, not like by the collage algorithms. The image remains recognizable because of the same look, even though the most important areas are less readable.

To select areas, which needs to kept same, Forras uses several attention models, which defines Attention Objects (AO). AOs usually are objects from the real world, which catch the human eye, because of their familiarity, shape, color etc. With three values AOs can be easily parametrized, these are ROI, Attention Value (AV) and Minimal Perceptible Size (MPS). The attention models fit the AOs into the context. Forras works with three different attention model at the same time, with saliency, face and text attention. Having the importance value of every pixel the most important area can be detected.

The algorithm above after carefully calculation chooses only one area, which contains as many AOs as possible. So some possibly important AOs have to be ignored and cut off. With Feature awared Texturing described in Forras this do not have to be the case. The algorithm expects an input image and a feature mask. A grid is generated, which lies on the input image. This grid can be modified in an optional shape, but the gridpoints on the feature mask are not allowed to change their proportion to each other. In that way the picture elements between the AOs fill the new shape, but the AOs get barely distorted

A detailed importance is essential by creating thumbnails, since a screen usually contain more information and ROIs than a common picture. But the algorithms described above has aspects, like face recognition and grid calculation, which overcomplicate the

calculations, causing performance issues without reaching better quality. A face on a computer screen is not so common as on usual photos, and it is not so important too than text for example. With a grid the input image can get reshaped in any other form, and in addition no important part needs to get damaged. But it is meant to form the input in completely other shape, not for resize it according to aspect ratio. Some aspects however, like definition of AOs, could be changed with the actual used approaches, this possibility will be discussed in the future work section.

2.2.3 Feature combining methods

All algorithms mentioned before works with some kind of feature combining algorithms. After choosing the interesting areas, which need to stay recognizable on the resulting image too, they will be merged according to their methods into one output picture. In this process it is usual that some artifacts show up near the joining are.

A possible solution for this problem is the Piosson equation for image processing, introduced by forras. A modified Poission equation using a guidance field needs to be solved for every color in the color space, where the guidance field is calculated right from the gradients of the input image. This algorithm is actually invented for seamless cloning, but it is helpful, when some not neighboring parts of the same image needs to be placed near each other. To make the Poisson calculation more efficient an algorithm suggested by forras can help. It proposes to solve the equation on an image pyramid built from the source and its destination image, so from the two regions, which needs to be combined in this case. To reach the smoothest result forras works with gradient-domain fusion developed from the Poisson equation. It collects the color gradients of the source images into a composite vector field. Afterwards a color composite is calculated, where the gradient fits in the vector field as much as possible.

Combining and accumulating the color data promises very smooth results, joining areas, which appears naturally. These algorithms are invented however for combining pictures from the real world, and not for computer generated images. In case of thumbnails, it is less important to make the transitions less edgy, since the source does not seem to be natural neater. So the use of such advanced methods like Poisson equation is pointless, since the offer, providing natural edges and smooth transition, matches not even the input screenshot.

CHAPTER 3

Methodology

The algorithm, which creates thumbnails more illustrative, has three main steps. At the beginning the UI elements are cut off. In the case of thumbnails there are other ways to get information about what application are actually opened, for example the labeling of the thumbnails. In addition it is usual that the same software is running many times for other purposes, for example the usage of text editor for writing and for reading some document. The content not the surface of these application matters, although reading the title of the thumbnails is smaller than the software recognition through visual data, this aspect needs to take count for the better visualization of the actual content area. The second step is the calculation of the importance map weighted on the place of text data. Unlike to common pictures from the real life the occurrence of text on computer screens is higher and more important. For this reason the calculation of the final importance map have two steps. First the importance value of every picture is generated according to the color changes of the source image. Second the color value is increased at the places where text is found. In that way not only the silhouette but the whole body of the letters seams to be salient. Lastly seam carving and simple resampling is performed until the output size is reached. In this section these three main sections is discussed, and an overview about the algorithm is shown.

3.1 Eliminating UI elements

For the elimination of UI elements, they need to get identified first. In the search for UI widget, it is supposed, that they are placed near the border of the screen but not in the central area. The middle of a screen works as reference data for the investigation, and it is not checked if there are some unimportant UI elements placed. The UI elimination algorithm works right on the source window, no further premodification is needed for this process.

Cropping the border first the horizontal then the vertical margins are checked, where the order is relevant. The algorithm is based on the observation, that there are the upper and the lower bars, which cover the whole width of the display. The sidebars, if they exist, run however only between the the upper and the lower bars, and cover the remaining area. Furthermore it is common that the sidebar have slightly different style that the other UI widgets mentioned above. The UI matching method investigates slices of the screen running along the margins and checking if they belong to the UI or to the middle area of the window. To get better results, it is important to investigate coherent data, and to not mix parts of different UI bars in one slice. That is why the order of margin cropping is chosen in the mentioned way.

To find the best cropping line, where the UI meets the content area a double checking method is performed. The first step is searching for a line having the same color and no interruption along the horizontal sides of the screen. This method is based on the consideration that toolbars often have a one color background. Therefore the task is to find a line, where no more widget is placed, but it still belongs to the UI area, so it is filled with a background color.

If there are a defined UI are the method above works well, but there are several cases, where it is not able to provide any result. Game applications usually uses their own graphical UI, but even by more traditional applications it can happen that the UI area is so overloaded, that no background line can be found. For that reason is it important to perform an other checking loop too. This step is based on the correlation value of the histograms of the border and the center. The margin is split into thin slices. Initially the first slice, near the border is checked. The histogram of this slice is compared to the histogram of the center. If the correlation is high, it means that the two areas have no big difference, so nothing should get cut off, the algorithm returns. But if low correlation is shown, they are not part of the same unit, the slice comes supposedly from UI area. In this case the two histograms, the one of the first slice and the other of the center region, is kept as reference value. In the next step the other slices are compared with the two reference histograms starting at the border and heading to the middle area. At the beginning the correlation with the border histogram is high and with the center is low, which shows that the slice is still in the UI area, it matches the theme of the border widgets. But by the slices being near to the center this tendency turns. At the slice whose correlation with the center histogram is higher than the one with the border the algorithm stops, cuts every slices which were checked before, and it terminates.

This method is executed four times. In the first two cases the upper and the lower borders are investigated. The third and the fourth loop is adjusted to the sidebars. In the first step not a horizontal but a vertical line is searched, looking for a straight route, where the background color of the sidebar takes the whole space between the, already cropped, upper and lower borders of the image. Finally in the second checking loop the slices are defined vertically and not horizontally this time.

3.2 Saliency calculation

In order to ignore that parts of the source image, which actually are unimportant, saliency calculation is needed. A saliency map is a grayscale image where the bright color of an area means that that region seems to be more interesting than other dark colored pixels. There are however many definitions and approaches, which pixels should be evaluated for salient or not salient. In this case the saliency is calculated in the way presented by Forras. There are two reasons, why this algorithm is chosen. First it evaluates the low level visual informations. It means only the pixel is valued as important, which attracts the low level human visual system, like edges and color changes. Secondly it is scale invariant, so it is more robust than other similar but only singe scale approaches.

The algorithm converts the source into a uniform colorspace (Lu*v) first. After that, to make the approach scale invariant, a Gaussian contrast pyramid is built. Forras calculates the contrast value from the weighted sum of the differences between the pixel and its neighborhood. This approach needs to get slightly modified, and the calculation have to ignore the weighting value. This parameter is used because of the fact, that the central area of common real word images is more important than the margin regions. Investigating computer screens this is however not the case. A screenshot often does not have one defined focus point, where the most important information is shown. So the use of a weighting value would prefer the inner window, even though there is no significant correlation between the importance of the data and its placement on the screen. At the end the importance of each pixel is calculated summing up all levels of the pyramid into one final saliency map.

In the case of computer screens the occurance of text is common, and its information content is usually very high. Therefore it is not permissible that regions with text content appear not so important on the saliency map. For that reason a further evaluation of the saliency map is needed, to check if text areas were evaluated as important data.

The text recognition algorithm is based on the one introduced in Forras. For the first step the whole input image is horizontally blurred, so that the letters and the neighboring words give a string together. After that is all coherent string is investigated if they possibly are words or they derive from other visual features of the source. The method, which verify the words, checks two aspects. The first is if the strings are high enough but not too high for being real letters. Furthermore if their width is not too small but not too big to be one word at least but not an endless sentence. The second loop evaluates the histograms of the area, where the strings are found. It is usual, that the letters being part of the same text data have the same color, while the background, for readability, does not have big color changes neither. Therefore the histogram of these areas has two accumulations, one for the letters and one for the color of the background. So if a string has a so special histogram it is evaluated as word, but even if it managed to pass the first check, without such a histogram, it is not possible to be labeled as word.

Finishing to calculate the final saliency map, which is used in the whole application, the regions evaluated as text need to get weighted. All area which passed the word checking

test is automatically evaluated as highly important, and gets the biggest saliency value. In addition, to make sure that not even the space between the words is damaged, which is important because of readability, the area neighboring the text data is also weighted.

3.3 Seam Carving

To resize the image without damaging the important regions, seam carving Forras is used. Based on the saliency map paths are calculated, and these with minimal importance value are eliminated. This step is repeated until the desired size is reached or the importance value of the chosen path exceeds a predefined threshold. Because seam carving does not necessary eliminates only straight lines, the algorithm effects the layout of unimportant regions noticeably. Furthermore if the whole image has very high salient values, the resulting seams do not have remarkable smaller salient value than any randomly chosen straight line, chosen for resampling elimination. To save the image from unnecessary artifacts and make the application more performant, a threshold value is defined as proposed in Forras. To reach the final output size common resampling is used.

To find the most appropriate path, every possible solution needs to get checked using the backpacking method. A path map is generated, where the past possibilities are stored, and every new try can use it as a look up. In this was performance can be saved, and the algorithm gets faster.

To find the next pixel of a path a five-neighborhood of the previous member is examined. The algorithm runs from the top to the bottom, where the next member is chosen from the five nearest pixels of the next line. To calculate the costs of a possible switch between the columns the importance value of the neighboring pixels is weighted accordingly. In every case the pixel is chosen, whose importance value with the weighting parameter is the smallest. Two paths is calculated per loop, one horizontally and one vertically. But at the end only the pixels being part of the less salient path are eliminated.

Because the horizontal and vertical paths are eliminated independently, the picture being processed usually does not hold the aspect ratio. For that reason it is possible that the weight and height parameters do not reach the output size or the threshold value in the same time. In this case the seam carving algorithm continues for only one parameter until it reaches one of the conditions above. Since the threshold parameter controls the algorithm, when to switch between seam carving and resampling, its value is essential. It is calculated from the maximum salient path found after the first loop of the seam carving algorithm. The common resampling method eliminates straight lines chosen from the source image until the desired size is reached. Seam carving is used to avoid that paths with high salient value are cut off. But if the seams have as high importance value as any other from the source image, it does not make a difference if seam carving or usual resampling is used.

CHAPTER 4

Implementation

The thumbnail creator application is developed in the programming language C++ using Microsoft Visual Studio 2015. Except the source image loading window, no operating system specific calls are performed. To make the application platform independent, only this part needs changes. For image processing purposes Open Source Computer Vision Library (OpenCV) is used.

OpenCV is apart of the computer vision a machine learning library too. It has more than 2500 algorithms, which can be used for image processing or for machine learning tasks, just like object identifying, finding similar images, image stitching etc. Since the library supports not only Windows, but Linux, Mac OS and Android too, all methods taken from OpenCV count as platform independent. The library is used for loading and saving the images, and to perform several image processing tasks like converting between color spaces and image editing like bluring or filering. The actual use of the library will be discussed in the following section.

4.1 Working Pipeline

In the following the actual implementation of the application is described. Apart form the functionalities mentioned in the previous chapter, other vital helper methods are also discussed. To make the application configurable some essential parameters are set outside the application code. These parameters will be highlighted in the following sections.

4.1.1 Image loading

Before the thumbnail algorithm can start the source image needs to get loaded. To make the application more flexible a Windows call is performed, that after every start of the application a new source image can be chosen. The type `OPENFILENAME`, part of the Windows API, launches a file window, where the files can be browsed. It shows only

image files with the extension `.jpeg` or `.png`. After the choice of any source, the file path can be read, and the OpenCV function `loadImage` opens the picture.

4.1.2 UI elimination

Near the margins the occurrence of UI elements is investigated. The application uses a configurable parameter, which defines which sections of the screen count as margin area. Its default value is 20%. Starting from the border in direction to the most central point every row, and later column, is checked, if there are a line on the source having the same color and filling the whole space between the vertical, and later the horizontal, borders. In the case of success the OpenCV methods `rowRange` and `colRange` cut the input image. After that the histogram of the remaining margin area is examined and compared to the central region, where the margin is split into thin slices. The function `calcHist` calculates the histograms of the marginal and of the central window. If according to the function `compareHist` they do not correlate, the comparison is performed for the other slices and they are assigned as margin or as central window. If a slice is found which correlates with the margin histogram the image is cropped by the slice again. The parameters of correlation and the width of the slices are also configurable. According to the tests, the best results are provided, if the correlation is between 1.3 and 2 and the width is 0.25% of the window.

4.1.3 Saliancy map calculation

The saliency map is calculated from a Gaussian pyramid. But before the building of the pyramid the cropped source image have to get converted into a uniform colorspace. For this purpose the OpenCV function `cvtColor` is called. Then the levels of the pyramid is calculated using the `pyrDown` method. Afterwards a contrast map is calculated with the L2 norm from the four neighboring pixels for each level. The last step is the composition of the contrast images into one saliency map. Upsaceling of the levels is performed, so they have the same size like the the the cropped image. The pixel values of the saliency map is given from the sum of the pixels with the same coordinates of every level of the contrast pyramid. In order to avoid any overflow, the pixel values are divided by the amount of the pyramid levels.

4.1.4 Text detection

To find as many words as possible the grayscale cropped source image is processed with the Laplacian operator, with the kernel size of three. Thank to the edge detection the layout of the letters is highlighted now. To make the series of letters related a horizontal blur is needed. The function `blur` is called with a configurable kernel size, but by default the width is set to 15 pixels. The OpenCV function `findContours` is able to find the related regions and save their silhouette points in a `vector`. For each region the width and the height is calculated. If a region is at least double so wide than high, these attributes are examined, if they reach the minimum size for being an actual word. The

size of the minimal word is configurable and set to 5X5 as default. In the case a region passes the test, it is marked as possible word and it is forwarded for further investigation. After labeling every region as word or non word, the average height of the possible words is calculated. If the height of a possible word is too much smaller or bigger than the average, it automatically gets sorted out. This confidence interval is also configurable and it is set for 50% and for 1000%. The maximal word height is defined as big because of the possible size of title or headline words. After the height test the histogram of the possible word regions is checked. If it have exactly two accumulations, for the color of the letters and for the background, the region stays in the list of possible words, but otherwise it is sorted out. Finally the space above and below of the possible words is investigated. Words are usually written in lines, where because of the readability there is some space between them. Therefore if two possible word region have common points, they can not be real words and so they also need to get sorted out of the list. When the final list is ready, the regions of the words need to get marked on the saliency map. The value of every text data pixel is automatically increased. At the very end to avoid value overflow and other irregularities the whole map is normalized with the OpenCV function `normalize`.

4.1.5 Seam carving

Seam carving is implemented only in one, vertical direction. To be able to eliminate not only vertical, but also horizontal seams the cropped source image and the saliency map is rotated with the functions `transpose` and `flip`. For every loop a vertical and a horizontal seam is calculated, and the one with smaller saliency value is cut from the cropped source and from its saliency map. Until both sides reach the resampling value or the desired output size the seam carving algorithm is executed. The resampling value is calculated from the average pixel saliency value of the most salient seam on the cropped source image. The desired size and the resampling parameter are both customizable, they are set by default to 50%. The seam calculation algorithm works by filling a path map, where the previous paths and its saliency value is saved. At the beginning the first row of the map is filled with the saliency values of the saliency map. Afterwards for the next row for every pixel the less salient predecessor is searched, and the saliency value at the coordinates of the previous pixel is added to the current saliency value. The data about the predecessor pixel and the new saliency value is stored in the path map at the coordinates of the current pixel. By looking for the previous pixel the algorithm works with 5-neighborhood. The saliency values of the five nearest pixels are weighted according to their position with $\sqrt{5}$, $\sqrt{2}$ and 1. That pixel is chosen as predecessor which finally the least salient is. When the algorithm reaches the last row, in the end row of path map the saliency values of the possible ways starting at the first row are stored. The final task is to find the smallest saliency value, and to follow the predecessor entities starting at the least salient pixel until the first row is reached again. The seam found in that way is afterwards eliminated not only from the cropped source image, but from its saliency map too. But if the saliency value of even the less expensive seam exceeds the resampling value, instead of cutting the path off, the usual resampling algorithm is

applied using `resize` and the algorithm terminates.

CHAPTER 5

Results and Evaluation

CHAPTER 6

Future Work

This application was proved to be useful for creating illustrative thumbnails, but there are certain implementations to extend its scope. There are two main areas, where certain improvements can be applied, performance and the software methodology. Although the current application provides reasonable results, described in the chapter before, there are several approaches for the development, which make the present method even robuster and easier to use. The most important directions for development will be discussed in the following section.

6.1 Performance improvement

The core task of a seam carving algorithm is the evaluation of the saliency map. In the best case every pixel and every possible path needs to be examined to find the most favorable path. Depending on the size of the input, the visiting and checking all pixels individually can take a long time. This process can be get however easily parallelized. In 2007, Nvidia released a parallel computing platform, called CUDA Forras, which exploits the computing performance of the GPUs. An application, using GPU computing, executes single threaded tasks on the CPU, while it splits the parallel precesses on the GPU. In addition, the CUDA system can be simply integrated into the current application, since it is developed for the programming languages C, C++ and Fortran, and also C++ is used in this project.

6.2 Methodology improvement

An essential challenge in seam carving problems is the construction of the saliency map and the calculation of the seams. The methods implemented in the current project are able to handle the common cases, but with some improvement it can became more robust and usable in even more special cases.

There are several approaches, which besides the low level pixel data examination, also tries to find semantical objects on the input. They are building on the observation, that some objects, for example faces, even if their coloring is not especially particular, seem to be very important for the human viewer. The perceptual seam carving algorithm Forras based on the Human Attention Model calculates not only the color changes but the occurance of the facial information too. It includes the already implemented feature of OpenCV, so the application is able to generate a face map. The energy function, which calculates the pixel values in the saliency map, is then weighted with the data from the face map, so a more detailed saliency map is constructed.

Furthermore Forras uses apart from the face information, the gradient magnitude, Canny edge detection and Hugh line detection to make the saliency map as meaningful as possible. It examines thoroughly both the semantic and the low level pixel data. There is however an other method to evaluate the pixel informations even more profoundly. Putting the information into a context makes the measurements more accurate, because it controls the information extremes. So no pixel seems to be more important or unimportant as it really is. Forras reaches it with the calculation of so called neighborhood inhomogeneity factor, which denotes the amount of inhomogeneous neighbors of every pixel. This approach helps to find the important areas inside an object, where a simple line detection algorithm fails and indicate no salient regions. To find the coherent areas, which later can be defined as object, the mean shift algorithm can be used like Forras does. Furthermore it makes the saliency calculation scale invariant, and decreases the chance to find misleading high important edges.

When the saliency map is final, the next step is to calculate the seams. In this project seams are defined as paths with the width of one pixel. But the release of this rule can provide better results with less artifacts and also increase of performance, like is does by Forras. This approach tries to find streams, seams wider than only one pixel, to eliminate bigger areas of the picture at once. Even if the cheapest seams are not neighboring, for the cost of eliminating a bit more salient regions, the number of needed loops for one image can be decreased. The most important precondition is, that the stream is not allowed to cross any salient line or region, for example faces, so the relative saliency value still needs to stay low.

To eliminate the possible artifacts caused by cutting of seams and streams an algorithm presented in Forras can help. This approach is also based on the saliency calculation. If there is a big area, which needs to be filled up, it investigates the saliency map and the texture of the neighboring regions. The most salient points are labeled then as anchor points, and the actual task is to find a way between these anchors and to color the background. By seam carving it is not the goal to recreate the eliminated area, but with this algorithm it is possible to make the transition smoother. If a stream is taken a narrow, only a few pixel width path can be left behind, to let the algorithm work. In the case of seams however the neighboring edges can be modified according to the filling method taking some wider area around the seam as reference.

A further possibility to make the current algorithm faster and able to deal with the

artifacts better is in Forras presented. This method has to slightly modify the pipeline of the project, since it employs usual resampling in the seam carving phase already. It classifies every pixel on the input image as foreground, very salient, related objects, and background, less salient, probably split up parts of the image. The seam carving method is applied only on the foreground data, the non salient regions are processed with simple resampling. With this approach the properties of the background have no influence on the actual sequence of a seam. The calculation needs to take only the saliency values of the important foreground objects into account, so the resulting path can be more accurate and save the most important regions of the picture. Furthermore considering that an average seem is not as long as if it would be applied on the whole input the artifacts became smaller and less noticable. As an additional favorable side effect also the performance increases, since several areas are ignored for the seam calculation.

CHAPTER 7

Conclusion

To make thumbnails more illustrative and usable even in small screens a thumbnail creating method was presented using seam carving. In order to have as informative results as possible the saliency calculation was adjusted to the special requirements of a screenshot. In addition to seam calculation the presence of UI elements on a computer screen were also taken into account. Because of the customized seam carving algorithm and its extension with other helpful methods, like UI parts elimination, text recognition and common resampling, the resulting thumbnails were more illustrative than their usual relatives, since their information content stayed better recognizable and the text region easier readable.

List of Figures

List of Tables

List of Algorithms

Index

Glossary

editor A text editor is a type of program used for editing plain text files.. 5

Acronyms

CTAN Comprehensive TeX Archive Network. 11

FAQ Frequently Asked Questions. 11

PDF Portable Document Format. 6, 10, 11, 15

SVN Subversion. 10

WYSIWYG What You See Is What You Get. 9

Bibliography

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58:345–363, 1936.