Photo Montage

School of Computer Science and Informatics

Cardiff University

CM3203 – One Semester Individual Project (40 Credits)

Student: Xinying Wang (C1466248)

Supervisor: Paul L Rosin

Moderator: Federico Cerutti

May 2016

Acknowledgement

Firstly, I would like to express the deepest gratitude to my supervisor - Professor Paul L Rosin, for his professional guidance and continual support throughout this project.

Secondly, I would like to thank my personal tutor – Professor David Walker, for providing valuable suggestions.

Abstract

Photomontage is the process and the result of making a composite photograph by cutting, joining and rearranging a group of photographs into a new image to show more of the theme than can be shown in a single photograph.

This project implements a photomontage system for automatically creating a composite picture from a collection of input images. The synthesised image can be viewed as a visual summary, and the resulting image should be representative of the collection, summarising its salient elements. However, if we create an image summary with original input images, it will contain lots of uninformative regions, and the only few images can be selected. Firstly, in order to maximise visible visual information on a given space, we crop the input images by using a quick and computationally efficient salient region detection algorithm. Second, join images together without overlay by using two-dimensional rectangle bin packing algorithm. This step aims to summarise the informative regions and maximise canvas coverage. However, the chance of perfectly fitting all images into a rectangle canvas is very tiny, and inevitably there will be some holes produced by image arrangement. Third, in order to hide the holes generated by rectangle packing, a digital inpainting technique is used.

This project implements a frequency- tuned salient region detection algorithms with an additional face detection function, two different rectangle packing algorithms, three different inpainting algorithms and five methods to improve the runtime of the exemplar-based inpainting. The report also explains and discusses several related algorithms – including concepts relating to k-means clustering, integral image and face detection. Due to the high runtime, a sequence of speed optimization methods has been developed to improve the system performance.

Visual and runtime evaluation indicate the efficiency of the photo montage system, this project shall be evaluated on its ability on each sub-system (saliency extraction, two-dimensional rectangle packing, digital inpainting) respectively and also be assessed as a whole.

Contents

A list of figures

Introduction

With the rapid development of digital cameras and popularisation of mobile phones, people are more willing to take pictures because of the convenience and high resolution. With a large amount of pictures produced every day, organising and summarising this vast amount of data is important. Photomontage is a technique to make a composite photograph by cutting, cropping, joining and rearranging a set of pictures.

1.1 Aims and Objectives

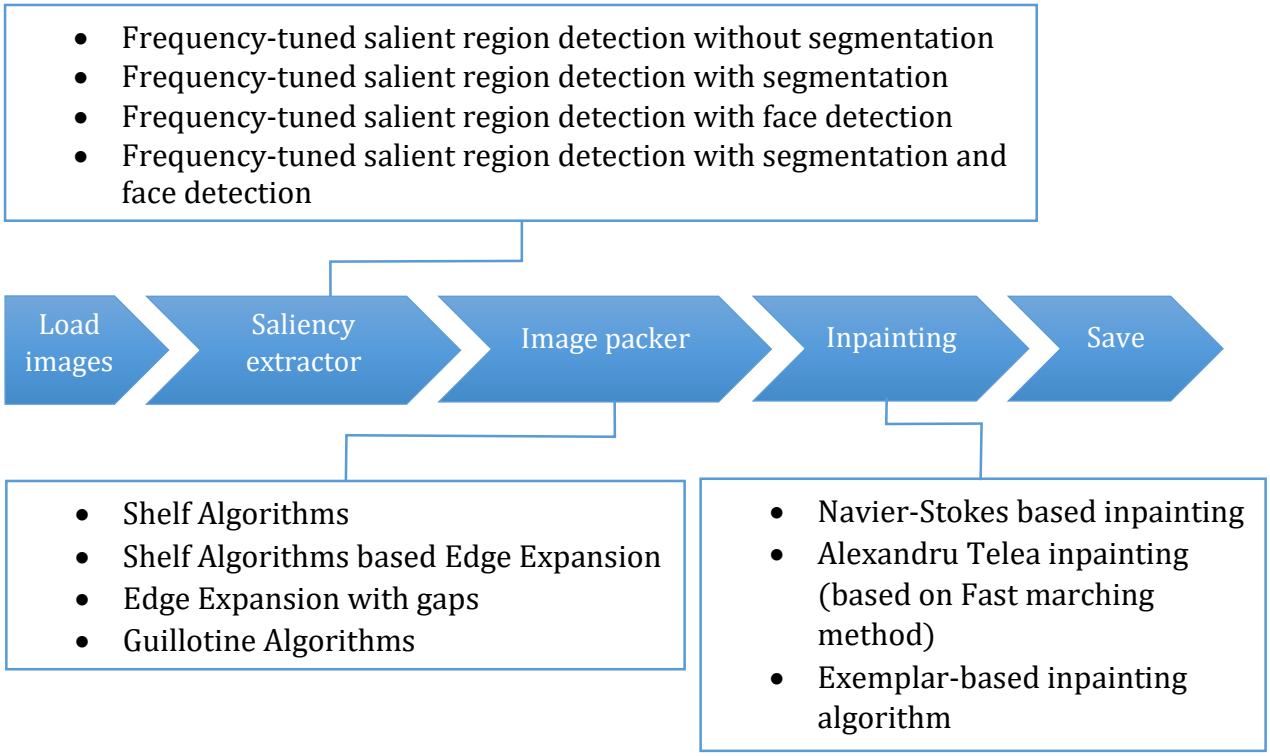
The aim of this project is to devise and implement a photomontage system that can automatically specify salient regions on a set of photographs and construct them into a single composite seamless picture. In order to create an ideal image summarization which contains as many as possible informative regions, input images are cropped with considering image content by using saliency region detection methods. To summarise these representative regions, different two-dimensional bin rectangle packing algorithms are used with considering space utilisation. This step aims to pack rectangles together into a container as densely as possible. As producing gaps is a nature problem when packing rectangles, we need an inpainting method to fill these extra space.

The main objective can be split into a series of goals, including researching approaches surrounding visual saliency detection, rectangle packing, research digital inpainting techniques; implementing the system itself; improving performance and speeding up running speed; testing and evaluating the functionality of the system created.

Desirable properties include: meaningless image fragments are avoided, faces are preserved whole, holes are minimised and inpainting can produce a good result. The system should be time efficiency and quality guaranteed, for which system testing, debugging and optimization are required.

1.2 Project structure

This photo montage system can be split into three modules, including saliency extractor, image packer and digital inpainting. Each module contains different methods to achieve its goal.



1.3 Approach

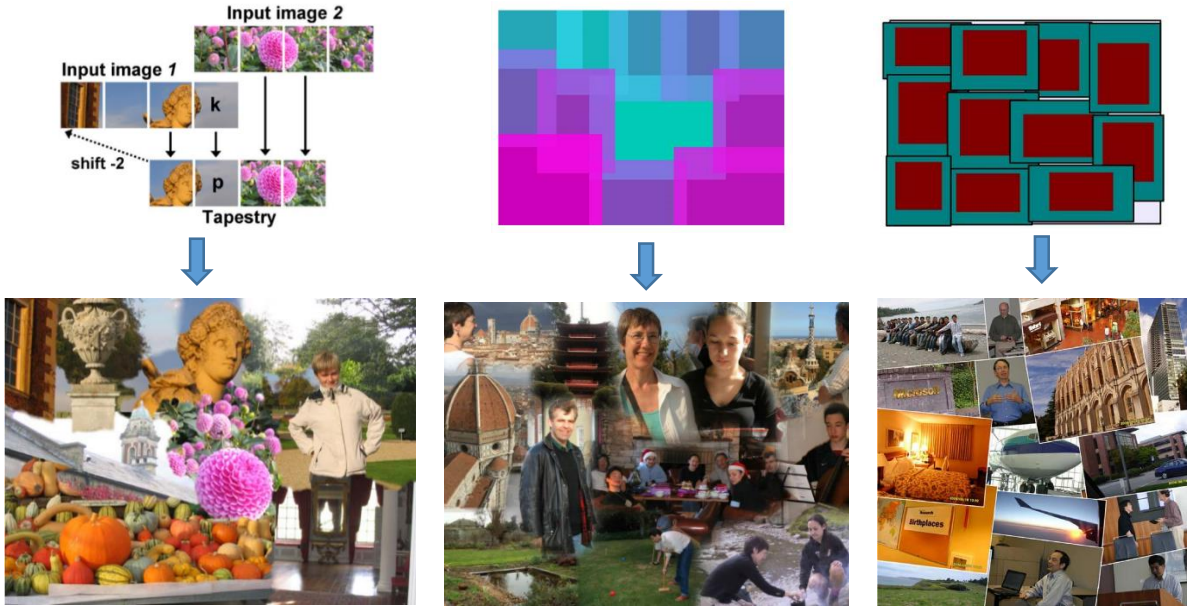
I use a top-down approach while designing my system. It breaks the system into three chunks and allows me to tackle each problem one at a time. The initial system includes only the relatively simple algorithms which get the system up as quickly as possible. It also allows me to refine each algorithm, build upon it and add more approaches in any order after the basic framework of the system is formed and works successfully.

In short, break down the system to gain sub-systems, tackle each sub-system with a relatively simple approach and get the entire system working before optimise it.

In this project, I use Java to develop my system. There are some open source computer libraries can be used, for example, BoofCV and OpenCV. OpenCV has an official version for Java while BoofCV is a new real-time computer vision library written in Java. Both of them contain lots of different image processing algorithms implemented and high performance. Consider that I have some grounding in OpenCV, I take OpenCV as my external library to support the programming of this project.

Background

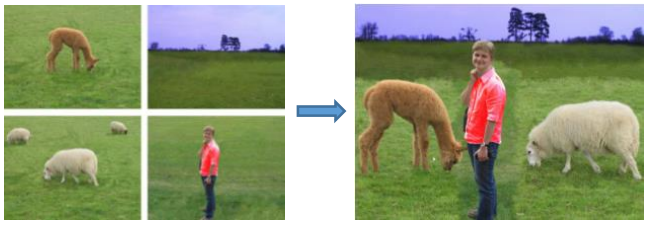
2.1 Related Work



(a)Digital Tapestry [2] - 2005 (b) Auto Collage [3] - 2006 (c) Picture Collage [4] - 2009

Figure 1. Related works. Image source: [2][3][4]. (a) Digital Tapestry: assemble patches from a batch of repeat shots of a scene. (b) Auto Collage extracts rectangle saliency regions from a set of image, arrange images with overlay style and performs edge-sensitive blending in the α-channel. (c) Picture Collage defines saliency regions using a set of weighted rectangles and joins image using an album style.

There have been many methods developed to summarise visual data automatically. Digital Tapestry [2] presents each image as a set of blocks and match a block from an input image to the tapestry. The advantage of this method is that if the input set of images are of the same scene (as shown below), it may produce a composite picture with relatively high coherence and fewer artefacts.



(a) Input images (b)Result

Figure 2. Digital Tapestry. Input images are of the same scene.

The limitation of Digital Tapestry is the computational complexity. Each region in the tapestry is assembled from salient input image block considering uniqueness constraint and block coherence constraint, it searches over input image blocks and finds a matching neighbouring block, hence is computationally intensive. And also, artefacts are introduced along the joins from two different images.

Auto Collage [3] was researched in 2006 by Microsoft and later, AutoCollage 2008, a Microsoft photomontage desktop application was initially released in Sep, 2008 and lastly updated in 2011. It sales 19.95 dollars at the Microsoft Store online which shows that Auto Collage is an existing state of software and has been used commercially.

Auto Collage automatically creates blended image collages, it defines four terms of energy to encourage the selection of representative images from the input image set, ensure informative and salient regions from selected image is extracted, treat certain materials with particular respect(e.g. human face, sky), and encourage a seamless and smooth layout. There are four steps in the Auto Collage framework, from the static ranking of images, through salient region detection, images packing by the branch-and-bound algorithm and lastly edge-sensitive blending in α-channel. Auto Collage can have a beautiful result with its blending technology but the blending still introduces artefacts on the boundaries of different images. In addition, Auto Collage weaken image boundaries, making the result feeling soft and smooth, so that it may confusing people when the number of images increases because the unclear boundaries. Another limitation of this method is that it cannot deal with images with multiple salient regions.

Picture Collage [4] use multiple weighted rectangles to indicate salient regions, and greater weight is assigned to the centre. It joins image with overlay style and considering salience ratio balance. This collage style is more common in people's daily life which generally can be seen in albums.

Compared with Digital Tapestry, Auto Collage and Picture Collage use overly style to avoid artefacts caused by tapestry; however, Auto Collage introduces transparent blending to soft image boundaries and the blending may also bring artefacts. Picture Collage is much faster than Auto Collage when dealing with a large number of images, and can clearly summarise hundreds of images because it retains image boundaries.

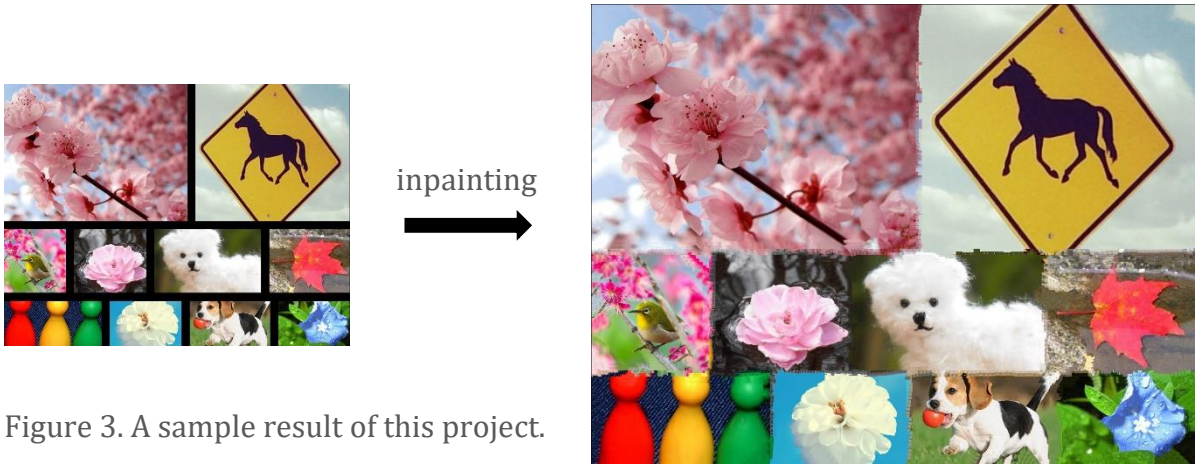
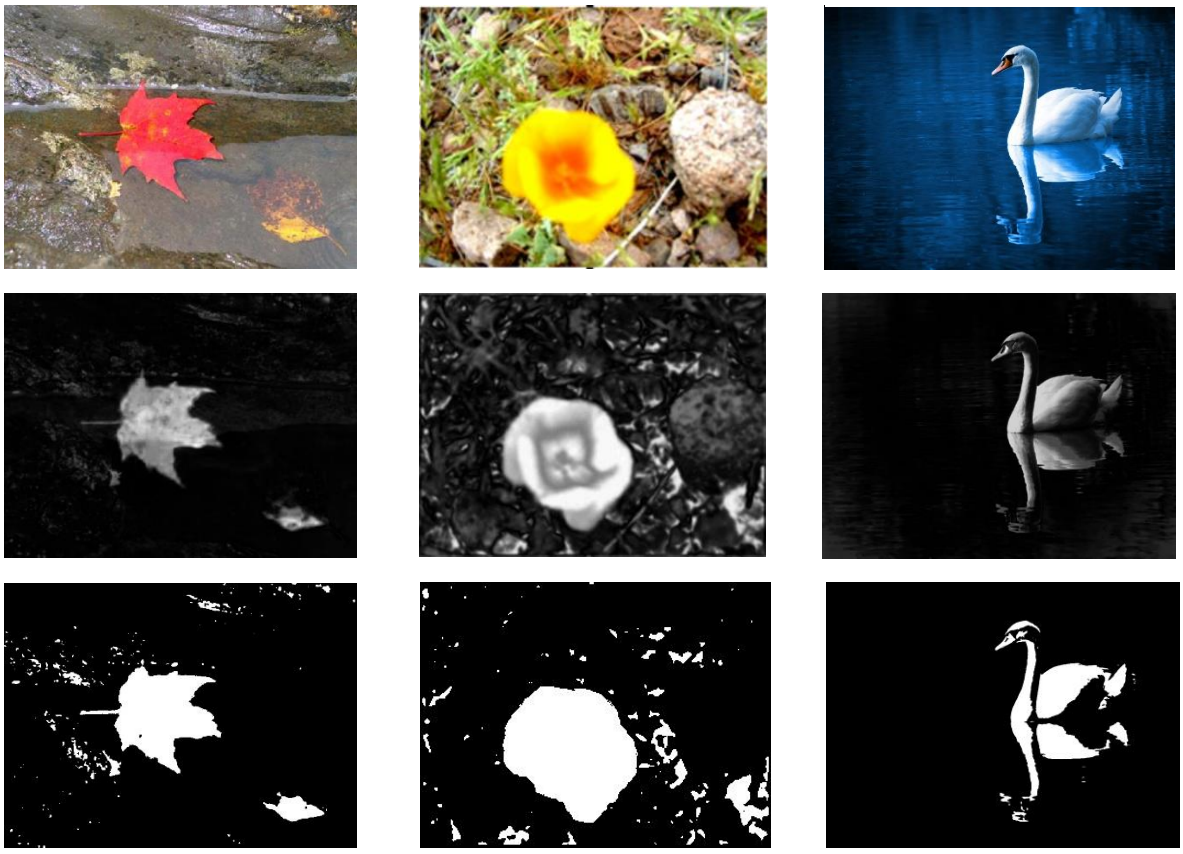


Figure 3. A sample result of this project.

In this project, each image is represented by a rectangle enclosing the most salient region, which is similar to Auto Collage and Picture Collage. Compared with the blending style of Auto Collage and overlay style of Picture Collage, this project packs images without allowing overlay and uses inpainting technique to inpaint the gaps between images.

2.1.1 Salient Region Detection



(a) (b) (c)

Figure 4. From top to bottom: input images, saliency maps and binary images generated by Frequency-tuned salient region detection algorithm.

The first step that most of the paper detailed is to determine which regions of the input images are representative or salient and should be selected. In Figure2, the leaf, flower and swan stand out relative to its background and attract the most visual attention, and they are the salient objects we need to detect and extract. The saliency map indicates the importance of each pixel.

There are several methods for automatic detection of visually salient regions, most of them are bottom-up framework based [6]-[9], which can split into three steps:

1. Feature extraction: by using one or multiple features of intensity, colour, luminance, texture, orientation and motion to determine the contrast of objects relative to their surroundings. (This project uses colour and luminance features).
2. Saliency computation: compute saliency map by using centre-surround feature distances [6]-[9] or other methods. (This project measures centre-surround feature distances using a Difference of Gaussians approach).
3. Identify the location of the salient region on the salient map. The approach in [7] introduces a set of linearly weighted rectangles to enclose the regions. In this project, the salient region is enclosed by a rectangle.

In this project, the approach in [9] is adopted to use for saliency detection.

2.1.2 Rectangle packing

This project is also related to two-dimensional rectangle bin packing algorithm for image arrangement. Two-dimensional rectangle bin packing problem is known to be NP-complete. In this problem, given a sequence of rectangles of different size and the goal is to find a packing of these rectangles into a minimum number of bins of size. Intersection and overlapping between rectangles are not allowed. For the two-dimensional rectangle bin packing problem, one version is called online rectagnle bin packing, receiving one rectangle at a time and placing it into bins immediately without any knowledge of the next items. The opposite to this is called offline rectangle bin packing, in which the whole sequence to pack is known in advance. In this project, we consider the latter one because the input images are known. And also, we are not considering rotatable rectangle bin packing which allows that rectangle may be rotated by 90 degrees because the input images of this project contain natural scenery and people.

The two-dimensional rectangle bin packing problem does not exist an optimal solution currently but lots of work has been done to develop efficient algorithms that can produce good result.

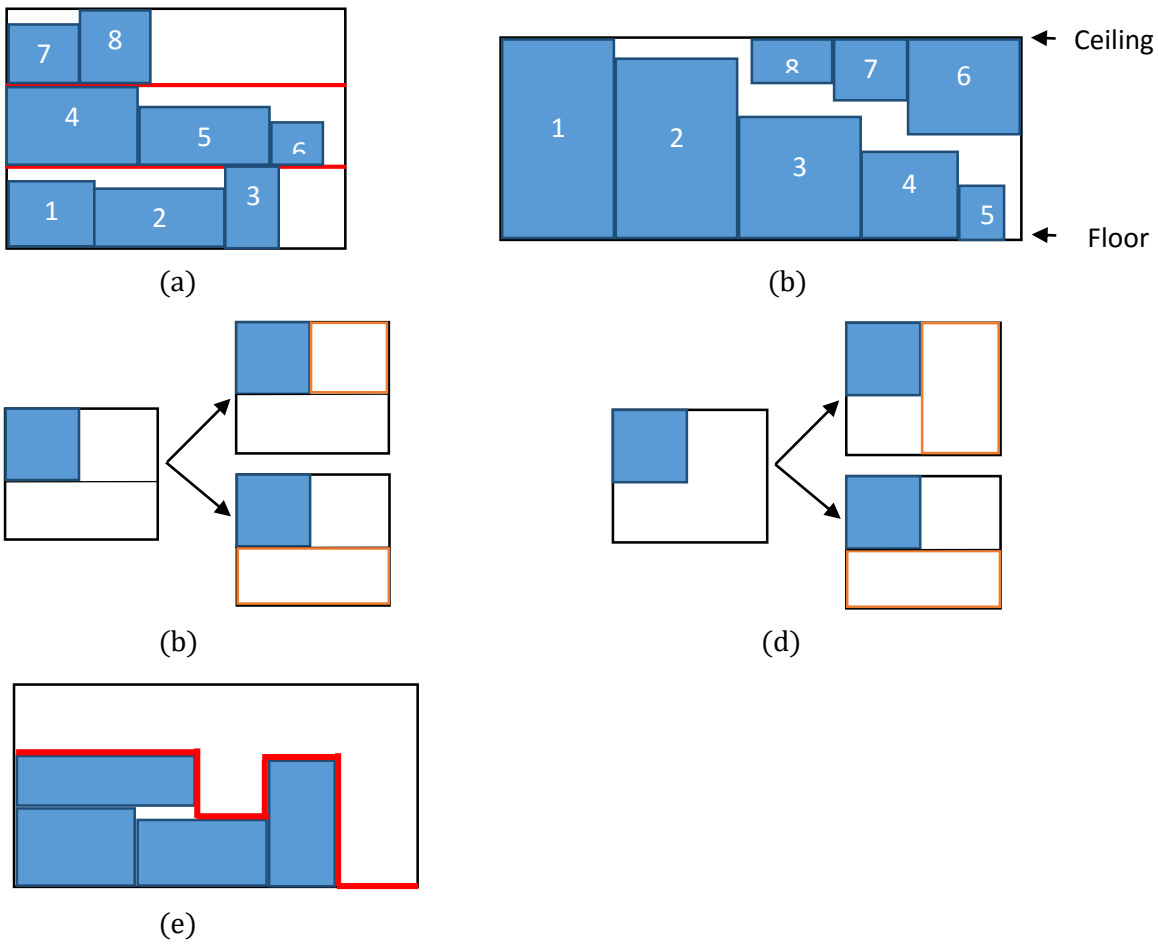


Figure 4. Two-Dimensional Rectangle Bin Packing algorithms. (a) Shelf algorithm, (b) Floor-ceiling algorithm, (c) Guillotine algorithm, (d) Maximal rectangle algorithm, (e) Skyline algorithm.

There are several algorithms for two-dimensional rectangle bin packing [10]:

- Shelf algorithms (or level algorithms): the simplest methods to produce packings. Pack rectangles from left to right, in rows forming level. It places the next rectangle R on the current level if fits, otherwise create a new level and accommodate R in new level. Time complexity: O(n · log n).
- Floor-ceiling algorithm: Consider a particular level. Place the rectangle R with decreasing size on the floor from left to right and if the new rectangle does not fit on the floor in the same level them place it on a ceiling.
- Guillotine algorithm: After placing an item, split the rectangle into 2 smaller rectangles to store the remaining free space. Time complexity: $O(n^2)$.
- Maximal rectangle algorithm: based on an extension of the guillotine algorithm. After placing an item, split the rectangle into 2 smaller rectangles in both split axes.
- Skyline algorithm: it maintains a list of the 'skyline' edges formed by the topmost edges of previously packed rectangles.

2.1.3 Digital inpainting

Image inpainting is the process of reconstructing missing or corrupted parts of the images and videos; it provides a means for concealment of a damaged image.

Auto Collage [3] allows image overlay; it creates a soft, transparent blend to hide the seam by producing an alpha mask (defines the way of the pixel's colour merged) for each image. Picture Collage [4] use overlay style to maximise the canvas coverage and it does not attempt to hide the borders of images so there are not inpainting methods used. In this project, image overlay is not allowed, and holes will always exist as the different size of image so the inpainting is required to fill holes generated by the image packing.

There are different categories of image inpainting algorithms, including texture synthesis based image inpainting, Exemplar and search based image inpainting.

- **Texture synthesis based image inpainting**

In this method, holes are filled by sampling and copying neighbouring pixels. [13][14]

The Navier-Stokes based inpainting method [13] uses ideas from classical fluid dynamics to maintain continuity between hole's pixel and original image pixels. It views the image intensity as a stream function and utilizes partial differential equations. It propagates isophote lines continuously from the outside into the target inpainting region, filling pixels from surrounding pixel data.

The Telea inpinting technique [14] is based on the fast marching method. Compared with the Navier-Stokes based inpainting method which are complex to understand, the Telea inpinting technique is simple. This method propagates pixel data inward from the boundary of target inpainting region, the missing pixel is filled by normalized weighted sum of all the known pixels in a small neighbourhood around the pixel to be inpainted.

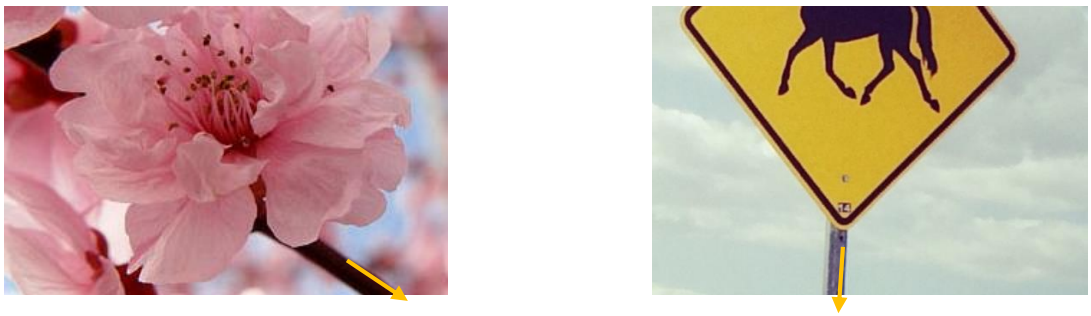
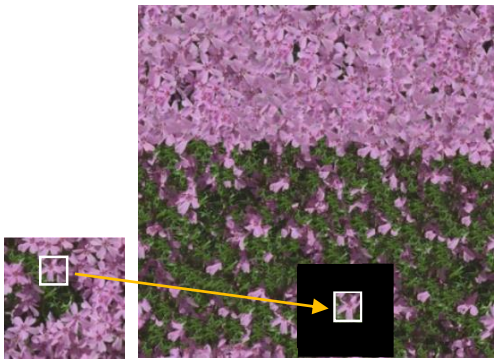


Figure 5. Isophote definition

There are some 'level lines' of the same intensity or a contour of equal luminance in an image, they are called isophotes.

- **Exemplar and search based image inpainting**

In this method, holes are filled by searching and matching a patch to be inpainted from a known patch.



Sample Damaged image

Figure 6. Searching patch in a sample image

Damaged image

Figure 7. Searching itself

- With "pure" textures – starts from a small source sample image and copy the similar patch from sample image to the damaged image [11]. It suits for an image which has a repetitive occurring textural patterns while it has difficulty filling missing data in pictures of real world scenes.
  This project contains different kinds of image, there are photographs of the natural sights, animal, plants and people. Because of the constraint of image category, we are not consider the pure textures in this project.
- Exemplar based – starts from searching itself to find a similar patch and copying colour values to fill the missing data from the undamaged part of the image. [12]

The inpainting approach in [12] combine both exemplar based method and linear structures for which following the isophote direction. In this project, we adopt it to do the digital inpainting.

Methods & Algorithms

This section details each algorithms that have been used in this project. Equations and graphs are used to support my explanation and also intended to assure readers can fully understand the way of these approaches work.

3.1 Saliency extraction

The approach in [9] is adopt to use for saliency detection because it is efficiency and can uniformly highlight the salient object. This approach considers colour and luminance features in Lab colour space and use the Euclidean distance to estimate centre-surround contrast. To highlight the salient object, this algorithm produces a saliency map by combining the outputs of several band pass filters.

3.1.1 Why combine DOG band pass filters

DOG (different of Gaussian) filter and LOG (Laplacian of Gaussian) filter are band pass filters which can be used for edge detection and intensity changes detection. In this project, a summation over DOG is used for saliency detection. The equations of DOG filter is given below:

$$\mathrm{DOG}(\mathrm{x},\mathrm{y}) = \frac{1}{2\pi}\left[\frac{1}{{\sigma_1}^2}\, e^{\frac{-(x^2+y^2)}{{\sigma_1}^2}} - \frac{1}{{\sigma_2}^2}\, e^{\frac{-(x^2+y^2)}{{\sigma_2}^2}}\right]$$

$$= \mathrm{G}(\mathrm{x},\mathrm{y},\sigma_1) - \mathrm{G}(\mathrm{x},\mathrm{y},\sigma_2)$$

$\boldsymbol{\sigma_1}$ and $\boldsymbol{\sigma_2}$ are the standard deviation of the Gaussian ($\boldsymbol{\sigma_1} > \boldsymbol{\sigma_2}$). $\boldsymbol{\sigma_1}$ and $\boldsymbol{\sigma_2}$ control the passband width of the DOG filter.

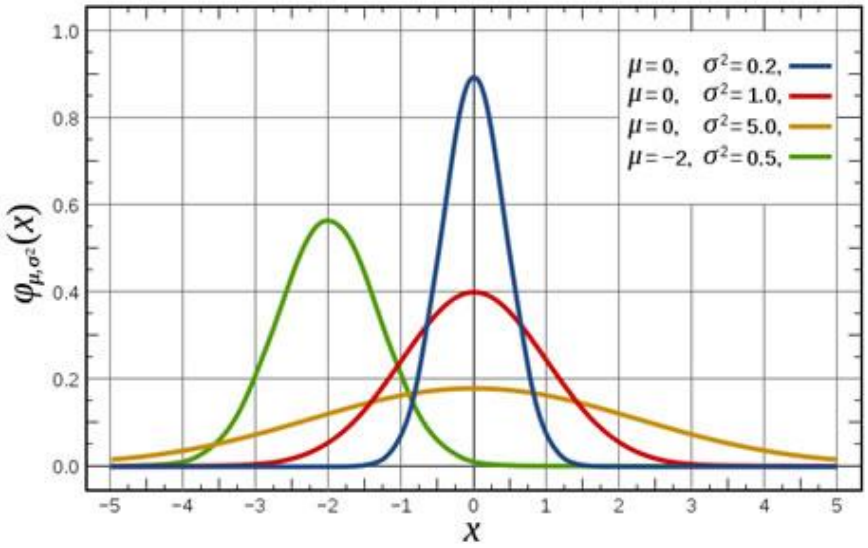


Figure 8. The probability density function of Gaussian distribution. The Image comes from wikipedia.org

If define $\frac{\boldsymbol{\sigma_1}}{\boldsymbol{\sigma_2}} = \boldsymbol{\rho}$ (keep $\boldsymbol{\rho}$ constant at 1.6, as need for ideal edge detector), we have $\boldsymbol{\sigma_1} = \boldsymbol{\rho\sigma}$ and $\boldsymbol{\sigma_2} = \boldsymbol{\sigma}$. Considering a combination of several DOG filters, then we have ($\mathbf{n}$ is an integer and $\mathbf{n}\geq 0$):

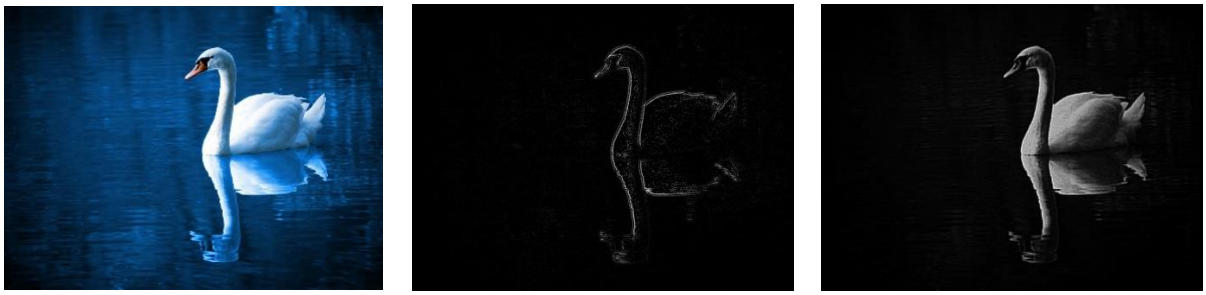
$$\sum_{0}^{n}[G(\mathrm{x},\mathrm{y},\rho^{n+1}\sigma) - \mathrm{G}(\mathrm{x},\mathrm{y},\rho^{n}\sigma)]$$

$$= G(\mathrm{x},\mathrm{y},\rho^{n+1}\sigma) - \cancel{\mathrm{G}(\mathrm{x},\mathrm{y},\rho^{n}\sigma)} + \cancel{G(\mathrm{x},\mathrm{y},\rho^{n}\sigma)} - \cancel{\mathrm{G}(\mathrm{x},\mathrm{y},\rho^{n-1}\sigma)} + \cdots$$

$$+\cancel{G(\mathrm{x},\mathrm{y},\rho^{n+1}\sigma)} - \cancel{\mathrm{G}(\mathrm{x},\mathrm{y},\rho^{n}\sigma)} + \cancel{G(\mathrm{x},\mathrm{y},\rho^{1}\sigma)} - \mathrm{G}(\mathrm{x},\mathrm{y},\rho^{0}\sigma)$$

$$= G(\mathrm{x},\mathrm{y},\rho^{n+1}\sigma) - G(\mathrm{x},\mathrm{y},\sigma)$$

One DOG filter can be an edge detector, adding up several DOG filter which means summarizing the output of several edge detectors. So that rather than just highlight the object edge, the salient regions will be uniformly covered.



(a) (b) (c)

Figure 9. (a) Original image. (b) Apply one DOG filter, highlight only object edges. (c) Apply several DOG filters (Gaussian($\boldsymbol{\sigma}$=40) – Gaussian ($\boldsymbol{\sigma}$=5)), highlight whole object.

3.1.2 Computing saliency and highlight the salient regions

As can be seen from Figure 7, the probability density function of Gaussian distribution tends to be flat with the increasing **σ.** Diving ($\boldsymbol{\rho^{n+1}\sigma}$) to infinity means a large ratio in standard deviations. In image, which is 2-dimensional space, it means infinitely large Gaussian kernel standard deviation, which is close to image average. So use image average to estimate Gaussian filter whose kernel standard deviation is infinitely large.

$\boldsymbol{I}_{\mu}$ presents the arithmetic mean pixel value of the image, which is determined by ($\boldsymbol{\rho^{n+1}\sigma}$). $\boldsymbol{I_{whc}}$ presents the pixel value of Gaussian blurred image (using a 5×5 kernel size), which is determined by $\boldsymbol{\sigma}$.

Use the Lab colour space and find the Euclidean distance between the Lab pixel vectors (L_{μ} is the mean image vector and $I_{whc}(x,y)$ is corresponding image pixel vector in the Gaussian blurred version of the original image).

The formula is written as:

$$S(x,y) = \| I_{\mu} - I_{whc}(x,y) \| = [I_{\mu} - I_{whc}(x,y)]^2$$
$$= [L_{\mu} - L_{whc}(x,y)]^2 + [a_{\mu} - a_{whc}(x,y)]^2 + [b_{\mu} - b_{whc}(x,y)]^2$$

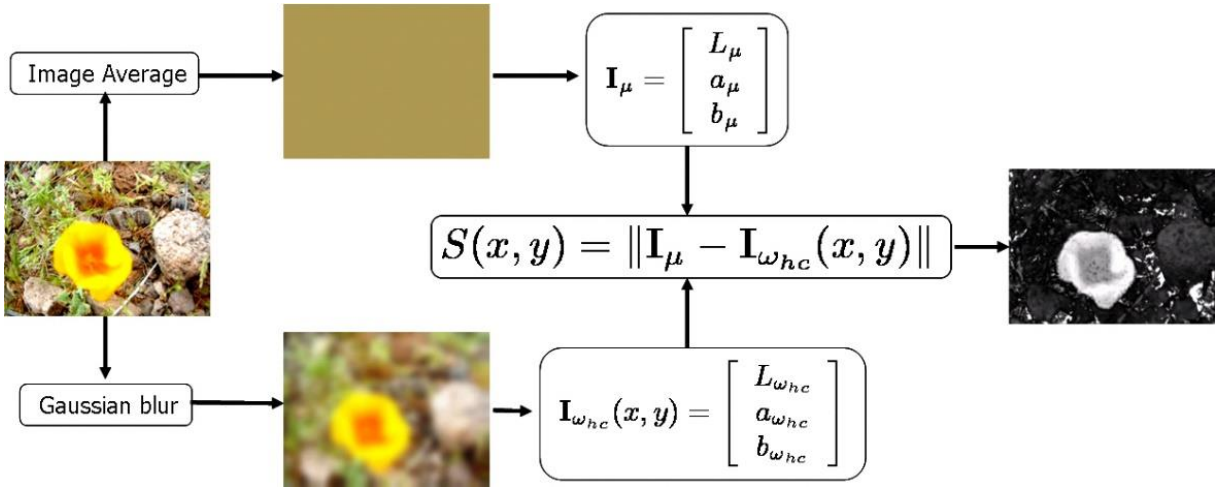


Figure 10. Flow chart of the Frequency-tuned saliency detection algorithm. Image from IVRG (Images and visual representation group).
ivrlwww.epfl.ch/supplementary_material/RK_CVPR09/

3.1.3 Threshold to binarize the saliency map.

Thresholding is the simplest method of image segmentation. In order to segment salient object from the background, we need to binarize the saliency map by assigning ones to salient object pixels and zeros to the background.

Rather than using a fixed threshold to binarize the saliency maps, an adaptive threshold ($\boldsymbol{T_a}$) value is used because it is image saliency dependent. The adaptive threshold ($\boldsymbol{T_a}$) is defined as two times the average saliency of the input image:

$$T_a = 2 \times \frac{\sum_0^{width-1} \sum_0^{height-1} Saliency(x,y)}{width \times height}$$

And also, two approaches for saliency map generation are employed. One is a simple thresholding which is quick and easy to implement; another is thresholding with image intensity segmentation, which can eliminate some noises.

- Simple thresholding: binarize the saliency map by each pixel, retain only those pixels whose saliency value is greater than T_a. The binary maps are generated by assigning ones to the chosen pixels and zeros to the rest of pixels.
- Thresholding with image segmentation: binarize the saliency map with intensity segmentation [7][9]. Use the k-means clustering to segment image and retain only those segments whose average saliency is greater than T_a. The binary maps are generated by assigning ones to the chosen segments and zeros to the rest of segments. (K-means clustering is explained in 3.1.5)

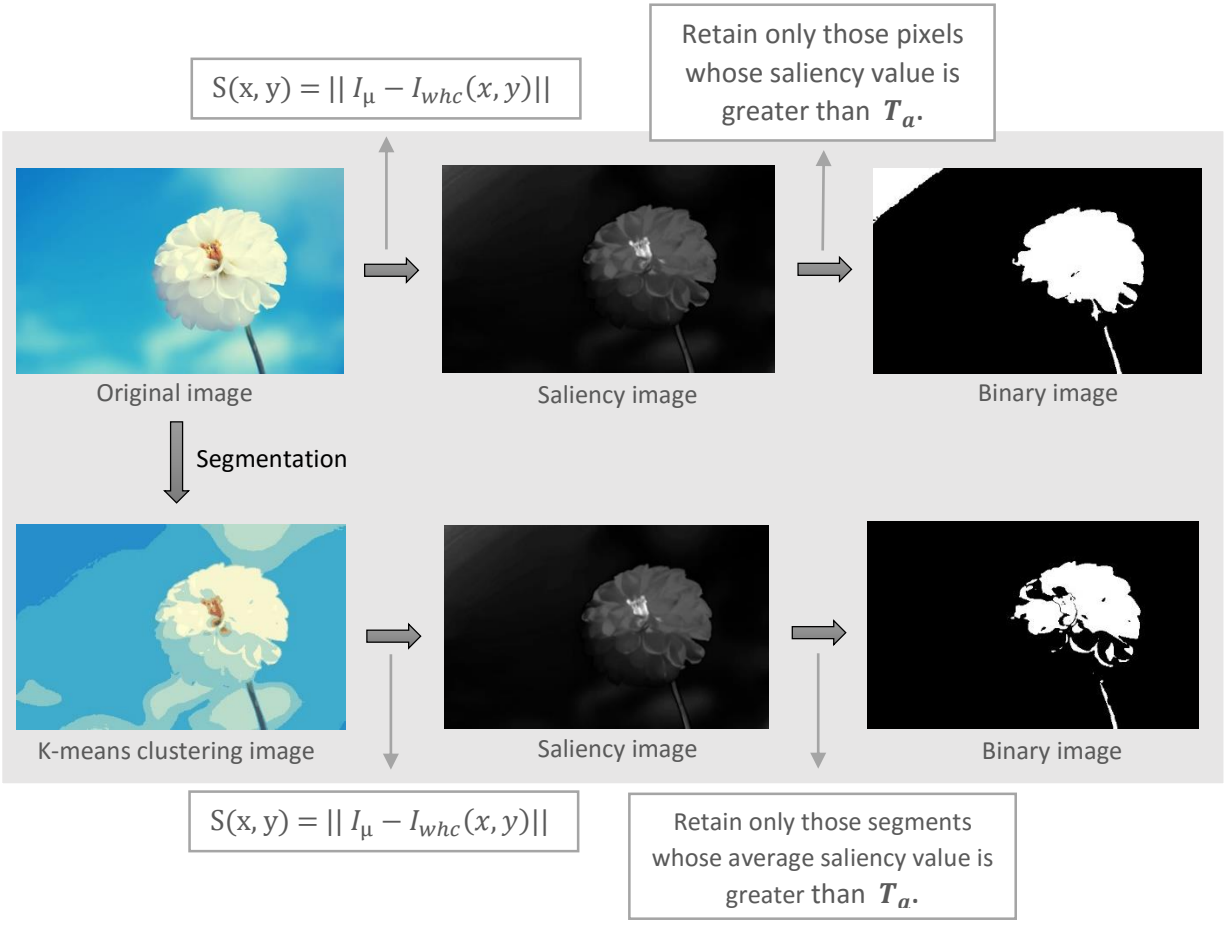
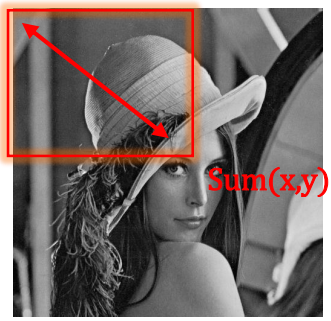


Figure 11. Threshold to binarize saliency maps.

3.1.4 Computing integral image (summed area table).

After we got binary maps, the next step is to use a rectangle to enclose the most salient region. Instead of recomputing sums of rectangles for saliency value at every re-scaling, form an integral image (summed area table) which computes all sums at the beginning. The value at any point (*x*, *y*) in the integral image (summed area table) is the sum of all the pixels above and to the left of (*x*, *y*) in the original image.



The integral image algorithm can quickly and efficiently generate the sum of values in a rectangular subset of a grid. It effectively reduces the computational complexity from O(n) to O(1).

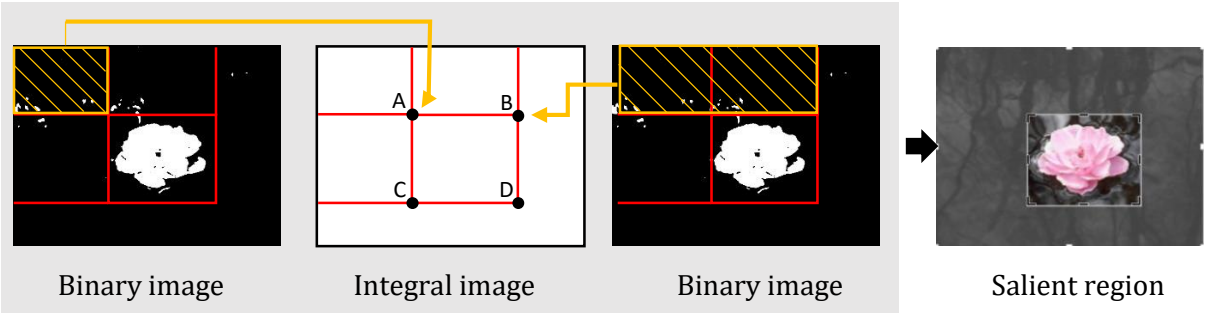


Figure 12. Integral image (summed area table) algorithm

After finishing the integral image generation, we search for the minimal rectangle which contains more than 85% of the highlighting area, in other word, find 4 points in integral image on the condition that I(sum)> I(max)×85%. I(max) is the point in the bottom right which is the sum of all the pixels in the binary image.

3.1.5 K-Means clustering algorithm

K-means clustering aims to partition n observations into k cluster and find centres of clusters and groups input samples around the clusters. K-means clustering is an iterative algorithm, it works as follow:

1. Decide the number of clusters k.
2. Randomly choose the centre of the clusters.

3. Attribute the closest cluster to each data point.
4. Set the position of each cluster to the mean of all data points belonging to that cluster.
5. Repeat steps 3-4 until convergence (until the position of each cluster unchanged).

In this project, K-means clustering algorithm is used in saliency extraction and digital inpainting. K-means clustering algorithm used in saliency extraction aims to produce the segmented image which may eliminate certain noises and generate better binary image. K-means clustering algorithm used in digital inpainting aims to classify patches into several clusters which in order to narrow down the search and increase the speed.

3.1.6 Viola Jones face detection algorithm

Face detection is a specific case of object detection, it aims to identify human face in digital images. In this project, faces should be regards as saliency and preserved whole. Compute saliency map only using colour and luminance features that may fail to highlight the face region in some cases. In order to ensure that the face region is highlighted in saliency maps, apply face detection algorithm to indicate the position of face.

The Viola–Jones object detection framework [16] is a popular and effective object detection method. In face detection, Viola–Jones requires full view frontal upright faces.

This method includes 4 concepts: Haar features; Integral image; Adaboost; Cascading.

- Haar feature

Human faces have some common properties, for example, the eye region is darker than the upper-cheeks and the nose bridge region is brighter than the eyes.

(a) (b)

Figure 13. Haar features and human face features. (a) A Haar feature that looks similar to the eye region. (b) A Haar feature that looks similar to the bridge of nose

Haar feature are similar to convolution kernels used in edge detection (e.g. sobel kernel). These black and white rectangles is to analyse the differences between the dark and light regions of a face. Each feature is calculate by subtracting the sum of pixel under white region from the sum of pixels under black region.

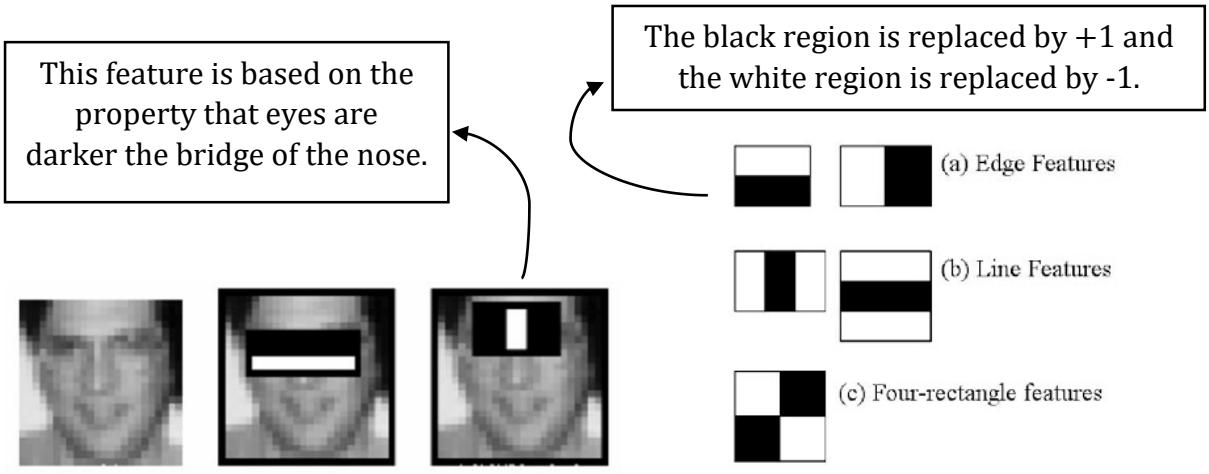


Figure 14. Haar features used in Viola Jones face detection. Image from OpenCV document.

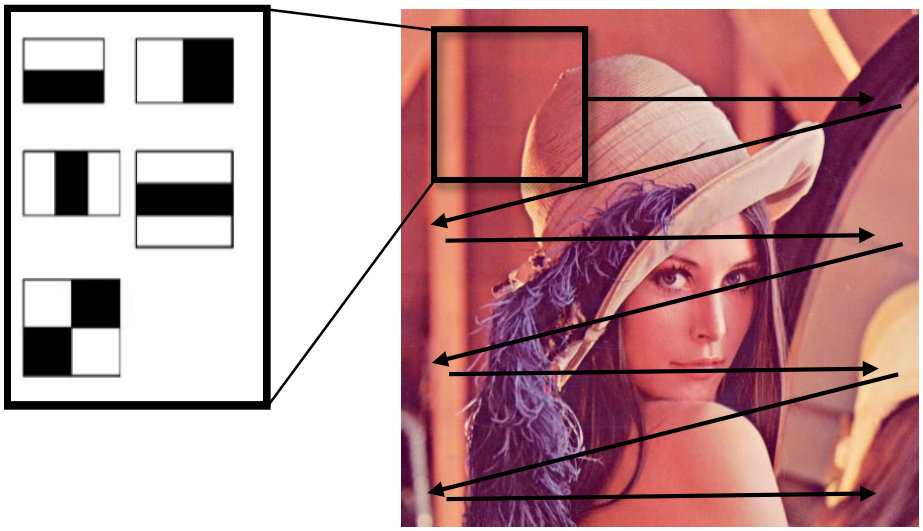


Figure 15. The mechanism of the detector scanning an image.

The sub-window is scanned across the image at various scales (start with a 24×24 window) to detect if there is a potential face within the window

- **Integral image**

The basic principle of integral image already explained in 3.1.4.

In the face detection, using integral image simplifies calculation of the sum of the pixel under white and black rectangles and makes things fast.

- **Adaboost**

Consider all possible sizes and location of each kernel, there can be around 160,000+ features values within a detector at a 24×24 window need to be calculated, most of them are irrelevant. Adaboost is a machine learning algorithm which helps on finding only a few set of useful features among all these features and decide which features to consider.

These selected features are called as weak classifiers because each one of them cannot work alone but together with others forms a strong classifier. Each selected feature is assigned a weighted value to form a final classifier. Final classifier is a weighted sum of these weak classifiers.

$$F(x) = w_1 f_1(x) + w_2 f_2(x) + w_3 f_3(x) + \cdots$$

final classifier(strong classifier) weak classifier

This step bring a great reduction of features to be calculated and time saving.

- **Cascading**

Group the features into different stages where each stage has a certain number of features. If a window fails the first stage, discard it, remaining features are not considered. If it passed, goes into the second stage of features and continue the process. If the window passes all stages, it is a face region.

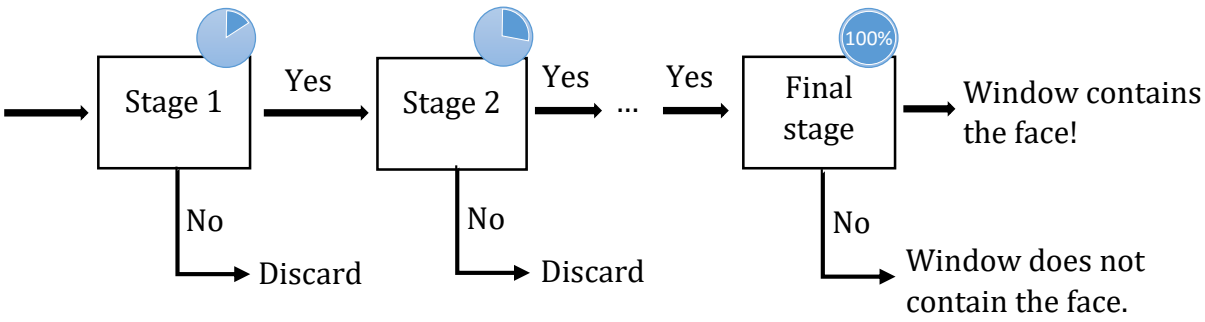


Figure 16. Cascade of classifiers

3.2 Rectangle Packing

This project provides three approaches based on rectangle packing algorithm to arrange the images. The first one is Shelf Next-Fit Decreasing Height (NFDH) algorithm which is fast and easy to implement. The second one is based on NFDH, it expands image boundaries to maximise page coverage. The third one is the Guillotine Algorithms, which utilise space usage.

3.2.1 Shelf Next-Fit Decreasing Height (NFDH) algorithm

First, create a rectangle canvas with fixed width and infinite height. Place images with decreasing height order, if the size of the image is unfit in current level, create a new level. Third, after finishing the arrangement of all images, cut off the excess height and width. The time complexity of NFDH algorithm is O(n log n).

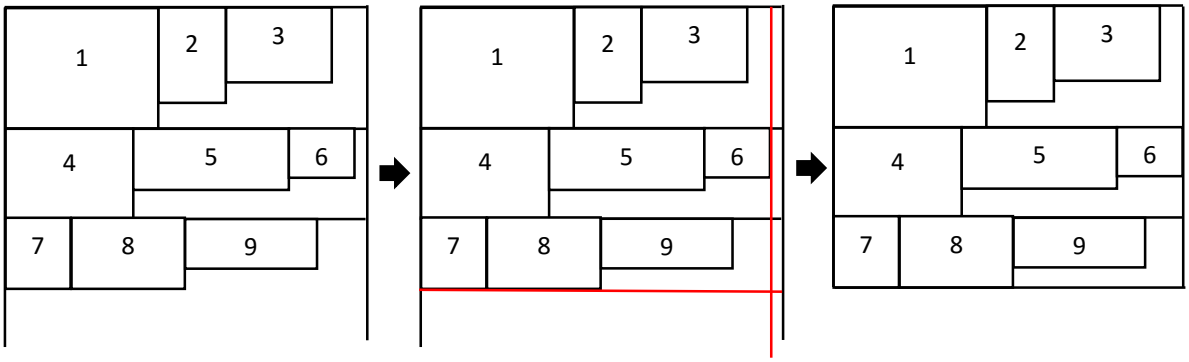


Figure 17. Shelf Next-Fit Decreasing Height (NFDH) algorithm.

3.2.2 Expand image boundaries based on NFDH

Increasingly extend the edge of the salient images until they cover the whole canvas or unable to expand (has reached boundaries of the original image). In most of cases, it creates a packing image without holes if there are enough space to expand.

Figure 18. Sample packing images of edge extension based on the NFDH algorithm.

3.2.3 Expand image boundaries and allow gaps.

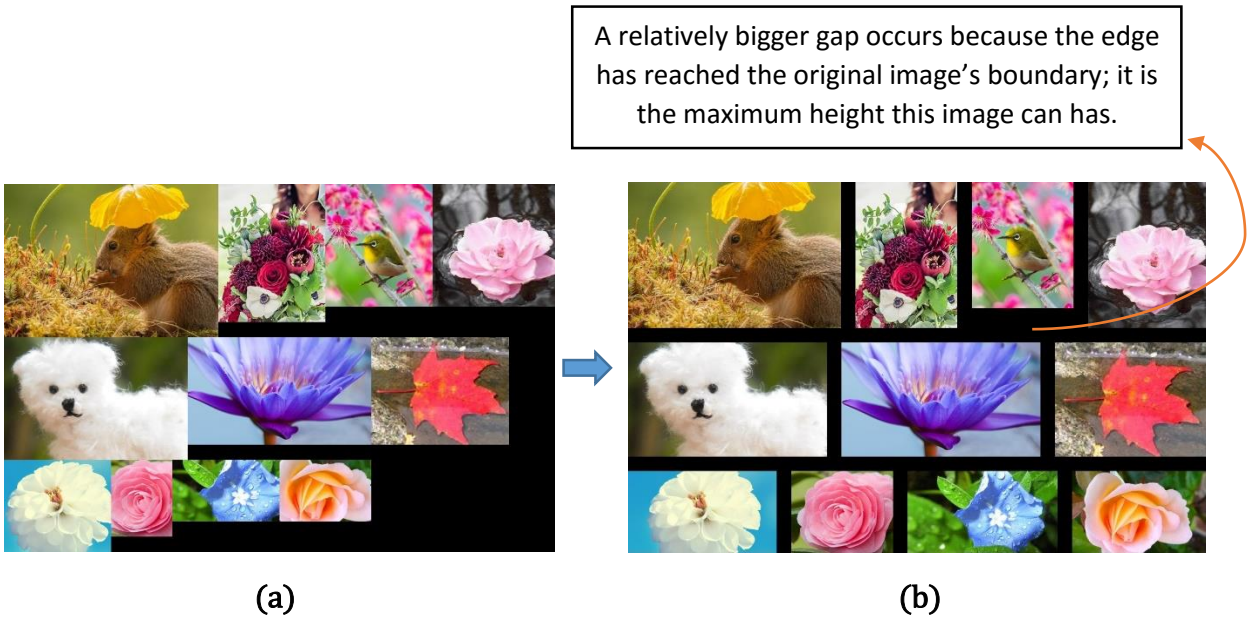


Figure 19. NFDH + Expand image boundaries with gaps. (a) Saliency images packed by the NFDH algorithm. (b)Expand boundaries and allow gaps between images.

To increase the space usage and has a better image arrangement, this project expand the boundaries of saliency images based on the Shelf Next-Fit Decreasing Height (NFDH) algorithm. It creates narrow gaps which may be used to create a joins between images by inpainting.

3.2.4 The Guillotine Algorithms

The Guillotine algorithm is a totally different approach to the shelf algorithm and superior to the shelf algorithm in space usage. It records each free areas of the bin and searches any free space.

Theoretically there are two possible split axes, vertical split and horizontal split. In this project we use only the way of horizontal split to store the remaining free space, because rather than using a fixed size container, a rectangle container with fixed width and extra-large height results in better utilization of free space. The priority is to fill the space horizontally.

- Each item (in decreasing height order) finds the smallest free rectangle in which enable to accommodate it.
- Horizontal subdivide the area into two free rectangle area.
- After finishing the placement of all images, cut off the excess height and width.

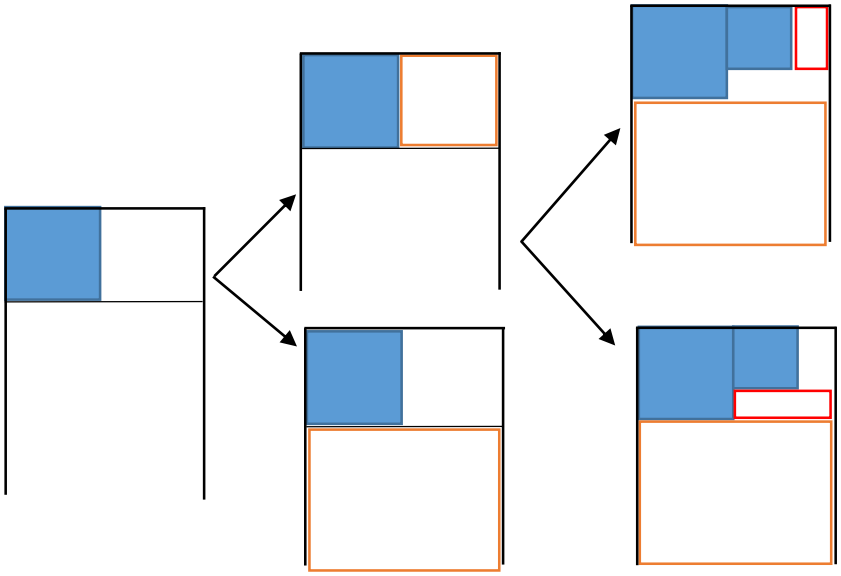


Figure 20. Guillotine Algorithms (horizontally split placement).

3.3 Inpainting

The Navier-Stokes based inpainting method and the inpainting method by Alexandru Telea which is based on Fast Marching Method both reconstruct the selected image area from the pixel near the area boundary and fill one missing pixel at each time. The difference between these two methods is that the Navier-Stokes based inpainting fills pixels along the edges from known regions to unknown regions while the Alexandru Telea inpainting method fills everything in the current boundary and goes into the regions. The Exemplar-based inpainting method reconstructs the missing image area from the patches searching in itself and fills several pixels in one patch each time.

The Navier-Stokes based inpainting method and the Alexandru Telea inpainting method both have been implementing in OpenCV. The Exemplar-based inpainting method is computationally intensive because it searches over the images to find the best patches.

3.3.1 Navier-Stokes based method [13].

This algorithm is based on fluid dynamics and views the image intensity as a streamline of the flow. It matches gradient vectors at the boundary of the region to be inpainted and continues isophotes, travelling along the edges from the known region to the region to be inpainted.

3.3.2 Method by Alexandru Telea, based on Fast Marching Method [14].

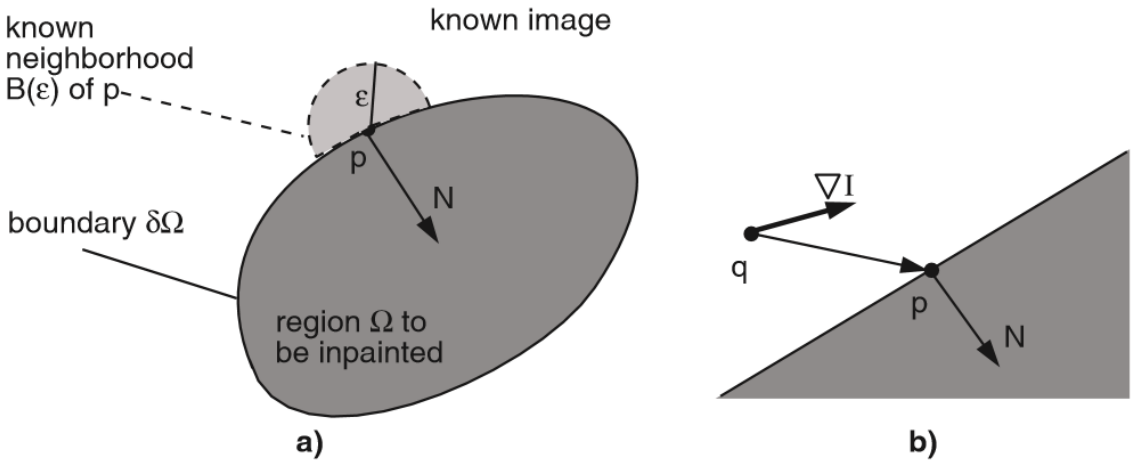


Figure 21. Alexandru Telea inpaiting algorithm. The image comes from [14].

This algorithm starts from the boundary of the inpainting region, fill every point in the front boundary and goes inside to gradually fill inner boundaries. For a point p on the boundary **B**, a small known neighbourhood **q** around the point is considered. The value of the pixel is determined by its known neighbourhood.

The priority of the point **p** on the boundary is (∇I(q) is the gradient of point **q**):

$$I_q(p) = I(q) + \nabla I(q)(p - q)$$

The weighting function is defined as:

$$\text{w(p,q)=directional(p,q)} \times \text{ distance(p,q)} \times \text{level (p,q)}$$

From the above equation, more weightage is given to those pixels near to the normal direction [**directional (p, q)**], close to the point [**distance (p, q)**], and those close to the boundary contours [**level (p, q)**].

$$I(p) = \frac{\sum_{q \in B(\varepsilon)} w(p,q)[I(q) + \nabla I(q)(p - q)]}{\sum_{q \in B(\varepsilon)} w(p,q)}$$

A weighted sum of all points **q** (the known pixels in the neighbourhood) is normalized to fill the pixel.

3.3.3 Exemplar-based inpainting [12].

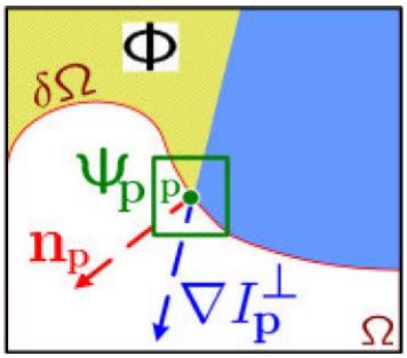


Figure 22. Schematic diagram of Exemplar-based inpainting algorithm. Patch Ψp is the patch to be inpainted, ∇I_p is the unit vector of isophote at point **p**, n_p is the unit normal vector of the front contour **δΩ** at point **p**. The image comes from [12].

This algorithm focus on both surrounding linear structure and exemplar-based texture synthesis. This algorithm can be concluded in 6 steps:

1. Fill every point in the front boundary and goes inside to gradually fill inner boundaries. (onion peel method)
2. For patch **Ψp** centred at the point **p** in current boundary, compute their priorities to determine the fill order.
3. Find the patch **Ψp** in the targeted area (damaged part of image) with maximum priority.
4. Find the patch **Ψq** in the source area (undamaged part of image) that minimizes the difference between these two patches.
5. Copy pixel data to from patch **Ψq** to patch **Ψp**
6. Update confidence values

The fill order of the patches in the same level boundary is determined by priority **P(p)**, which is defined as the product of the confidence term – **C(p)** and the data term – **D(p)**:

$$P(p) = C(P) \times D(p)$$

C(p) is the percentage of the known pixels in inpainting patch **Ψp.** Patches that are surrounded by more known pixels will tend to be filled first.

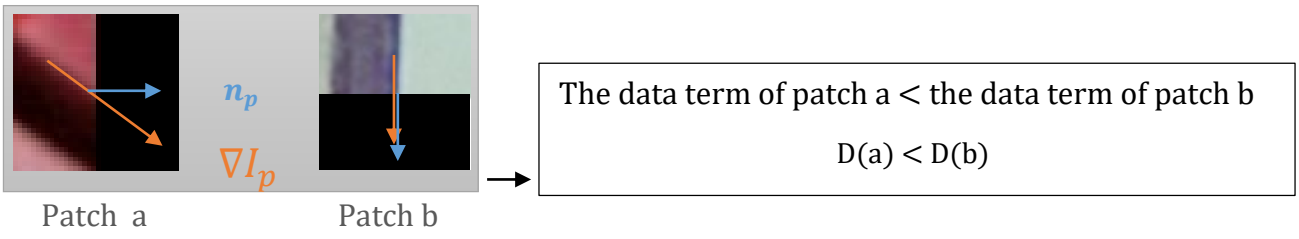


Figure 23. Data term is calculated by isophote vector and normal vector of the front contour.

The data term **D(p)** is the strength of isophote hitting the boundary, it boosts the priority of a patches that an isophote flows into.

$D(p) = \frac{|\nabla I_p \cdot n_p|}{\alpha}$, $\boldsymbol{\nabla I_p}$ is the unit vector of isophote at point **p**, $\boldsymbol{n_p}$ is the unit normal vector of the front contour at point **p**, **α** is a normalization factor (**α**=255 for grey image and **α**=3×255 for 3 channel image like RGB image).

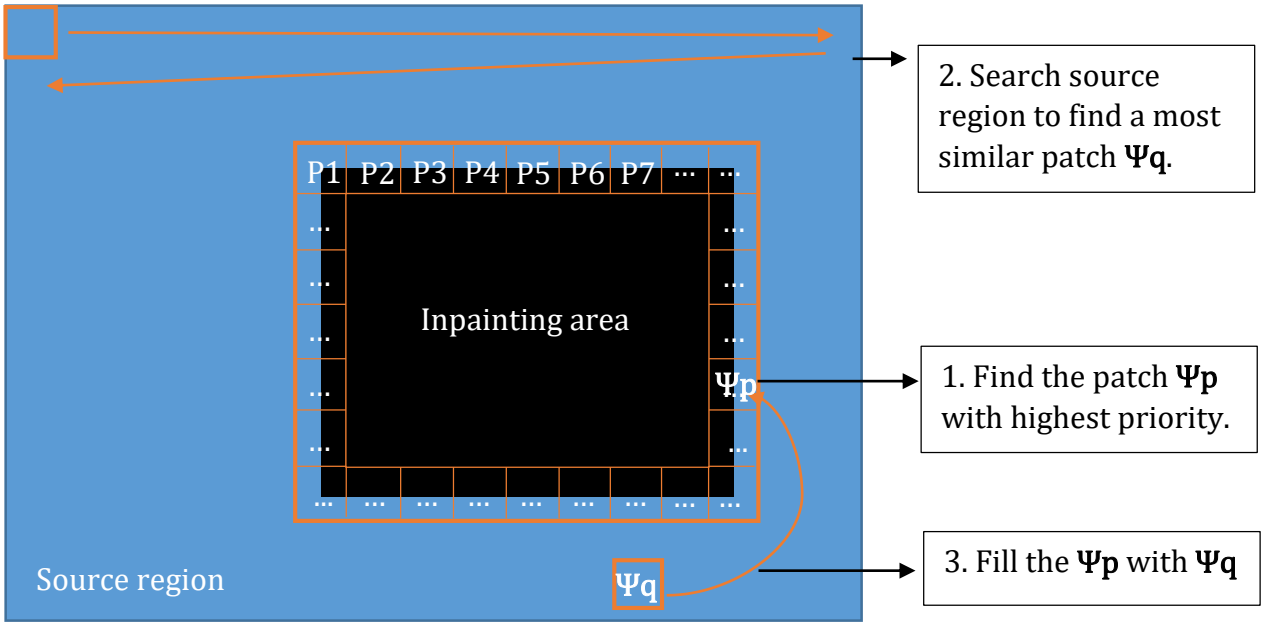


Figure 24. The processes to filling missing data. Replace the inpainting patch Ψp with the source patch Ψq.

Once all priorities on the fill front have been calculate, the patch **Ψp** with the highest priority has found, then we search the patches **Ψq** in source region (undamaged part of image) and find the most similar and fill the **Ψp** with **Ψq**.

The way to measure the similarity between two patches is the sum of squared differences of the known pixels in these two patches. The smaller sum of squared differences, higher similarity.

Implementation

4.1 Technical Background

OpenCV 2.4.11:

This project is written in Java and externally calls OpenCV as a support library. OpenCV is an open source computer vision library and has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. It implements lots of algorithms for computer vision. It helps to develop this project, for example, it contains the Gaussian filter, K-means clustering, Viola–Jones face detection, Navier - Stokes inpainting method [13], Telea inpainting method [14] used in this project.

OpenCV 3.0.0 was used in this project in the period of first three weeks because it is new and brings new functionalities, but it is currently beta and not the official release, it was unstable and the functionality in some modules has been split into other modules. In addition, the documentation and tutorial of OpenCV 3.0.0 online is limit compared with OpenCV 2.4.11. Consider that I have encountered few bugs in 3.0.0 and no additional needs is required in this project, so I replace OpenCV 3.0.0 with OpenCV 2.4.11, which is the last official stable release.

4.2 Images sources

Before the coding and testing could commence it is necessary to prepare digital image data to be used. All images processed in this project are found and downloaded by using an advanced search filter called "usage rights" on Google image search, which can filter the results to find images that you have permission to use.

4.3 System environment

The runtime performance cost of algorithms externally depends on hardware. This report will carry out comparison of runtime performance cost of different algorithms, a declaration of the system environment is shown below:

Computer: Microsoft Surface Pro 3

Processor: Inter(R) Core(TM) i5-4300U CPU @ 1.90GHz 2.50 GHz

RAM: 4.00GB

Operating system type: Windows 10. 64-bit, X64-based processor

4.4 System framework

Before any coding work start, it is essential to understand fully the algorithms which are intended to be used. The following is to create the basic framework of the system. The coding work of the system is mainly split into four modules, they are:

- Graphical user interface which includes loading images, displaying images, processing images and saving images.
- Saliency extractor
- Rectangle packer
- Digital inpainting

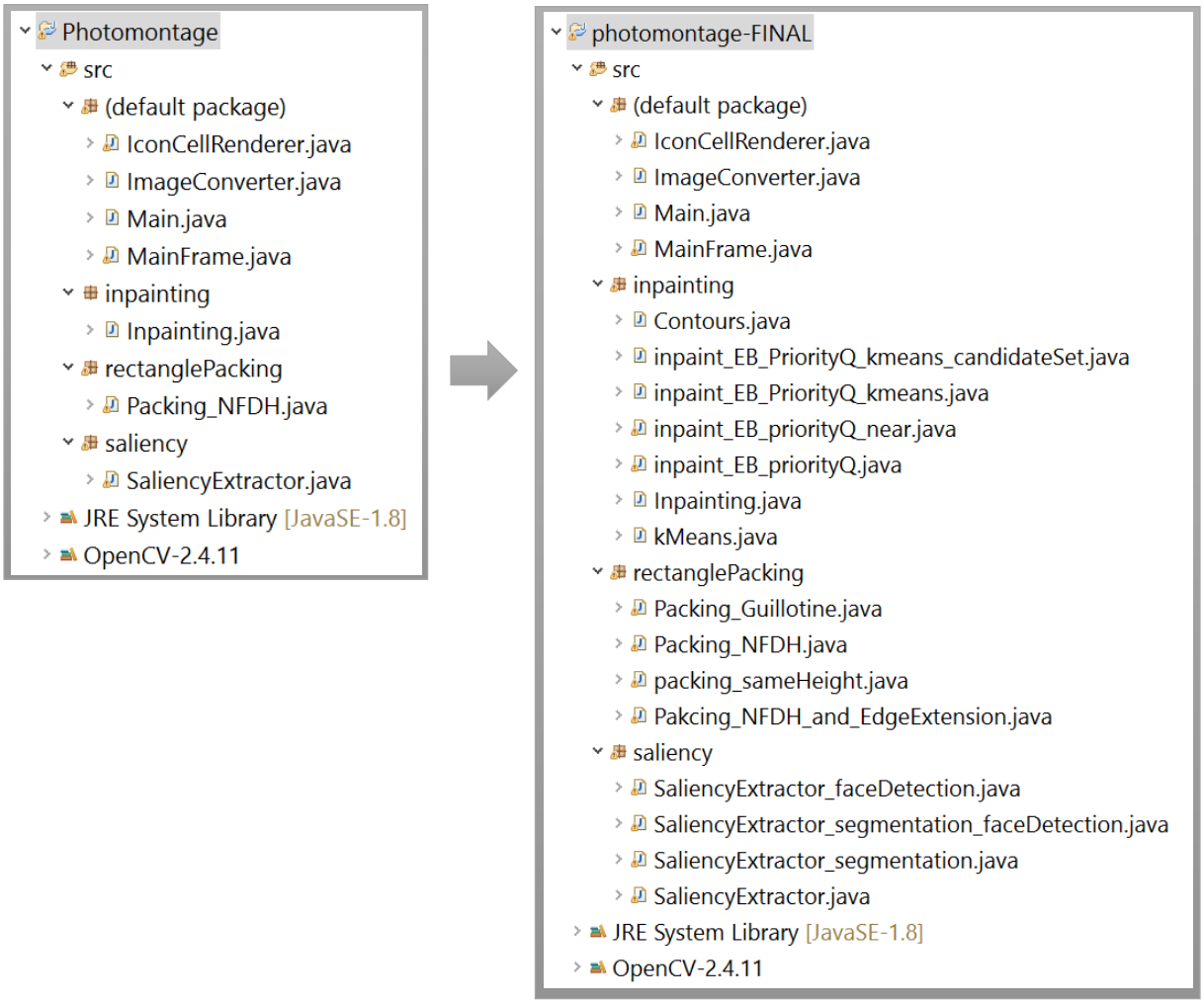


Figure 24. System framework. Initial system -> final system

Each sub-system was created by a relatively simple algorithm to get the whole system work as quick as possible. After finishing the basic system, optimise each sub-system and add more methods. A simple graphical user interface is used to run each sub-system. Once the user opens and loads a directory of the image set, different methods in each module can switch to run; the user does not need to reload the image or restart the

application. The user can click the image icons in the left lists and see their detail in the right window.

A screenshot of the user interface is shown below.

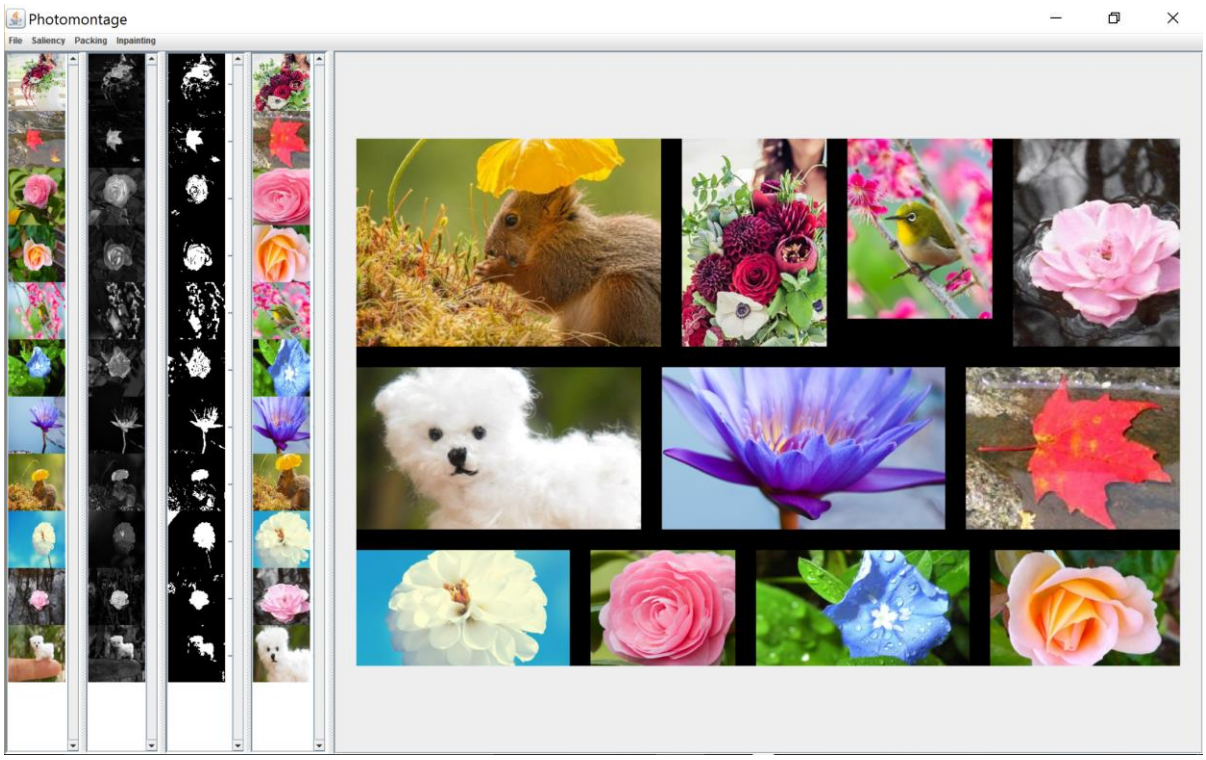


Figure 26. Graphical user interface

4.5 Mat image and buffered image convertor

The class Mat in OpenCV represents an n-dimensional dense numerical single-channel or multi-channel array, and it can be used to store real or complex-valued vectors and matrices, grayscale or colour images. Images are loaded and processed as mat images, in order to display the result images on the UI, a convertor for converting buffered image to mat image and mat image to buffered image is provided.

4.6 Saliency extractor

There are four choices of saliency extractor can be used, the user can select which one to use depends on the type of images. For example, if the set of images are all the natural scenery and without people, there is not necessary to use saliency extractor with face detection.

- **Frequency-tuned salient region detection without segmentation**

1. Convert the image from RGB colour space to Lab colour space.
2. Apply Gaussian filter with kernel size 5×5.
3. Calculate mean image colour vector.
4. Find the Euclidean distance between the Lab pixel vectors of Gaussian image and mean colour vectors. Then normalise it to generate saliency image.
5. Generate binary image by assigning ones to the pixels whose saliency value are greater than 2 × mean saliency (adaptive threshold) and zeros to the rest of pixels.
6. Computing integral image based on the binary image.
7. Search with a 32k×32j window in integral image(k and j are increasing integers and k>0, j>0) and find a minimum window with Sum(window) > 85%× Sum(all)

- **Frequency-tuned salient region detection with segmentation**

1. Convert the image from RGB colour space to Lab colour space.
2. Apply Gaussian filter with kernel size 5×5.
3. Calculate mean image colour vector.
4. Find the Euclidean distance between the Lab pixel vectors of Gaussian image and mean colour vectors. Then normalise it to generate saliency image.
5. Segment image with K-means segmentation algorithm and retains only those segments whose average saliency is greater than 2 × mean saliency value. The function of K-means clustering has already implemented by OpenCV.
6. Generate binary image by assigning ones to the pixels in selected segments and zeros to the rest of pixels.
7. Computing integral image based on the binary image.
8. Find a minimum window.

- **Frequency-tuned salient region detection with face detection**

1. Convert the image from RGB colour space to Lab colour space.
2. Apply Gaussian filter with kernel size 5×5.
3. Calculate mean image colour vector.
4. Find the Euclidean distance between the Lab pixel vectors of Gaussian image and mean colour vectors. Then normalise it to generate saliency image.
5. Load a Cascade classifier from an OpenCV file. Apply Cascade classifier class for face detection. Face regions are represented by a list of rectangles.
6. Generate binary image by assigning 1s to the pixels whose saliency value are greater than 2 × mean saliency (adaptive threshold) and 0s to the rest of pixels. Assign ones to the pixels whose are in the face region to highlight the face region.
7. Computing integral image based on the binary image.
8. Find a minimum window.

- **Frequency-tuned salient region detection with segmentation and face detection**

Combine both K-means clustering and Viola–Jones face detection

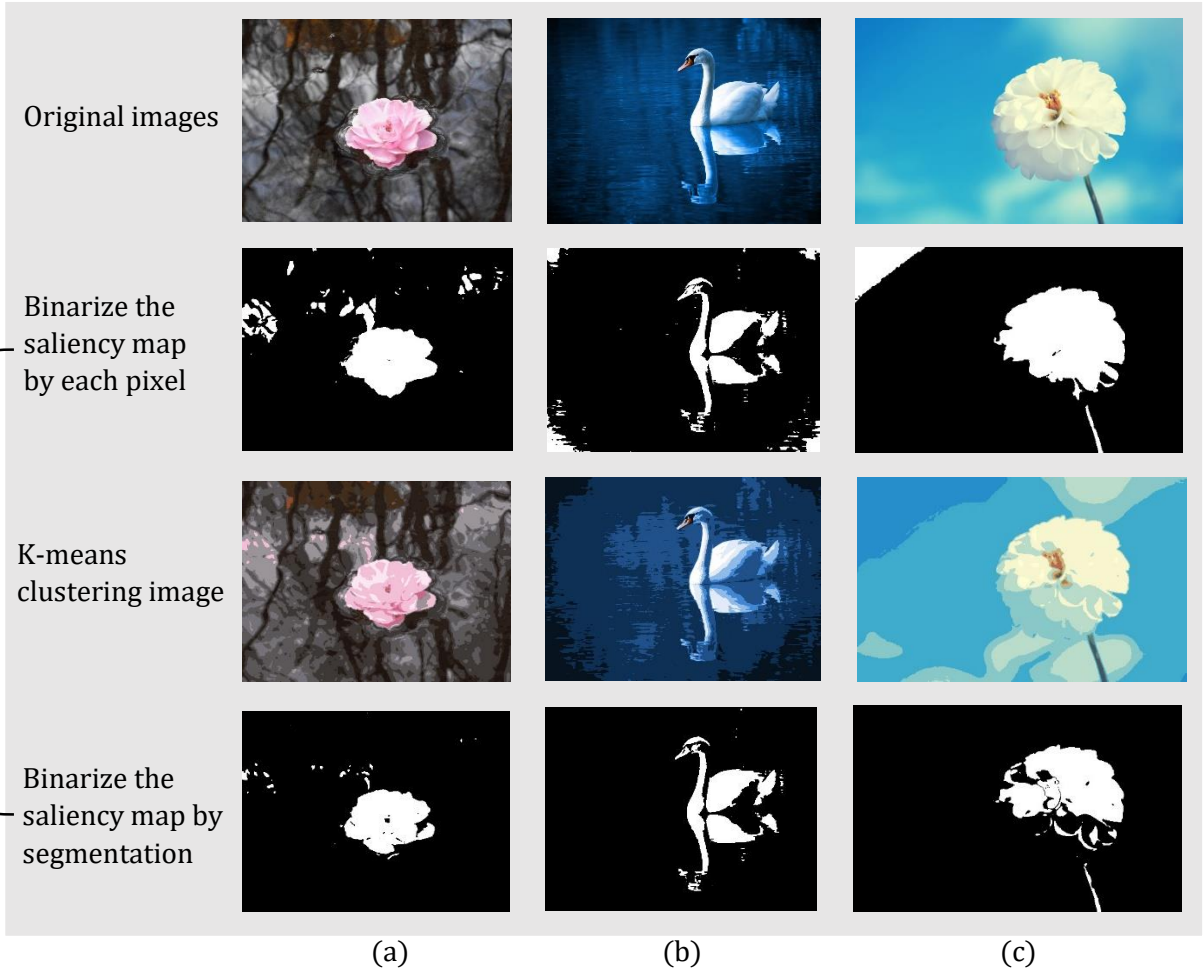
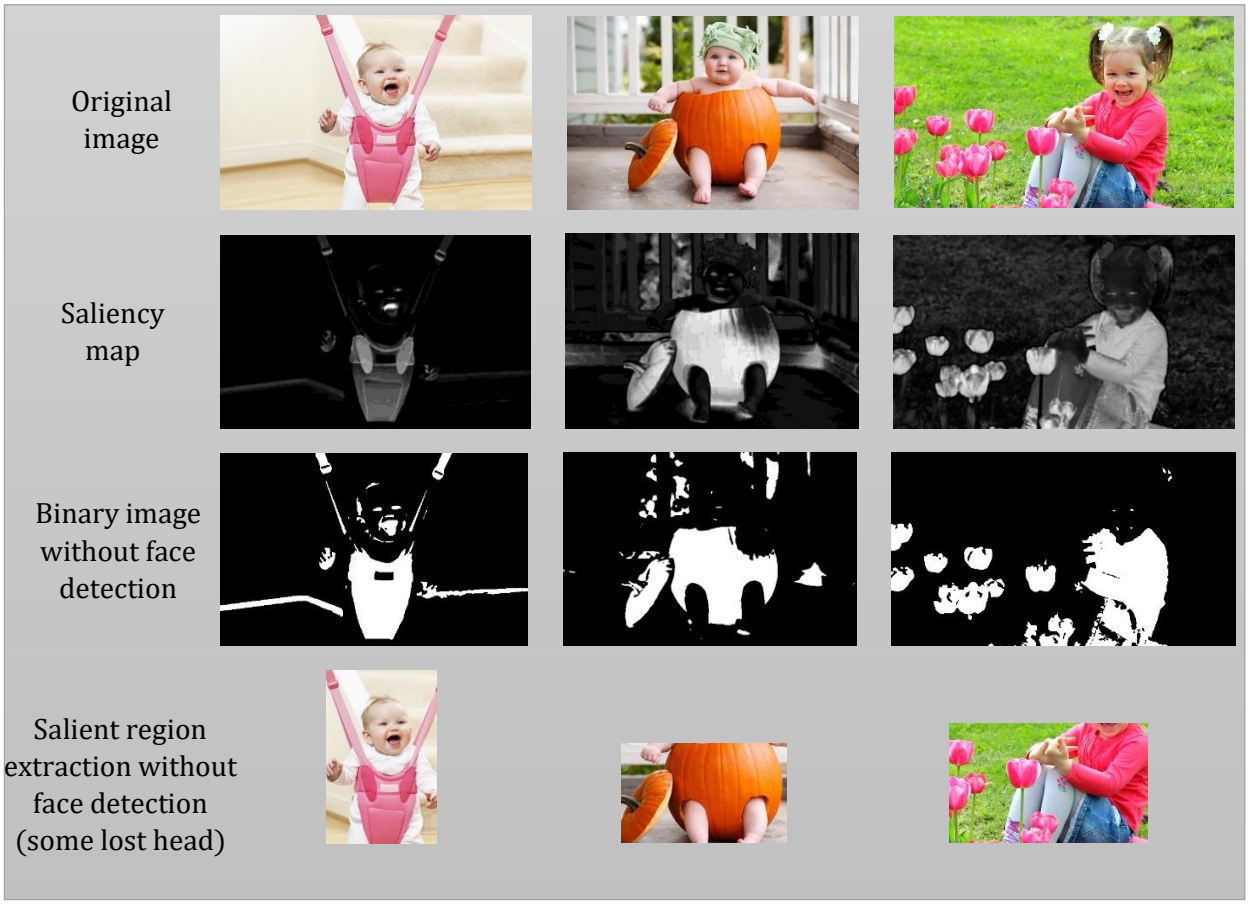
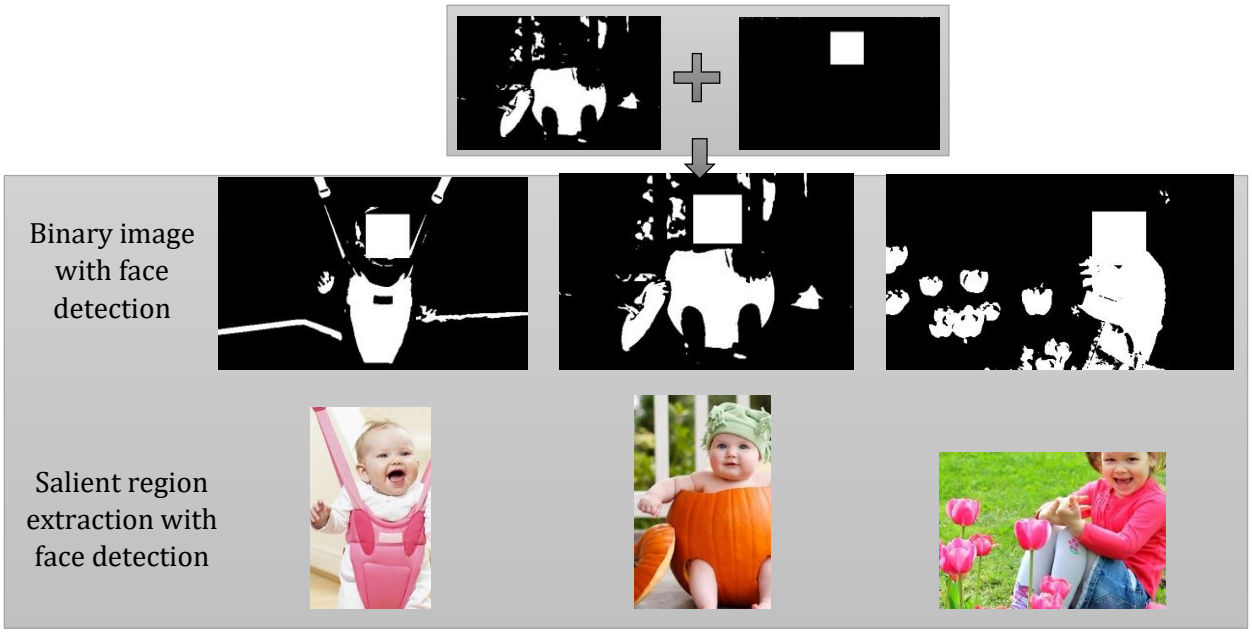


Figure 27. Results of the saliency extraction without segmentation and with segmentation.

Generating binary image by segmentation can eliminate some noises in the background. For example, it reduces the noises in (a) and removes the noises around the image corner of (b) and in the top left corner of (c).





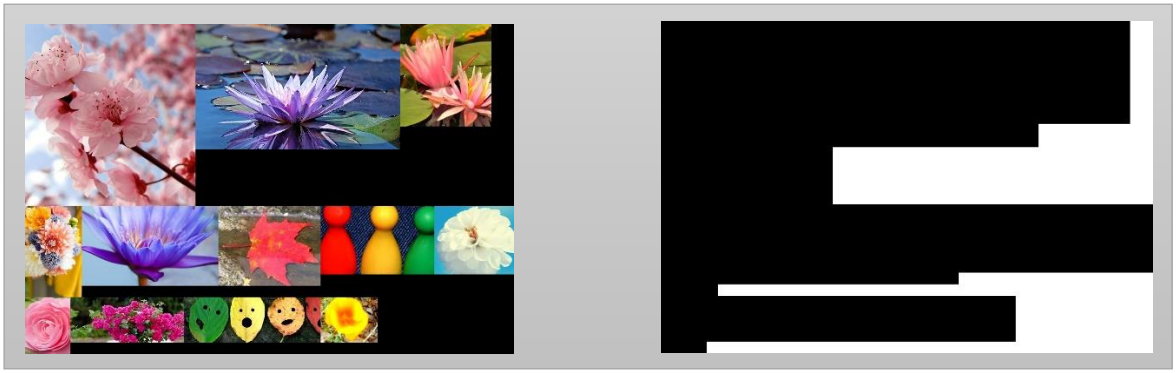
(a) (b) (c)

Figure 28. Results of saliency extraction without face detection and with face detection

The salient region extraction without face detection fails to enclose the face region in some images because low saliency value in face region in the saliency map and face region did not get the highlight in the binary image.

4.7 Rectangle packer

A packing image and an inpainting mask are generated in this section. Inpainting mask is an 8-bit 1-channel image which is used to indicate the area that needs to be inpainted. Non-zero pixels indicate the area that needs to be inpainted.



Packing image Mask

Figure 29. Packing image and mask.

- **Shelf Next-Fit Decreasing Height (NFDH) algorithm**

The first rectangle packer is using the shelf algorithm (also called level algorithm) which is unarguably the fastest and simplest method one can use to produce packings.

1. Estimate a fixed width by a square root of the sum of the width of all images, so that the result can lead as close to square.
2. Rank images with decreasing height order
3. Place image in the current level, if the width of the image is unfit in the current level, create a new level.
4. After finishing the placement of all images, cut off the excess height and width.

- **Expand image boundaries**

1. Expand image boundaries based on NFDH algorithm

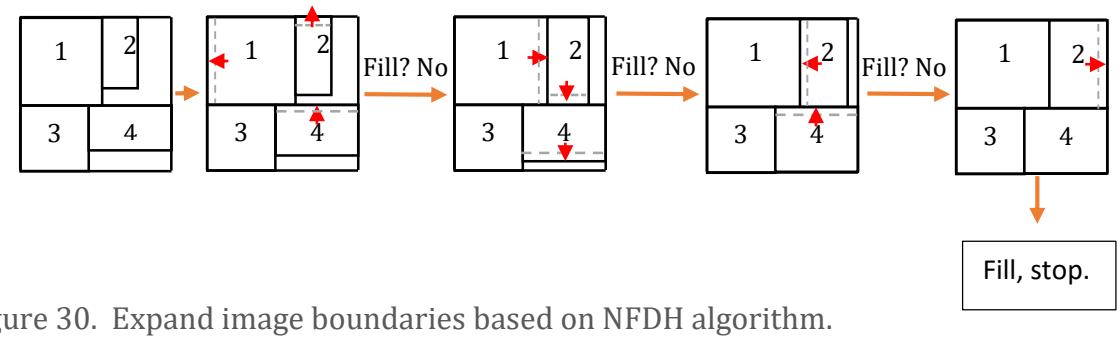


Figure 30. Expand image boundaries based on NFDH algorithm.

- **The Guillotine Algorithm**

  Use a binary tree to store free rectangles, create a class Node to store the position and size of rectangles, use a Boolean data type to identify the rectangle if is free to use.

1. Set the rectangle canvas by estimating a fixed width by a square root of the sum of the width of all images, so that the result can come as close to square. Set the height of canvas equals to 10×width.
2. Initialize the root of the binary tree = canvas(the biggest free rectangle)
3. Traverse the tree to find a free node with a minimal size which enables to accommodate the image.
4. After placing the image, mark the node as used and split the node to two nodes
5. Repeat step 3-4

```
private class Node {

    private Node down;
    private Node right;
    private int row;
    private int col;
    private int blockHeight;
    private int blockWidth;
    private boolean used;
```

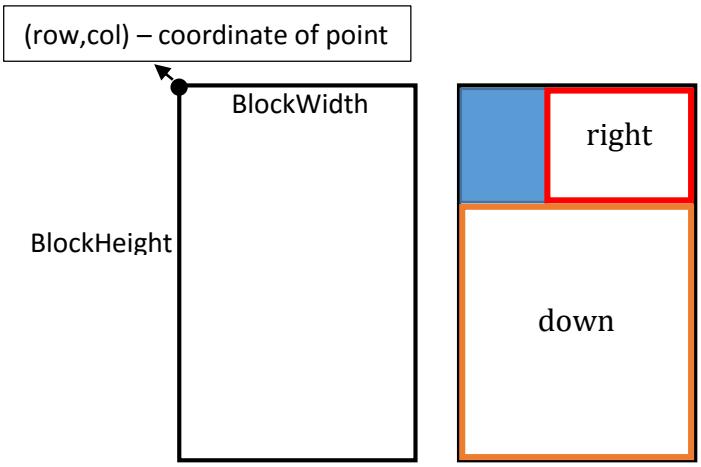


Figure 31. Binary tree used in the Guillotine algorithm

4.8 Digital inpainting

4.8.1 NS inpainting and Telea inpainting

The Navier-Stokes based inpainting method and Alexandru Telea inpainting method which is based on Fast Marching Method are both provided by OpenCV.

4.8.2 Exemplar based inpainting

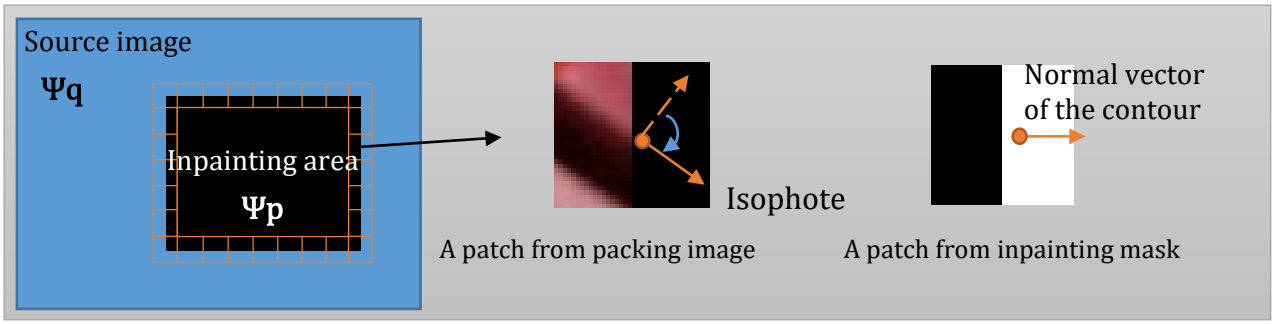


Figure 32. The isophote vector and normal vector of the front contour at point p

1. Initialize confidence term **C(p)**. **C(p)** equals to the percentage of the known pixels in inpainting patch. Use a mat to save them.

2. Calculate $\boldsymbol{n_p}$ (the unit normal vector of the front contour at point **p**) by using Sobel kernel in inpainting mask. Use two mats to save them, one for the horizontal direction and one for the vertical direction.

3. Compute gradient in packing image by using Sobel kernel and them rotate gradient 90 degrees to get $\boldsymbol{\nabla I_p}$ (the unit vector of isophote at point **p**). Use two mats to save them, one for the horizontal direction and one for the vertical direction.

4. Found the front contour, use a List<MatOfPoint> to save their position.

5. Calculate priority value **P(p)** of points along the fill front which determines their order to be inpainted. $\mathbf{P(p)}=\frac{|\boldsymbol{\nabla I_p} \times \boldsymbol{n_p}|}{255\times\mathbf{3}} \times \boldsymbol{C(p)}$

6. Create a priority queue to store the position and priority value of contour points. The Point with the highest priority at the top of the queue.

7. After priority queue construction, use poll () method to retrieve and remove the head of this queue.

8. If the point with the highest priority value (the head of this queue) has already filled, then remove the head of this queue and do nothing. If not, search the most similar patches in the source image (known data) and fill the inpainting patch **Ψp** with matched patch **Ψq.**

9. After inpainting a patch, update confidence term - C(p) and data term D(p).

10. Repeat step 8-9 until this queue is empty.

11. Repeat step 4-10 until not front contour is found.

In the beginning, a simple array is used to store the priority and position of points. In order to find the point with the highest priority, it needs to traverse the array every time. For a size 928×1376 image, it cost 303 seconds to inpaint 30 patches (patch size is 11×11 pixels and jump a step of 11 pixels in source area to search a matched patch), which means about 10 seconds spent for one patch. And there are thousands of patches need to be inpainted so that it is very important to increase the inpainting speed.

In order to improve efficiency, there are six methods below (patch size are all 11×11 pixels):

- **Priority queue**

A priority queue is an abstract data type which an element with high priority is served before an element with low priority.

A priority queue is used to store the priority and position of points rather than using a simple array to store and traverse the array each time to find the maximum priority. It speeds up the time for finding highest priority point without affecting inpainting quality.

- **Early jump-out**

An early jump-out technique for speeding up the time of testing similarity of image patches without affecting inpainting quality. The algorithm is shown below:

S_min is a previously selected patch whose has the minimum sum of Euclidean distance so far.

For pixels in current patch **Ψq** {

The sum of Euclidean distance between two patches: **S_current** = **S_current** + **Distance(next pixel)**

If **S_current** is greater than **S_min**, then break the loop. This patch has a greater difference than previously selected patch. Give up this patch and go to next patch.

}

For an image with size 928 pixels ×1376 pixels:

Simple array: **303 seconds** to inpaint 30 patches

Priority queue: **16 seconds** to inpaint 30 patches

Priority queue + Early jump-out: **7 seconds** to inpaint 30 patches

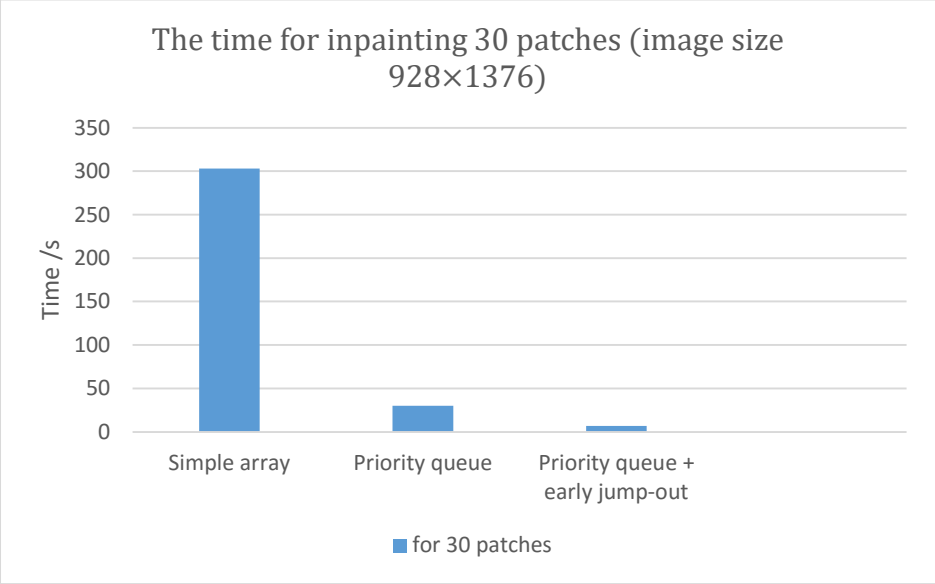


Figure 32. The time for inpainting 30 patches in a 928×1376 image.

The time complexity for using simple array to find the highest priority point is O(n).

The time complexity for using priority queue to find the highest priority point is O(1).

Using priority queue greatly reduces the time in finding the highest priority and using early jump-out reduces the time for testing similarity of image patches. Both of them does not affect the inpainting quality.

7 seconds to inpaint 30 patches which means 4 minutes for relatively small holes which has 1000 patches and 27 minutes for big holes which has about 7000 patches. Obviously, it is not quick enough.

A 928×1376 image with big holes (about 7000 patches)

The following 4 methods for speeding up time may affect the inpainting quality.

- **Search patches with a big step**

Searching every possible patch in source image can testing all possible patches and find the best one. It produces the better result but low searching speed.

For an image with size 928 pixels ×1376 pixels:

Searching every possible patch (step =1 pixel) to fill 20 patches: 448 seconds

Searching patches with a big step (step =11 pixels) to fill 20 patches: 4 seconds

Small step Big step

Figure 34. Search patches in image with steps. Small step and big step

- **Search neighbours**

Only search the patches near the hole.

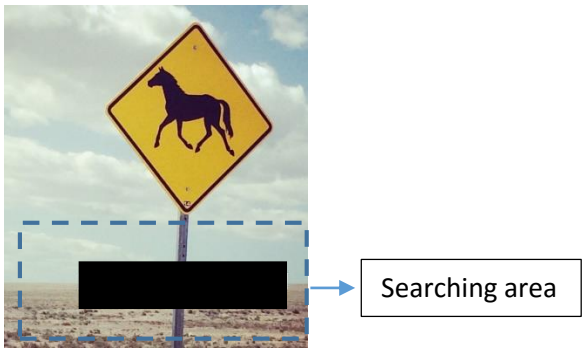


Figure 35. Searching neighbouring area of the hole

The searching area is depends on the patch size. It is defined as 5 × patch size. For example, if patch size = 11×11, the searching radius around the inpainting patch is 5×11=55 pixels.

- **K-means clustering**

The K-means clustering in this section use a fixed K-seeds rather than an adaptive K-seeds by adding up histogram peaks in three channels (channel-L, channel-A and channel-B) to obtain the total number of peaks and indicates the value of K. Because the packing image summarise different images together which includes a variety of tone.

The inpainting patch firstly searches the similar cluster and then go into the cluster and search the patches around the points in this cluster. For example, the inpainting patch is a part of the earth which is brown; then we are not looking for patches in the sky which are blue.

The searching step is adaptive by the size of clusters.

The adaptive step =20 * cluster size/maximum cluster size;

Bigger cluster, bigger step.

- **Candidate set**

After filling the front part of patches, add their matched patches into candidate set and fill the following patches by searching the similar patch in the candidate set. This method can greatly reduce the time cost in searching patches from whole images.

Results & Evaluation

5.1 Saliency extraction

The frequency-tuned salient region detection algorithm is fast and provides full resolution saliency maps which may suited to image segmentation. The segmentation is based on the intensity and colour properties of the pixels which aims to take out some noises.

- **Compare saliency extraction without segmentation and with segmentation**

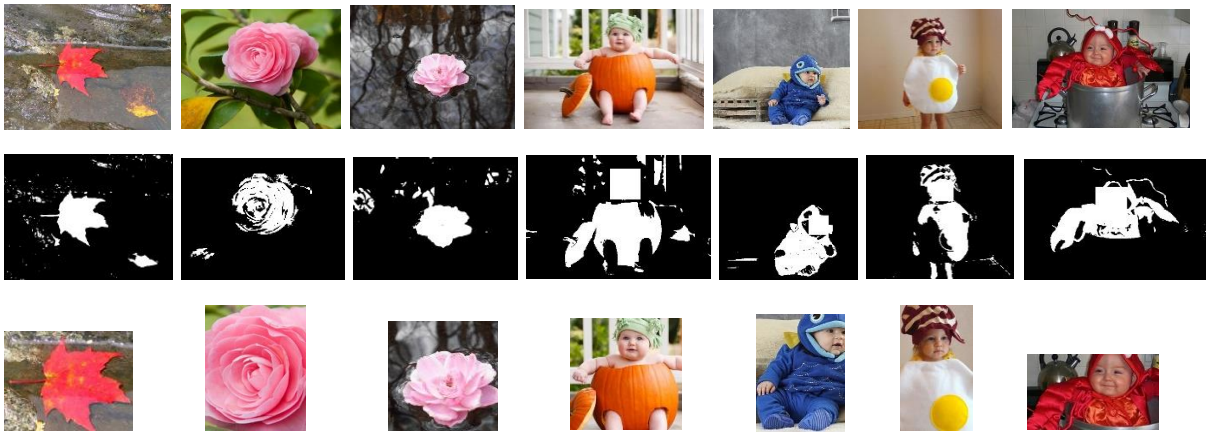


Figure 36. Results of Saliency extraction without segmentation. The first column are the original images, the second column are the binary images, and the third column are saliency images extracted.

Figure 37. Results of Saliency extraction with segmentation. The first column are the clustering images, the second column are the binary images, and the third column are saliency images extracted.

Saliency extraction with segmentation may reduce noises in the background but also introduce noises in some cases. Because it segments images based on the intensity and colour of pixels and the binary image is generated considering the mean saliency of the segments, the noise may be reduced by classifying the noise pixel to a segment which has a low mean saliency value. Also, the mean saliency of the segment may be raised by the noise pixel whose has a relatively higher saliency value, once the mean saliency of the segment exceeds a threshold, it will result in a bigger noise.

- **Viola–Jones face detection**

This algorithm is fast and efficient. To detect a face, Viola–Jones requires full view frontal upright faces.It is not perfect, given 80 images which contains human faces and apply Viola–Jones face detection to detect faces, 61 of them are correctly detected. There are two cases of failures:

- images that do include faces that are mistakenly classified by the face detector as not including faces
- images that do not include faces that are mistakenly classified by the face detector as including faces

The first case of failure:

Figure 38. Images that the face detector detects as not including faces.

In these images, face detection fails to detect a face from the side, a face with strong facial expressions, and a face covered by some items. The failures due to the distortion of certain face properties like irregular shape and size of eyes, mouth, and bridge of nose, etc. The distortion of face properties results unmatchable facial features so that the score representing the confidence of the face cannot reach the threshold at the end of the classifier cascade of the face detector.

The second case of failure:

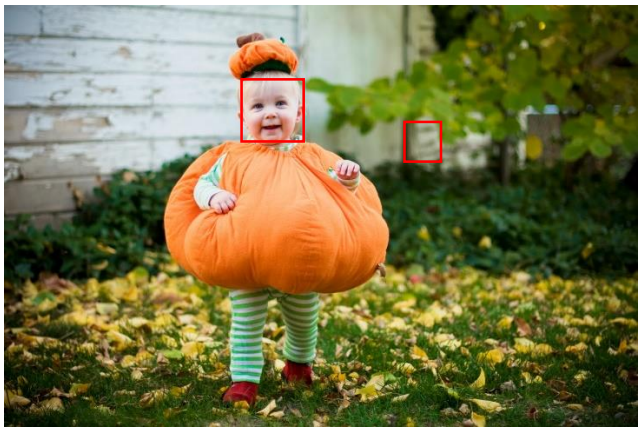


Figure 39. Images that the face detector detects the mistaken face region

In this case, a non-face region is mistakenly classified as a face. This non-face region contains intensity features which similar to a face. This non-face region contains darker regions and brighter regions which may mistakenly match to the human eye and mouth. The detector returns a score of high confidence level about these matchings. So that the score may reach the threshold at the end of the classifier cascade of the face detector and classify the non-face region as a face.

In this two cases of the incorrect face dictation, the threshold at the end of the classifier cascade of the face detector plays a key role in the accuracy. By decreasing the threshold, the failure of the first case can be reduced and also increases the failure of the mistaken face. Reversely, increasing the threshold increases the failure of the first case but decrease the failure of the mistaken face.

In the Viola–Jones face detection, the detector is most effective only on full view frontal upright faces without strong expressions. And also, it is sensitive to lighting conditions because it uses features to match face properties based on pixel intensities.

5.2 Rectangle packing

Shelf algorithm

Guillotine algorithm

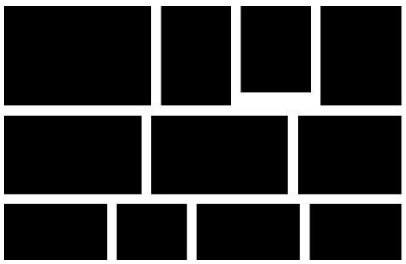
Figure 40. Shelf algorithm and Guillotine algorithm

Compared with the Shelf algorithm, Guillotine algorithm keeps exact track the free spaces of the bin and allows more remaining space to use. Guillotine algorithm has a higher availability of space and produces smaller holes. Guillotine algorithm utilises the free space for smaller rectangles. However, if the sizes of rectangles are the same or similar, the results of shelf algorithm and guillotine algorithm would be the same.

Based on the maximum size of each bin in Shelf algorithm, I expand the boundaries of these saliency images to create a better image layout.

(a)Shelf algorithm

(b) Expand boundaries with gaps



(c) Inpainting mask

Figure 41. Different image arrangements.

5.3 Digital inpainting

- **Navier-Stokes based inpainting method and Alexandru Telea inpainting method**

Packing image (Shelf algorithm)

Packing image (expand boundaries)

Navier-Stokes based inpainting method with an inpaintRadius=5 pixels

Alexandru Telea inpainting method with an inpaintRadius=5 pixels

Figure 42. Results of Navier-Stokes based inpainting and Alexandru Telea inpainting

Navier-Stokes based inpainting and Alexandru Telea inpainting both inpaint the missing pixel with its neighbours. They are fast and displays strong visual artefacts when inpainting a big hole. The curve joins between images may due to that the missing pixel is filled with the image which are more close to it.

- **Exemplar-based inpainting method applied on packing image generated by Shelf algorithm**

Image 1:

For an image with size 256×416 pixels (115 patches need to be inpaintde, patch size = 15×15 pixels; Step=11 pixels or use an adaptive step in the k-means clustering)

Search whole image to inpaint 115 patches: 3s

Search neighbours to inpaint 115 patches: 1s

Search whole image to inpaint 115 patches with the k-means clustering: 1s

Search whole image to inpaint 115 patches with a candidate set: 1s

Search whole image to inpaint 115 patches with the k-means clustering and a candidate set: 1s

Image 2:

For an image with size 512×1056 pixels (1123 patches need to be inpainted, patch size = 15×15 pixels; Step=11 pixels or use an adaptive step in the k-means clustering)

Search whole image to inpaint 1123 patches: 140s

Search neighbours to inpaint 1123 patches: 8s

Search whole image to inpaint 1123 patches with k-means clustering: 16s

Search whole image to inpaint 1123 patches with a candidate set: 16s

Search whole image to inpaint 1123 patches with k-means clustering and a candidate set: 8s

Image 3:

For an image with size 768*1152pixels (1150 patches need to be inpainted, patch size = 15×15 pixels; Step=11 pixels or use an adaptive step in the k-means clustering)

Search neighbours to inpaint 1150 patches with the k-means clustering: 15s

Search whole image to inpaint 1150 patches with the k-means clustering and a candidate set: 75s

Figure 43. Results of Exemplar-based inpainting in a big hole surrounded by different images

The strong visual artefacts can be clearly seen when inpainting a large hole surrounded by different images. The visual artefacts increase with the distance increases from boundary. The boundaries data of the large hole surrounded by different images are very spread and sparsely distributed. The colour and intensity features of surrounding data of holes differ greatly.

- **Exemplar-based inpainting method applied on packing image generated by Shelf algorithm and with expand boundaries**

For an image with size 1026×908 pixels (1659 patches need to be inpainted, patch size = 15×15 pixels; Step=11 pixels or use an adaptive step in the k-means clustering))

Search neighbours to inpaint 1651patches: 25s

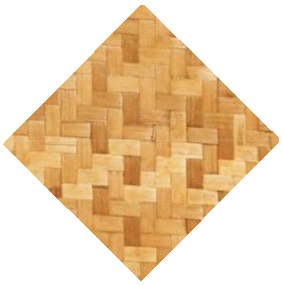
Search whole image to inpaint 1651 patches with the k-means clustering and a candidate set: 155s

Figure 44. Results of Exemplar-based inpainting in gaps betwwen different images

When doing the inpainting with a big hole surrounded by different images, it always has strong artefacts and creates an awful image collage. Currently there is few of even no researches conducted in inpainting with a large hole surrounded by different images. The results of the Exemplar-based inpainting applied in big hole are disappointing because the strong artefacts.

Surprisingly, the Exemplar-based inpainting applied in narrow gaps between different images produces an interesting result although still artefacts exist. I call it basket weave-effect because it is similar to a woven basket texture. To some extent, it can be used in visual arts can be improved in the future to create a fantasy effect.

Joins of different images

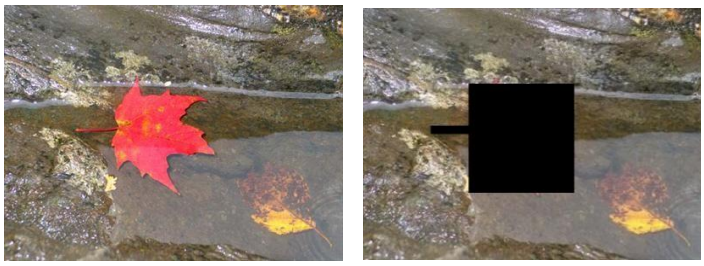


A woven basket texture

Figure 45. Basket weave-effect

Although the Exemplar-based inpainting can introduces strong artefacts when inpainting a big hole surrounded by different images, it can produce a good result when inpainting a hole at the centre of an image and it is effective when applying at an image which has strong linear structures.

- **Inpainting with a hole at the centre of an image (patch size=15*15)**



Original Image Remove object

(a) (b) (c)

(d) (e) (f)

(a) Apply Exemplar based inpainting (search whole image)
(b) Apply Exemplar based inpainting (search neighbours)
(c) Apply Exemplar based inpainting (search whole image with the k-means clustering)
(d) Apply Exemplar based inpainting (search whole image with the k-means clustering and a candidate set)
(e) Apply Navier-Stokes base inpainting
(f) Apply Alexandru Telea inpainting

Figure 46. Inpainting a hole at the centre of an image

The Navier-Stokes based inpainting method and Alexandru Telea inpainting method are not working well with big holes. Because they fill missing pixels by sampling and copying neighbouring pixels, the artefacts increase with the distance of the missing pixel and boundaries increases.

The Exemplar-based inpainting can get a better result when used to inpaint a hole at the centre of an image because there is more reliable pixel data along the boundaries of the hole and a single image has coherent textures. However, there are still artefacts due to incorrect selection of patches. Theoretically, search whole image with a small searching step will produce the best result because it search patches in all probability but it is computationally intensive and causes run-time inefficiency.

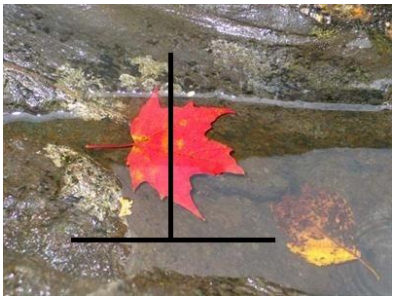
(a) Original image (b) Damaged image (c) inpainting result

Figure 46. Inpainting a hole at the centre of an image with strong linear structures

The Exemplar based inpainting method pays special attention to linear structures and fills patches along the isophote. It is very effective in strong linear structures.

- **Inpainting with a small damaged portions of an image**

Original image



Small damaged portions of an image

(a)

(b)

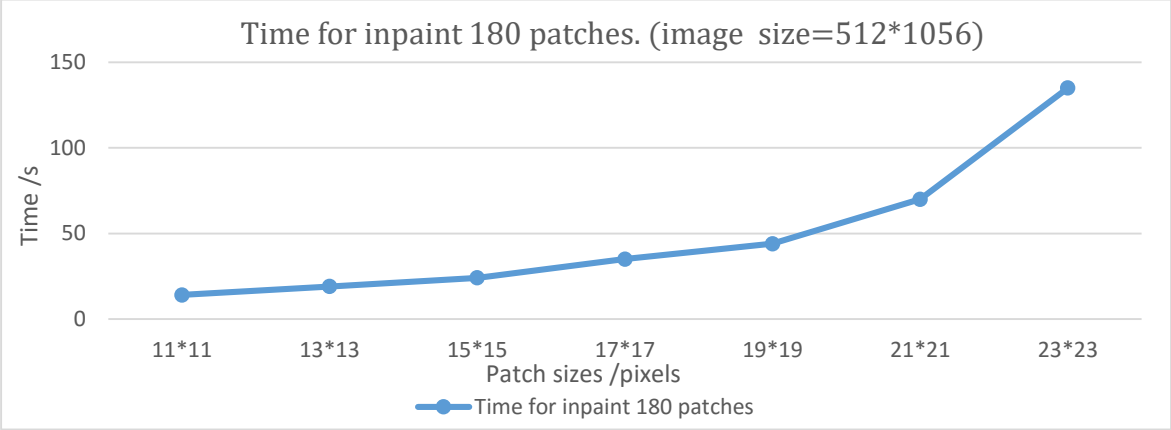
(c)

Figure 47. Inpainting used in a small damaged portions of an image. (a) Navier-Stokes based inpainting with an inpaintRadius=5. (b) Alexandru Telea inpainting with an inpaintRadius=5. (c) Exemplar based inpainting with patch size 11×11 pixels.

Navier-Stokes based inpainting, Alexandru Telea inpainting and Exemplar-based inpainting works well in a small damaged portions of an image because there are more reliable surrounding data and also more information is provided.

- **Runtime of the Exemplar-based inpainting method**

The patches size and the searching step can both effect the runtime:



The time increases with the patch size increases. Because a bigger patch size results in increasing computations when testing similarity of patches.

Figure 48. Inpainting time with different patch sizes.

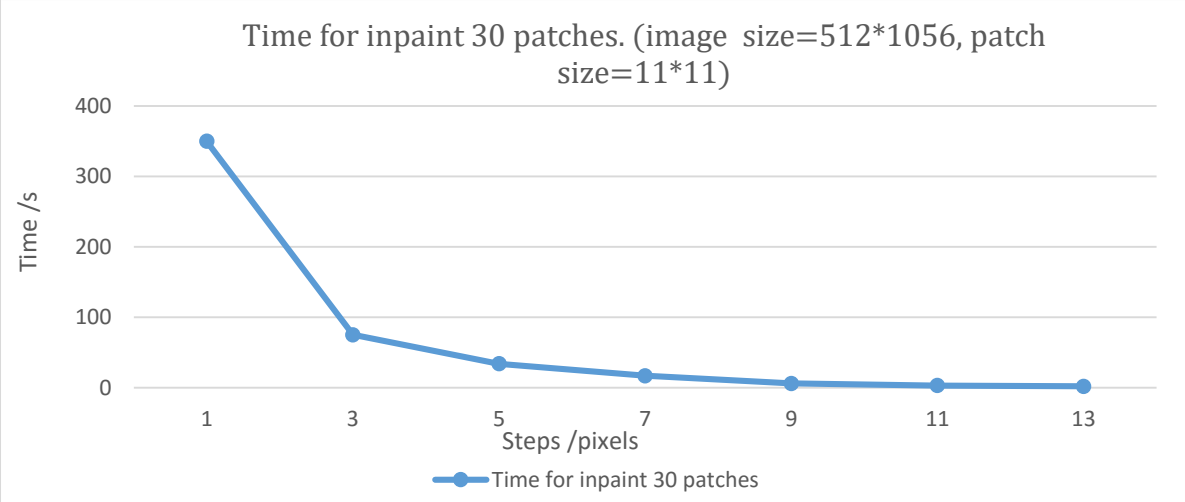


Figure 49. Inpainting time with different searching steps.

Step = 1 means search patches in all probability.

Step = 3 means search patches doing:

```
Loop1：  (row+=3 pixels)
    Loop2：  (column+=3 pixels)
    Test similarity of patches
```

The time decreases with the searching step increases. Because a bigger searching step results in decreasing patches to be tested.

Future work

Saliency extraction:

Although the saliency extractor (frequency tuned method) have achieved good results with images, it can be insufficient to analyse complicated variations common in natural images because it only considers first order average colour. Future work on saliency extraction would consider more image features like spatial relationships across image parts.

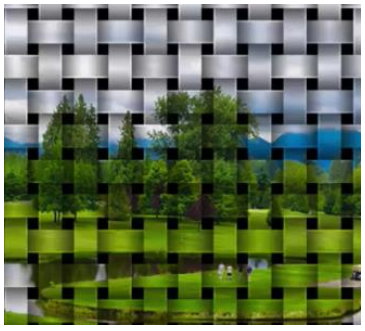
Two-dimensional bin rectangle packing:

In order to improve the quality of inpainting, it is essential to minimise holes.

Alternatively, there are other image arrangement methods like page layout or layout style that allowing overlap or packing the images together with irregular shapes.

Digital inpainting:

There was very few or even not papers to do digital inpainting with big holes created by different images. It was originally hoped that given a seamless good inpainting result using Exemplar-based inpainting method. Pixel-based inpainting methods and Exemplar-based inpainting methods can be efficiency in filling small holes but have disappointed results in big holes. The joins of images cannot be perfectly hided by inpainting. In this project, inpainting gaps between different images using Exemplar-based inpainting produces a sense of "interweaving" effect and looks interesting. It could be a good way to deal with the joins between different images.



There are some artworks to do an "interweaving" effect of an image in Photoshop. In the future, we can improve the Exemplar-based inpainting and investigate an interweaving way to create fantasy joins between images and apply it to the photomontage automatically.

Conclusion

As a whole, this project implements a photo montage system which has attempted different solutions to create an ideal image summary. The first part of this project is to extract salient region. I believe that the first part of this project has been a success. The frequency –turned approach of computing saliency using low-level features of colour and luminance is easy to implement and fast. In particular, the human face is treated with particular respect and preserved whole.

The second part is two-dimensional rectangle bin packing which mainly aims to summarise saliency regions of images and minimise holes without allowing overlap.

The final part is digital inpainting, and it attempted three different inpainting methods to restore the big hole generated by different images. Unfortunately, none of those solutions has proved successful. The big hole inpainted using those method has strong visual artefacts, the results are disappointing. In addition, inpainting gaps between different images using Exemplar-based inpainting produces a sense of "interweaving" effect and produces a better result.

Reflection on Learning

Having come towards the end of this project, I shall come to the reflection on what I have learnt over this course.

My first exposure to image processing was about two years ago, in order to get a higher mark, I chose a coursework about object recognition which aims to recognise one or more oranges from a uniform green background and draw circles to enclose the oranges. At that time, I have no relevant knowledge about image processing, in order to complete the task which was a challenge for me, I found tutorials online and followed the steps. Although I did lots of work to figure out how to use edge detector, Gaussian filter and what is Hough transform, I still missed some fundamental concepts and details of how they work.

The module 'Scientific Computing and Multimedia Applications' I took in my second year introduced mainly general concepts of image format, image compression and something about the video, it did not focus much on image processing. During my final year in the module 'Computer vision', I was introduced many theories and concepts of image processing. I also got some vital knowledge that was essential for this project.

Think back the orange recognition that I was struggling two years ago, and it has become very easy for me now because I have got more understanding of image processing and grew.

In this project, I have greatly developed my knowledge of image processing and deeper understanding of code optimization. I have taken several optimization methods in the Exemplar-based inpainting algorithm and stayed motivated to reduce the running time, having never felt that the running speed of an algorithm is so important. Each time I meet with the supervisor, it is the time for me to expand knowledge and gain a guidance. Computer vision has become my favourite subjects and I decided to take computer vision as my subject in the postgraduate study.

I have enjoyed working on this project immensely, having improved my problem-solving skill and increased my knowledge about image processing. If I go back to when the project starts, I would have done things differently.

Firstly, I would have managed my time more effectively because things always take longer than I expect. It is better to have a tight schedule in the first few weeks and allow more time in the end.

Secondly, I would have written a diary to track each process I made during the course of the project. I did write some keywords to record my processes but it is not enough to remind me of details.

Reference

[1] Agarwala, A., Dontcheva, M., Agrawala, M. Drucker, S., Colburn, A., Curless, B., Salesin, D., Cohen, M. 2004. Interactive Digital Photomontage. ACM Transactions on Graphics vol. 23(3), SIGGRAPH '04, p.294-295.

[2] Rother, C., Kumar, S., Kolmogorov, V., and Blake, A. 2005. Digital tapestry. In Proc. Conf. Comp. Vision and Pattern Recog.

[3] Rother, C., Kumar, S., Kolmogorov, V., and Blake, A. 2006. Auto Collage. In Proc. SIGGRAPH '06, New York, pp. 847-852

[4] Liu, T., Wang, J., Sun, J., Zheng, N., Tang, X., and Shum, H. 2009. Picture Collage. In IEEE Transactions on Multimedia, 11(7): 1225 - 1239.

[5] Ohashi, T., Aghbari, Z., and Makinouchi, 2003. A. Hill-climbing algorithm for efficient colorbased image segmentation. In IASTED International Conference On Signal Processing, Pattern Recognition, and Applications (SPPRA 2003), June 2003.

[6] Itti, L., Koch, C., and Niebur. 1998. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(11):1254-1259.

[7] Liu, T., Sun, J., Zheng, N., Tang, X., and Shu,H. 2007. Leaning to detect a salient object. In Proc.CVPR, 2007.

[8] Achanta, R., Estrada, F., Wils, P., and Susstrunk, S. 2008. Salient Region Detection and Segmentation. International Conference on Computer Vision System.

[9] Achanta, R., Hemami, S., Estrada, F., and Susstrunk, S. 2009. Frequency-tuned salient region detection. In IEEE CVPR, 2009, pp. 1597–1604.

[10] Jylänki,J. 2010. A Thousand Ways to Pack the Bin - A Practical Approach to Two-Dimensional Rectangle Bin Packing.

[11] Ashikhmin, M. 2001. Synthesizing Natural Textures. In proc. ACM Symp. On Interactive 3D Graphics, pp.217-226, Research Triangle Park, NC, Mar 2001.

[12] Criminisi,A., and Perez, P. 2003. Object Removal by Exemplar-Based Inpainting. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03).

[13] Bertalmio, M., Bertozzi, A. L., and Sapiro, G. 2001. Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting. In IEEE CVPR,vol. I, pp.355-365(2001).

[14] Telea,A. 2004. An Image Inpainting Technique Based on the Fast Marching Method. Journal of graphics tools 9.1 (2004): 23-34.

[15] Patel, P., Prajapati, A., and Mishra, S. 2012. Review of Different Inpainting Algorithms. International Journal of Computer Applications (0975 – 8887), Volume 59– No.18.

[16] Viola,P. and Jones,M. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features. International Journal of Computer Vision.