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

Thumbnails werden benutzt um eine Liste von geöffneten Fenstern und Tabs anzuzeigen, wenn auf Computern oder mobilen Geräten zwischen ihnen gewechselt wird. Diese Bilder erleichtern das Erkennen der offenen Applikationen, und helfen, dass das nötige Fenster schneller gefunden wird. Thumbnails sind aber nur ein verkleinerter Screenshot von den Fenstern; wenn aber Tabs oder die selbe Applikation mehrmals offen sind, werden sie leicht unübersichtlich. Abhängig von der Auflösung des Bildschirms werden die Thumbnails kleiner, wenn die Anzahl der offenen Fenster steigt. Außerdem sind Screenshots von der selben Applikation sehr ähnlich, zum Beispiel die Seite und Toolbar in MS Office Word, der Text auf der Seite ist aber nicht lesbar. Es gibt bereits mehrere Möglichkeiten, wodurch die beim Bearbeiten entstehenden Artifakte weniger auffällig und die wichtigen Regionen hervorgehoben werden können. In dieser Bachelorarbeit wird eine Applikation entwickelt, welche diese Methoden auf Screenshots anwendet und Thumbnails erstellt. Die unwichtigen Teile werden weggeschnitten, dannach wird das Bild mit Seam Carving bearbeitet, und wenn keine unrelevanten Informationen mehr übrig bleiben, wird herkömmliches down-sampling benutzt. Die Thumbnails zeigen also nur salient Informationen an, wodurch sie illustrativer sind und ihren Zweck besser erfüllen können.

Abstract

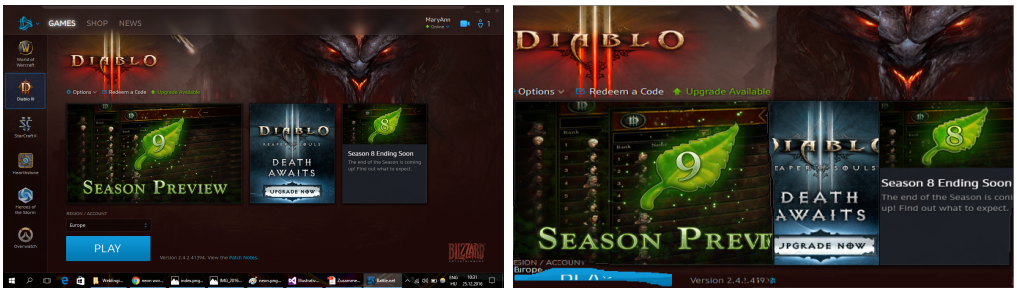
Thumbnails are used to display a list of open windows or tabs when switching between them on computers and on mobile devices. These images make it easier to recognize the opened applications, and help to find the needed window quicker. Thumbnails display however only a screenshot of the windows, so they 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 captured images. The less important areas of the screenshots are cut off, then seam carving is applied 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. However, the current thumbnail system is not 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. Figure 1.1 shows a possible output of the current (Figure 1.1a) and the new (Figure 1.1b) thumbnail creating algorithms.



(a) Currently used thumbnail (b) Thumbnail created by the new algorithm

Figure 1.1: The result using the two different approaches

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 thesis 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, additionally, 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. Chang et al. [CYM11] 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, Prefab [DF10] 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.

Sikuli [YCM09] 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 in terms of time and space, after accomplishing a training pattern, the Sikuli system is able to create new object models too. Although in the original 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 of the database and thus creating a more detailed database entries.

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. The algorithm of Dixon et al. [DLF11] 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 manipulated as needed. They can cut off as whole or processed according to the information content. Mirkamali et al. [MN15] 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 single 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. Rother et al. [RBHB06] takes the parameters representativeness, importance costs, transition cost and object sensitivity into account.

$$E(L) = E_{rep}(L) + w_{imp}E_{imp}(L) + w_{trans}E_{trans}(L) + w_{obj}E_{obj}(L)$$

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.

Egorova et al. [ESK08] 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.

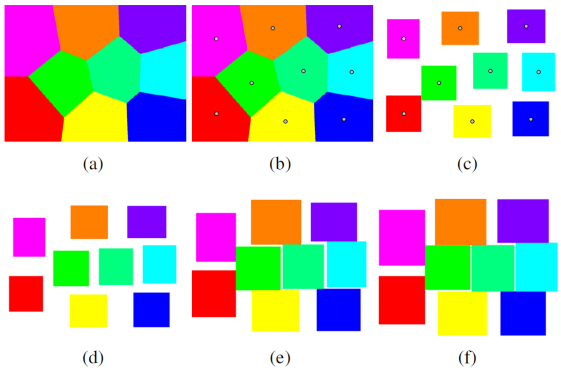


Figure 2.1: The whole ROI packing process [LSCP10] (a) K-means clustering (b) ROI's center at the beginning of the algoritm (c) Layout of the ROIs at the beginning of the algorithm (d) Shifting of the ROIs (e) Expanding the ROIs (f) Output after some iterations

Having the best ROIs for the collage the last task is to merge them into one output picture. For this purpose Lee et al. [LSCP10] 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. Figure 2.1 shows the different steps of the ROI packing algorithm.

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 Chen et al. [CXF+03] 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. The algorithm 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.

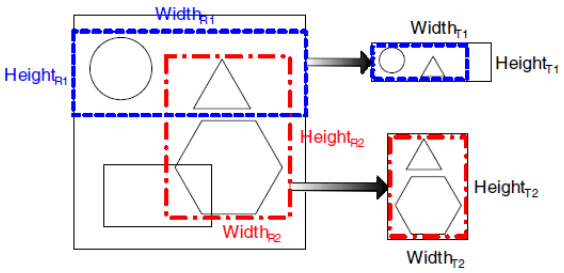


Figure 2.2: Possible solutions of the algorithm of Chen et al. [CXF+03]

The approach 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, like shown on Figure 2.2. With Feature-aware Texturing described in [GSCO06] 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, illustrated on Figure 2.3.

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

Figure 2.3: The resulting image and grid from a certain input image and its feature mask of the Feature-aware Texturnig algorithm [GSCO06]

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 Perez et al. [PGB03]. A modified Poisson equation using a guidance field needs to be solved 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 Hussain et al. [H+16] offers help. It proposes to solve the equation 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, Agarwala et al. [ADA+04] 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 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 illustrative thumbnail creating algorithm has three main steps, as shown on Figure 3.1. 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 the 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 re-sampling 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.

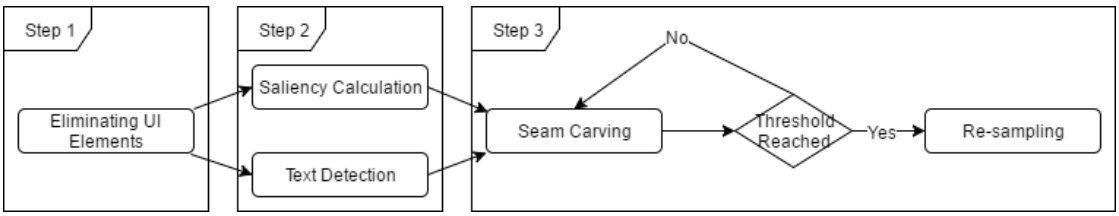


Figure 3.1: Flow chart of the algorithm

3.1 Eliminating UI elements

For the elimination of UI elements, first their identification must be accomplished. For the search of UI widget a heuristic is developed, which implies that these elements are located near the border of the screen and not in central areas. Thus, the middle of the screen is taken as reference data for the investigation and is not checked for the unimportant UI elements. The UI elimination algorithm works right on the source window in a predefined, parametrized border area, i.e. no further premodification is required for this process.

Cropping the borders, first the horizontal, then the vertical margins are investigated, allowing where their order is relevant. The algorithm is based on the observation that upper and lower bars expand through the whole width of the display. The sidebars, if they exist, run however between the upper and the lower bars, and cover the remaining areas of the margin. Furthermore, occasionally the sidebar have a slightly different style than the other UI widgets mentioned above. The UI matching method investigates slices of the screen running along the margins and examines whether they belong to the UI or to the middle area of the window. To get better results, it is important to investigate a coherent data, and not to mix parts of different UI bars to the same slice. The above described process with an established order of margin cropping is fit for this purpose.

To identify the best cropping line, indicating where the UI meets the content area, a double validation method is performed. The first step evolves searching for a line in the predefined border area, located however near to the center of the source, 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 unicolor background, as the examples illustrates on Figure 3.2. Therefore, the task is to find a line, where widgets are no longer located, but it still belongs to the UI area, so it contains a background color.

In case of predefined UI areas the method above works well, on the other hand, there are several specific occasions where it is not able to provide any results. Outstandingly, game applications usually use their own graphical UI, but even at more traditional applications it is plausible that the UI area is so overloaded or designed that no background line can be found. For this reason it is important to perform an additional checking loop, too. This step is based on the correlation value of the border and center's color histogram. The margin is split into thin slices. Initially the first slice near the border is examined. The histogram of this slice is then compared to the histogram of the center. If their correlation is high, it implies that the two areas are not significantly different, therefore nothing should be cut off, and so the algorithm returns. Alternatively, if low correlation is found, the examined strips are not part of the same unit, indicating that the slice is supposedly from within the UI area. In this case, the two histograms, the one with the first slice and the other from the center region, are kept for/as reference values. In the next step, the remaining slices are compared with the two reference histograms, starting at the border and heading towards 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 i.e. it matches the theme of the border widgets. This tendency reverses

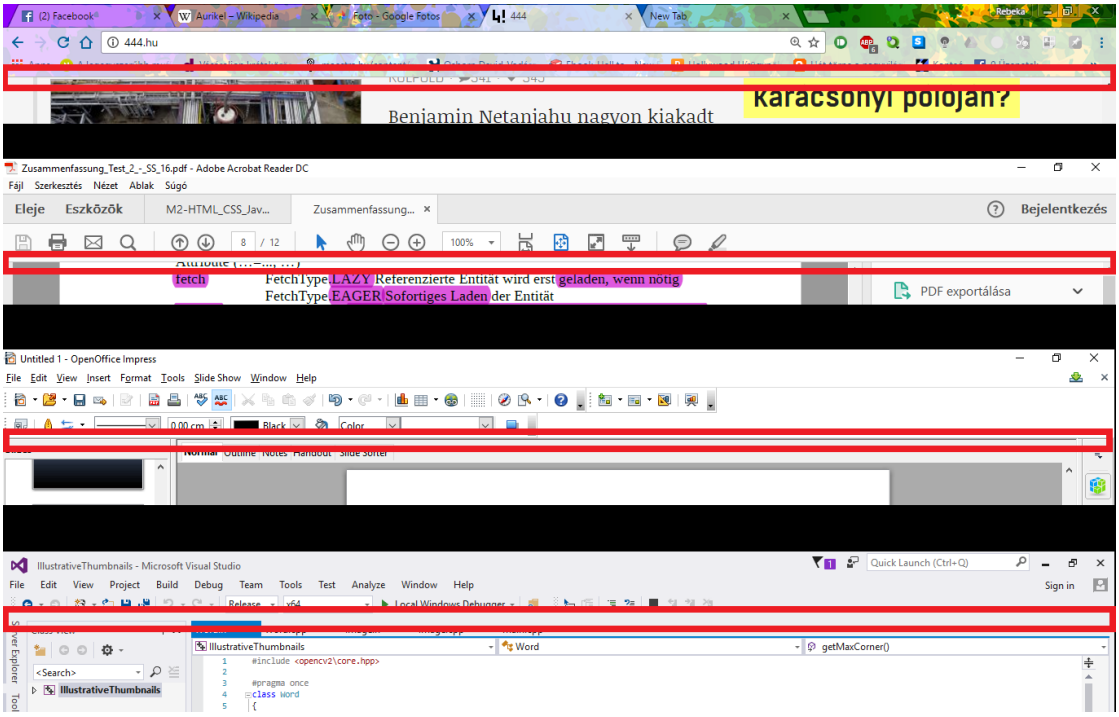


Figure 3.2: Unicolor line at the edge of the UI area

however as the slices approach the center regions. At the slice whose correlation with the center histogram is higher than the one with the border histogram, the algorithm stops, cuts all of the previously examined slices, and then terminates. Figure 3.3 shows the borders between the content and UI regions found by the UI cropping heuristic.

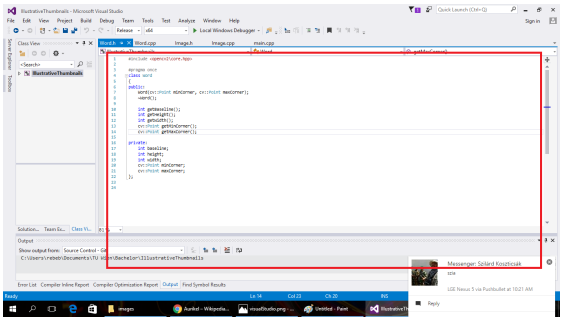


Figure 3.3: The border of the content

3.2 Saliency calculation

In order to ignore that parts of the source image that are actually unimportant, saliency calculation is applied. 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 the region is more interesting than other pixels with lower values. There are, however, many definitions and approaches about which pixels should be evaluated as salient or not salient. In this case the saliency is calculated as presented by Niu et al. [NLLG12]. There are two reasons, why this algorithm is chosen. First, it evaluates the low level visual informations. It means that only those pixels are valued as important that activate the low level human visual system, for example due to their sharp edges and color change characteristics. 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. It calculates the contrast value from the weighted sum of difference between the pixel and its neighborhood, using the L2 norm:

$$C_{i,j,l} = \sum_{q \in \Theta} w_{i,j,l} d(p_{i,j,l}, p_q)$$

$$w_{i,j,l} = 1 - \left(\frac{r_{i,j,l}}{r_{l,max}} \right)$$

where:

C	= Contrast value
(i, j)	= pixel coordinates
l	= pyramid level
Θ	= neighborhood
w	= weight
d	= difference
p	= color of the pixel
r	= distance from the image center
$r_{l,max}$	= maximum possible distance on the image

This approach needs to get slightly modified, in way that the calculation ignores the weighting value. The weighting parameter is used because of the fact that the central area of regular real word images is more important than the margin regions. When investigating computer screens, however, this is not the case. A screenshot often does not have a defined focus point where the most important information is shown. By using the weighting value, the focus would be on the inner window, despite the fact that there is no actual significant correlation between the importance of the data and its position on the screen. At the end, the importance value of each pixel is calculated by summing up all levels of the pyramid into one final saliency map.

In the case of computer screens, texts are fairly common, and their information content is usually also very high. Therefore, it would be rather impractical not to reflect the

(a) Input image (b) Output

Figure 3.4: Result of the text detection algorithm applied on the given source

importance of such text regions on the saliency map. For that reason a further evaluation of the saliency map is needed, to check whether text areas were evaluated as important data.

The text recognition algorithm is based on the one introduced in [CYM11]. For the first step the whole input image is horizontally blurred, so that the letters and the neighboring words form a string together. After that, all of these coherent strings are investigated and consorted depending on whether they are representations of actual words or if they derive from another visual feature of the source. The method that identifies the words examines two aspects. The first is if the strings are high enough but not too high to indicate real letters. Second, whether their width is not too small but also not too big to form at least one word but not an endless sentence. Both of these size examining values are parametrized, with their default value based on the size of one letter, as explained in the next chapter. The second loop evaluates the histograms of the area, where the strings were found. Normally the letters, if belonging to the same text data, have the same color, while the background, for optimal readability, does not change its color underneath the text, either. Therefore, the histograms of these areas are bimodal, namely one for the letters and one for the color of the background. To sum up, to identify a string as a word, it is not sufficient to pass the first check, it also need to have a corresponding special histogram to be labeled as word. Figure 3.4 shows the output of the text detection algorithm. The light blue color labels the possible word regions, the darker color denotes the actual words found by the algorithm.

The last step for the calculation of the final saliency map, which is consequently used in the whole application, is to get the identified text regions weighted. All area which passed the word checking test is automatically evaluated as highly important, and gets the highest saliency value. In addition, to make sure that even the space between the words remains undamaged, which is important because of greater readability, the area neighboring the text data is also weighted. Figure3.5 is the final saliency map of Figure 3.4a.

Figure 3.5: The final saliency map

3.3 Seam Carving

To resize the image without damaging the important regions, seam carving [AS07] is used. Seam carving calculates paths according to the saliency map, and eliminates those 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 the straight lines, the algorithm noticeably effects the layout of the unimportant regions. Furthermore, if the whole image has very high salient values, the resulting seams do not have a remarkably smaller salient value than any straight line would have had when chosen randomly for re-sampling. To save the image from unnecessary artifacts and increase the application performance, a threshold value is defined as Dong [DZPZ09] proposed. To reach the final output size common re-sampling is applied.

To find the most appropriate path, every possible solution need to be examined using the backpacking method. A path map is thus generated, where all past possibilities are stored, and that every new try can use as a look up. This way performance can be saved and the algorithm becomes faster.

To find the next pixel of the path, five-neighborhood of the next possible member is examined. The algorithm runs from the top to the bottom, where the path to the next member is chosen from the five nearest pixels of the previous line. To calculate the costs of a possible switch between the columns the importance value of the neighboring pixels are weighted accordingly. In each case the chosen pixel is the one whose importance value with the weighting parameter is the smallest. Algorithm 3.1 shows the flow of the seam carving process. Two paths per loop are calculated, one horizontally and one vertically, however, in the end only the pixels of the least salient path become eliminated.

Because the horizontal and vertical paths are eliminated independently, the picture being processed usually does not hold the aspect ratio. For this reason it is plausible that the weight and height parameters do not reach the output size or the threshold value at the same time. In this case the seam carving algorithm continues only for the remaining parameter until it reaches one of the conditions above. The value of the threshold parameter is essential, since it determines when the algorithm is to switch between seam carving and re-sampling. It is calculated from the maximum salient path found after

Algorithm 3.1: The seam carving algorithm

Input: the cropped source image *image*, the importance map *saliencyMap*, two dimensional vector of the size of *image* storing the saliency value and the previous path *pathValues*

Output: seam carved *image*

```
1  for y < width of image do
2      for x < height of image do
3          previousPixel = detect the less salient pixel of the five-neighborhood in the
           previous line in pathValues, the saliency value of pathValues at (x,y) =
           previousPixel + saliencyMap at (x,y), the path value of pathValues at (x,y)
           = position of previousPixel, increase x
4      end
5      increase y
6  end
7  lessSalientPixel = detect the smallest saliency value in the last row of pathValues
   for j = height of image, j ≥ 0 do
8      remove the pixel from image with the coordinates of lessSalientPixel,
       lessSalientPixel = the pixel in pathValues with the coordinates of the path
       value of lessSalientPixel, decrease j
9  end
10 return (image)
```

the first loop of the seam carving algorithm. Since with every loop one unsalient pixel is eliminated from the horizontal or from the vertical path, the relative saliency of the seams constantly increases. Therefore, in most cases, even the least salient seam will get more salient than the most salient seam of the original image. The common re-sampling method eliminates straight lines chosen from the source image until the desired size is reached. Seam carving is applied to prevent paths with high salient value to be cut off. On the other hand, if the seams have as high importance value as any other of the source image, regular re-sampling has additional advantages. First, the algorithm is faster than the implemented seam carving method, and second, it applies an interpolating algorithm between the borders of the eliminated area, so in this case it ultimately saves more information than seam carving.

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 in the future, only this part needs customization. For image processing purposes Open Source Computer Vision Library (OpenCV) is used [Bra].

OpenCV is apart from the computer vision also a machine learning library. It has more than 2500 algorithms that can be used for image processing or for machine learning tasks, among object identification, finding similar images, image stitching and so on. Since the library offers support not exclusively for Windows, but also for Linux, Mac OS and Android, all methods taken from OpenCV can be regarded as platform independent functions. The library is applied when loading and saving the images, and to perform several image processing tasks like converting between color spaces, and image editing task such as blurring or filtering. The actual use of the library will be discussed in the section below.

4.1 Working Pipeline

In the following the actual implementation of the application is described. Apart from 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 source code. A list of these parameters are highlighted below.

4.1.1 Image loading

Preceding the start of the thumbnail algorithm it is essential to load the source image. To make the application more flexible a Windows call is performed, allowing a new source image to be chosen after each start of the application. The type `OPENFILENAME` [Win17],

part of the Windows API, launches a file window to browse the required file. It shows only image files with the extension `.jpeg` or `.png`. After choosing the preferred source, the file path is established 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 works with a configurable parameter, which defines the sections of the screen that are to be regarded as a margin area, using the default value of 20%. Starting from the border towards the direction of the middle of the source, every row and after then, every column, are searched for a specific source line that first, has the same color and second, a completely filled space between the vertical - later the horizontal - borders. Once accomplished, the OpenCV methods `rowRange` and `colRange` cut the input image. After cutting, the histogram of the remaining margin area is examined and compared to the central region, the margin is split along the nearest border into thin slices. The function `calcHist` calculates histograms of the marginal and of the central window. The function `compareHist` measures the correlation of two histograms with the equation [Ope17]:

$$d(H_1, H_2) = \left(\frac{\sum_J (H_1(J) - H_1)(H_2(I) - H_2)}{\sqrt{\sum_J (H_1(J) - H_1)^2 \sum_J (H_2(J) - H_2)^2}} \right)$$

where:

$$H_k = \left(\frac{1}{N} \right) \sum_J H_k(J)$$

$$N = \text{Amount of the bins}$$

If according to it they do not correlate, the comparison is performed on other slices that are consequently labeled as margin or central windows. If a slice is found that 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 testings, the best results are provided when the correlation is between 1.3 and 2 and the width is the 0.25% of the window.

4.1.3 Saliancy map calculation

The saliency map is calculated from a Gaussian pyramid. But before building the pyramid the cropped source image has to get converted into a uniform colorspace. For this purpose the OpenCV function `cvtColor` is called. Then the levels of the pyramid are determined 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 to merge the contrast

images into one complete saliency map. Upscaling of the levels is performed, so they have the same size to the cropped image. The pixel value of the saliency map is given from the sum of pixels with the same coordinates for every level of the contrast pyramid. In order to avoid any overflow, the pixel values are divided by the amount of 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. As a result of the edge detection the layout of the letters are now highlighted. To make the series of letters related to each other, a horizontal blur is applied. The function `blur` is called. Although its kernel size is configurable, by default the height is set to one, the width to 15 pixels. Finally, the OpenCV function `findContours` identifies the related regions and save their silhouette points into a 2 dimensional `vector`. For each region the width and height values are calculated. If a region is at least double as wide as high, the function proceeds with the examination of these attributes, namely whether they reach the minimum size for being an actual word. The size of the minimal word is configurable and by default set to 5x5, approximately the font size of 5pt, in order to be able to take even the smallest fonts into account. In case a region passes the test, it is marked as a possible word and is forwarded for further investigation. After labeling every region as word or non word region, the average height of the possible words is calculated. If the height of a possible word is significantly smaller or greater than the average, it is automatically sorted out. This confidence interval is also configurable and it is set to 50% and 1000%. The maximal word height is defined generously in order to be able to find also the potentially large-sized titles and headlines. After the height test, the histogram of the possible word regions is examined. If it has exactly two accumulations, one for the color of the letters and one for the background, the region stays in the group of possible words, otherwise it is sorted out. Finally, the space above and below the possible words is investigated. Words are usually written in lines, with some space for readability placed between them. Therefore if two possible word regions overlap i.e. have common points at the top or at the bottom, it is inconceivable that the region in question represents a real words and thus is sorted out of the word list. When the final list is ready, the regions of the words are to be marked on the saliency map. The value of every text data pixel is automatically increased. At the very end the whole map is normalized with the OpenCV function `normalize` to avoid value overflow and other irregularities.

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 re-sampling value or

the desired output size the seam carving algorithm is executed. The re-sampling value is calculated from the average pixel saliency value of the most salient seam on the cropped source image. The desired size and the re-sampling parameter are both customizable, they are set by default to 50%. The seam calculation algorithm works by writing a path map, where the previous paths and their saliency value are 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 the least salient predecessor of every pixel is searched, and the saliency value at the coordinates of the current pixel is increased with the predecessor saliency value. If the new saliency value is smaller than the one already stored at the coordinates of the current pixel, then the entry is updated and thus a better path is saved. By looking for the predecessor pixel the algorithm works with 5-neighborhood. The saliency values of the five nearest pixels are weighted according to their distance from the current pixel with $\sqrt{5}$, $\sqrt{2}$ and one. The pixel that is finally the least salient is chosen as predecessor. When the algorithm finishes, in the end row of the path map the saliency values of the possible ways for 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 identified this way is afterwards eliminated not only from the cropped source image, but also from its saliency map. But if the saliency value of even the least expensive seam exceeds the re-sampling value, instead of cutting the path off, the usual re-sampling 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 including 14 pictures, showing seven different applications, captured on Windows 10. The test images and the results in case of the use of the recommended `config` file are listed in Appendix A. In the following the test cases of the elimination of UI elements, text detection and for the value of re-sampling threshold are presented and evaluated in respect to which of them is best able to provide the desired results. Then the application is compared to Adobe Photoshop [Inc], with their advantages and disadvantages are described. At last the performance issues of the application is discussed briefly.

5.1 Elimination of UI elements

Depending on the investigation area for UI elements, various regions of the border area may be cropped. If the border region is defined too small (10% of the image size) the algorithm terminates too early and not all UI widgets are cut off. But setting the border value too big (40% of the image size) is also dangerous, because eventually the reference value for the central area can be chosen wrong, resulting in cropping from additional, non-border areas.

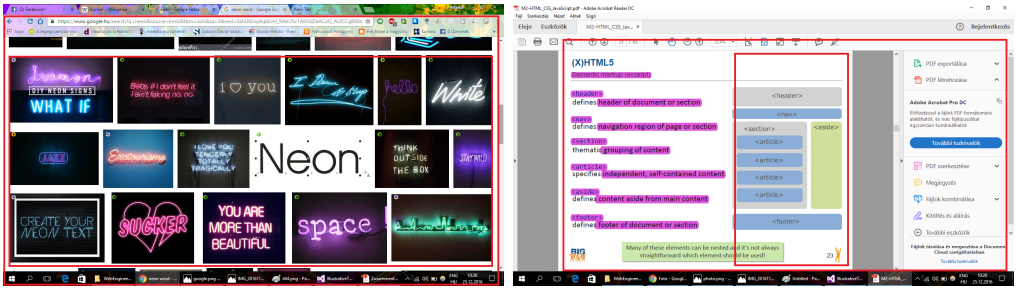


Figure 5.1: The cropping points according to the border area 10% and 40%

Starting from the histograms of the border area, the central area and the slices between them are sequentially compared, the value of their correlation is crucial for assigning them to one or another 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 be eliminated, at low correlation (2.5) however some UI widgets can be labeled as important content.

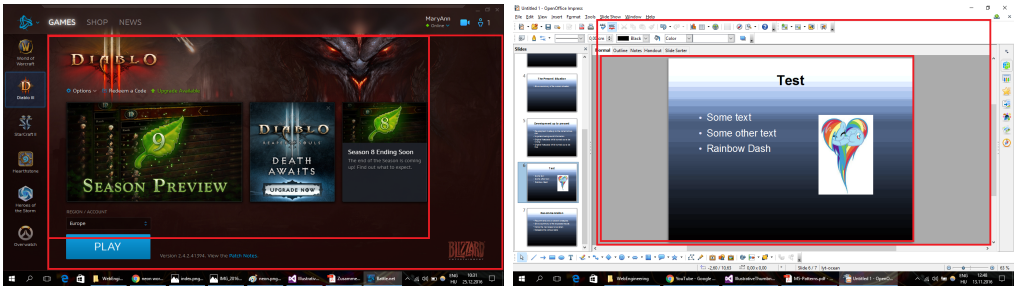


Figure 5.2: The cropping points according to the correlation 1 and 2.5

The thickness of the slices influences how refined throughout the algorithm is when searching 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, i.e. where the UI and the content actually meets.

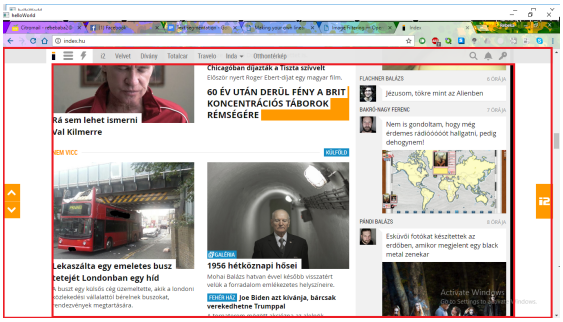


Figure 5.3: The cropping points according to the thicknes of the slices 12% and 5%

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 the implementation of a threshold range a minimal pixel color value is defined that is in place to have the pixel labeled as possible text. If this parameter is small (50) any bright area can be classified as text, even if the given region is only neighboring a word. But if it is set to a high value (200), only the core of a word will reach it, so no region will actually contour its related word's shape. Bright blue contours the related area, the darker color however shows the regions classified as actual word.

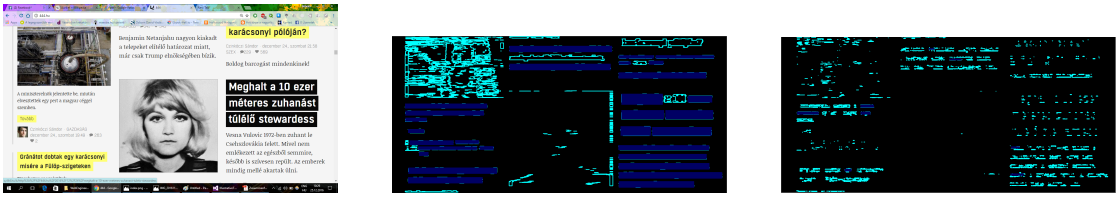


Figure 5.4: The related areas according to the brightness threshold value 50 and 200

The properties of a possible word are also parametrized. The set of possible words needs to be sorted first, depending on how likely they are potential words when considering their size. The parameters for minimum height and width need to be chosen carefully, since if they are too big (width is set to 100, font size of 12pt, more than 6 letters or height to 20, font size of 14pt), almost every word will get excluded from the list.

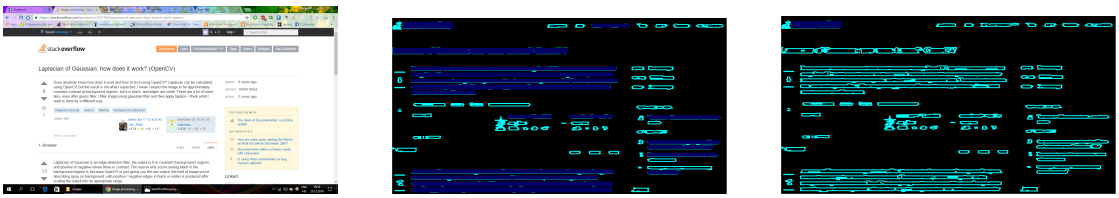


Figure 5.5: The actual words according to the minimal width 100 and minimal height 20

From the set of the possible words the average word size is calculated, but after then it is still further customizable how much the words are allowed to vary from this value. For example if the allowed difference is set to 80% of the original size the smaller words can easily get sorted out, while at 20% they also manage to stay in the possible words set.

Figure 5.6: The actual words according to the maximal allowed difference 80% and 20% from the average

With the inversion of the algorithm, setting the weight of the words for negative, the behavior of the application can be changed. For experimental investigation the text areas were set non-salient, so only images and no words are saved. When really small thumbnails are required, it is advisable to save and present the image data instead of texts, since they would be illegible due to their minuscule size. The experimental results are presented below.

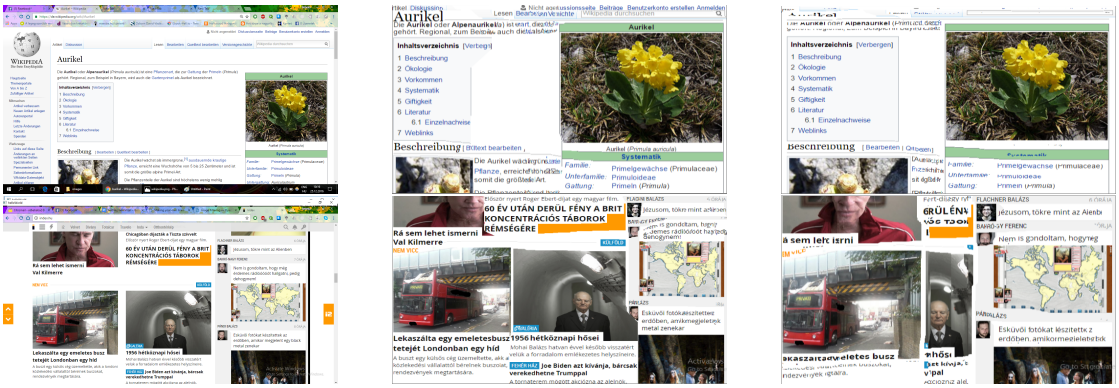


Figure 5.7: The output of the current and the modified algorithms

5.3 Re-sampling threshold

The re-sampling threshold determines when the application switches from seam carving to usual re-sampling. If the value is small (0.1) only a few seam carving loops are 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 re-sampling methods, with the important areas being no longer easily recognizable. When the threshold is set, however, to a bigger value (0.4), the

application switches at the very end of the process. In this case the application works noticeably slower and additionally in causes more artifacts than usual re-sampling.

Figure 5.8: The results according to the re-sampling threshold 0.1 and 0.4

5.4 Comparison to Adobe Photoshop

To compare the algorithm to an already existing tool the seam carving feature of Adobe Photoshop CS 5 [Inc] was used.

Figure 5.9: Source images used for testing

There are two test scenarios; the first takes the same source to the illustrative thumbnail creator algorithm. The input image in 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.

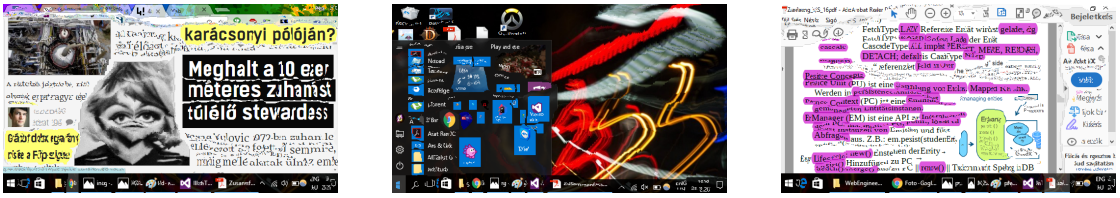


Figure 5.10: Results of the first scenario

Comparing the images to outputs of the thumbnail algorithm, it is striking how much smaller and less readable the words appear. 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 starts the seam calculation. The second test scenario is in place for this reason.

Figure 5.11: Results of the second scenario

The quality of the output definitely improves, since an already smaller image needs to shrunk to the same size as before. The main difference between the two methods seems to lay on the weighting of the content. This approach pays more attention to the text, with image data being easily ignored.

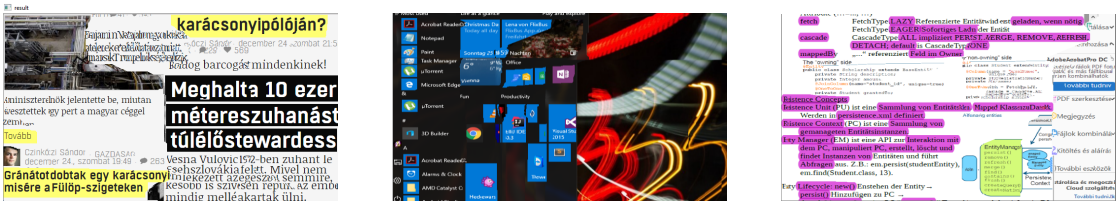


Figure 5.12: Result of the thumbnail algorithm

Photoshop, however, attempts to sustain both regions, perhaps with a slightly more emphasize on the image content. Since the thumbnail creator algorithm has a ranking between the elements, holding texts before images, the output is not dubious: even in case of image data loss, the text remains readable. The output of Photoshop is rather disorganized, the text and image regions flow into each other. The images are significantly better recognizable, but in exchange the words are noticeably more damaged. All results produced by Photoshop are listed in Appendix B.

5.5 Performance evaluation

The tests were performed on Windows 10 with CPU Intel i3-2310, 2.1GHZ and 2 Cores, using Grafic card AMD Radeon HD 7400M. During the tests some performance issues occurred, the application occasionally needed up to 3.5 minutes performance time, however remaining in a one-minute termination average.

The performance bottleneck is caused by 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 need to get redetermined. If the UI cropping algorithm is not

able to cut a bigger area or the re-sampling threshold is reached later, several seam carving loops are performed. Therefore, screenshots with bigger and clearly defined UI areas have shorter run time, whereas inputs like images of gallery applications and of computer games take longer.

CHAPTER 6

Future Work

This application, according to the chapter above, is effective at creating illustrative thumbnails, but there are certain implementations to extend its scope. There are two main improvement areas to investigate: performance and software methodology. Although the current application provides reasonable results, as described in the preceding chapters, there are several approaches, to make the present method even more robust 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. Optimally, every pixel and every possible path are needed to be examined in order to find the most favorable path. Depending on the size of the input, 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 [Coo13], which exploits the computing performance of the GPU. In addition, the CUDA system can be simply integrated into the current application, since it is developed for programming languages C, C++ and Fortran, that includes the C++ used in this project.

6.2 Methodology improvement

An essential challenge in seam carving is the construction of the saliency map and the calculation of the seams. The methods implemented g in the current project are able to handle most standard cases, on the other hand, with some improvements it has the potential to became more robust and usable in even more special cases.

There are several approaches that besides the low level pixel data examination, aims to find also semantical objects in the input. They are building on the observation, that

some objects, for example faces, even if their coloring is not special, are very important for human viewers. The perceptual seam carving algorithm from Hwang et al. [HC08], based on the Human Attention Model, calculates not only the color changes but also the occurrence of facial information. It includes an already implemented feature of OpenCV, with which 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 an even more detailed saliency map is constructed.

Furthermore, apart from the face information, Domingues et al. [DAV10] also uses gradient magnitude, Canny edge detection and Hugh line detection to make the saliency map as meaningful as possible. It thoroughly examines both the semantic and the low level pixel data. There is, however, another method to evaluate the pixel information in an even more profound manner. Putting the information into a context makes the measurements more accurate, because it controls for information extremities. Consequently, no pixel will appear more important - or unimportant - than it really is. Guo et al. [GDT+15] obtains this with the calculation of a 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 i.e. indicate no salient regions. The saliency map, where accordig to the algorithm also the insight of the object is filled, is showed on Figure 6.1.

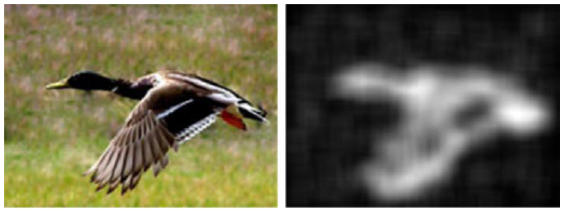


Figure 6.1: The source image and the saliency map of the NIF algorithm [GDT+15]

To find the coherent areas that later can be defined as objects, the use of a mean shift algorithm, as Liu et al. [LG06] does, would be effective. Furthermore, this algorithm makes the saliency calculation scale invariant, and decreases the chance of finding misleadingly high important edges. The resulting saliency map is showed on Figure 6.2.

Figure 6.2: The source image and the saliency map of the mean shift algorithm [LG06]

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 information of each pixel. On the other hand,

moderation of this rule can provide better results with less artifacts and also increases of performance, as obtained by Domingues et al. [DAV10]. 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 slightly greater salient region, the number of needed loops per 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.

In order to eliminate the possible artifacts caused by cutting of seams and streams an algorithm presented by Sun et al [SYJS05] offers help. This approach is also based on 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 then labeled as anchor points; the actual task is to find a connection between these anchors and to color the background, as shown on Figure 6.3.

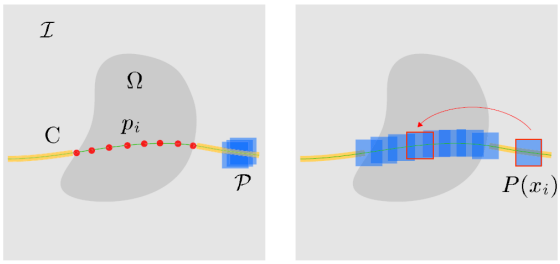


Figure 6.3: The structure propagation algorithm, where Ω denotes the unknown regions and p_i the anchor points [SYJS05]

In case of seam carving the goal is not to recreate the eliminated areas but with the above algorithm it is possible to make transition areas smoother. If a stream is taken, a narrow, only a few pixel wide 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 presented by Dong et al. [DZPZ09]. This method has to slightly modify the pipeline of the project, since it employs usual re-sampling 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 re-sampling. With this approach the properties of the background have no influence on the actual sequence of seam. The calculation needs to take only the saliency values of the important foreground objects into account, therefore the resulting path will be more accurate and consist the most important regions of the picture. Furthermore, in this case a seam is shorter than it would be when it was be applied to the whole input, as it is only calculated for certain foreground objects, causing

significantly smaller and less noticeable artifacts. As an additional favorable side effect, also the performance increases due to the decreased amount of areas to be considered for the seam calculation.

CHAPTER 7

Conclusion

To make thumbnails more illustrative and applicable even on small screens, a thumbnail creating method using seam carving was presented in this paper. In order to obtain the most informative results 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. As a combined results of the customized seam carving algorithm and its extensions with other helpful methods such as UI part elimination, text recognition and common re-sampling, the created thumbnails are more illustrative than their usual relatives, since their information content is better recognizable and the text regions are easier to read.

Bibliography

[ADA+04] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 294–302. ACM, 2004.

[AS07] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM Transactions on graphics (TOG)*, volume 26, page 10. ACM, 2007.

[Bra] G. Bradski. *Dr. Dobb's Journal of Software Tools*.

[Coo13] Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[CXF+03] Li-Qun Chen, Xing Xie, Xin Fan, Wei-Ying Ma, Hong-Jiang Zhang, and He-Qin Zhou. A visual attention model for adapting images on small displays. *Multimedia systems*, 9(4):353–364, 2003.

[CYM11] Tsung-Hsiang Chang, Tom Yeh, and Rob Miller. Associating the visual representation of user interfaces with their internal structures and metadata. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 245–256. ACM, 2011.

[DAV10] Daniel Domingues, Alexandre Alahi, and Pierre Vandergheynst. Stream carving: an adaptive seam carving algorithm. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 901–904. IEEE, 2010.

[DF10] Morgan Dixon and James Fogarty. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1525–1534. ACM, 2010.

[DLF11] Morgan Dixon, Daniel Leventhal, and James Fogarty. Content and hierarchy in pixel-based methods for reverse engineering interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 969–978. ACM, 2011.

[DZPZ09] Weiming Dong, Ning Zhou, Jean-Claude Paul, and Xiaopeng Zhang. Optimized image resizing using seam carving and scaling. In *ACM Transactions on Graphics (TOG)*, volume 28, page 125. ACM, 2009.

[ESK08] Marta Egorova, Ilia Safonov, and Nikolay Korobkov. Collage for cover of photobook. *Proc. GRAPHICON-2008*, pages 160–163, 2008.

[GDT+15] Dongyan Guo, Jundi Ding, Jinhui Tang, Min Xu, and Chunxia Zhao. Nif-based seam carving for image resizing. *Multimedia Systems*, 21(6):603–613, 2015.

[GSCO06] Ran Gal, Olga Sorkine, and Daniel Cohen-Or. Feature-aware texturing. *Rendering Techniques*, 2006(17th):2, 2006.

[H+16] Khaled Hussain et al. Efficient poisson image editing. *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, 14(2):45–57, 2016.

[HC08] Daw-Sen Hwang and Shao-Yi Chien. Content-aware image resizing using perceptual seam carving with human attention model. In *Multimedia and Expo, 2008 IEEE International Conference on*, pages 1029–1032. IEEE, 2008.

[Inc] Adobe Systems Incorporated. Adobe photoshop cs5.

[LG06] Feng Liu and Michael Gleicher. Region enhanced scale-invariant saliency detection. In *Multimedia and Expo, 2006 IEEE International Conference on*, pages 1477–1480. IEEE, 2006.

[LSCP10] Man Hee Lee, Nitin Singhal, Sungdae Cho, and In Kyu Park. Mobile photo collage. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 24–30. IEEE, 2010.

[MN15] Seyed Saeid Mirkamali and P Nagabhushan. Object removal by depth-wise image inpainting. *Signal, Image and Video Processing*, 9(8):1785–1794, 2015.

[NLLG12] Yuzhen Niu, Feng Liu, Xueqing Li, and Michael Gleicher. Image resizing via non-homogeneous warping. *Multimedia Tools and Applications*, 56(3):485–508, 2012.

[Ope17] OpenCV. *Histograms*, accessed January 28, 2017.

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 313–318. ACM, 2003.

[RBHB06] Carsten Rother, Lucas Bordeaux, Youssef Hamadi, and Andrew Blake. Autocollage. In *ACM transactions on graphics (TOG)*, volume 25, pages 847–852. ACM, 2006.

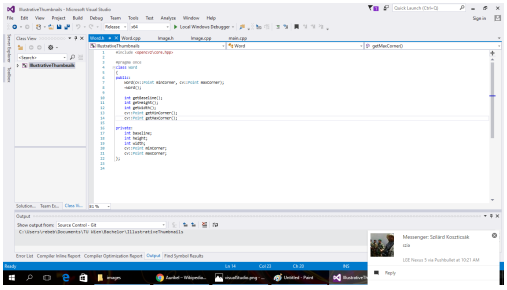
[SYJS05] Jian Sun, Lu Yuan, Jiaya Jia, and Heung-Yeung Shum. Image completion with structure propagation. *ACM Transactions on Graphics (ToG)*, 24(3):861–868, 2005.

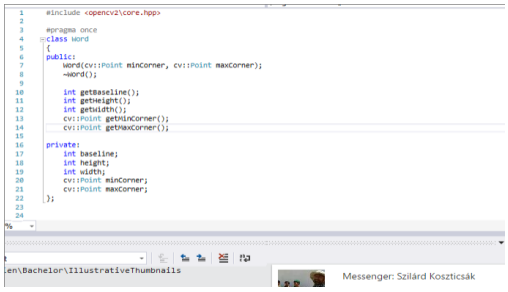
[Win17] Windows. *OPENFILENAME structure*, 2008 (accessed January 28, 2017).

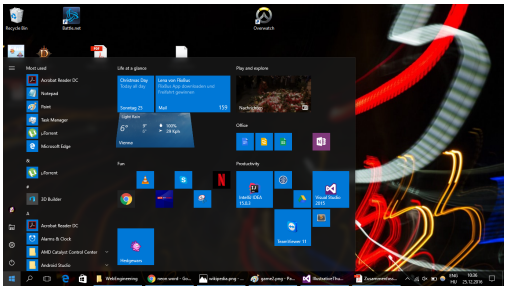
[YCM09] Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller. Sikuli: using gui screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 183–192. ACM, 2009.

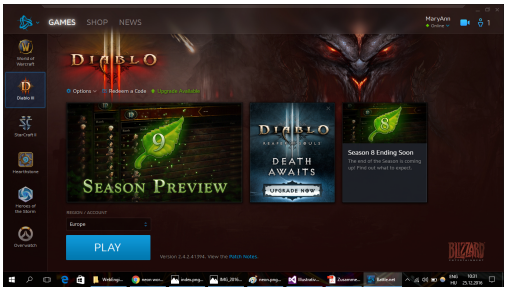
Appendix A

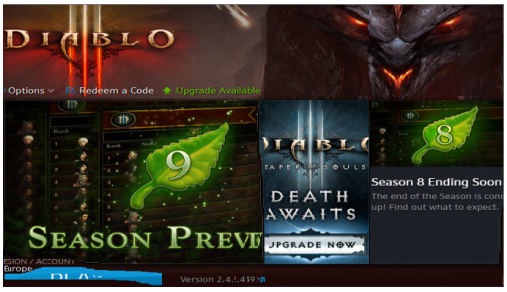
This section contains the source and the result images using the recommended `config` settings. The border area is 20%; the correlation of the lower and the right corner is 1.3, the upper and the left however 2.0; the thickness of the slices is 2.5%; the brightness threshold for text detection is 100; the minimal word height and also the width is defined as 5; the difference to the average height is 50% and 200% and the re-sampling threshold is set to 50%.







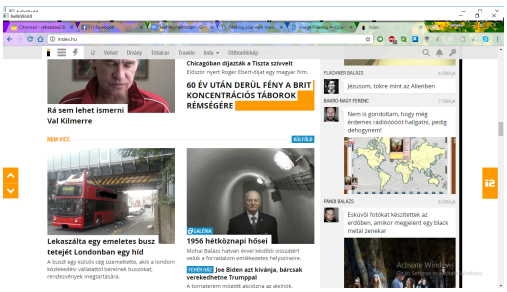

GAMES
SHOP
NEWS
DIABLO
SEASON PREVIEW
DEATH AWAITS
UPGRADE NOW
Season 8 Ending Soon
PLAY


DIABLO
Options
Redeem a Code
Upgrade Available
DEATH AWAITS
UPGRADE NOW
Season 8 Ending Soon

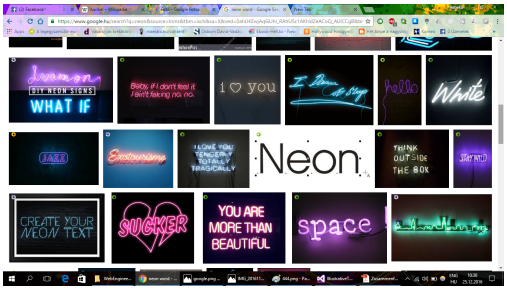

DIABLO
GAME MENU (ESC)
A ROYAL AUDIENCE
HARD


DIABLO
GAME MENU (ESC)
OPTIONS
ACHIEVEMENTS
CUSTOMER SERVICE
LEAVE GAME
EXIT DIABLO III
A ROYAL AUDIENCE
HARD

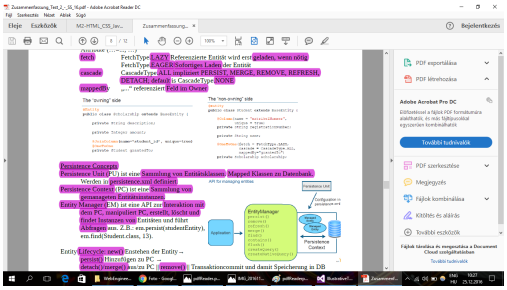

OBJECTIVES
A Royal Audience
NEW! Escape the Imperial Palace

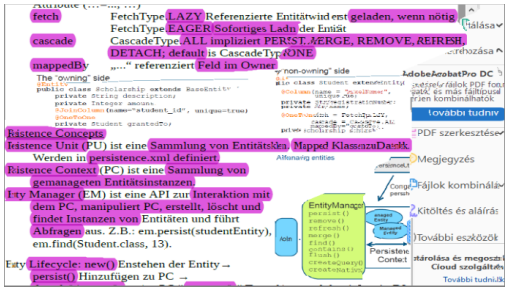

Chicagóban díjazták a Tiszta szívvel
60 ÉV UTÁN DERÜL FÉNY A BRIT KONCENTRÁCIÓS TÁBOROK RÉMSÉGÉRE
Rá sem lehet ismerni Val Kilmerre
Lekaszálta egy emeletes busz tetejét Londonban egy híd
1956 hétköznapi hősei
Joe Biden azt kívánja, bárcsak verekedhetne Trumppal

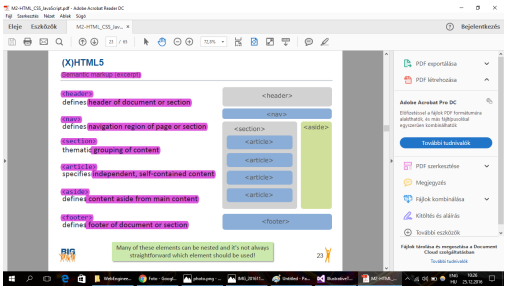

60 ÉV UTÁN DERÜL FÉNY A BRIT KONCENTRÁCIÓS TÁBOROK RÉMSÉGÉRE
Rá sem lehet ismerni Val Kilmerre
Lekaszálta egy emeletes busz tetejét Londonban egy híd
1956 hétköznapi hősei
Joe Biden azt kívánja, bárcsak verekedhetne Trumppal

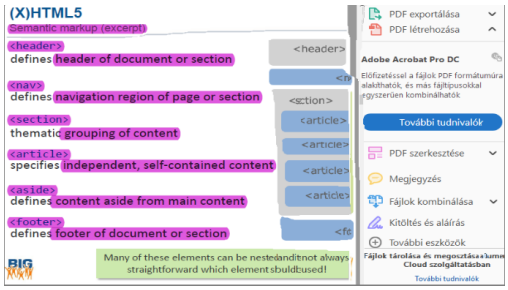

DIY NEON SIGNS
WHAT IF
Baby, if I don't feel it I ain't faking no no.
I ♡ you
Hello
Write
JAZZ
Encourage
I LOVE YOU SO TOTALLY TRAGICALLY
Neon
THINK OUTSIDE THE BOX
CREATE YOUR NEON TEXT
SUCKER
YOU ARE MORE THAN BEAUTIFUL
space


DIY NEON SIGNS
WHAT IF
JAZZ
Encourage
REATE YOUR
/EON TEXT
YOU ARE MORE THAN BEAUTIFUL
space

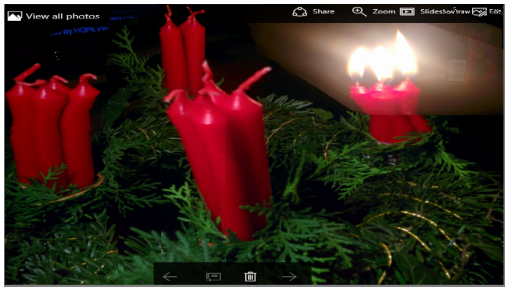

fetch
cascade
mappedBy
Persistence Context
Persistence Unit (PU) ist eine Sammlung von Entitätsklassen, Mapped Klassen zu Datenbank.
Werden in persistence.xml definiert.
Persistence Context (PC) ist eine Sammlung von gemanageten Entitätsinstanzen
Entity Manager (EM) ist eine API zur Interaktion mit dem PC, manipuliert PC, erstellt, löscht und findet Instanzen von Entitäten und führt Abfragen aus. Z.B.: em.persist(studentEntity), em.find(Student.class, 13).
Entity Lifecycle: new() Erstehen der Entity →
persist() Hinzufügen zu PC →
remove()
Persistence Context

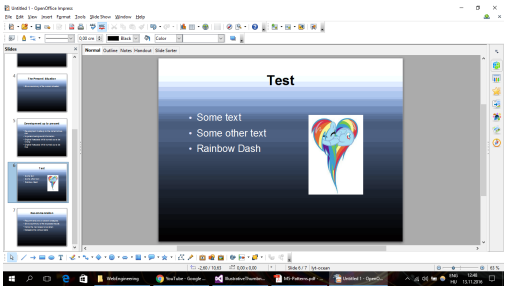

fetch
cascade
mappedBy
Persistence Concepts
Persistence Unit (PU) ist eine Sammlung von Entitätsklassen, Mapped KlassenzuDatenb.
Werden in persistence.xml definiert
Persistence Context (PC) ist eine Sammlung von gemanageten Entitätsinstanzen
Entity Manager (EM) ist eine API zur Interaktion mit dem PC, manipuliert PC, erstellt, löscht und findet Instanzen von Entitäten und führt Abfragen aus. Z.B.: em.persist(studentEntity), em.find(Student.class, 13).
Entity Lifecycle: new() Erstehen der Entity →
persist() Hinzufügen zu PC →
Persistence Context


(X)HTML5
Semantic markup (excerpt)
<header>
defines header of document or section
<nav>
defines navigation region of page or section
<section>
thematic grouping of content
<article>
specifies independent, self-contained content
<aside>
defines content aside from main content
<footer>
defines footer of document or section
Many of these elements can be nested and it's not always straightforward which element should be used!

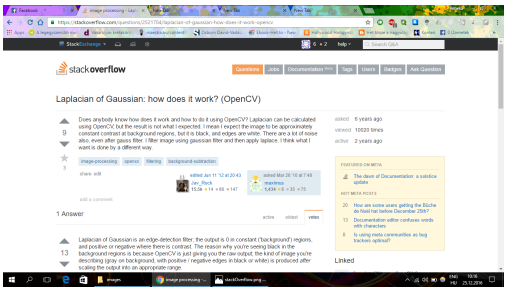

(X)HTML5
Semantic markup (excerpt)
<header>
defines header of document or section
<nav>
defines navigation region of page or section
<section>
thematic grouping of content
<article>
specifies independent, self-contained content
<aside>
defines content aside from main content
<footer>
defines footer of document or section
Many of these elements can be nested and it's not always straightforward which element should be used!

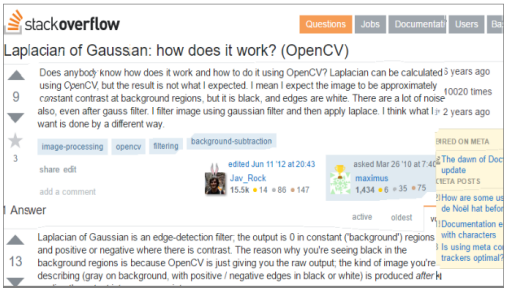

View all photos

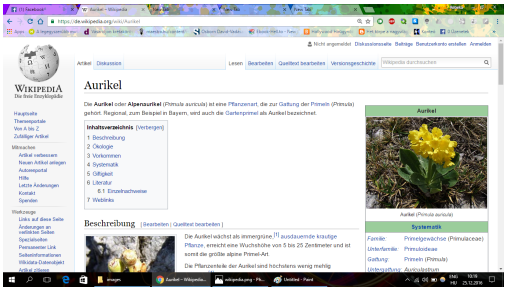

View all photos
Share
Zoom
Slideshow
Edit


Test
Some text
Some other text
Rainbow Dash


Test
Some text
Some other text
Rainbow Dash


stackoverflow
Laplacian of Gaussian: how does it work? (OpenCV)

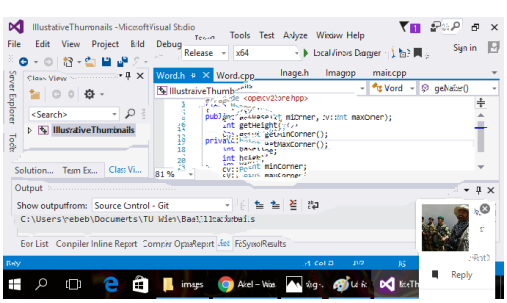

stackoverflow
Laplacian of Gaussan: how does it work? (OpenCV)

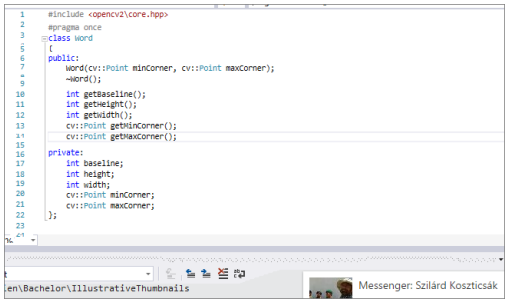

Aurikel
Inhaltsverzeichnis
Beschreibung

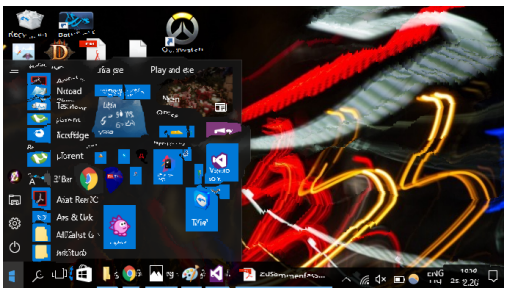

Aurikel
Inhaltsverzeichnis
Beschreibung
Systematik

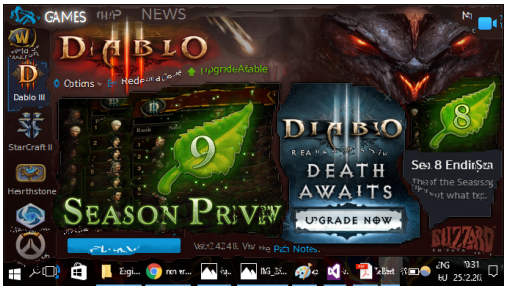
Appendix B

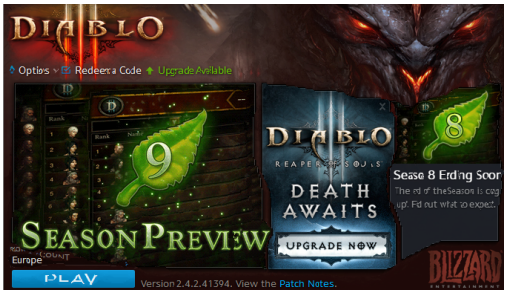
This section lists all reference images created by Photoshop. In case of the first column the input was the source image, in the second column the cropped image. The order of the sources is the same like in the section above.








GAMES
NEWS
DIABLO
Options
Redeem a Code
Upgrade Available
Diablo III
StarCraft II
Hearthstone
SEASON PREVIEW
DEATH AWAITS
UPGRADE NOW
PLAY
Patch Notes.
BLIZZARD


DIABLO
Options
Redeem a Code
Upgrade Available
SEASON PREVIEW
DEATH AWAITS
UPGRADE NOW
Europe
PLAY
Version 2.4.2.41394. View the Patch Notes.
BLIZZARD

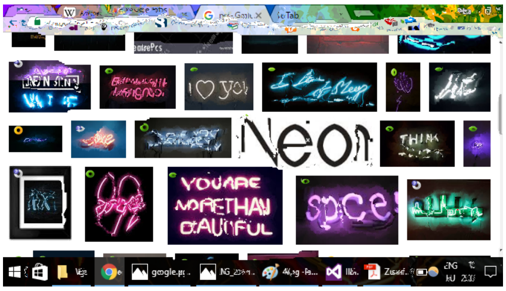

DIABLO
GAME MENU (ESC)
OPTIONS
ACHIEVEMENTS
CUSTOMER SERVICE
LEAVE GAME
EXIT DIABLO III
CURRENT SETTINGS
A ROYAL AUDIENCE
OPEN GAME TO PUBLIC
LOWER DIFFICULTY
HARD
RETURN TO GAME


DIABLO
GAME MENU(ESC)
OPTIONS
ACHIEVEMENTS
CUSTOMER SERVICE
LEAVE GAME
EXIT DIABLO III
CURRENT SETTINGS
A ROYAL AUDIENCE
OPEN GAME TO PUBLIC
LOWER DIFFICULTY
HARD

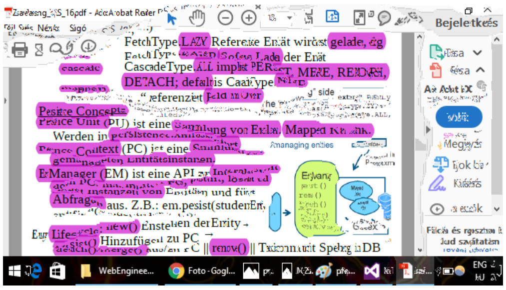

OBJECTIVES
A Royal Audience
NEW
Escape the Imperial Palace

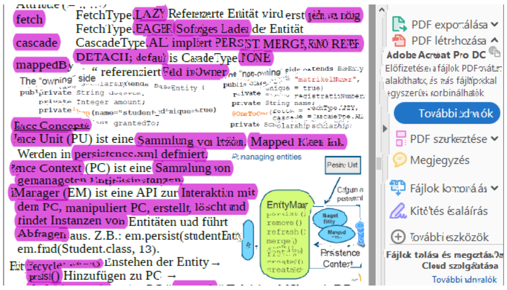

Chicagóban díjazták a Tisza szívvel
60 ÉV UTÁN DERÜL FÉNYA
KONCENTRÁCIÓS TÁBOROK
RÉMSÉGÉRE
Rá sem lehetsmeni
Val Kilmerre
NEM VICC
KÜLFÖLD
BELÉPÁS
96 hétköznapi hősei
Joe Biden azt ívánja bácsi
verekedhet-e Trumppal
tetejét.ondonbanegy híd

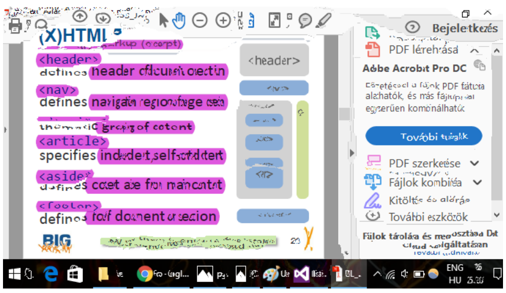

60 ÉV UTÁN DERÜLFÉNYA
KONCENTRÁCIÓSTÁBOROK
RÉMSÉGÉRE
Rá sem ehet meg
Valkilmerre
NEM VICC
KÜLFÖLD
BELÉPÁS
96 hétköznapi hősei
Joe Eden azt kránja, bácsi
verekedhetne trumppal
Lekaszáta egy eneletes bu
tetejét.ondonbanegy híd

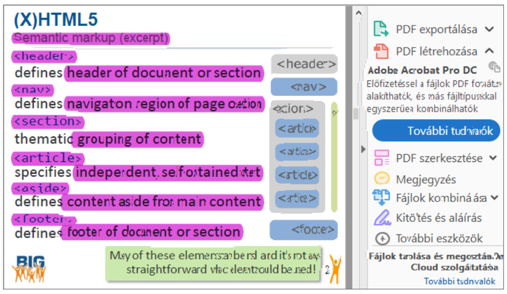


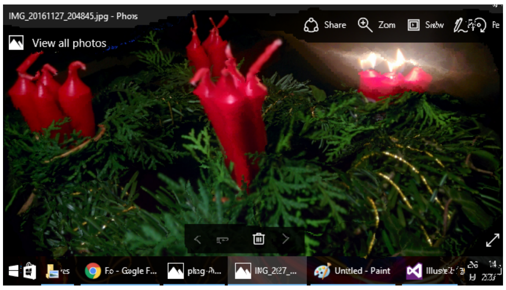

WHAT IF
JAZZ
I LOVE YOU TENDERLY TOTALLY TRAGICALLY
Neon
THINK OUTSIDE THE BOX
YOU ARE MORE THAN BEAUTIFUL
space

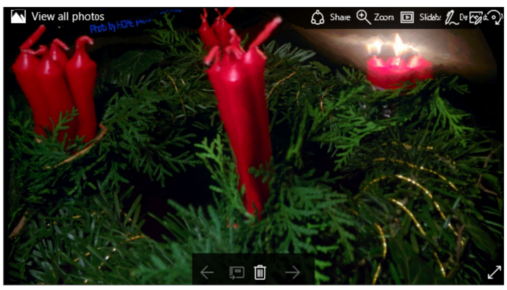


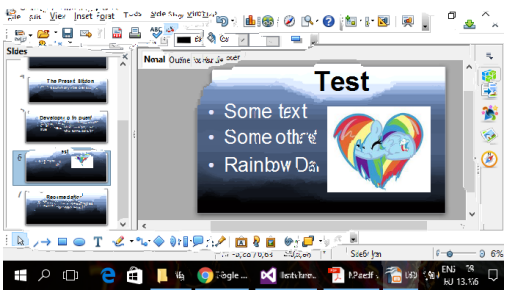

fetch
cascade
Werden in persistence.xml definiert.
em.find(Student.class, 13).




(X)HTML5
Semantic markup (excerpt)
<header>
defines header of document or section
<nav>
<section>
thematic grouping of content
<article>
<aside>
<footer>
defines footer of document or section
PDF exportálása
PDF létrehozása
Adobe Acrobat Pro DC
További tudnivalók
PDF szerkesztése
Megjegyzés
Fájlok kombinálása
Kitöltés és aláírás
További eszközök

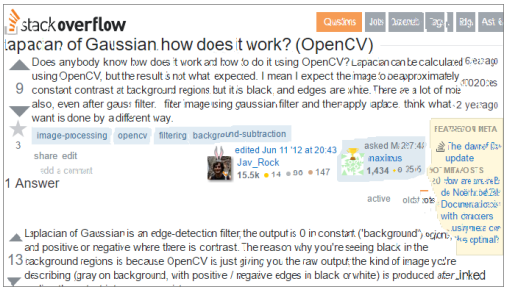





Test
Some text
Some other
Rainbow Da


Test
Some text
Some other text
Rainbow Dash


stackoverflow
Laplacian of Gaussian: how does it work? (OpenCV)
Laplacian of Gaussian is an edge-detection filter, the output is 0 in constant ("background") regions
and positive or negative where there is contrast. The reason why you're seeing black in the
13 background regions is because OpenCV is just giving you the raw output; the kind of image you're
describing (gray background, with positive / negative edges in black or white) is produced after scaling the output to an appropriate range.


stackoverflow
Laplacian of Gaussian how does it work? (OpenCV)
Does anybody know how does it work and how to do it using OpenCV? Laplacian can be calculated
9 using OpenCV, but the result is not what expected. I mean I expect the image to be approximately
constant contrast at background regions but it is black, and edges are white. There are a lot of noise
also, even after gauss filter. filter image using gaussian filter and then apply laplace. think what
want is done by a different way.
image-processing opencv filtering background-subtraction
share edit
edited Jun 11 '12 at 20:43
Jay_Rock
15.5k
1 Answer
Laplacian of Gaussian is an edge-detection filter, the output is 0 in constant ("background") regions
and positive or negative where there is contrast. The reason why you're seeing black in the
13 background regions is because OpenCV is just giving you the raw output; the kind of image you're
describing (gray on background, with positive / negative edges in black or white) is produced after scaling


WIKIPEDIA
Die freie Enzyklopädie
Aurikel
Inhaltsverzeichnis
1 Beschreibung
4 Systematik
5 Giftigkeit
Beschreibung


Aurikel
Die Aurikel oder Alpenaurikel
gehört. Regional, zum Beispiel
Inhaltsverzeichnis [Verbergen]
1 Beschreibung
2 Ökologie
3 Vorkommen
4 Systematik
5 Giftigkeit
6 Literatur
6.1 Einzelnachweise
7 Weblinks
Beschreibung [Bearbeiten Quelltext bearbeiten]
Aurikel (Primula Auricula)
Familie Primelgewächse (Primulaceae)
Unterfam. Primuloideae
Gattung Primeln (Primula)