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 a list of open windows or tabs 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 only if no more irrelevant information remains, a conventional downsampling algorithm is applied. So the thumbnails show only salient information, which makes them more illustrative and easier to fulfill their purpose.

Contents

CHAPTER 1

Introduction

With the increased use of mobile devices and reliance on multi-tasking thumbnails are becoming increasingly important. Thumbnails appear when switching between tasks, representing the concurrent windows i.e. the running applications. To keep a continuous and effective workflow, it is essential to make the process of switching as fast and smooth as possible by allowing the user to quickly identify and choose between the given tabs. The current thumbnail system is not, however, fit for this purpose. Due to the fact that standard thumbnail applications simply take the screenshot of each application and present them in a smaller size, the relevant parts of the window are no longer recognizable. For example, the user interface widgets, especially in case of the multiple windows! of the same application!, take up valuable place that could otherwise be used to present more important content elements, such as actual file names or other unique characteristics of the window. Furthermore with the use of tablets, smart phones and smart watches the display, thus the thumbnail size needs to shrink even more, but the original calculation algorithm does not take this 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 re-sampling method, where not necessarily straight lines are determined, and the least important of them are then eliminated. The definition of importance depends on the implementation, but it in any case evaluates color changes and identifies the edges that shape the outline of the element on the source image. Following a similar logic, the algorithm presented in this paper uses a customized seam carving method. Furthermore, it takes into account that the given input is essentially a screenshot thus will probably contain some UI elements on the picture. Additionally, considering that letters are more common on computer screens than on usual images, the text content is also investigated to calculate its salient regions.

1.1 Definition of the word illustrative

To provide satisfactory results a detailed and clear definition of the word illustrative is needed. Not only it influences the calculation of the pixel and region saliency value, but also acts at the selection of result outputs, and !is useful for future software development! . ! Also, for a fitting definition it needs to be taken into account that in this case the seam carving algorithm is used for screenshots and not for usual photographs.

The expression 'illustrative thumbnail' also comprehends that it is able to represent more information on a same sized picture than the other not illustrative thumbnail. Unlike usual visual features observed on real world images, such as color changes and location of edges, as mentioned above, computer screens have additional special properties that require further consideration. For example, most screenshots include UI widgets, that are classified as not illustrative. Namely, it is rarely the case that the relevant part of an application is its UI and not its the actual content processing features, furthermore, the identification of the a window in question is also possible by reading its title. 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, i.e. even after size reduction it needs to stay readable.

Summing up, in this project a thumbnail is called 'illustrative' if its informative content preserves. To measure this property, contrast values, color changes, location of edges, text data and the presence of UI elements are evaluated and considered.

CHAPTER 2

Related Work

There are several image processing algorithms that can be helpful at creating illustrative thumbnails. The main difference between thus algorithms is whether they consider the input as an actual application window or just as a regular image. Therefore, this section is divided into two parts. The first one discusses algorithms with UI processing segments. Algorithms, invented for resampling natural images, are examined in the second section. Moreover, there are two classes of information presentation methods are to be distinguished, namely, simple resizing and collages, the latter combining the most important parts of the image. In the following, these methods are compared.

2.1 Processing as UI

When the input is a screenshot, there is a high chance that the usual UI elements, like buttons and menu bars, are shown on the screen. Exceptions for this are only cases when graphics, images, videos or application are presented, such as video games, gallery program or video players. Such applications tend to hide all UI elements and operate in "fullscreen" mode, or use a redefined UI e.g. game menu bars.

Labeling the image parts as content and non-content, the metadata about the UI elements can be helpful. Forras uses already existing accessibility APIs, tested on Mac OS X and on Microsoft Windows, 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 however several disadvantages, that need to be taken into account when using such APIs. The range and the granularity of the support is often not wide enough. The use of an accessibility API is unable to ensure that every UI elements are recognized, because some metadata are not reachable or they will be ignored by the application.

Consequently, Forras Prefab uses an adapted of such prototypes. In the database, models and prototypes of common UI elements are saved. The (components of) UI elements are then matched with the prototypes from the database, allowing a consequent access to the predefined metadata. Since the Prefab system is able to split complex widgets to their constructing elements, the database does not need to be unnecessary big, though 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 widely used software, the system fails to recognize them, since these rare widgets are not included in the given database.

Forras Sikuli offers a solution not only for the incompleteness of !such! database, but for issues around granularity, 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/demanding/not cost-efficient/expensive IN TERMS OF TIME AND SPACE/ would not be time and storage efficient , after accomplishing a training pattern, the Sikuli system is able to create new object models too. Although in the original/Sikuli application this feature is used for another purpose, i.e. to reduce matching costs, it can effectively used to solve the problem of database and granularity. With other words, Sikuli allows the expansion the of database and thus creating a more detailed database !entry bemenetet? !.

On the other hand, in case of accessibility APIs, it is not only the availability of the metadata that may cause problems, but also it does not provide information about the actual visibility of the UI elements. One window or rather one widget can overlap with or even fully cover another, some content can be out of the range of the borders of the screen etc. Forras is built on the Prefab system, but additionally it creates a hierarchical tree of 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 'process' helyett: applied/manipulated/operated with as needed. They can cut off as whole or processed according to the information content. Forras invented an algorithm that eliminates the picture objects 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 easily be eliminated and more important elements of the UI can thus get more visible.

According to the definition of "illustrative", the UI elements of a screen in any case are to be excluded, consequently the segmentation of the UI elements is not required. Although the Prefab and Sikuli systems are proved to be helpful at distinguishing between UI and content parts of the image, these approaches have their disadvantage at their reliance on database usage and at their slow template matching performance. Furthermore they are actually designed for another purpose, namely to segment and classify UI elements, so before their employment significant reworking is needed. Since it is not essential to know, which exact widgets appear on the screen, and processing their actual content may cause performance issues, the use of the above methods would overcomplicate the application without providing noteworthy advantages.

2.2 Processing as a regular picture

There are several information retrieving methods for processing images with any kind of content. Furthermore, these methods can also handle a series of important tasks such as interesting point recognition, Region of Interest (ROI) selection, image or feature composition, i.e. tasks that are in order to make any kind of images more illustrative. Based on the type of input data, there are two groups of the above algorithms that will be discussed in the following sections. The first category works with more than one picture at the same time. Its strength is to choose singe features that best represents the whole input data. In exchange it is likely that no input image will be recognizable on the result. To the contrary, there are the methods in the second group that take only one picture for input and process it as one unit. The resulting image is similar to the input data, however thus it is likely that not only the unimportant but also the areas with high information ratio are damaged. Finally, in the subsection below some approaches are presented that assist with 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 more illustrative thumbnails where such methods can be helpful. On the one hand the actual information of a screenshot image is presented 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 thumbnails for desktop switching can be easily generated using collage creator algorithms, where the input is screenshots of the open applications 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 collects 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 every object has a reasonable placement.

Forras concentrates however only at the first two parameters above. It clusters the images according to 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 is selected and every pixel on the canvas needs to get assigned to one of them 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 are increased, keeping the aspect ratio, until they overlap. Then every ROI is shifted to the middle of its area. This method is repeated until there are no further increases in the ROI anymore. To fill the white areas and eliminate any empty place, neighboring ROIs are allowed to partially cover each other. Collage methods are excellent at representing a large image dataset in a small place. They work with numerous input data, take the most important parts of them and create a new image, that is not similar to any previous one. That is why they are more useful for making a thumbnail for a desktop while not being much so in case of applications. Even though with taking the most important ROIs of one screen it would be possible to create a more illustrative image than any other, since the important content could stay large and so well readable, classic collage assembly methods were developed for natural images. Screenshot collages have however different requirements for image and text regions. In addition, cutting a screen apart and arranging its parts willingly has a potentially confusing result for the user, requiring them to spend even more time with the screen recognition.

2.2.2 Resampling methods

To attain a constant relative position among regions, applying a resampling method is more effective than the above described collage creating approaches. Resampling means that some equally distributed parts of the image will be eliminated, thus, unlike by the collage algorithms, all remaining areas will have the same relation to each other. Therefore, the image itself remains recognizable because it has a highly similar appearance, in spite of having the most important areas less readable.

To select the invariable areas Forras suggests various attention models that are able to define the so-called Attention Objects (AO). AOs are usually real-world objects that due to their familiarity, shape, color etc., attract the human eye. AOs can easily be parametrized using three values: ROI, Attention Value (AV) and Minimal Perceptible Size (MPS). The attention models fit the AOs into their context. Forras works with three different attention models at the same time: saliency, face and text attention. The most important areas can be detected according to the importance value of each given pixel.

The algorithm above, after careful calculations, chooses the only area that contains the highest possible amount of AOs. To accomplish this, some possibly important AOs have to be ignored and cut off. With Feature awared Texturing described in Forras this does 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 at creating thumbnails, since a screen usually contains a greater amount of sensitive information. For example, a text can get easily destroyed, since they do not include such a clear focus point as regular pictures, where the density of ROIs is high. But the algorithms described above has aspects like face recognition and grid determination that over-complicate the calculations, causing performance issues without reaching better quality. A face on a computer screen is not as frequent as on usual photos, and in addition it is not as sensitive as for example a text data. In the case of down sampling a human face can stay recognizable, whereas texts quickly become illegible. With a grid the input image can get reshaped to any other form with no additional damage to the important areas. !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 expand the actual used approaches, this possibility will be discussed in the future work section.

2.2.3 Feature combining methods

All the above mentioned approaches work with some kind of feature combining algorithms. After choosing the relevant areas that need to stay recognizable also on the resulting image, they are merged into one output picture according to their methods. It is usual in this process that some artifacts show up near the joining area.

A possible solution for this problem is the Poisson equation for image processing, introduced by forras. A modified Poission equation using a guidance field needs to be solved/applied for every color in the color space where the guidance field is calculated 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 need to be placed near each other. To make the Poisson calculation more efficient an algorithm suggested by forras offers help. It proposes to solve the equation of/with! an image pyramid that is built from the source and the destination image, or, as in this case, from the two image regions to be combined. To reach the most optimal 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!composition? is calculated, where the gradient fits as well in the vector field as possible.

Combining and accumulating the color data promises very smooth results with a natural appearance at the joining areas. These algorithms however are invented 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 to create illustrative thumbnails 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 by giving a title for it, containing the running application's name. In addition, it is rather usual that the same software is running multiple times, possibly for other purposes, e.g. using the text editor for both writing one and reading another document. Consequently, the actual content and not the surface of these applications makes a difference. ! Although reading the title of the thumbnails is slower 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! based on? the place/location? of text data. Unlike at regular real-life pictures 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 an image based energy function described in the next sections. Second the importance value is increased at the places where text is found. This way not only the silhouette but also the whole body of the letters seems to be salient. Lastly, seam carving and simple resampling is performed until the correct output size is reached. In this section these three main sections are discussed, and an overview of the algorithm is presented.

3.1 Eliminating UI elements

For the elimination of UI elements, they need to get identified first. In the search for UI widget a heuristic is developed, which supposes, 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 in a predefined, parametrized border area, 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 in the predefined border area, but near to the center 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 is a defined UI area 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 color 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.

3.2 Saliency calculation

In order to ignore that parts of the source image, which actually are unimportant, saliency calculation is needed. The saliency value of a pixel measures, how important the pixel is, how noticeable it is and how much attention it affects for the human eye. A saliency map is a grayscale image where the high value of an area means that that region seems to be more interesting than other pixels with lower value. 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 using the L2 norm. This approach needs to get slightly modified, in way that the calculation has to ignore the weighting value. Forras uses this weighting parameter 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. All these size checking values are parametrized, their default value based on the size of one letter is mentioned in the next chapter. 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 are bimodal, 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. Seam carving calculates paths according to the saliency map, and eliminates these with minimal importance value. 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. Since with every loop one unsalient pixel is eliminated from the horizontal or from the vertical paths, the relative saliency of the seams increase permanently. Therefore in most of the cases even the less salient seam needs to hit the relative value of the most salient path on the original image, where every unsalient pixel were included. 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, regular resampling has already more advantages. First the algorithm is faster then the implemented seam carving method. And second it applies an interpolating algorithm between the borders of the eliminated area, so it saves ultimately more information than seam carving in this case.

CHAPTER 4

Implementation

The illustrative thumbnail creator application is developed in the programming language C++. Except the source image file browsing 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, which measures the correlation of two histograms,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 in the default settings of the project the height is set to 1, the width 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, approximately the font size of 5pt in order to take also the smallest fonts into account, 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 at the top or at the bottom, 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

To make the project configurable the application uses a `config.txt` file, where every parameter can be set as needed. In order to find the configuration, which provides reasonable results in as many cases as possible several tests were performed. There is a test database having 14 pictures, showing seven different applications, captured on Windows 10. In the following the test cases of the elimination of UI elements, text detection and for the value of resampling threshold are presented and evaluated, which possibility provides the desired results. At last the application is compared to Adobe PhotoShop Forras, their advantages and disadvantages are discussed.

5.1 Elimination of UI elements

Depending on the investigation area for UI elements different regions of the border area can be cropped. If the border region is defined as too small (10% of the image size) the algorithm terminates too early and not all UI widget is cut off. But determining the border too big (40% of the image size) is also dangerous, because eventually the reference value for the central area can be chosen wrong, and finally not only the border area get cropped. Foto google, pdf

Since the histograms of the border area, the central area and the slices between them is sequentially compared, the value of its correlation is crucial by assigning them to one or to the other part of the image. If the correlation between the central area and the slice is set for too high, it means their correlation value has to be small (1), some content elements near the border can get rid of, by low correlation however some UI widgets can be labeled as important content. foto ppt, blizzard

The thickness of the slices influences, how refined the algorithm is when looking through the border area. If the slices are really slim (5% of the image size) the performance slightly decreases, but in exchange it is able to find a really close cropping point, where the UI and the content actually meets. foto desktop

5.2 Text detection

If the lines are already blurred on the grayscale image, the color of the letters and their area is slightly modified. With a threshold a minimal pixel color value is defined to get the pixel labeled as possible text. If this parameter is small (50) every bright area can be classified as text, even if the region is only near to a word or part of a text. But if it is set to a bigger value (200) only the core of a word can reach it, so no region will actually contour its related word's shape. foto 444

The properties of a possible word is also parametrized. The set of possible words need to be sorted first, if they according to their size they actually can be words. The parameters for minimum height and width need to be chosen carefully, since if they are too big (width is set to 100, by font size of 12pt more than 6 letters or height to 20, font size of 14pt), almost every word can get excluded from the list. foto mindegy From the set of the possible words the average size is calculated, and it is customizable how much the words are allowed to vary. For example if the difference can be only 80% the smaller words can easily get sorted out, while by 50% they can also manage to stay in the possible words set. foto pdf2

With the inversion of the algorithm, setting the weight of the words for negative, the functionality of the application can be changed. For experimental investigations the text areas were set for not salient, so the images and not the words on the image is saved. To save the image data is useful, when the thumbnail needs to get really small, so that it is no more possible, that the text remains readable. The experimental results are presented below. foto

5.3 Resampling threshold

The resampling thershold controls, when the application switches from seam carving to usual resampling. If the value is small (0.1) only a few seam carving loop is performed, and the application behaves similar to a common down sampling algorithm. the performance is positively affected by fewer seam carving loops, but it also loses its advantage against simple resampling methods, and the important areas are no more so easily recognizable. foto diablo

5.4 Comparison to Adobe Photoshop

To compare the algorithm to an already existing tool the seam carving feature of Adobe Photoshop CS 5 Forras was used. There are two test cases, the first one takes the same source like the illustrative thumbnail creator algorithm. The input image of the second case is however the cropped source image produced by this application after the elimination of the UI elements. In the first test it is investigated, how seam carving invented for usual photos works on screenshots. Comparing the images to the outputs of the thumbnail algorithm, it is striking how much smaller and less readable the the word are. Because this project applies not only seam carving but also heuristics to cut off the border region, to compare the actual seam carving algorithms Photoshop needs the same input as this application has, when it start the seam calculation. The second test case is needed for this reason. The quality of the output improves definitely, since a smaller image needs to get shrunk to the same size. The main difference between the two methods seems to be the weighting of the content. This approach pays more attention for the text and image data can easily get ignored. Photoshop however tries to keep both of the regions, perhaps the image content seems to be slightly more important. Since the thumbnail creator algorithm has a ranking between the elements, text before images, the output is clear, even if the image data is lost, the text stays readable. The output of Photoshop is rather disorganized, the text and image regions flow in each other. The images are significantly better recognizable, but the words are notably more damaged.

5.5 Performance evaluation

The test are performed on Windows 10 with the CPU Intel i3-2310, 2.1GHZ and 2 Cores and with the Grafic card AMD Radeon HD 7400M. During the tests some performance issues turned up, the application needed up to 3.5 minutes but in average 1 minute to terminate.

The performance bottleneck is caused by the seam calculation. Every time, when a seam is eliminated, the whole saliency map is recalculated. Having a new saliency map the possible seam paths also needs to get redetermined. If the UI cropping algorithm is not able to cut a bigger area or the resampling threshold is reached later, then many seam carving loop is performed. So screenshots with bigger and clearly defined UI areas have shorter run time, whereas inputs like gallery applications and computer games take longer time.

CHAPTER 6

Future Work

This application according to the vhapter before is 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. 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.