FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Generating Expressive Window Thumbnails through Seam Carving

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Rebeka Koszticsák
Matrikelnummer 1325492

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

Wien, 6. Februar 2017

Rebeka Koszticsák	Manuela Waldner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Generating Expressive Window Thumbnails through Seam Carving

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 Koszticsák
Registration Number 1325492

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

Vienna, 6th February, 2017

Rebeka Koszticsák	Manuela Waldner

Erklärung zur Verfassung der Arbeit

Rebeka Koszticsák

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, 6. Februar 2017

Rebeka Koszticsák

Danksagung

Eszter Takács, vielen Dank für die "Übersetzung"!
Peter Pollak, ich habe gewonnen...

Acknowledgements

Eszter Takács, thank you so much for the "translation"!
Peter Pollak, I won...

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 ein gesuchtes Fenster schneller gefunden wird. Thumbnails sind aber nur verkleinerte Screenshots von Fenstern; wenn aber Tabs oder Applikationen mehrmals geöffnet 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 derselben Applikation oft 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 Artefakte weniger auffällig gemacht und die wichtigen Regionen hervorgehoben werden können. In dieser Bachelorarbeit wird eine Applikation entwickelt, welche diese Methoden auf Screenshots anwendet und Thumbnails erstellt. Screenshots von Applikationsfenster werden durch eine Kombination aus Cropping, Abschneiden von irrelevanten Elementen an der Seite, Seam Carving, Verkleinern durch Herausschneiden unwichtiger Pixel-Pfade, und herkömmlichem Down-Sampling zu Thumbnails verkleinert. Die Ergebnisse zeigen also nur relevante Informationen an, wodurch sie expressiver sind und ihren Zweck besser erfüllen können.

Abstract

Thumbnails are used to display lists 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 however only display 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 (e.g. : 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 enhance the main region. In this bachelor thesis an application is implemented that uses the above methods on screenshots. Screenshots of windows are reduced by cropping the irrelevant elements of the margin area using seam carving, i.e. by eliminating the non-important pixel paths; and by common down-sampling. As a result the thumbnails show only relevant information, which makes them more expressive and easier to fulfill their purpose.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**

2 Related Work **5**
2.1 Processing as UI 5
2.2 Processing as a Regular Picture 7

3 Methodology **15**
3.1 Eliminating UI Elements 16
3.2 Importance Map Calculation 19
3.3 Seam Carving 22

4 Implementation and Working Pipeline **25**
4.1 Image Loading 25
4.2 Saliancy Map Clculation 26
4.3 Text Detection 26
4.4 Seam Carving 27

5 Results and Evaluation **29**
5.1 Elimination of UI Elements 29
5.2 Text Detection 31
5.3 Re-Sampling Threshold 33
5.4 Comparison to Adobe Photoshop 33
5.5 Natural Images 36
5.6 Performance Evaluation 37

6 Future Work **39**
6.1 Performance Improvement 39
6.2 Methodology Improvement 39

7 Conclusion **43**

Bibliography **45**

Appendix A **49**

Appendix B **57**

CHAPTER 1

Introduction

With the increasing use of mobile devices and reliance on multi-tasking thumbnails are becoming more 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 and windows. However, classic thumbnails are sometimes not sufficient for this purpose. Due to the fact that standard thumbnails simply take the screenshot of each application and present them in a smaller size, the relevant parts of the window may no longer be recognizable. Figure 1.1 shows the windows-switching tool of Windows 10.

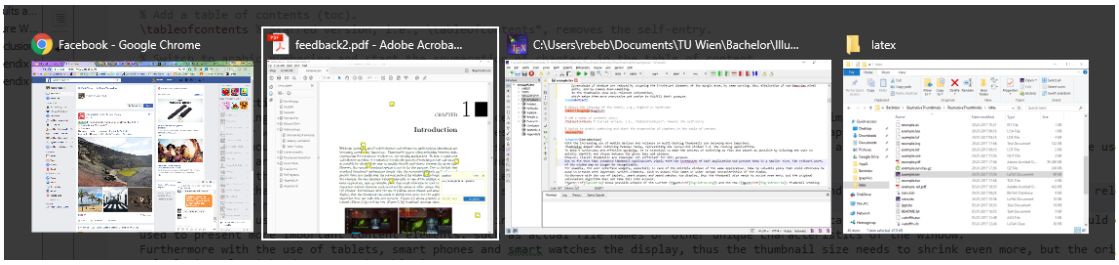
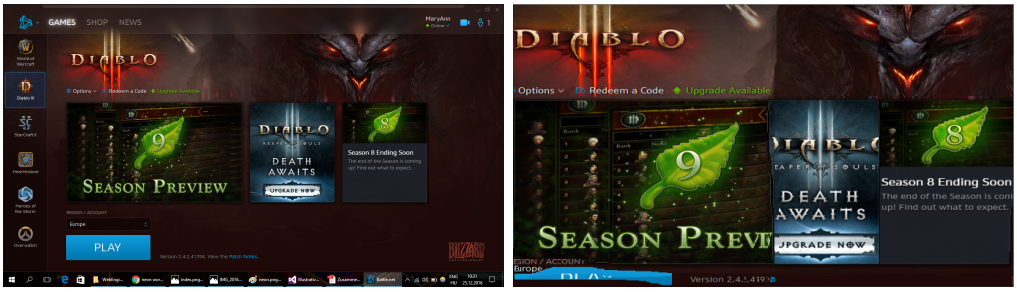


Figure 1.1: Visualization of the currently used thumbnail system on windows 10 in case of four running applications.

For example, the user interface widgets, especially in case of the multiple windows of the same application, take up valuable space that could otherwise be used to present more important content elements, such as actual file names or other unique characteristics of the window. Furthermore, although with the use of tablets, smart phones and smart watches the display, thus also the thumbnail size, need to shrink even more, the original calculation algorithm does not take this into account. Figure 1.2 shows possible outputs of the current (Figure 1.2a) and the new (Figure 1.2b) thumbnail creating algorithms.



(a) Classic thumbnail (b) Expressive thumbnail

Figure 1.2: The result using the two different approaches.

Seam carving is a special re-sampling method, where seams along the least important image regions are eliminated to down-sample the image. 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 elements on the source image. It is implemented for natural images, however, and not for screenshots, where the usual saliency calculation is not adequate enough. Following a similar logic, the algorithm presented in this thesis uses the seam carving method, but it takes into account that the given input is essentially a screenshot. So it will probably contain some UI elements on the picture, too. Additionally, considering that letters are more common on computer screens than on usual images, the text content is also investigated to determine them as important regions.

In summary, the thumbnail creating algorithm of this thesis performs cropping first to eliminate the UI elements of the margin area, then seam carving using a customized importance map and finally common down-sampling. During seam carving, a modified importance map is built from the usual saliency values of the screenshot and from the results of the text detection.

To provide satisfactory results a detailed and clear definition of the word expressive is needed. Not only it influences the calculation of the pixel and region importance value, but also acts at the selection of result outputs, and is useful for future software development. Moreover, 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 "expressive thumbnail" also comprehends that it is able to represent more information on a same sized picture than another not expressive 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 expressive. Namely, it is rarely the case that the relevant parts of an application includes its UI and not its actual content processing features; additionally, the identification of the window in question is also possible by reading its title. Furthermore, the importance of text data on the computer screens is assumably much higher than on regular pictures or photographs. Therefore, in order to transmit as much information as

possible, text data distortion is to be avoided, i.e. even after size reduction it needs to stay readable. Figure 1.3 shows a possible output of the seam carving algorithm using different importance maps.

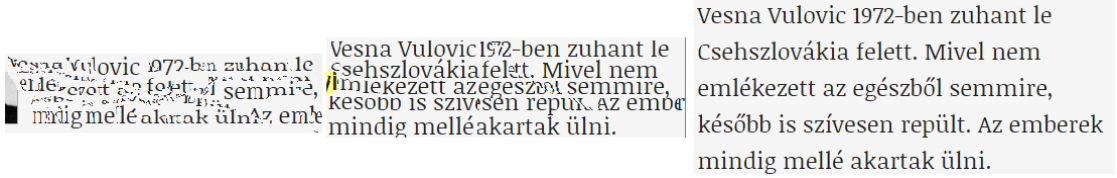


Figure 1.3: The text damage caused by seam carving using traditional saliency map and modified importance map of this application.

Summing up, in this project a thumbnail is called "expressive" if it well preserves the information content. 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 these 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 re-sampling natural images, are examined in the second section. Moreover, there are two classes of information presentation methods 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 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 programs 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] use 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 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 while it is still able to cover the most common UI elements. Figure 2.1 a possible entity of the Prefab database.

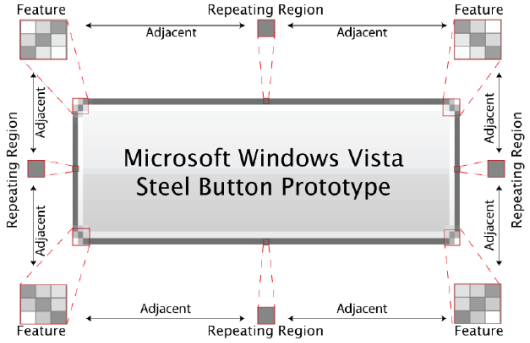


Figure 2.1: Prototype of Windows Vista Steel Button in the Prefab database [DF10].

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 databases, but for issues around granularity, too. Similar to the Prefab system it uses its own templates, however 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 be used effectively for solving the problems around database space 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 from 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 children are built in. Using this tree the order of the UI elements becomes clear, and the misleading information can be eliminated.

After labeling the UI elements correctly, they can be manipulated as needed. They can be cut off completely or processed according to the information content. Even a full size

UI widget can be highly detailed, therefore by resizing them the originally small elements become barely recognizable; in addition, they potentially also interfere with each other by taking space up from their competitors, leading to even worse legibility. 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. In this case, the tree mentioned above is applied as a z-buffer. With this cut off algorithm, unnecessary widget elements are easily eliminated and more important elements of the UI become more apparent as they can expand on the increased display space available.

According to the definition of "expressive", 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 applying them significant reworking is needed. Since it is not essential to know, exactly which 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 saving 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 place 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 none of the input images 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. Although the resulting image is similar to the input data, it is thus likely that not only the unimportant areas but also those with high information ratio will be damaged. Additional distortions might also occur.

2.2.1 Collage Creating Methods

A collage is an assembled image, containing parts of a bunch of input images and it is representative for the whole input data. In case of expressive thumbnails there are two scenarios when 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. An alternative solution is to retrieve the content in form of ROIs, that can afterwards be combined arbitrarily. On the other hand, thumbnails for desktop switching can be easily generated using collage

creator algorithms, where the inputs are screenshots of the open applications of the desktop, instead of some ROIs of one screen. The following algorithms are implemented for natural images however, therefore appropriate method modifications are required in order to use them for creating thumbnails.

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 E_{rep}, importance costs E_{imp}, transition cost E_{trans} and object sensitivity E_{obj} into account.

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

$$L = \{L(p), p \in \wp\}$$

$$L(p) = (n, s)$$

where:

$\wp$: domain
n : input image
s : pixel-wise 2D shift of n
w : adjusted weight by testing.

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 (assuring 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 arranges a reasonable placement for every object.

Egorova et al. [ESK08] concentrates however only at the first two parameters of the above. It clusters the images according to their source and time, when they were taken, and measures their quality. As a result of the clustering, when it comes to choosing the the final images it is already clear which images are the same or have similar content. This feature, accordingly modified, can be useful at sorting the ROIs of the screenshot, i.e.: text content, image content etc., or of the running applications of the desktop, i.e.: text processing, gallery application etc. The parameter quality summarizes the resulting values 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.

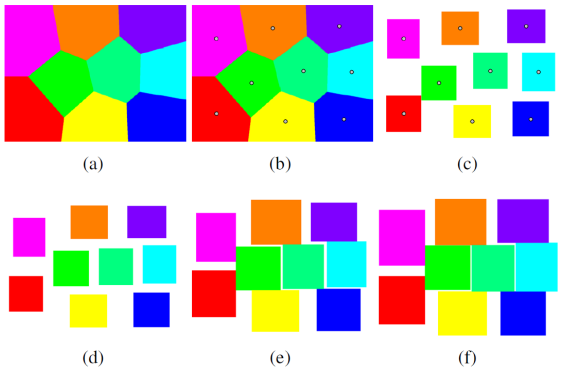


Figure 2.2: The whole ROI packing process [LSCP10] (a) K-means clustering (b) ROI's center at the beginning of the algorithm (c) Layout of the ROIs at the beginning of the algorithm (d) Shifting of the ROIs (e) Expanding the ROIs (f) Output after some iterations.

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 their aspect ratio constant, until they eventually overlap. Then, every ROI is shifted to the middle of its area. This method is repeated until there are no further increases in the ROIs anymore. To fill the white areas and eliminate any empty place, the neighboring ROIs are allowed to partially cover each other. Figure 2.2 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 of the previous ones. That is why they are more useful at making a thumbnail for a desktop while they do not necessarily provide as much advantages in case of applications. With taking the most important ROIs of one screen it would be possible to create a more expressive image than any other, since the important content could stay large and well readable, but classic collage assembly methods were developed for natural images. These approaches need adjustment according to the different requirements for image and text regions. On the one hand the collage method could minimize the distortion of important regions since they stay rigid. On the other hand, 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 screen recognition.

2.2.2 Re-Sampling Methods

To attain a constant relative position among regions, applying a re-sampling method is more effective than the above described collage creating approaches. Re-sampling

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.

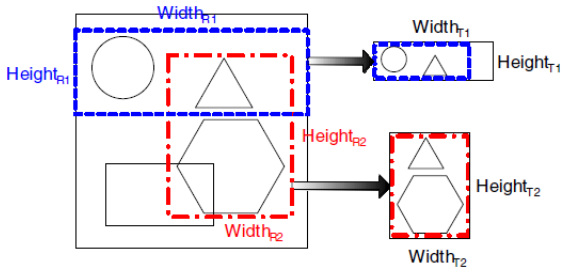


Figure 2.3: Possible solutions of the algorithm by Chen et al. [CXF+03].

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, as shown on Figure 2.3. 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 into an optional shape, but the gridpoints on the feature mask are not allowed to change their proportion to each other. This way the picture elements between the AOs fill the new shape while the AOs get barely distorted, as illustrated on Figure 2.4.

A detailed importance is essential at creating thumbnails, since a screen usually contains a greater amount of sensitive information. Text data is not allowed to be ignored, so they need to be part of one or more ROIs. But the algorithms described above have aspects like face recognition and grid determination that over-complicate the calculations. 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 uniform 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. Originally this algorithm is meant to fit the input into completely other shape and not to resize it according to its aspect ratio. Some of its aspects however, such as

Figure 2.4: The resulting image and its grid created by the Feature-aware Texturing algorithm; on the left: the input and its feature mask [GSCO06].

the definition of AOs, could expand the approaches used in this thesis; this possibility will be discussed in the future work section.

Seam Carving

Seam carving is a method, where unimportant pixel-paths are eliminated in order to down-sample the input image. Every pixel is first examined in terms of how much information they contain and how important they are. Afterwards, horizontal and vertical seams, i.e. pixel-paths where the following pixels are neighbors on the source image, can be calculated, and those with low importance value are cut off. With this approach the size of the input decreases without significant information loss. Figure 2.5 shows the resized version of the same input using Seam carving 2.5a and simple down-sampling 2.5b.

(a) Seam carved image (b) Down-Sampled image

Figure 2.5: The results of seam carving and common down sampling.

The importance map calculation depends on the individual implementation, but they are normally based on the saliency model introduced by Itti et al. [IKN98]. Itti's saliency calculation takes the function of the low-level human visual system into account, so it channels attention to color, intensity and orientation of the pixels in the first row. In the calculation every pixel and their neighborhood is investigated in terms of their

relationship to each other. In this way the attention catching elements of an image, like color and intensity changes, lines and edges, can be easily found. The architecture of this saliency model is illustrated on Figure 2.6.

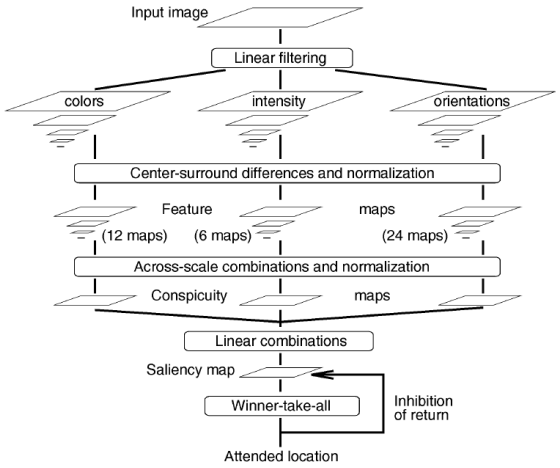


Figure 2.6: The architecture of the saliency model [IKN98].

According to the importance map Seam carving is performed as by Avidan et al.[AS07], where the least important seams are simply eliminated. A seam runs from the top to the bottom or from the left to the right side and every pixel is part of the 8-neighborhood of the previous seam member. The final importance cost of every seam is calculated by summing up the importance values of the containing pixels. When this value is low, it means that the seam does not cross regions with high information content, so in case of elimination the data loss remains minimal. Figure 2.7 shows the least important seams on a possible input image. Repeating this calculation, the image size can be reduced drastically without damaging the important regions as much as common down-sampling would do.

Figure 2.7: The least important seams of the image [AS07].

Since Seam carving pays attention to the importance of one and other image regions in

order to create more expressive thumbnails it is preferable over down-sampling. Seam carving tries to save the high-information-content of the source, and eliminates only the non-relevant parts of the image. But its level of efficiency is strongly depends on the importance map. In case of thumbnails, the calculation of the saliency is not sufficient, since for example it evaluates text data as not important, to the contrary as it should be. Therefore, also in case of Seam carving, to apply it for thumbnail creation the importance map calculation needs to get adjusted to the special requirements of a screenshot.

CHAPTER 3

Methodology

The illustrative thumbnail creating algorithm has three main steps, as shown in Figure 3.1. At the beginning the UI elements are cropped. 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. The appearance of the applications is thus very similar, and they may no longer be discriminable when scaled down. Consequently, the actual content and not the surface of these applications makes a difference. Although reading the title of the thumbnails may be slower than the software recognition through low-level 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 by the location of text data. Unlike at regular real-life pictures the occurrence of text on computer screens is very common, and possibly is the main discriminating content. For this reason the calculation of the final importance map has two steps. First the importance value of every pixel 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 and the space between them is marked 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.

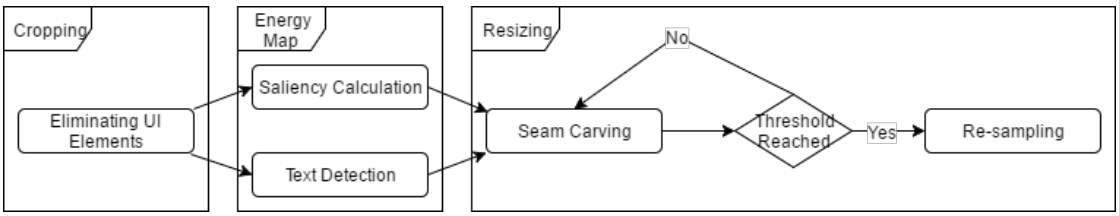


Figure 3.1: Flow chart of the algorithm.

3.1 Eliminating UI Elements

For the elimination of UI elements, first their identification must be accomplished. For the search of UI widget three heuristics were developed. The first one implies that these UI elements are located near the border of the screen and not in central areas. Secondly, it is assumed that the UI has similar background or theme color, and it differs from the rest of the window. Finally, the UI area is bounded by a straight line, while the horizontal one is longer than the vertical. Thus, the middle of the screen is taken as reference data for the investigation and is not checked for UI elements. The UI elimination algorithm operates right on the source window in the border area, which size is predefined and parametrized, i.e. no further premodification is required for this process.

In case their order is relevant, while cropping the borders first the horizontal, then the vertical margins are investigated. The cropping algorithm, described in the following, is based on the assumption 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 has a slightly different style than the other UI widgets mentioned above. The UI matching method investigates slices of the screen running along the margins; examining their color histograms determines whether they belong to the UI or to the middle area of the window. Because of the use of color histograms and the assumption that the UI has similar background color, which is different from the content, it is important to investigate a coherent data, so the slices should cover only UI or only content area, and not a mix of them. The above described process with an established order of margin cropping is fit for this purpose.

To identify the best cropping line, indicating the edge where the UI meets the content area, a double validation method is performed. The first step evolves searching for a row of pixels in the border area, which size is configurable and predefined as mentioned above, located however near to the center of the source, having the same color and no interruption along the horizontal sides of the screen. It means, the length of the row is equal to the width of the source, it has exactly one pixel from every column which are actually taking place in one line on the input image. But this pixel row is not necessarily an actual line or edge of the image, it is only a horizontal path, where every member has the same color. This method is based on the observation that toolbars often have a unicolor background, furthermore the UI is normally bounded by a straight line, where such a pixel row can easily be found, as the examples illustrate on Figure 3.2. Therefore, the task is to find a horizontal pixel path, which has already passed the widget and icon location area, but still belongs to the UI area, i.e. it contains the background color or the line at the edge of the UI and content area.

In case of predefined UI areas the above method 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

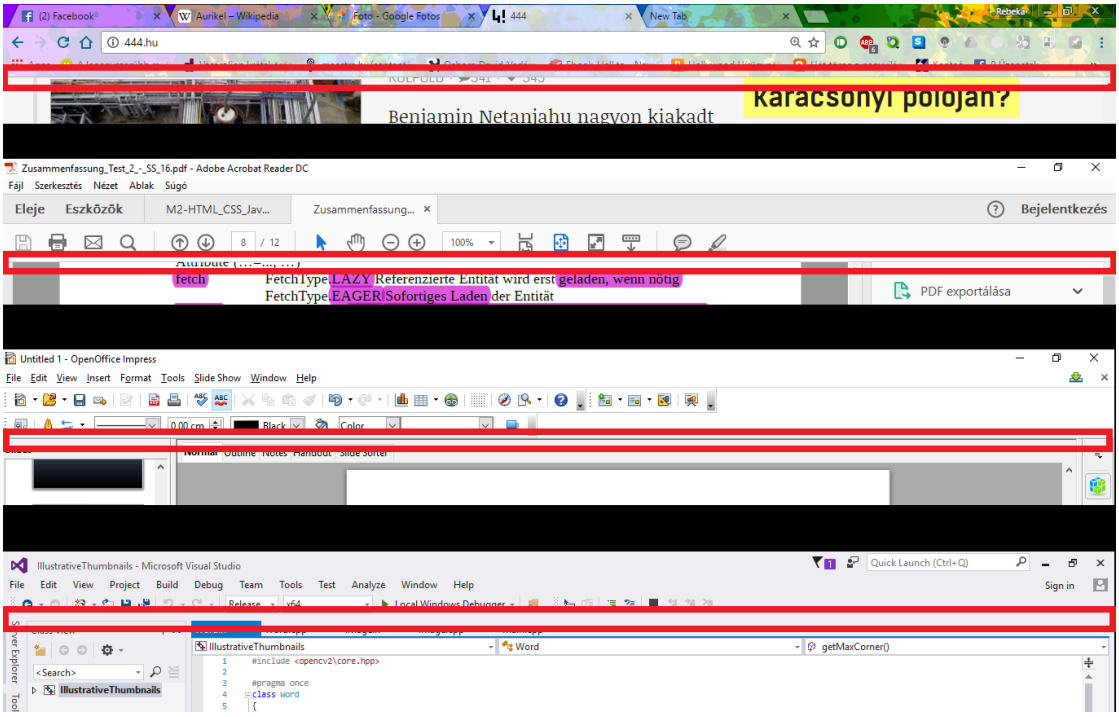


Figure 3.2: Unicolor line at the edge of the UI area.

be found or the UI is simply not bounded by the edge mentioned before. 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 calculated by the following 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 : Amount of the bins.

The result of this equation is between 1 and 0, if it is high it means that the two histograms are very similar, if it is low, however, it implies that they have no similarities. The margin is split into thin slices, where the actual size is parametrized and can be set as required, as a percentage portion of the actual height of the source. 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, where this threshold value can be defined, 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 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. Initially, 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.

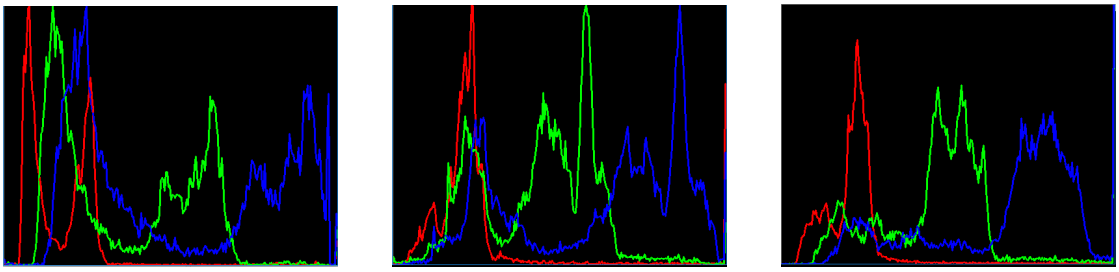


Figure 3.3: Histogram of the margin area, of the slice where the source 3.4 is cropped and of the content region.

This tendency reverses 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 displays the histograms of the margin, of the cropping and of the content area of Figure 3.4. Figure 3.4 shows the borders between the content and UI regions found by the UI cropping heuristics.

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

Figure 3.4: The border of content.

3.2 Importance Map Calculation

In order to ignore the parts of the source image that are actually unimportant, importance calculation is applied. The importance map calculation has two main steps: saliency calculation and text detection. In the following these two components of the algorithm is described.

3.2.1 Saliency Calculation

The saliency value of a pixel describes how much it stands out from its surrounding, how noticeable it is and how prompting it is for the human eye. A saliency map in this project is a matrix where the high value of an area means that the region is more salient than any other pixels with lower values. There are, however, many definitions and approaches about which pixels should be evaluated as salient or not salient, some of them is already mentioned in the Related Work section. 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 information just like the traditional method introduces by Itti et al. [IKN98]. It means that only those pixels are valued as important that activate the low-level human visual system, for example due to their intensity or color change characteristics. In addition however, it is scale invariant, so it is more robust than other similar approaches mentioned before.

The algorithm converts the source into a perceptually linear, device-independent colorspace (Lu*v) first. After that, to make the approach scale invariant, a Gaussian contrast

pyramid is built. The algorithm of Niu et al. [NLLG12] 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 a 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. 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.

3.2.2 Text Detection

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 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 detection 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 using connected component analysis. Thank to

(a) Input image (b) Output

Figure 3.5: Result of the text detection algorithm applied on the given source.

this blur operation very short words like "the", "a" or "I" do not get lost, because they will be connected to the words next to them. 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, the thresholds can be set as needed, with their default value based on the size of one letter, in default case 5pt, 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, meaning that they have one peak for the letters and one for the background color. The two largest bins of the histogram are detected, and if they contain a predefined percent of the pixels, it is set for 65% as default, then is the area labeled as actual word. 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. Figure 3.5 shows the output of the text detection algorithm. The light blue color labels the possible word regions, thus in the result of the connected component analysis the darker color denotes the actual words found by the algorithm.

The last step for the calculation of the final importance map, which is consequently used in the whole application, is to get the identified text regions weighted in the saliency map. All areas that passed the word checking test are automatically evaluated as highly important, and get the highest saliency value in the importance map. In addition, to make sure that even the space between the words remains undamaged, which is important for greater readability, the area neighboring the text data is also weighted. Figure3.6 is the final importance map of Figure 3.5a.

Figure 3.6: Final saliency map.

3.3 Seam Carving

To resize the image without damaging the important regions, seam carving is used, mentioned in the Related Work section. Seam carving calculates paths according to the importance 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 eliminate only the straight lines, the algorithm noticeably affects 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 to increase the application performance, a threshold indicating when to switch to re-sampling is defined, as Dong [DZPZ09] proposed. To reach the final output size common down-sampling is applied.

To find the most appropriate path, every possible solution need to be examined using the backtracking method. A path map is thus generated, where all past possibilities belonging to the previous start pixels are stored, and where every new try that occurs when examining the next start pixel can be used as a look up. This way performance can be saved and the algorithm becomes faster.

To find the next pixel of the path, the 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 by $\sqrt{5}$, $\sqrt{2}$ and 1 according to their distance. In each case the chosen pixel is the one whose importance value with the weighting parameter is the smallest.

Figure 3.7 is a visualization of the path map while running the algorithm. The yellow pixels are already set, the first row is automatically filled with the importance values from the importance map, the whites are however not known. The algorithm always takes the next unknown pixel running from left to right, from the top to the bottom and searches for an appropriate predecessor. The next point is displayed in the blue square. The possible predecessor of this point are shown in the red square. The importance of these pixels i.e. the yellow entities, saved in the path map, are compared, and the one with the smallest value is chosen. In case of the blue square pixel the coordinates of the chosen predecessor are stored and its importance value is set by adding the value in the predecessor pixel and the blue pixel value in the importance map together.

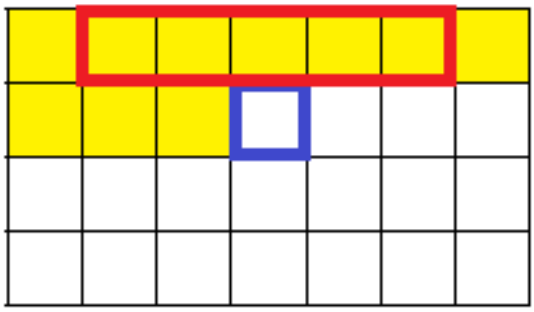


Figure 3.7: Visualization of the path map during the calculation.

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. Additionally, since the output needs to hold the aspect ratio, in the end same amount of horizontal and vertical seams are 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 width 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, thus it is only looking for either horizontal or vertical seams, since the width or the height already hit the preconditions, 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 the first loop of the seam carving algorithm. This threshold value is further customizable, the different outputs in case of varying its value is discussed in the Results and Evaluation section. 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

Algorithm 3.1: The seam carving algorithm

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

Output: seam carved *image*

```
1  while y < width of image do
2      while x < height of image do
3          previousPixel = detect the less salient pixel of the five-neighborhood in the
             previous line in pathValues
4          the saliency value of pathValues at (x,y) = previousPixel + saliencyMap at
             (x,y)
5          the path value of pathValues at (x,y) = position of previousPixel
6      end
7  end
8  leastSalientPixel; while i < width of pathValues do
9      Set leastSalientPixel if the i-th value in the last row of pathValues is smaller
         than the value of leastSalientPixel
10 end
11 for j = height of image, j ≥ 0 do
12     remove the pixel from image with the coordinates of leastSalientPixel
13     lessSalientPixel = the pixel in pathValues with the coordinates of the path
         value of lessSalientPixel
14 end
15 return (image)
```

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 and Working Pipeline

The expressive thumbnail creator application is developed in the programming language C++. Except for 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 will need customization. For image processing purposes Open Source Computer Vision Library (OpenCV) is used [Bra].

Apart from being a computer vision library openCV is a machine learning library too. 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 tasks such as blurring or filtering. The present use of the library is discussed in this section. 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. The application uses a `config.txt` file, where every parameter can be set as needed. A list of these parameters is highlighted below.

4.1 Image Loading

Preceding the start of the thumbnail algorithm it is essential to load the source image. The application is not able to capture screenshots, it only reads the already saved screenshot images. 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 source. It shows only image files with the extension `.jpg` or `.png`. After choosing the preferred source, the file path is established and the OpenCV function `loadImage` opens the picture.

4.1.1 UI Elimination

The default value of the border area is 20% of the source size. 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, is 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, then the margin is split along the nearest border into thin slices. The function `calcHist` calculates the histograms of the marginal and of the central window. The function `compareHist` measures the correlation between the two histograms. If 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 of the slices is the 0.25% of the window.

4.2 Saliancy Map Clculation

The saliency map is calculated from a Gaussian pyramid. However, 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 using the equation mentioned in the Methodology section, from the four neighboring pixels for each level. In order to avoid any overflow, by merging the levels into one saliency map, the pixel values are divided by the amount of pyramid levels.

4.3 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. The operator calculates the second derivate of the intensity values. In case of edges, there is an intensity change between the neighboring pixels, therefore the result of the Laplacian operator equals zero. Resulting the above, the layout of the letters are 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 connected regions and saves their silhouette points into a 2 dimensional `vector`. For each region the bounding rectangle is 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 a minimal word is configurable and by default set to 5x5 pixel, approximately the font size of 5pt, in order to be able to take even the smallest fonts into account. Once a region passed the test, it is marked as a possible word and is forwarded for further investigation.

After labeling every area as word or non-word region, the average height of the possible words is calculated. If the height of a possible word, compared to a threshold, is 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 potentially large-size titles and headlines. The smaller words are, however, allowed to be sorted ot. If they are written so small, it is likely that their information content is not as important as for example of a headline. Moreover it would be difficult to preserve them readable, and they would take space from other content.

After the height test, the histogram of the possible word regions is examined. If it has exactly two accumulations, one for the letter color and one for the background, the region stays in the group of possible words, otherwise it is sorted out.

Finally, the spaces above and below the possible words are 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 to the maximum value. At the very end the whole map is normalized with the OpenCV function `normalize` to avoid value overflow at visualization and other irregularities.

4.4 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 by 90 degrees 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%. At this point, usually the algorithm

has already switched to down-sampling. On the other hand, at bigger size the differences are easier recognizable, and further down-sampling can be performed at any time.

The seam calculation algorithm works by writing a path map, where the previous paths and their saliency values are saved. The identified seam 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 then the algorithm terminates.

CHAPTER 5

Results and Evaluation

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 when using the recommended `config` file and the experimental results on Linux are listed in Appendix A. In the following, the test cases of the elimination of UI elements, text detection and various value of re-sampling thresholds are presented and evaluated in respect to which of them are best able to provide the desired results. Then, the application is compared to Adobe Photoshop [Inc], with both applications' advantages and disadvantages described. Following that, some further results are presented when the algorithm is applied on natural images but not on screenshots. At last, the performance issues of the application are discussed briefly.

The recommended `config` settings are: The border area is 20%; the correlation between the lower and the right border is 1.3, between the upper and the left is 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%. Finally, the different settings of the application are compared to the output of the above described case. To avoid the exponential increase of the tests cases only one parameter is modified in the following section, the others are always reset to default.

5.1 Elimination of UI Elements

Depending on the investigation area for UI elements, various regions of the border area may be cropped. Any other parameter that investigate the UI regions, such as correlations and the thickness of the slices, are reliant on the border region, since it determines the region to be examined. 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) also has potential disadvantages, as eventually the reference value for the central area can be chosen wrong, resulting in cropping from additional, non-border areas. So de default border region is set to 20%.

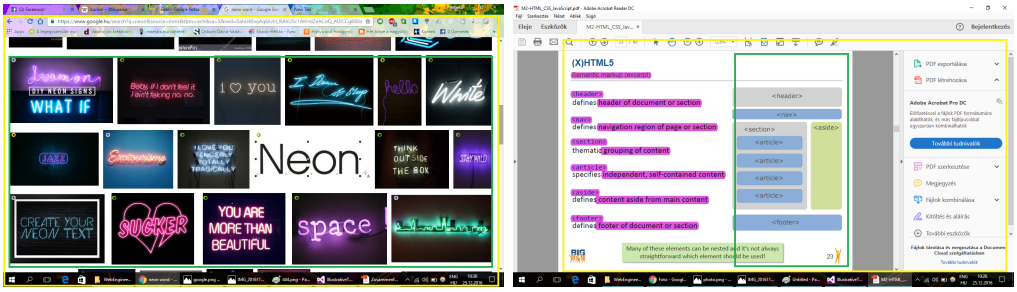


Figure 5.1: Cropping points according to the border area 10% (yellow) and 40% (green).

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 border area and the slice is set too low, their correlation value has to be small (1), thus some content elements near the border can be eliminated; at expected high correlation (2.5) however some UI widgets can be labeled as important content. Since the correlation is calculated for the three color channels, maximum value of the summarized correlation is three.

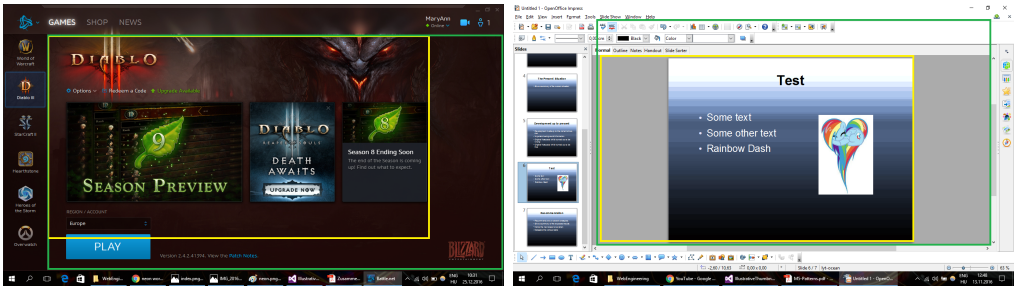


Figure 5.2: The cropping points according to the correlation of 1 (yellow) and 2.5 (green).

The thickness of the slices influences how 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 meet.

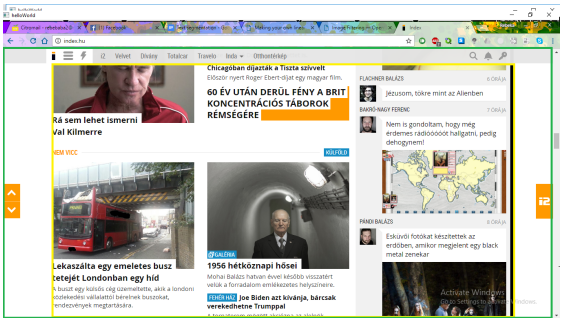


Figure 5.3: Cropping points according to the thickness of the slices 12% (yellow) and 5% (green).

5.2 Text Detection

As a result of the Laplacian operator, the silhouettes of the letters are highlighted, and a bright color is assigned to them. If these silhouettes 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. On the one hand, if this parameter is small (50) any bright area can be classified as text, even if the given region is only neighboring a word. On the other hand, if it is set to a high value (200), only the core of a word will reach it, therefore no region will actually contour its related word's shape. Bright blue contours the related area while the darker color shows the regions classified as actual words.

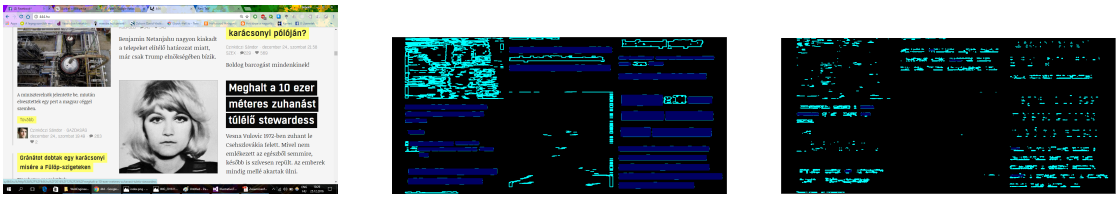


Figure 5.4: The related areas according to the brightness threshold value 50 and 200.

The properties of a possible word are also customizable in the `config` file. 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 is 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.

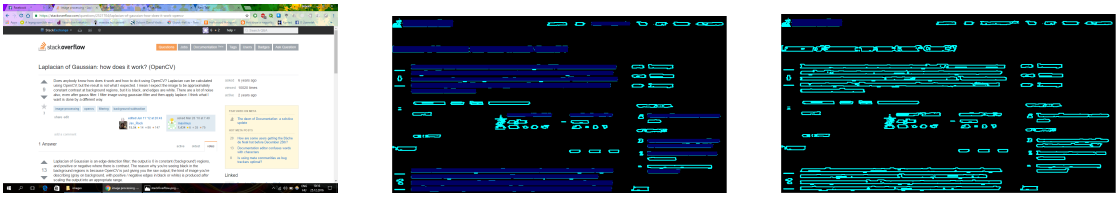


Figure 5.5: The actual words according to the minimal width 100 and minimal height 20.

From the set of the possible words the average word size is calculated, but thenafter the words' variance from this value is still further customizable. 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 of 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 to non-salient, i.e. 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 the letters would be illegible due to their minuscule size. The experimental results are presented below.

Figure 5.7: Different text weighting: high weights (middle) and low weights (right).

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, 10% of the average importance of the most important seam on the original image) 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, 40% of the average importance of the most important seam on the original image), 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. On Figure 5.8 the red lines indicate the edges, where the side-effects of elimination are obvious.

Figure 5.8: Results at re-sampling threshold of 0.1 and 0.4.

5.4 Comparison to Adobe Photoshop

To compare the algorithm to an already existing tool the content-aware resize 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 expressive 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: Photoshop 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 used when it began with the seam calculation. The second test scenario is in place for this reason.

Figure 5.11: Photoshop results of the second scenario.

The quality of the output definitely improves: faces are recognizable, the text is easier to read, icons are preserved, all due to the fact that an already smaller image needed to shrunk to the same size as in the first scenario. The main difference between the two methods seems to lay on the weighting of the content. The expressive thumbnail approach pays more attention to the text, with image data being easily ignored.

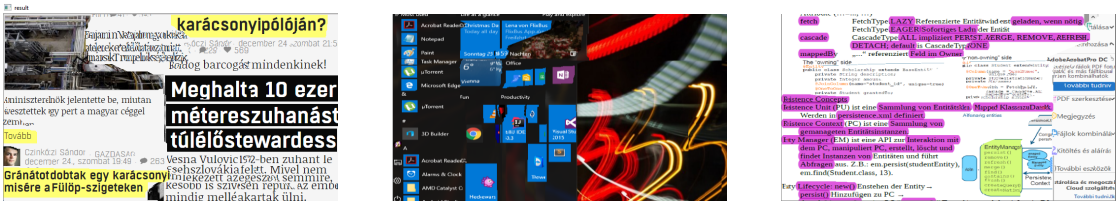


Figure 5.12: Result of the thumbnail algorithm.

Photoshop, however, attempts to sustain the image content. Since the thumbnail creator algorithm has a ranking between the elements, favoring texts before images, the output is not dubious: even in case of image data loss, the text remains readable, like the snippet of Figure 5.13 shows. The output of Photoshop is rather disorganized, the text and image regions flow into each other, illustrated on Figure 5.14. The images are better recognizable as already illustrated on Figure 5.13, but in exchange the words are noticeably more

damaged, as it appear on the code snippets on Figure 5.15. All results produced by Photoshop are listed in Appendix B.

Figure 5.13: Snippet of the same region of the first test image captured on the output of the Photoshop test scenarios and of the thumbnail creator algorithm.

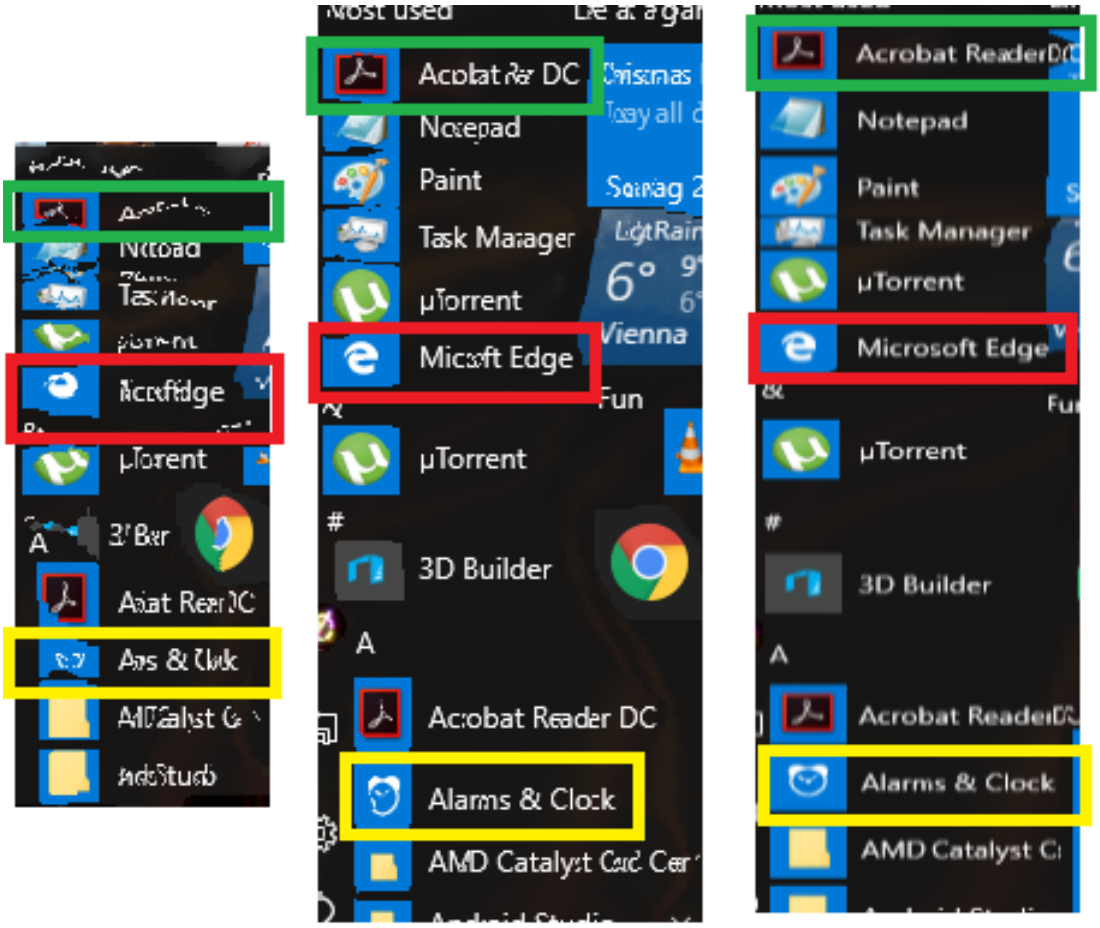


Figure 5.14: Snippet of the same region of the second test image captured on the output of the Photoshop test scenarios and of the thumbnail creator algorithm.

Figure 5.15: Snippet of the same region of the third test image captured on the output of the Photoshop test scenarios and of the thumbnail creator algorithm.

5.5 Natural Images

The algorithm is designed to create expressive thumbnails but nor to perform down-sampling on natural images. The importance map calculation is customized for the requirements of a screenshot, not of a common photograph. There are two aspects, where the thumbnail algorithm fails and is not able to provide results having the same quality as before. The first one is the UI cropping heuristic, as shown in Figure 5.16. It is also common on natural images, that the margin area has a slightly different color theme than the central region. At the margin normally the background is shown, whereas the focus objects of the picture are placed at the middle of the scene. Therefore it is possible, that since the UI cropping algorithm examines only the color histograms of both areas, it will assume that at the margin UI widgets are found, and thus some border regions get eliminated.

Figure 5.16: The UI cropping heuristic eliminates a person on the right side.

Other drawbacks of the thumbnail algorithm on natural images are the artifacts caused by seam carving. There are two explanations to why the algorithm is not able to work on photographs from the real word as smooth as on screenshot images. The first one is the importance map calculation, which, as mentioned above, is customized for computer screens and not for natural images. Therefore, in later steps the seam carving method does not have information about which objects are to be handled as "special" i.e. similar to how text data was prioritized before. It is thus possible, that the seam carving algorithm ignores human faces like it does on Figure 5.17.

Figure 5.17: Seam carving eliminates two faces and causes artifacts around the legs.

Moreover, there are no post-processing step after the seam elimination. The transition between the objects on computer screens is not as smooth as it is on natural images. Sharp edges and strong intensity changes on screenshots are usual, so even after seam carving the possible artifacts caused by pixel elimination are not outstanding. To the contrary, in order to obtain a natural-looking outcome in case of natural images, it would be essential to smoothen the resulting sharp edges of the pixel path elimination. Since this step is not implemented, the weaknesses of natural image processing are obvious on Figure 5.18 and on Figure 5.19.

5.6 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, the size of the source is 1366x768 pixel. 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 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 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

Figure 5.18: There is a horizontal line in the middle, where probably several seems were eliminated from.

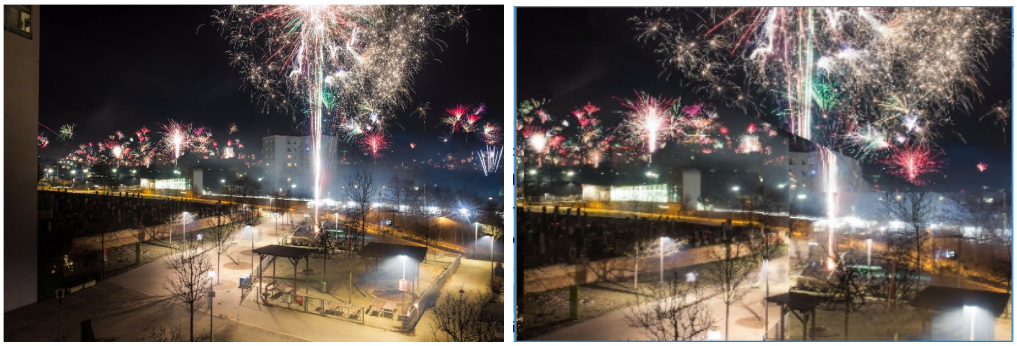


Figure 5.19: There are several artifacts on the firework bodies.

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 expressive 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 could be easily parallelized, however. In 2007, Nvidia released a parallel computing platform, called CUDA [Coo13], which exploits the computing performance of the GPU. 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. In addition, seam carving is already implemented on the system by Duarte et al. [DS12].

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 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 also aim to find semantically meaningful 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, thus for example the output in the case of Figure 6.1 would change. It includes an already implemented feature of OpenCV,

Figure 6.1: The current algorithm destroys the facial information.

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 use 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 than the current application, because it controls for information extremities. Consequently, no pixel will appear more important - or unimportant - than it really is. Guo et al. [GDT+15] obtained 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 also in cases when a simple line detection algorithm fails i.e. indicate no salient regions. A saliency map where also the inside of the object is filled according to the above algorithm is showed on Figure 6.2. To find the coherent areas that later can be defined as objects, the use of a mean shift algorithm, as in Liu et al. [LG06], 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 in Figure 6.3.

Once the final saliency map is reached, the next step is to calculate the seams. In this project seams are defined as paths with the width of a singular pixel. On the other hand, moderation of this rule can provide better results with less artifacts and also increases

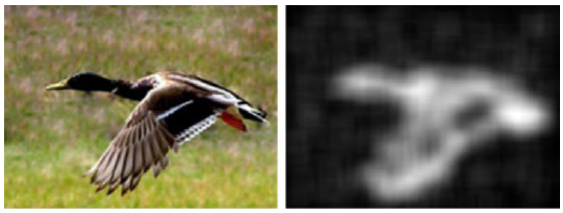


Figure 6.2: The source image and the saliency map of the NIF algorithm [GDT+15].

Figure 6.3: The source image and the saliency map of the mean shift algorithm [LG06].

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, therefore it is essential that the relative saliency value still remains 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 6.4. In case of seam carving the

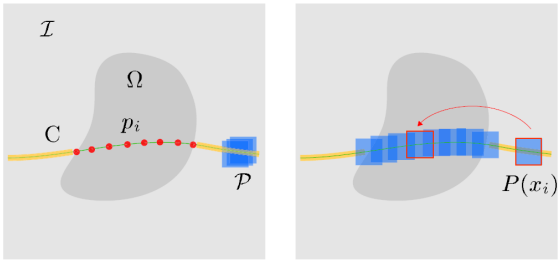


Figure 6.4: The structure propagation algorithm, where Ω denotes the unknown regions and p_i the anchor points [SYJS05].

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 in order 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. At this approach the properties of the background have no influence on the actual seam sequence. 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 will 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.

All improvements suggested in this section are developed for natural images. In the case of their use, these methods need some modification for an optimal functioning at thumbnails. When these approaches are implemented, it is to be considered that they are tested on natural images, and they may possibly have different behavior when using them for thumbnail creation.

CHAPTER 7

Conclusion

To make thumbnails more expressive 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 computer screens 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 expressive than their usual relatives, since their information content is better recognizable and the text regions are easier to read.

Bibliography

[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.

[DS12] Ronald Duarte and Resit Sendag. Accelerating and characterizing seam carving using a heterogeneous cpu-gpu system. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*

(PDPTA), page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

[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.

[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.

[IKN98] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

[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.

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

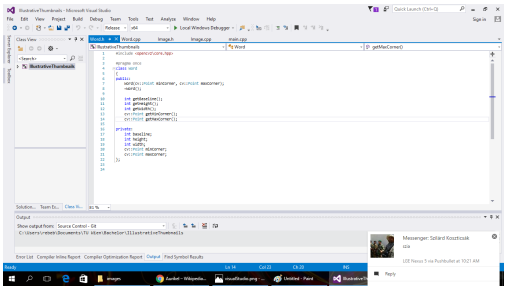
[SYJS05] Jian Sun, Lu Yuan, Jiaya Jia, and Heung-Yeung Shum. Image completion with structure propagation. *ACM Transactions on Graphics (ToG)*, 24(3):861–868, 2005.

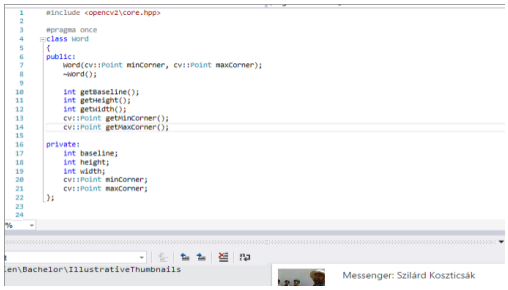
[Win17] Windows. *OPENFILENAME structure*, 2008 (accessed January 28, 2017).

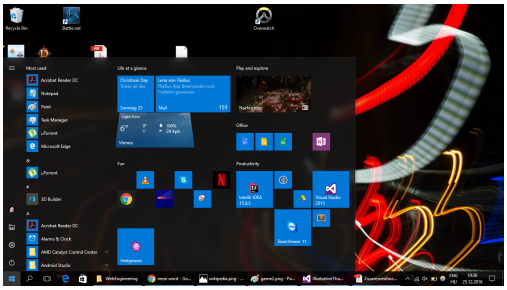
[YCM09] Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller. Sikuli: using gui screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 183–192. ACM, 2009.

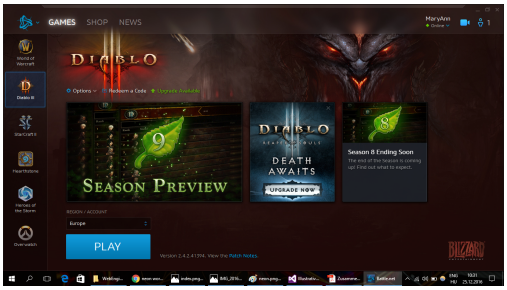
Appendix A

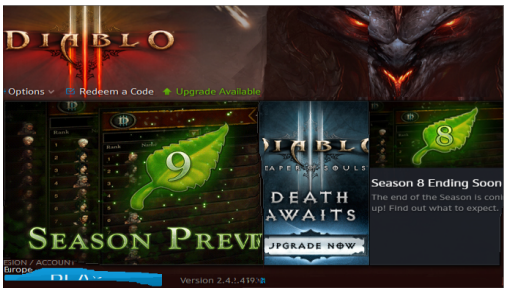
This section contains the source and the result images using the recommended `config` settings. The border area is 20%; the correlation of the lower and the right border is 1.3, the upper and the left however 2.0; the thickness of the slices is 2.5%; the brightness threshold for text detection is 100; 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%. At the end experimental results are listed, the screenshots were captured on the Linux system.







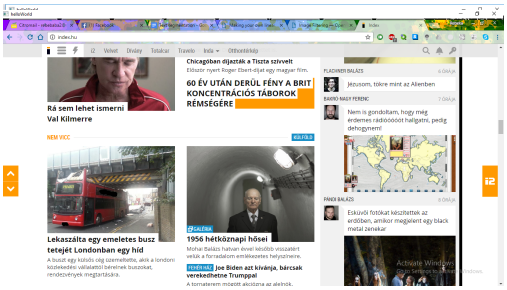

GAMES
SHOP
NEWS
DIABLO
SEASON PREVIEW
DEATH AWAITS
UPGRADE NOW
Season 8 Ending Soon
PLAY

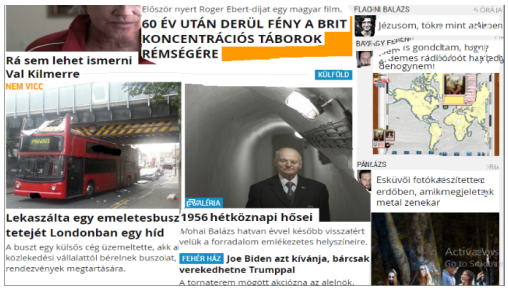

DIABLO
Options
Redeem a Code
Upgrade Available
DEATH AWAITS
UPGRADE NOW
Season 8 Ending Soon

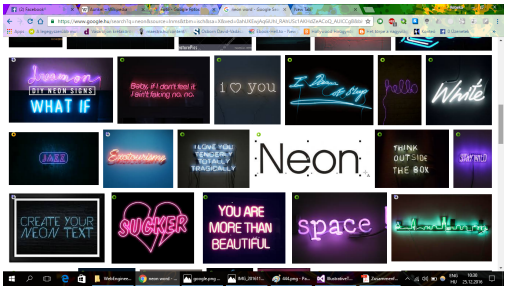
DIABLO
GAME MENU (ESC)
HARD

DIABLO
GAME MENU (ESC)
HARD

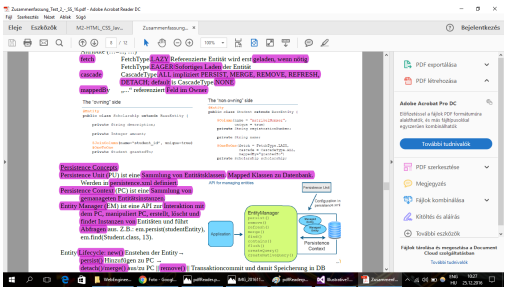
OBJECTIVES
A Royal Audience
NEW Escape the Imperial Palace

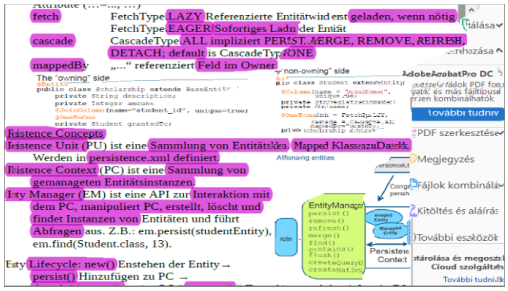

60 ÉV UTÁN DERÜL FÉNY A BRIT KONCENTRÁCIÓS TÁBOROK RÉMSÉGÉRE
Rá sem lehet ismerni Val Kilmerre
Lekaszálta egy emeletesbusz tetejét Londonban egy híd
1956 hétköznapi hősei
Joe Biden azt kívánja, bárcsak verekedhetne Trumppal

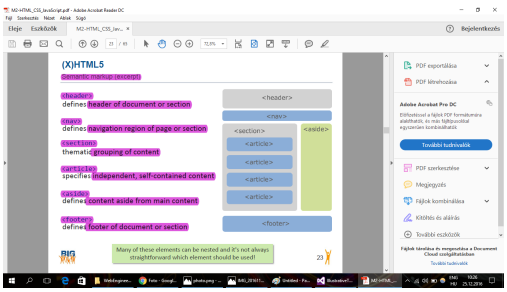

60 ÉV UTÁN DERÜL FÉNY A BRIT KONCENTRÁCIÓS TÁBOROK RÉMSÉGÉRE
Rá sem lehet ismerni Val Kilmerre
Lekaszálta egy emeletesbusz tetejét Londonban egy híd
1956 hétköznapi hősei
Joe Biden azt kívánja, bárcsak verekedhetne Trumppal

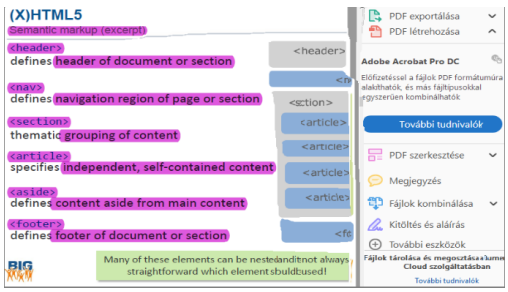

DIY NEON SIGNS
WHAT IF
Neon
CREATE YOUR NEON TEXT
SUCKER
YOU ARE MORE THAN BEAUTIFUL
space

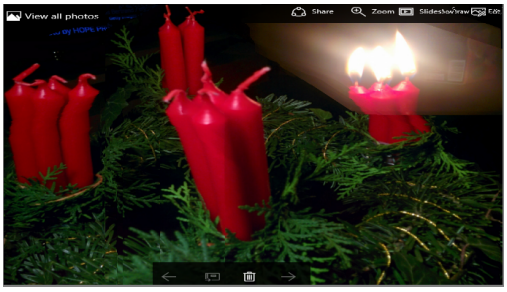
DIY NEON SIGNS
WHAT IF
JAZZ
CREATE YOUR NEON TEXT

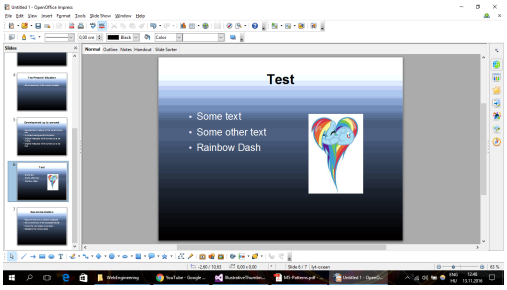

Persistence Concepts
Persistence Unit (PU) ist eine Sammlung von Entitätsklassen
Persistence Context (PC) ist eine Sammlung von gemanageten Entitätsinstanzen
Entity Manager (EM) ist eine API zur Interaktion mit dem PC, manipuliert PC, erstellt, löscht und findet Instanzen von Entitäten und führt Abfragen aus.
Entity Lifecycle: new() Erstehen der Entity → persist() Hinzufügen zu PC → remove() Transaktionscommit und damit Speicherung in DB


FetchType.LAZY Referenzierte Entität wird erst geladen, wenn nötig
FetchType.EAGER Sofortiges Laden der Entität
CascadeType.ALL impliziert PERSIST, MERGE, REMOVE, REFRESH, DETACH; default is CascadeType.NONE
mappedBy ... referenziert Feld im Owner
Persistence Concepts
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 →

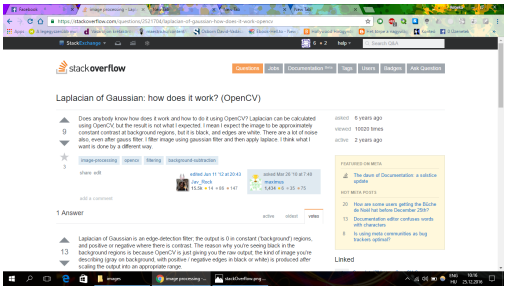

(X)HTML5
Semantic markup (excerpt)
<header> defines header of document or section
<nav> defines navigation region of page or section
<section> thematic grouping of content
<article> specifies independent, self-contained content
<aside> defines content aside from main content
<footer> defines footer of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

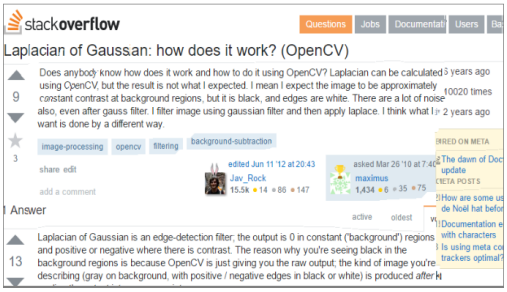

(X)HTML5
Semantic markup (excerpt)
<header> defines header of document or section
<nav> defines navigation region of page or section
<section> thematic grouping of content
<article> specifies independent, self-contained content
<aside> defines content aside from main content
<footer> defines footer of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

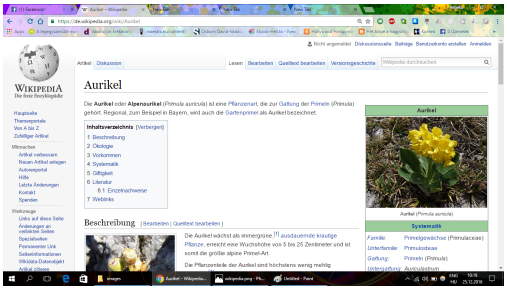

View all photos
Share
Zoom


Test
Some text
Some other text
Rainbow Dash

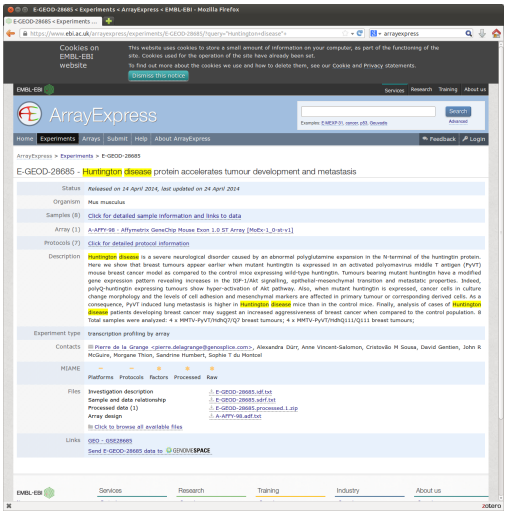
Test
Some text
Some other text
Rainbow Dash


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

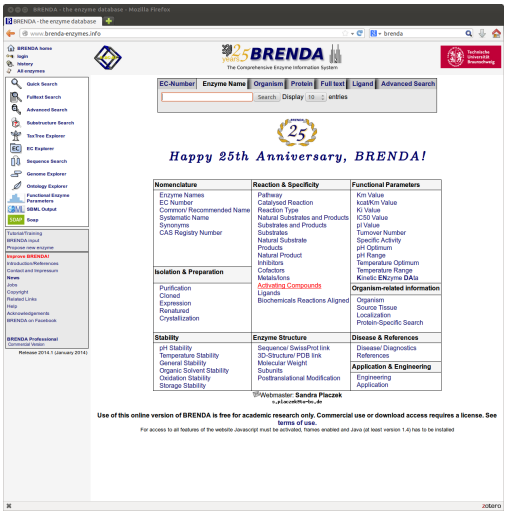

stackoverflow
Questions
Jobs
Laplacian of Gaussian: how does it work? (OpenCV)
Answer

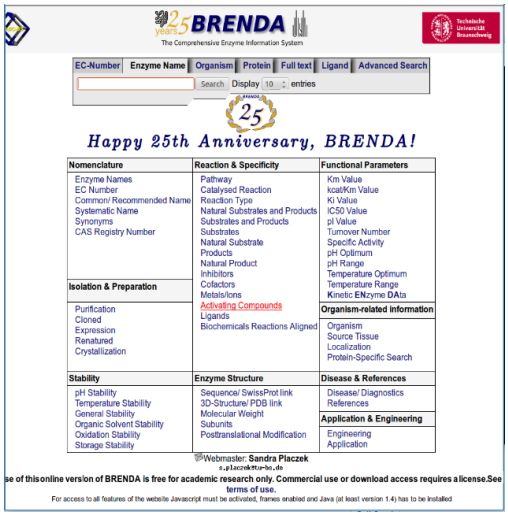

Wikipedia
Aurikel
Beschreibung

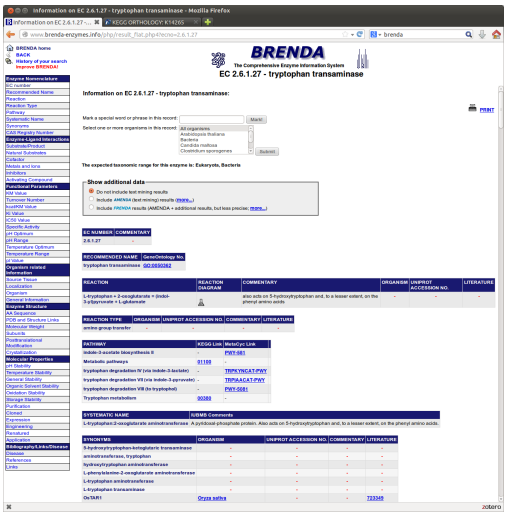
Aurikel
Inhaltsverzeichnis
Beschreibung
Systematik

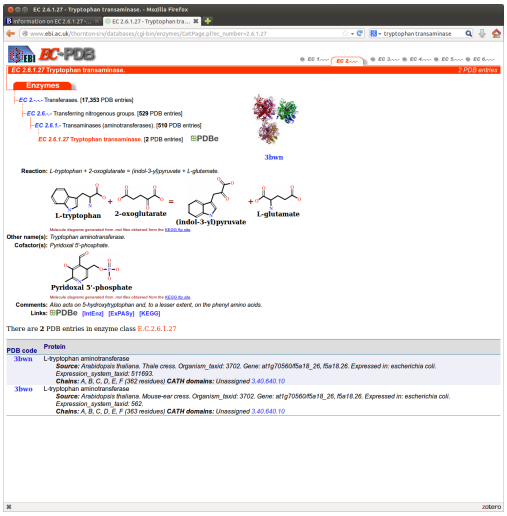

ArrayExpress
E-GEOD-28685 - Huntington disease protein accelerates tumour development and metastasis

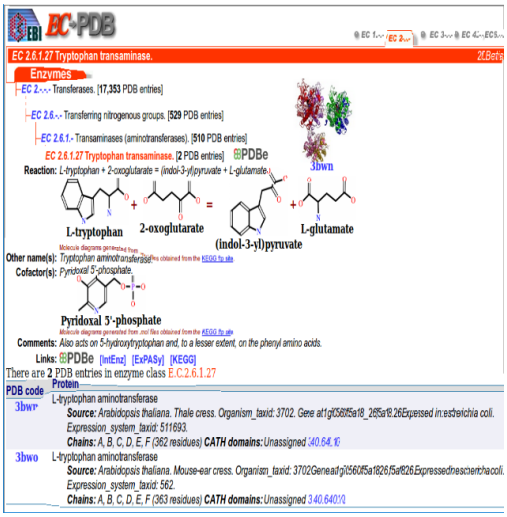
Huntington disease protein accelerates tumour development and metastasis

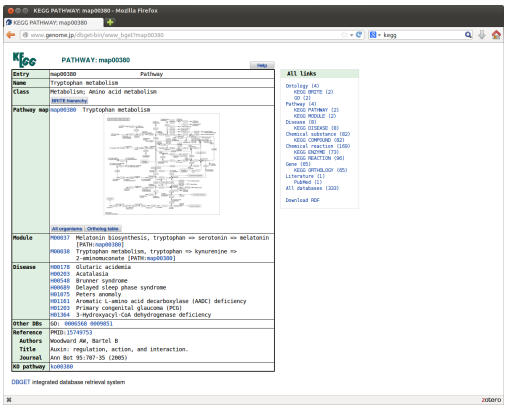

BRENDA
Happy 25th Anniversary, BRENDA!

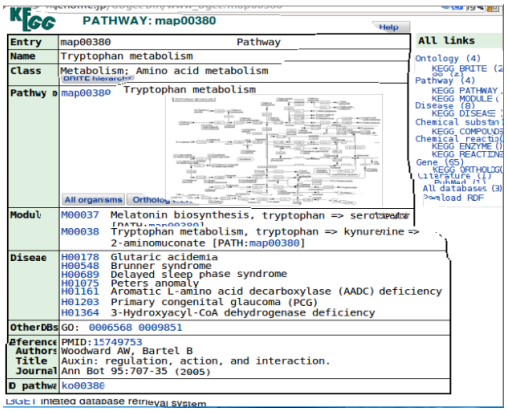

BRENDA
Happy 25th Anniversary, BRENDA!

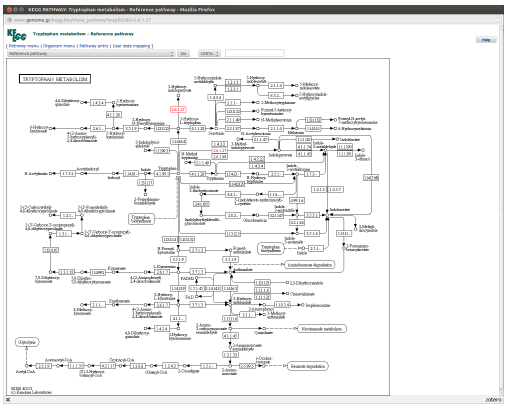


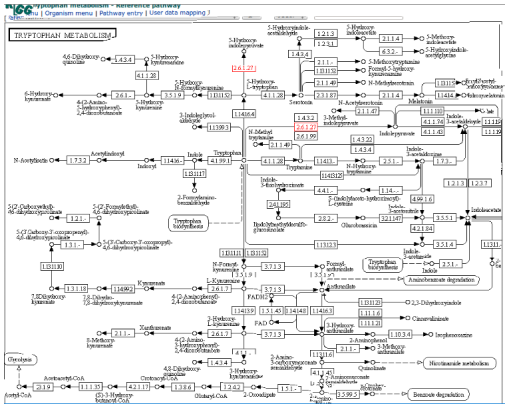




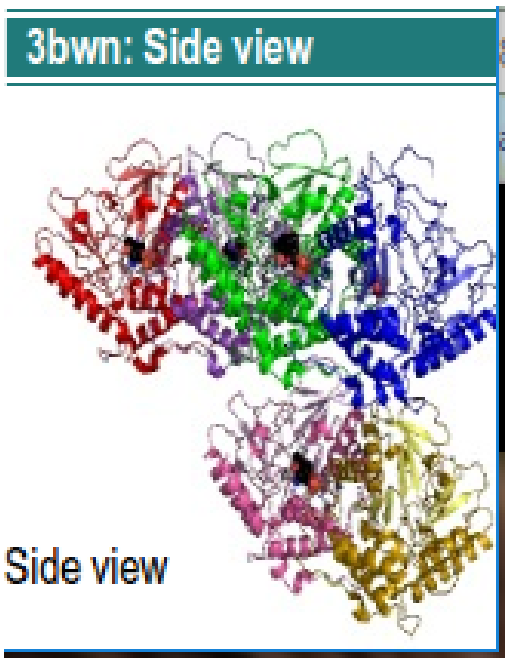






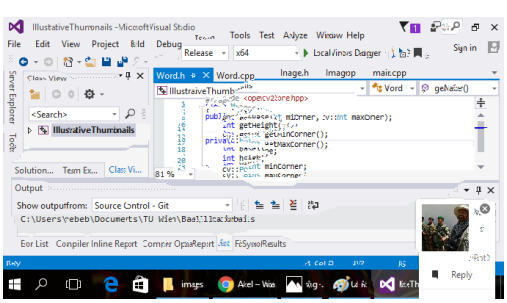

TRYPTOPHAN METABOLISM

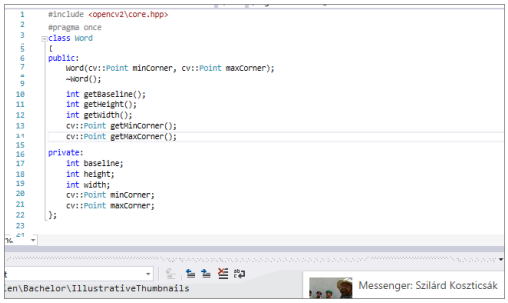
3bwn: Side view - Mozilla Firefox
www.ebi.ac.uk/thornton-srv/database
3bwn: Side view
Side view
zotero

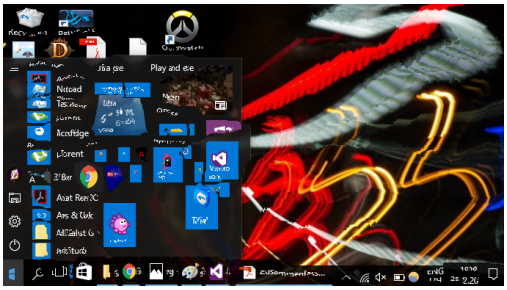

3bwn: Side view
Side view

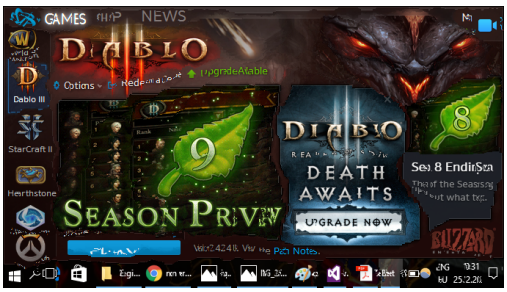
Appendix B

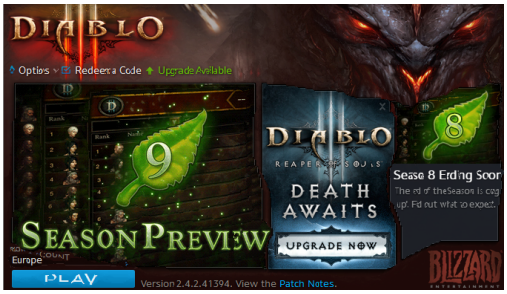
This section lists all reference images created by Photoshop. In case of the first column the input was the source image, in the second column the cropped image. The order of the sources is the same as in the section above.








GAMES
NEWS
DIABLO
Options
Redeem a Code
UpgradeAvailable
Diablo III
StarCraft II
Hearthstone
SEASON PREVIEW
DIABLO
REAPER OF SOULS
DEATH AWAITS
UPGRADE NOW
Season 8 Ending Soon
BLIZZARD


DIABLO
Options
Redeem a Code
Upgrade Available
SEASON PREVIEW
DIABLO
REAPER OF SOULS
DEATH AWAITS
UPGRADE NOW
Season 8 Ending Soon
Europe
PLAY
Version 2.4.2.41394. View the Patch Notes.
BLIZZARD

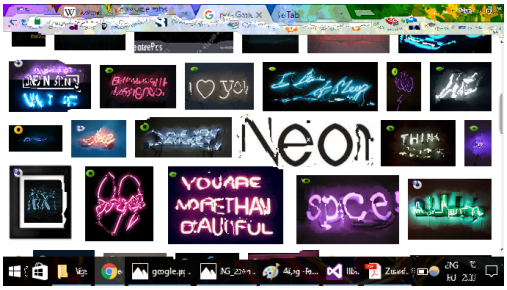
DIABLO
GAME MENU (ESC)
OPTIONS
ACHIEVEMENTS
CUSTOMER SERVICE
LEAVE GAME
EXIT DIABLO III
CURRENT SETTINGS
A ROYAL AUDIENCE
CAMPAIGN
PRIVATE GAME
OPEN GAME TO PUBLIC
LOWER DIFFICULTY
HARD
RETURN TO GAME

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

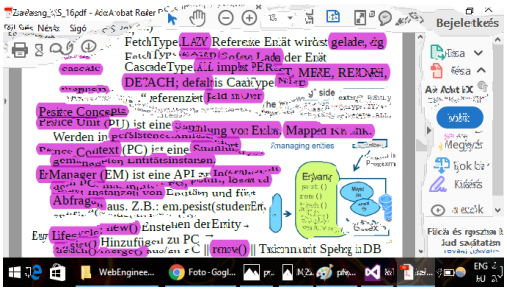
OBJECTIVES
A Royal Audience
NEW
Escape the Imperial Palace

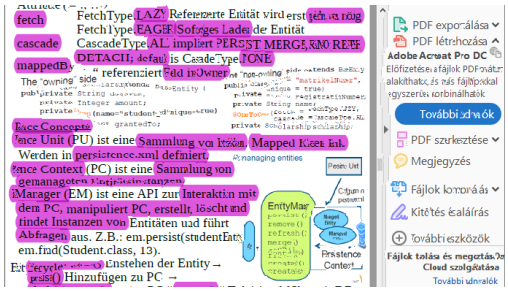
Chicagóban díjazták a Tisza szívvel
60 ÉV UTÁN DERÜL FÉNYRE A KONCENTRÁCIÓS TÁBOROK RÉMSÉGÉRE
Rá sem lehetsmeni Val Kilmerre
NEM VICC
KÜLFÖLD
Lekaszálta egy eneletes busz tetejét.ondonban egy híd
96 hétköznapi hősei
Joe Biden azt ívánja bácsa verekedhet-e Trumppal

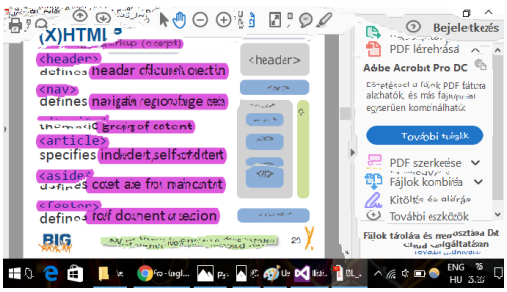
60 ÉV UTÁN DERÜL FÉNYRE A KONCENTRÁCIÓS TÁBOROK RÉMSÉGÉRE
Rá sem lehetsmeni Valkilmerre
NEM VICC
KÜLFÖLD
Lekaszálta egy eneletes busz tetejét.ondonban egy híd
96 hétköznapi hősei
Joe Biden azt ívánja, busz verekedhetne trumppal

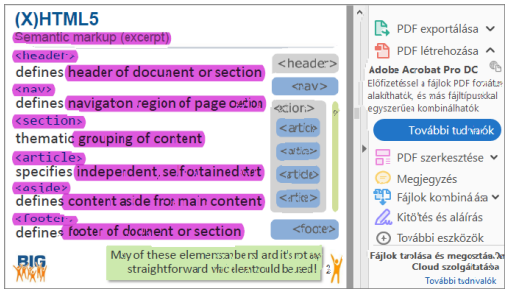

Neon
YOU ARE MORE THAN BEAUTIFUL
space

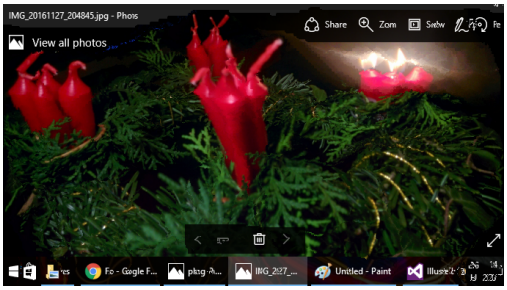
WHAT IF
JAZZ
I LOVE YOU TENDERLY TOTALLY TRAGICALLY
Neon
THINK OUTSIDE THE BOX
YOU ARE MORE THAN BEAUTIFUL
space


Bejelentkezés
FetchType LAZY Referenzte Entät wird erst geladen, eg
cascade
DETACH; defaltis CaadType
Persice Concepts
Werden in
Abfragen
Hinzufügen zu PC
remove()

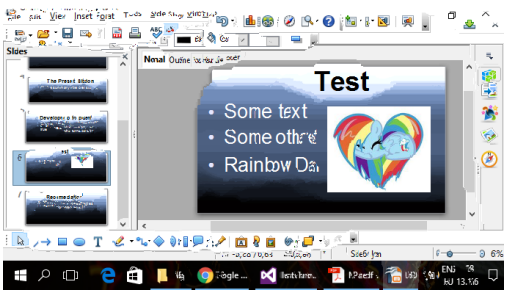

fetch
FetchType LAZY Referenzte Entät wird erst
cascade
DETACH; defaltis CaadType
mappedBy
Werden in persistence.xml definiert
Abfragen
em.fnd(Student.class, 13).
Hinzufügen zu PC
PDF exportálása
PDF létrehozása
Adobe Acroat Pro DC
További tudnivalók
PDF szerkesztése
Megjegyzés
Fájlok kombinálása
Kitöltés és aláírás
További eszközök


(X)HTML
<header>
defines header of document or section
<nav>
defines navigation region of page section
<article>
specifies independent, self-contained
<aside>
defines content as de from main content
<footer>
defines footer of document or section
Bejelentkezés
PDF létrehozása
Adobe Acrobat Pro DC
További tudnivalók
PDF szerkesztése
Fájlok kombinálása
Kitöltés és aláírás
További eszközök


(X)HTML5
Semantic markup (excerpt)
<header>
defines header of document or section
<nav>
defines navigation region of page section
<section>
thematic grouping of content
<article>
specifies independent, self-contained
<aside>
defines content as de from main content
<footer>
defines footer of document or section
<header>
<nav>
<section>
<article>
<aside>
<footer>
May of these elements are not always straightforward and should be used!
PDF exportálása
PDF létrehozása
Adobe Acrobat Pro DC
További tudnivalók
PDF szerkesztése
Megjegyzés
Fájlok kombinálása
Kitöltés és aláírás
További eszközök

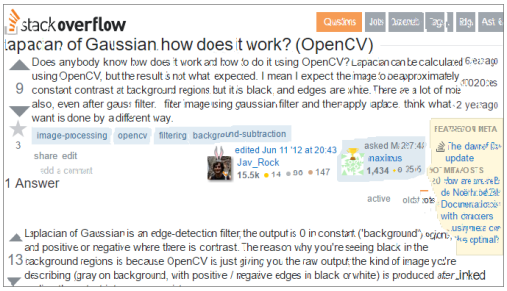

View all photos
Share
Zoom
Slideshow

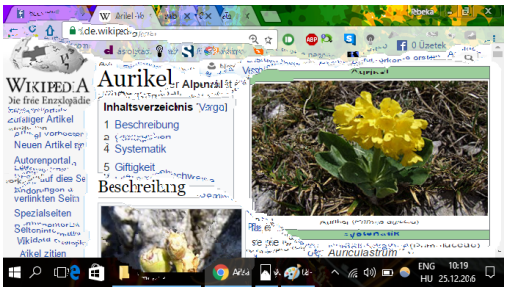
View all photos
Share
Zoom
Slideshow


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 Gaussian how does it work? (OpenCV)
Does anybody know how does it work and how to do it using OpenCV? Laplacian can be calculated
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
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
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 the output to an appropriate range.


WIKIPEDIA
Aurikel
Inhaltsverzeichnis
1 Beschreibung
4 Systematik
5 Giftigkeit
Beschreibung

Aurikel
Die Aurikel oder Alpenaurikel
Inhaltsverzeichnis [Verbergen]
1 Beschreibung
2 Ökologie
3 Vorkommen
4 Systematik
5 Giftigkeit
6 Literatur
7 Weblinks
Beschreibung [Bearbeiten Quelltext bearbeiten]
Aurikel (Primula Auricula)
Primelgewächse (Primulaceae)
Primuloideae
Primeln (Primula)