

**Skin feature analysis
and classification
Project Report – August 2004.**

Mathieu Albenque 53128893

Supervised by **Prof. Paul Whelan**

Table of contents

1	INTRODUCTION	- 3 -
1.1	PIGMENTED SKIN LESIONS CHARACTERISTICS	- 3 -
1.2	EQUIPMENT AND SOFTWARE	- 4 -
1.2.1	<i>The Images</i>	- 4 -
1.2.2	<i>The software used for the programmation</i>	- 5 -
2	THE DESIGN OF THE PROGRAM	- 6 -
2.1	THE VISUAL DESIGN	- 6 -
2.2	THE IMPLEMENTATION DESIGN	- 7 -
3	TECHNICAL BACKGROUNDS.....	- 9 -
3.1	TEXTURE ANALYSIS AND CLASSIFICATION	- 9 -
3.1.1	<i>Introduction</i>	- 9 -
3.1.2	<i>The feature extraction techniques</i>	- 9 -
3.1.2.1	The Local Binary Pattern (LBP)	- 9 -
3.1.2.2	The G-Statistic	- 11 -
3.1.3	<i>Unsupervised Texture Segmentation Algorithm</i>	- 12 -
3.1.3.1	Algorithm development	- 12 -
3.1.3.2	Hierarchical splitting	- 12 -
3.1.3.3	Agglomerative merging	- 14 -
3.1.3.4	Boundary Refinement	- 16 -
3.2	COLOUR ANALYSIS AND CLASSIFICATION	- 17 -
3.2.1	<i>Introduction</i>	- 17 -
3.2.2	<i>The k-means clustering algorithm</i>	- 17 -
4	IMPLEMENTATION	- 19 -
4.1	THE PROGRAM ARCHITECTURE	- 19 -
4.1.1	<i>The structures</i>	- 19 -
4.1.2	<i>The basic functions</i>	- 20 -
4.2	THE MAIN FUNCTIONS	- 21 -
4.2.1	<i>Texture analysis and classification</i>	- 21 -
4.2.1.1	The LBP calculation	- 21 -
4.2.1.2	The G-statistical value	- 21 -
4.2.1.3	The hierarchical splitting	- 21 -
4.2.2	<i>Colour analysis and classification</i>	- 22 -
4.2.2.1	RGB to XYZ transformation	- 22 -
4.2.2.2	K-means clustering	- 23 -
5	RESULTS AND DISCUSSION	- 24 -
5.1	RESULTS	- 24 -
5.1.1	<i>Texture analysis and classification</i>	- 25 -
5.1.2	<i>Colour analysis and classification</i>	- 28 -
5.2	DISCUSSION	- 31 -
6	CONCLUSION AND FURTHER RESEARCHES	- 31 -
7	REFERENCES	- 32 -
8	APPENDIX	- 33 -
8.1	C++ SOURCE CODE	- 33 -
8.1.1	<i>Skinfeature.c</i>	- 33 -
8.1.2	<i>Skinfeature_lib1.c</i>	- 36 -
8.1.2.1	Texture analysis source code	- 36 -
8.1.2.2	K-mean clustering source code	- 43 -

Abstract

Skin feature analysis and classification

The aim of this project is to provide an efficient way to segment skin cancer images in order to support practitioners in their medical diagnosis of skin melanoma. The DCU Vision Group proposes a novel method that combines colour and texture studies for the segmentation of the skin lesion from unaffected skin region in an image. A good discrimination of skin lesions is allowed by the combination of the distributions of colour and texture. This project consists in developing a medical imaging system in the area of skin feature analysis and classification with a view to supporting the early medical diagnosis of skin melanoma. This application has been developed in C++ language. Various approaches can be envisaged for the treatment of the skin feature images. But the method that consists in combining a colour and texture analysis to segment the skin lesion from the unaffected skin region seems to be the best adapted one. Series of tests have been done using this method and allow to consider this novel way to segment skin cancer images as an efficient way.

Keywords : Segmentation, Texture, Colour, Distributions, Features, Skin.

1 Introduction

As said before, this project consists in designing a program which can assist medical practitioners in their diagnosis of melanoma. The aim of this program is to provide an efficient way to segment the skin cancer images from a colour and texture analysis. This method provides a good meaning of discrimination of the skin lesions. The fact that there are different types of skin cancers which differ in their colour and texture composition justifies the use of the colour and texture analysis.

To understand why a such treatment is appropriated to the skin feature analysis and classification, the next point of this report deals with the pigmented skin lesions characteristics.

1.1 Pigmented skin lesions characteristics

Clinical features of pigmented lesions suggestive of skin cancer are the ABCD's of the skin cancer. Each of the four letters refers to a particularity of the melanomas ;

- ✓ **A** stands for Asymmetry.
- ✓ **B** stands for Border irregularity.
- ✓ **C** stands for Colour variation.
- ✓ **D** stands for Diameter greater than 6mm.

These previous characteristics are very important in the diagnosis of the skin cancers. Before the creation of our program, the techniques used to recognise the malignancy of a melanoma were based either on colour or on texture analysis. But the A and B characteristics refer to the texture of the pigmented skin lesion, and the C characteristic refers to the colour distribution. The pigmentation of the skin lesion represents a very important part of the diagnosis. For example, the asymmetry of pigmentation is indicative of malignant lesion, such as de-pigmentation, irregular bluish or grey-blue areas, or brown globules and black dots.

So, a combination of colour and texture analysis is providing a better diagnosis.

1.2 Equipment and software

In order to realise this diagnosis from an image, it's necessary for an automatic system to detect and localise the lesions within the image. In that case, the program should be able to segment automatically the malignant region from the rest of the image.

1.2.1 The Images

To apply the segmentation, we need to have very good images without any form of compression. The Targa (*.tga) format is one of these formats , and is this we've chosen to use for the application. Because the skin lesion have a complex structure in size and colour variations and because the diagnosis is mainly leaned on the variations of colour and texture, that is very important to have an image the most faithful as possible to the real case.

To realise the tests we use a certain number of images. All are in Targa format and are colour images. Here are these images :

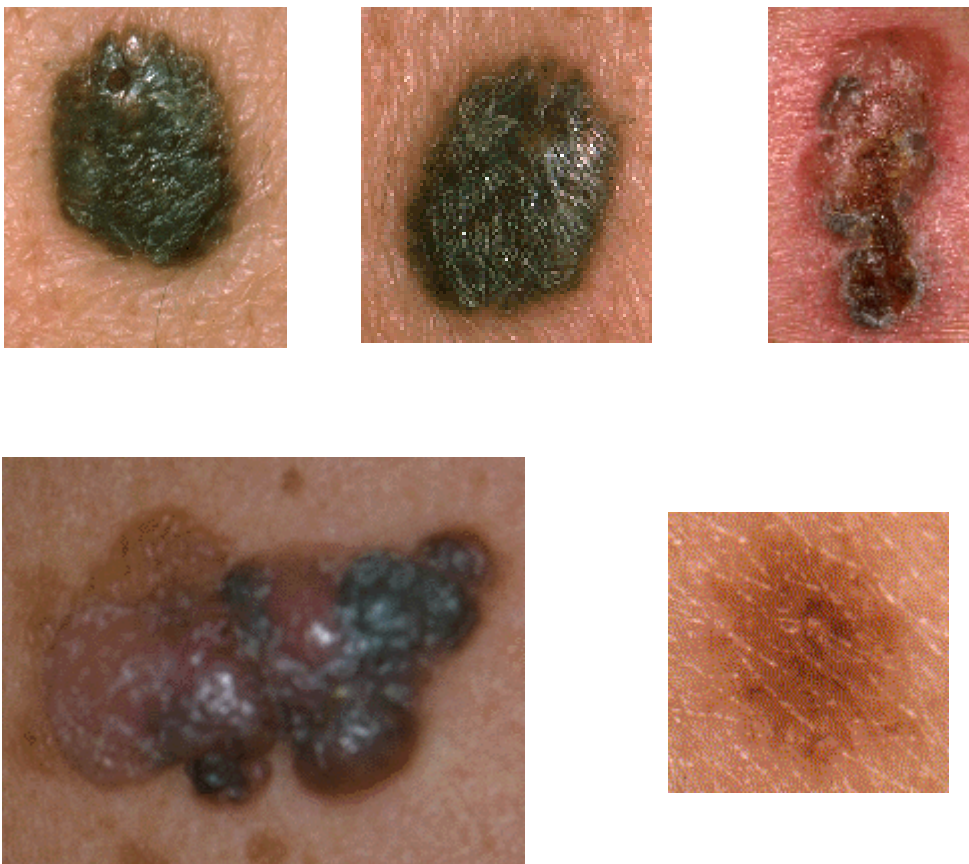


Fig1 : the sample images

1.2.2 The software used for the programmation

For implementing the different algorithms used for the colour-texture segmentation, we chose to use the C++ language. The program has been developed with a Microsoft Visual C++ 6.0 software (*Copyright © 1994-98 Microsoft Corporation*).

2 The Design of the program

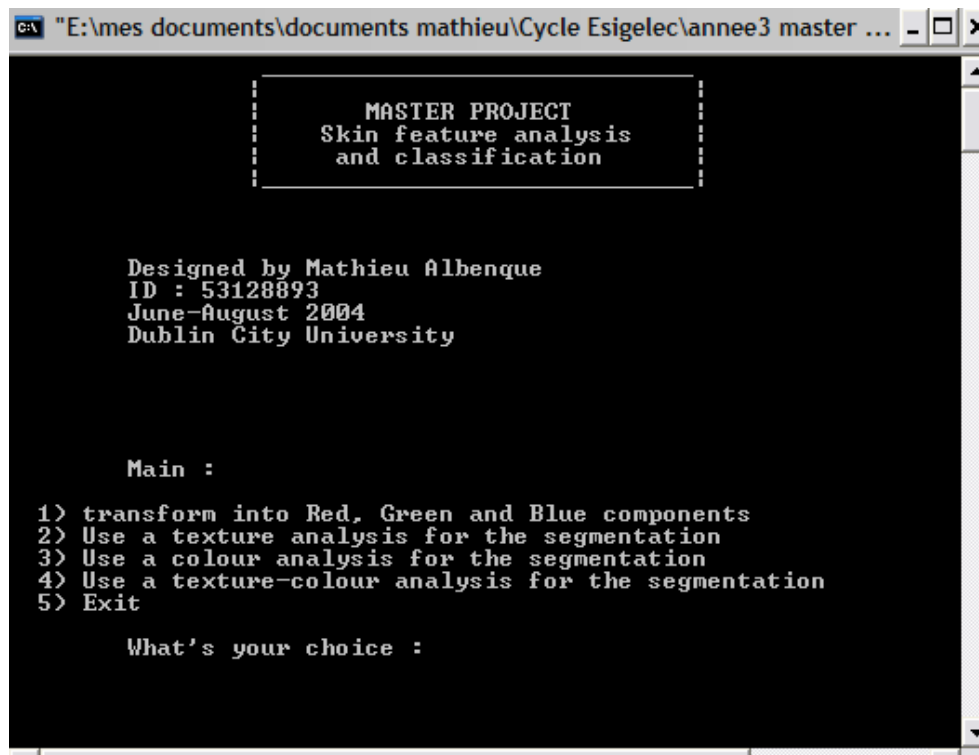
2.1 The visual design

Because our application performs a colour texture segmentation, the method is decomposed into three main parts :

- ✓ The texture analysis
- ✓ The colour analysis
- ✓ The colour-texture analysis

Each of these analysis is developed from a particular function (*void create_diff_imagebis(image *imgin)* for the texture and *void kmeanclustering(image *r, image *g, image *b, int w, int h, image *kmean)* for the colour and both for colour-texture analysis)).

When applying the application, a menu is proposed. Through it, we can chose between the three possibilities (**fig2**).



```
C:\ "E:\mes documents\documents mathieu\Cycle Esigelec\annee3 master ... - _ X
MASTER PROJECT
Skin feature analysis
and classification

Designed by Mathieu Albenque
ID : 53128893
June-August 2004
Dublin City University

Main :
1) transform into Red, Green and Blue components
2) Use a texture analysis for the segmentation
3) Use a colour analysis for the segmentation
4) Use a texture-colour analysis for the segmentation
5) Exit

What's your choice :
```

fig2 : visual approach of the application

2.2 *The implementation design*

The way of combining colour and texture information requires a preliminary study of texture in one hand, and colour in a second hand. So, we can decompose the program into two main parts which consist in the texture analysis and classification for the first one and colour analysis and classification for the second one. The colour-texture analysis and classification is just issued from the combination of the two main analysis.

The implementation method can be decomposed into several steps dealing either with colour or with texture or both. Here are the following steps that are implemented in the C programme :

- The Local Binary Pattern features are extracted from the average of the three planes in RGB colour space.
- The distribution of the texture features is used for texture discrimination.
- A G non-parametric statistical test is used as a similarity measure to discriminate the texture distributions.
- A hierarchical splitting method is used to split the image based on the texture descriptions using the similarity measure.
- An agglomerative merging procedure based on the merging criteria determines the similarity between two different regions using G statistic, producing the segmented image.
- A boundary refinement algorithm based on the colour histograms enhances the segmented result to obtain the final segmented image.

- An unsupervised k-means clustering algorithm has to be performed on the image to obtain the distribution of the colour clustered labels.
- Distribution of the texture features and distribution of the colour clustered labels is used to describe the texture and the colour. The Merger Importance (MI) values had been calculated before in using the G statistic. Weights are computed using the histograms of the colour labels.

The combination of all these steps allows to design a robust programme which is able to segment any skin cancer image.

3 Technical backgrounds

3.1 Texture analysis and classification

3.1.1 Introduction

A novel technique for implementing an algorithm for unsupervised texture segmentation has been proposed in the ISSC paper of the 25-26th June 2002^[1]. The paper stipulates that statistical approach is well suited to the analysis and classification of random or natural textures. This texture feature extraction technique is based on a non-parametric statistical approach. The feature extraction uses the Texture Spectrum method through the Local Binary Pattern (LBP) calculation. The texture description is allowed using the G-Statistic and the final segmentation comes from the implementation of a Split and Merge algorithm.

3.1.2 The feature extraction techniques

3.1.2.1 The Local Binary Pattern (LBP)

The Texture Spectrum method is a very useful technique for texture feature extraction. A texture image can be decomposed into a set of small textural units called *Texture Units* (TU). A texture unit is composed by eight elements (pixels) of the image. These eight elements come from a 3*3 neighbourhood where the central element is consider as a threshold element and is excluded from the Texture Unit. The neighbourhood is represented by $V = \{V_0, V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8\}$ where V_0 is the central element. The occurrence distribution of texture units is called the *Texture Spectrum*.

The Local Binary Pattern is a form of texture spectrum method. For Each 3*3 neighbourhood, we threshold the eight elements V_1 to V_8 of the neighbourhood with the central element of the neighbourhood, V_0 . Then the elements of the texture unit $\{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$, take one of the two possible values (0,1) using the following rule.

$$\begin{aligned} E_i &= 0 \text{ if } V_i < V_0 \\ E_i &= 1 \text{ if } V_i \geq V_0 \end{aligned}$$

From the new texture unit elements, the LBP value is calculated. To each element E_i of the texture unit corresponds a particular *weight* which determines the value of the LBP. The process is explain in **fig3** as following ;

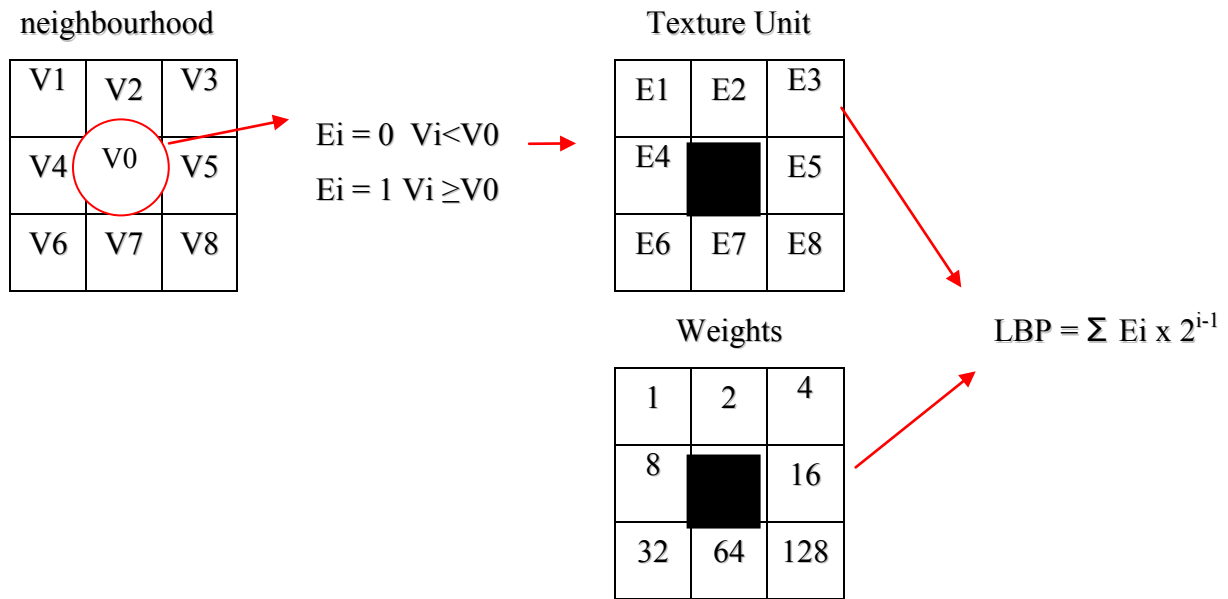


Fig 3 : Computation of Local Binary Pattern

Here is now an example :

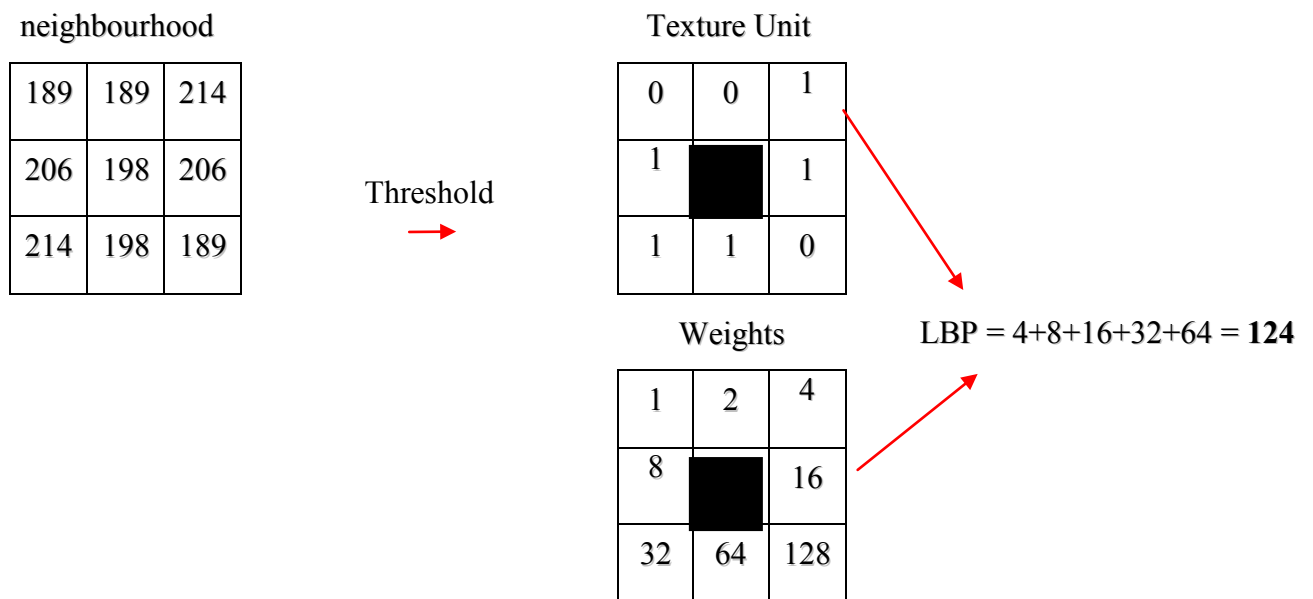


Fig 4 : Example of computation of Local Binary Pattern

In addition to the LBP calculation which represents the texture repartition of the image, this method provides another parameter which can be useful in the discrimination of textures. This is the Contrast measure (C), which is the difference between the average grey levels of pixel with value 1 and pixels with value 0 contained in the texture unit. In combining the LBP calculation and the C calculation, we change the size of the distribution which becomes a 2D distribution of size 256*8.

But a 1D LBP distribution of size 256 is enough for obtaining good results.

3.1.2.2 The G-Statistic

When the LBP distribution is calculated, the next step is the discrimination between two distributions in the images. If in the image we consider two neighbour regions, the texture distributions are extracted from the LBP calculation. But the problem is to know if the two samples are from the same texture or not. The uniformity of the region composed by the two neighbour samples is tested with the G-statistic calculation.

We consider the first sample as Model sample. The second sample is the same dimension as the Model and is named Sample. For each of them, we generate the LBP histograms of $n=256$ bins, such as m_i is the frequency of bin i in the Model, and s_i is the frequency of bin i in the Sample. Then for m_i and s_i not null, we calculate the G values with the following formula :

$$G = 2 \sum_{i=0}^n s_i \log(s_i / m_i)$$

From the G value, we can say if Model and Sample are drawn from the same population so have the same texture.

The LBP and G statistical feature extraction techniques are combined together to realise the final implementation of the unsupervised texture segmentation.

3.1.3 Unsupervised Texture Segmentation Algorithm

3.1.3.1 Algorithm development

The segmentation of an image implies a division of the image into regions of similar attributes. If no assumption can be made about the number and the type of textures, then an unsupervised texture segmentation is implemented. In the case of unsupervised texture segmentation, a statistical analysis is performed on the vectors distributions and the aim is to recognise clusters in the texture distribution and to assign labels to them. This method follows hierarchical splitting and agglomerative merging procedure. The algorithm used in the C++ program is this proposed in ^[1] and implements the splitting using quadtree structure and merging generates a forest structure with each tree of the forest representing a merged area in the image.

3.1.3.2 Hierarchical splitting

The principle of hierarchical splitting consists in dividing the image into blocks of roughly uniform texture. The whole image is considered to be the upper level of the splittree. The whole image is firstly divided into four subblocks. The uniformity of the region composed by these blocks is tested in using the G values issued from the LBP histograms. If the region tested is not uniform, it is splitted again into four subblocks which are tested by uniformity in using the same principle as before. This principle consists in calculating the G values between a model block (one of the blocks) and the other three blocks, in calculating $R = G_{\max}/G_{\min}$ and in thresholding the R value with a X value which lie between 0.9 and 1.2. The operation is repeated recursively until a minimum block size S_{\min} is reached. The value of S_{\min} is most of time 16×16 .

The algorithm is implemented using a quadtree structure. A quadtree structure is called a splittree where every parent node have four child nodes (**fig5**). This algorithm generates the splittree by using an iterative procedure which begin with the initialisation of the tree by the whole image and stops when the last iteration generates leaves with a size S_{\min} . The leaves of the splittree at the end of the iterations refers to homogeneous regions in the image. These leaves are denoted as $V = \{V1, V2, V3, \dots\}$, in **fig5** and **fig6**. The union of theses homogeneous leaves represents the whole image. Pseudo-code for the splittree is given in **fig7**.

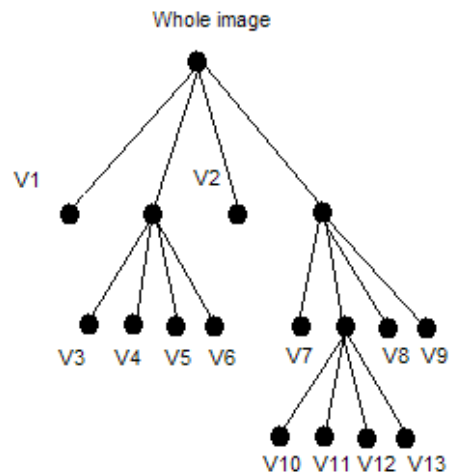


Fig 5 : An example of the splittree

V1	V3	V4	
	V5	V6	
V2	V7	V10	V11
		V12	V13
	V8	V9	

Fig 6 : the image representation

```

NodeType Head ;
//The Whole image is assigned to Head of tree
Head.Image = Image;
While(1)
  //start iteration
  int No_of_leaves
  //Scan through all the leaves
  NodeType *Leaf = GetLeaves(Head, &No_of_leaves);
  Flag=0;
  For(i=0;i<No_of_leaves;i++)
  {
    //check for homogeneity
    if(!Homogeneous(Leaves[i].Image))
    {
      //split the image of ith leaf into four
      split(Leaf[i])
      flag=1;
    } //endif
  } //endfor
  if (flag==0) //exit condition
  break;
} //end while
  
```

Fig 7 : Pseudo-code for splittree [1]

3.1.3.3 **Agglomerative merging**

After the hierarchical splitting, an agglomerative procedure is applied to the image which has been splitted into blocks of roughly uniform texture. This procedure merges similar adjacent regions until a stopping rule is satisfied. In the image, pairs of adjacent segments issued from the splitting procedure, are defined by a particular value named Merger Importance (MI). The Merger importance is calculated from $MI = p \times G$ where p is the smallest number of pixels among the two regions and G is the value of G-statistical between the two regions. The pair of adjacent segments which has the smallest Merger Importance value is merged. After merging, the two respective LBP histograms are summed to be the histogram of the new region. Then the G distribution between the new region and all adjacent regions needs to be computed again. This merging procedure is adopted until a stopping rule is satisfied. This rule is defined by the ratio MI_{curr}/MI_{max} , where MI_{curr} is the Merger Importance of the current merge and MI_{max} is the largest Merger Importance of all preceding merges. The ratio is thresholded by a threshold value Y . If the ratio exceeds Y , the merging procedure is halted and a new region which is not merged for the moment is then treated. The value of Y is found experimentally but lie between 1.2 and 1.4 in this type of application.

Theoretically, the adjacent regions with identical LBP histograms have a value of G which is null, so a zero MI value which will lead to the termination of merging prematurely. This particular case has to be considered.

The implementation of agglomerative merging is done by creating a mergegraph which is characterised by a set of nodes and two types of links which connect these nodes. A mergegraph is built from the triplet (V,A,E) as following :

- $V = \{V_1, V_2, \dots\}$ which are the leaves of the splittree.
- $A = \{a_1, a_2, \dots, a_n\}$ which represents the adjacent links between the leaves of the splittree.
- $E = \{e_1, e_2, \dots, e_n\}$ where e_i is the unidirectional link between V_i and $V_{(i+1) \bmod n}$. This forms the circular link list of nodes.

Fig8 shows the mergegraph corresponding to the splittree described in **fig5**. The mergenode procedure traverses through the circular link list. The Merger Importance value for all adjacent nodes are calculated and the region with the smallest Merger Importance value is merged with the current node. Then the circular link list is updated and the procedure is repeated until the stopping rule is satisfied. The pseudo-code given in ^[1] is presented in **fig9**.

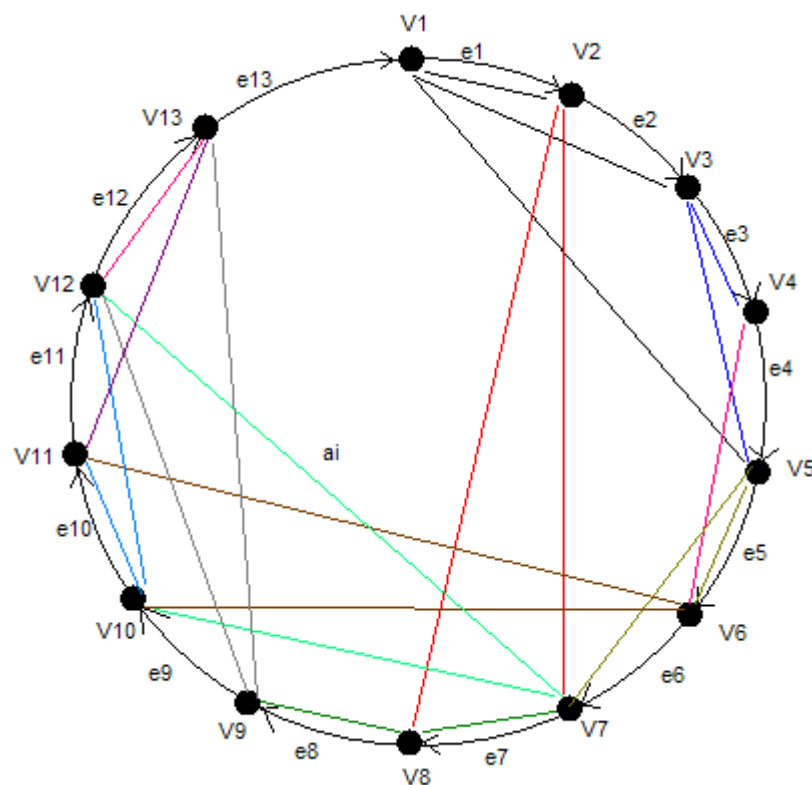


fig8 : example of mergegraph.


```
//Head points to the first node of Mergegraph
NodeType cnode;
//if current node is not equal to head
for(cnode = Head; cnode != Head ; cnode=cnode->next)
{
//Merge adjacent node of mallest merger importance with the
current node
Merge_Adj(cnode);
//update adjacent node for the new cnode
update_Adj_Information(cnode);
//update merge Information for cnode
update_merge_Information(cnode)
} //endfor
if (cnode(mergecondition))
{
AddForest(cnode);
} //endif
```

Fig 9 : Pseudo-code for mergegraph ^[1]

3.1.3.4 Boundary Refinement

The result of the split and merge algorithm gives a blocky segmented image. For the improvement at the boundaries between various regions we use a boundary refinement algorithm based on the neighbourhood study. A pixel in the segmented image is regarded as a boundary point if it is on the boundary of at least two distinct regions ,i.e., its region label is different from at least one of its four neighbours.

The procedure consists in placing around a studied pixel, a discrete square with a dimension d and in computing the colour histogram for this region. The corresponding colour histograms for the different neighbouring points are also calculated. The homogeneity of the square region and the i^{th} neighbouring region is tested in using a Merger Importance calculation. If the Merger Importance value between adjacent regions and the region around the considered pixel is lower than the merger threshold, the pixel is reclassified and is issued from another region. This procedure is iterative and proceeds until no pixels are relabelled.

3.2 Colour analysis and classification

3.2.1 Introduction

Clustering analysis is the process of grouping finite set of objects into subsets. The k-means method has been shown to be effective in producing good clustering results in colour analysis and classification applications.

3.2.2 The k-means clustering algorithm

The study realised through this project uses the unsupervised clustering technique based on the k-means algorithm [2],[3]. The k-means algorithm is the most popular among the iterative clustering algorithms. In most of cases, this technique is adopted to determine the colour clustered distribution.

If we consider an input colour image with dimension n , before applying the k-means algorithm, the image has to be transform in a suitable plane for the study. This transformation is a RGB to XYZ transformation [3]. The colour image is specified by its trichromatic coefficients (x,y,z) obtain from the following operations :

$$\checkmark X = 2.36R - 0.515G + 0.0052B$$

$$\checkmark Y = -0.89R + 1.42G - 0.014B$$

$$\checkmark Z = -0.46R + 0.88G + 1.009B$$

$$\bullet x = X/(X+Y+Z)$$

$$\bullet y = Y/(X+Y+Z)$$

$$\bullet z = Z/(X+Y+Z)$$

Then the image is represented in a (x,y,z) plane, but a (x,y) 2D plane is good enough good for the study.

Then, the first step of the k-means algorithm is to define the number of clusters k , needed. Determination of the optimal number of clusters is a very difficult task as this parameter is image dependent. In order to address this issue, the number of cluster is set to a particular value considered to be sufficiently large to assure that all important regions in the image with similar colour characteristics are clustered. Each k prototype is randomly initialised to one of the n input patterns and have a (x,y) coordinates pair which represents the centroïde of the cluster.

The next step is to assign each pixel of the picture to one of the k clusters. This is made by a distance calculation. For each pixel of the image, we calculate the distance between the pixel and each of the clusters centroids. The pixel belongs to the cluster for which the distance is minimum. We do this step for all the pixels of the picture.

Then the centroids of the clusters are calculated again from the new distribution of pixels belonging to the corresponding cluster such as x is equal to the mean of the x coordinates of the pixels and y is the mean of the y coordinates. A distance calculation is applied again, the pixels are re-labelled when necessary and the centroids are calculated again.

The procedure is repeated until no pixel is relabelled in the image. **Fig10** shows an example of k -means algorithm.

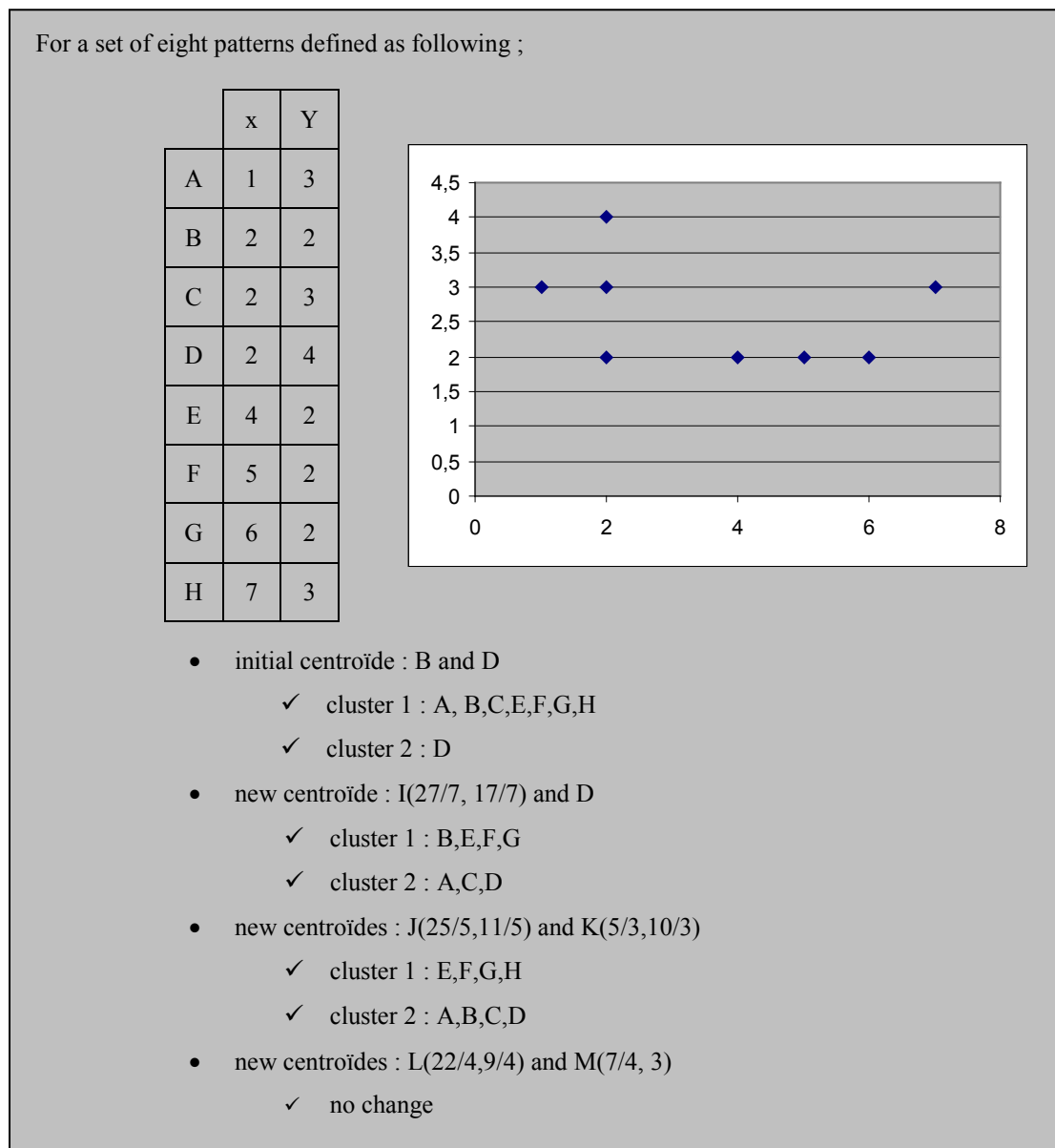


Fig10 : An example of k -means algorithm for $k=2$

4 Implementation

4.1 The program architecture

The program is built according to a relatively common architecture in C++ using classes with constructors and destructors, structure definitions and common functions.

For a better understanding a separate compilation principle is used. The **fig11** presents the global architecture of the programme.

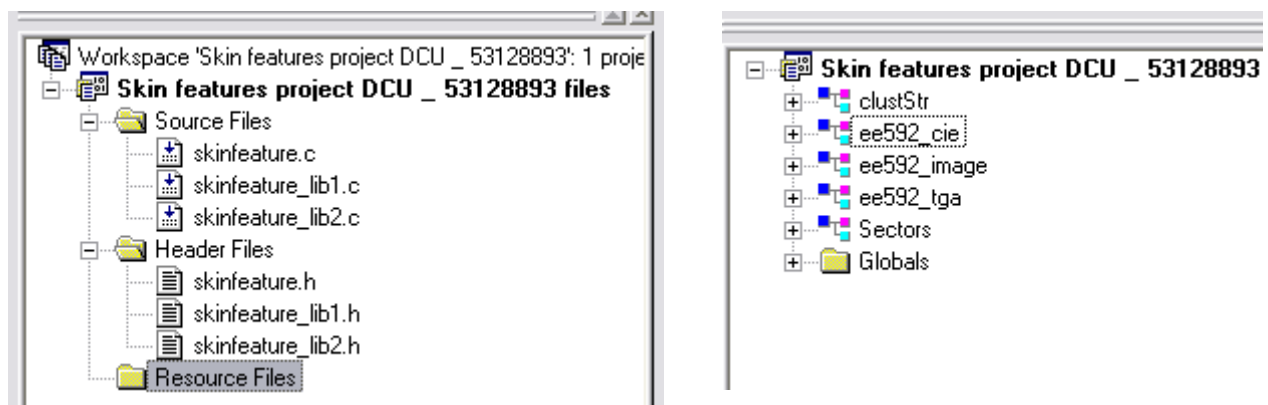


Fig11 : the programme architecture

4.1.1 The structures

As said in the beginning of this paper, the format we chose to use for input colour images is Targa format. But, for the treatment, the images are decomposed into their Red, Green and Blue components and these components are stocked into pgm images. So we create corresponding structures for targa images and pgm images.

A Targa image is characterised by five parameters ; the width, the height, the red component, the blue component and the green component. A pgm image is characterised by only three components because this image is a grey level image ; the width, the height and the pixel value. Here are the two main structures used (**fig12**).

```
struct ee593_tga
{
    int width;
    int height;
    pel *rpels;
    pel *gpels;
    pel *bpels;
};
typedef struct ee593_tga tga;

struct ee593_image
{
    int width;
    int height;
    pel *pels;
};
typedef struct ee593_image image;
```

Fig12 : the structures definitions

4.1.2 The basic functions

These functions are used during the manipulation of the images. They can be used for creating images, for cloning images either still for saving or for loading images, whether in Targa format or in pgm format. These functions are the following one and can be found in the source code.

- *alloc_image(int width, int height)*
- *alloc_tga(int width, int height)*
- *clone_image(image *img)*
- *get_image_width(image *img)*
- *get_image_height(image *img)*
- *get_image_pels(image *img)*
- *pgm_load_image(char *filename)*
- *pgm_read_image(char *filename, image *img)*
- *pgm_write_image(image *img, char *filename)*
- *tga_load_image(char *filename)*
- *tga_read_image(char *filename, tga *img)*
- *tga_write_image(tga *img, char *filename)*

The next part deals with the implementation of the different functions used to achieved a texture analysis and classification and a colour analysis and classification as described in the theoretical part of the report.

4.2 The main functions

4.2.1 **Texture analysis and classification**

The source code for the texture analysis is in appendix.

4.2.1.1 **The LBP calculation**

The Local binary pattern values are simply calculated as explained in the theoretical part. In order to calculate the LBP values, two functions are used. The first one is determining the Texture Units in thresholding the elements of a 3x3 neighbourhood with the central element (*void lbp_img(image *imgin, image *imgout)*). The other function is calculating the Local Binary pattern values from the Texture Unit and it's corresponding Weight matrix (*int lbp_block(int in[])*).

4.2.1.2 **The G-statistical value**

The G-statistical values are calculated from the pixel distributions. So a function which generates pixel colour distribution histograms is needed ; (*void histog(image *in, double histo[])*). And for calculating the G values we just need a function that takes as parameters the two histograms of the concerned regions ; (*double calcG(double histom[], double histos[])*).

4.2.1.3 **The hierarchical splitting**

The hierarchical splitting consists in dividing the image into blocks of roughly uniform texture. Each time a block is defined as non uniform, it is splitted into four subblocks. For this part of the implementation a class is used. In this class, the uniformity of images is tested and in case of non uniformity, the image is decomposed into four images with same size i.e. the quarter as the original image. Then the process is done to each of the four new images, until the images issued from the division have a size of 16x16. **Fig13** is showing the picture decomposition.

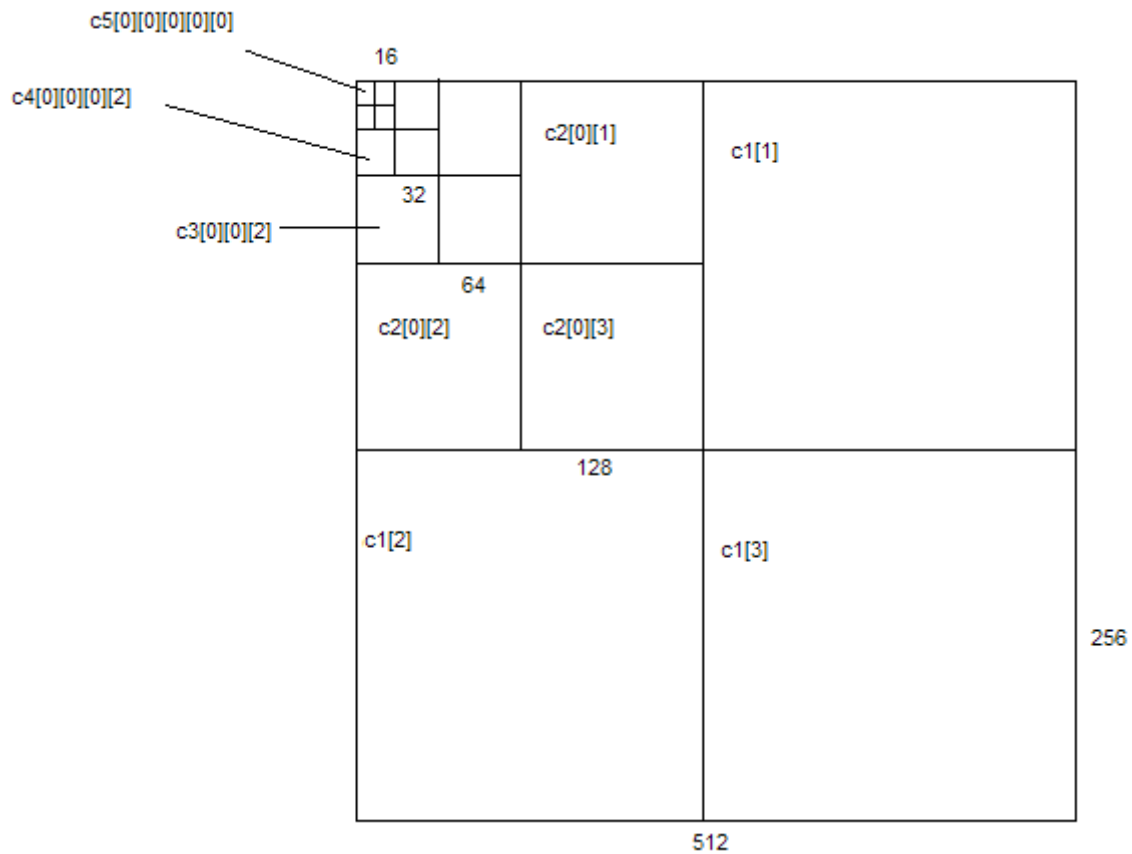


Fig13 : the splitting decomposition

4.2.2 Colour analysis and classification

4.2.2.1 RGB to XYZ transformation

The first step of the colour clustering consists in changing the plane of the study. Because the trichromatic coefficients (x,y,z) of the image are characterising the colour distribution, the x , y and z values are calculated from the Red, Green and Blues components of the input colour image. **Fig14** is the C++ source code that illustrates this step.

```
for (i=0;i<size;i++,pixelred++,pixelgreen++,pixelblue++)
{
    rgbval[0]=(double)*pixelred;
    rgbval[1]=(double)*pixelgreen;
    rgbval[2]=(double)*pixelblue;

    xyzval[0]=2.36*rgbval[0] - 0.515*rgbval[1] + 0.0052*rgbval[2];
    xyzval[1]=-0.89*rgbval[0] + 1.42*rgbval[1] - 0.014*rgbval[2];
    xyzval[2]=-0.46*rgbval[0] + 0.88*rgbval[1] + 1.009*rgbval[2];

    k=(xyzval[0]+xyzval[1]+xyzval[2]);

    if(k!=0)
    {
        x->pels[i]=xyzval[0]/k;
        y->pels[i]=xyzval[1]/k;
        z->pels[i]=xyzval[2]/k;
    }
    else
    {
        x->pels[i]=x->pels[i-1];
        y->pels[i]=y->pels[i-1];
        z->pels[i]=z->pels[i-1];
    }
}
```

Fig14 : RGB to XYZ transformation source code

4.2.2.2 K-means clustering

Once the RGB to XYZ transformation is done, the k-means procedure begins in initialising the first cluster centroïdes. They can be randomly chosen, but they can also be geometrically defined. This second solution is the adopted solution in the program. The first nine cluster centroïdes corresponds to particular chosen points from the image. **Fig15** presents the repartition of the first nine centroïdes.

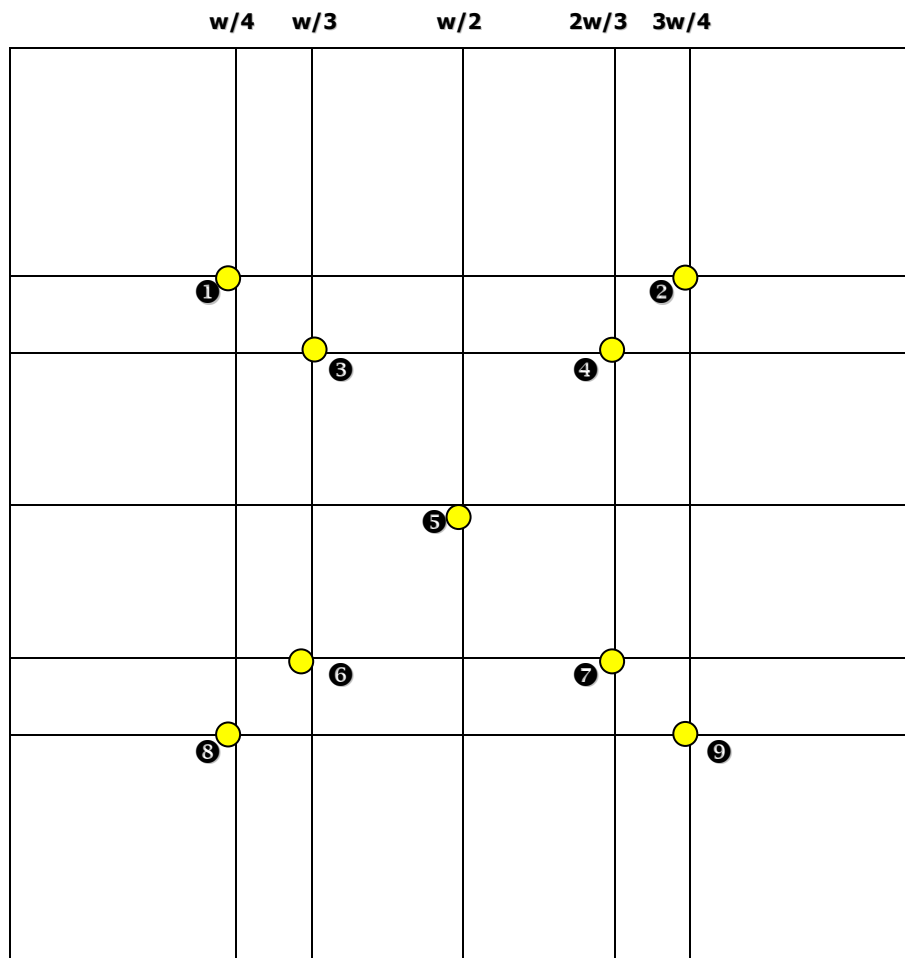


Fig15 : cluster centroids initialisations from the original image.

Then the centroids are redefined at each iteration until the stopping condition is reached.

5 Results and discussion

5.1 Results

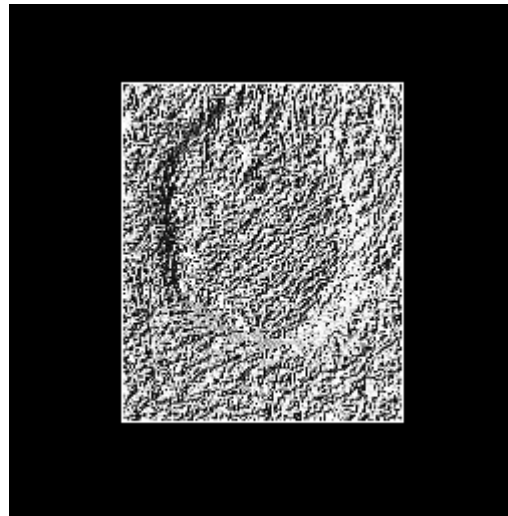
The images used for colour texture analysis are presented in **fig1**. Segmentation results for these images are presented in the following part. The results are all presented in grey scale images and are saved in hard disk in order to provide a view to supporting the early medical diagnosis of skin melanoma.

5.1.1 Texture analysis and classification

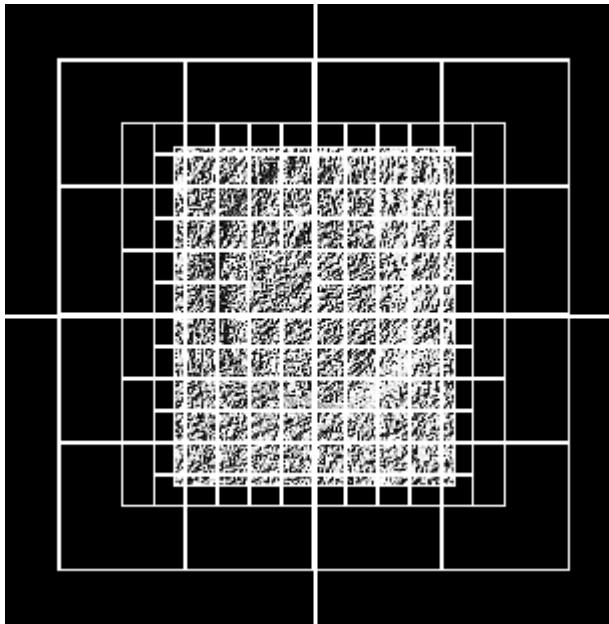
Here are the results of the unsupervised texture segmentation for the set of images. In the following results we can find for each image, the original image, the LBP image and the splitted image. The merging result is not presented here due to a problem in the program that can't have been solved due to a lack of time.



Original image



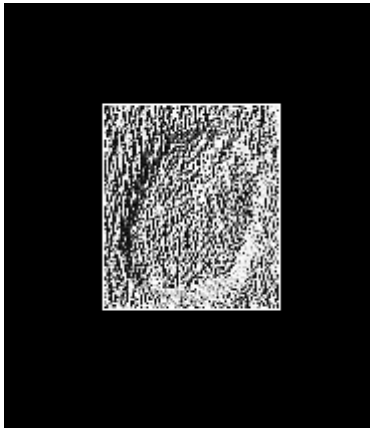
LBP image



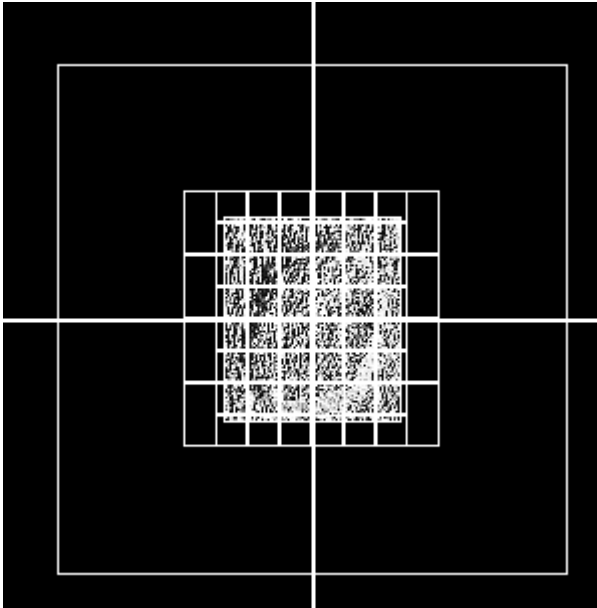
Splitted image



Original image



LBP image



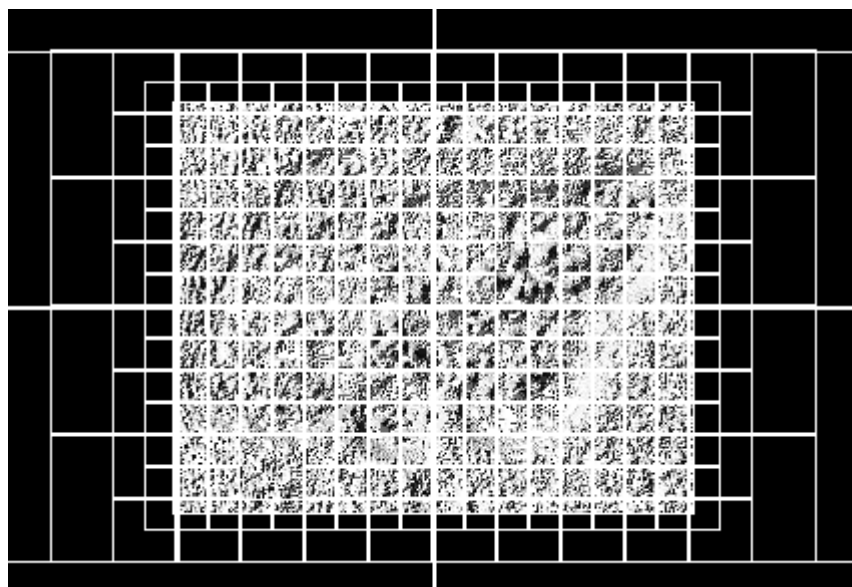
Splitted image



Original image



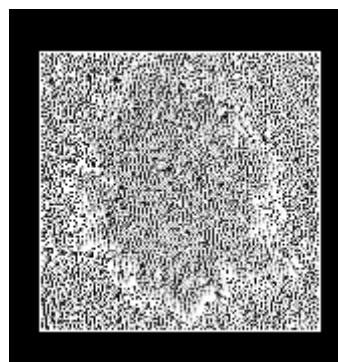
LBP image



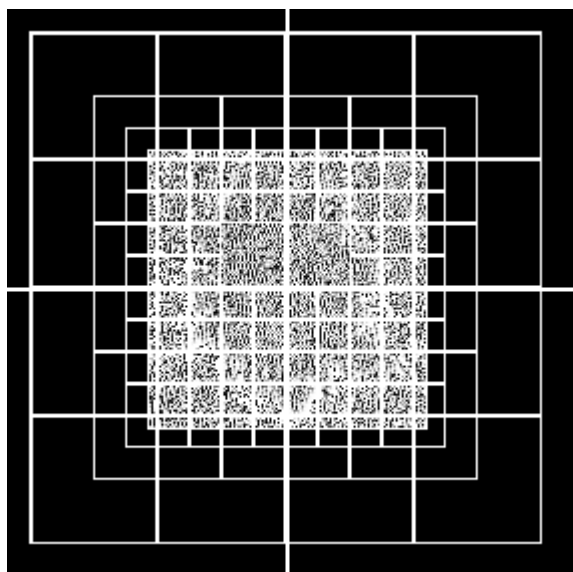
Splitted image



Original image



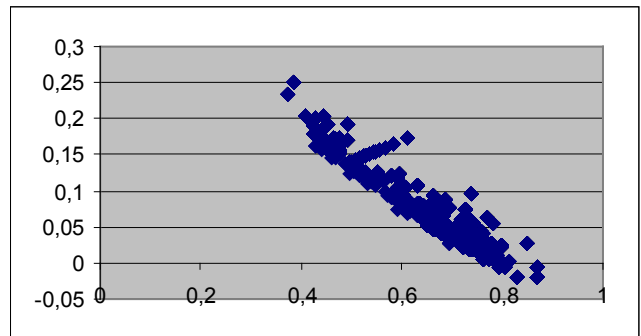
LBP image



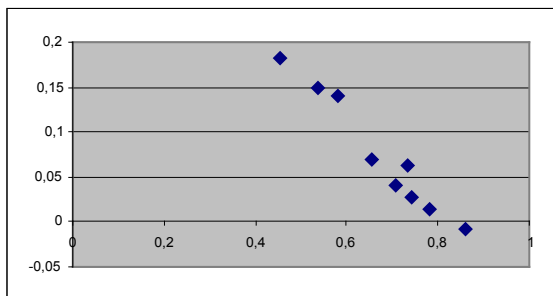
Splitted image

5.1.2 Colour analysis and classification

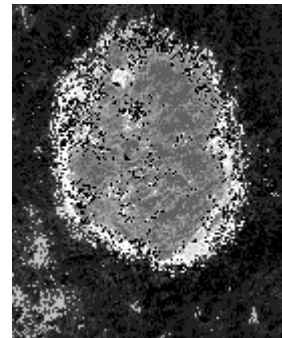
For each image, we can find the different steps of the algorithm. From the original image, the (x,y) pixel repartition is drawn, the final centroïdes are drawn and finally the k-means clusterised image is presented.



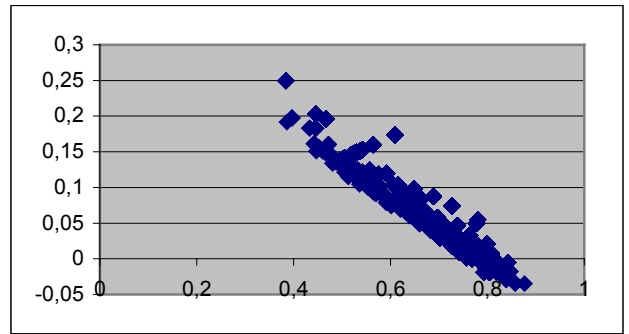
x,y pixel distribution



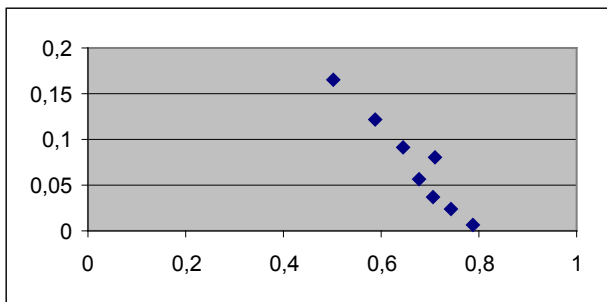
Centroides at the end



clusterised image



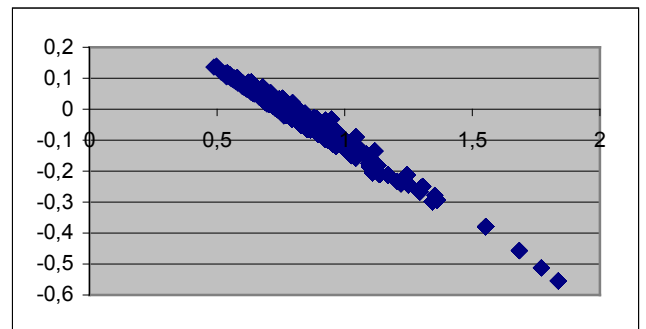
x,y pixel distribution



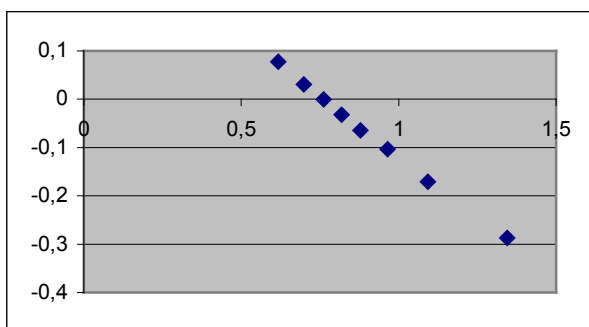
Centroides at the end



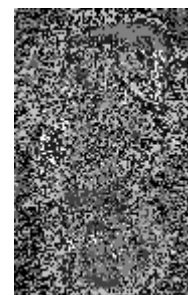
clusterised image



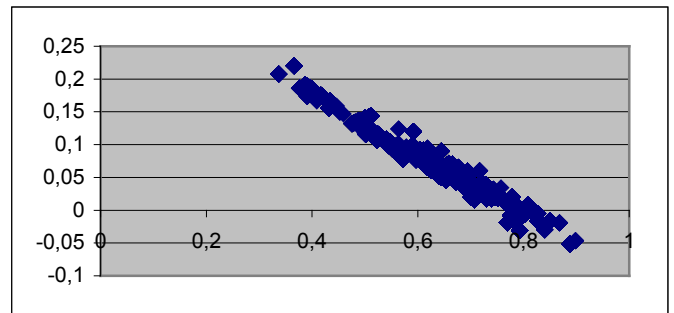
x,y pixel distribution



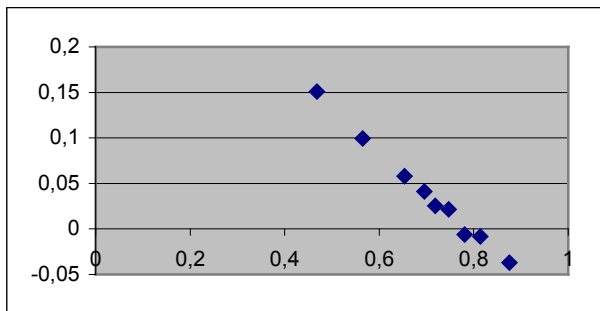
Centroides at the end



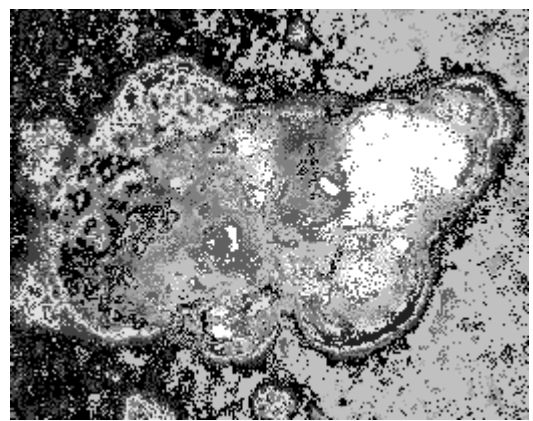
clusterised image



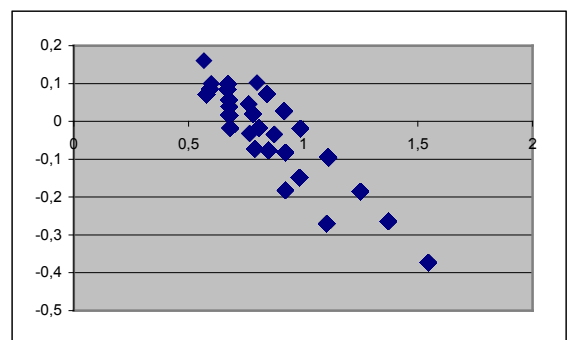
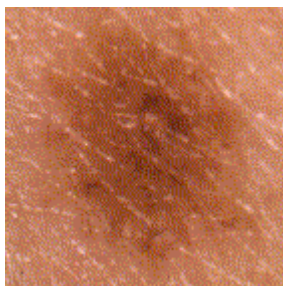
x,y pixel distribution



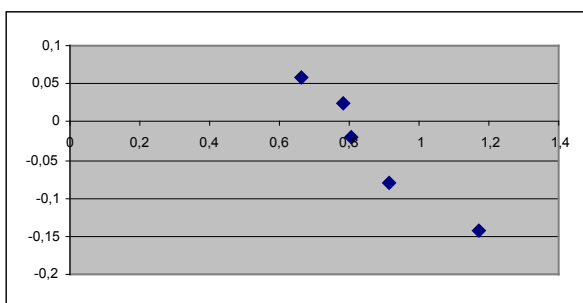
Centroides at the end



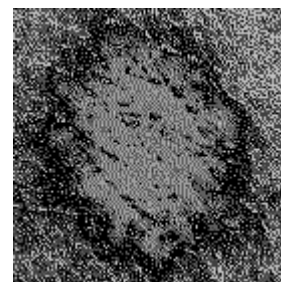
clusterised image



x,y pixel distribution



Centroides at the end



clusterised image

5.2 Discussion

The results obtained in applying a colour and a texture analysis are quite good. Because they depend on external parameters chosen by the user, these results are not the best we can have.

For the k-means clustering, we chose to use nine clusters which is good enough in this type of study. The threshold values used for the split and merge algorithm are also important in the result quality. Here we chose a splitting threshold value equals to 0.9, and a merging threshold value equals to 1.2. But because the merging process is not working, the results can't be compared with other results coming from different algorithms. But I'm still working on the merging procedure in order to provide final results which can be compared.

6 Conclusion and further researches

This project based on a colour and texture analysis of medical images is designed to provide an efficient way to segment skin cancer images in order to support practitioners in their medical diagnosis of skin melanoma. A good discrimination of skin lesions is allowed by the combination of the distributions of colour and texture. This project consists in developing a medical imaging system in the area of skin feature analysis and classification with a view to supporting the early medical diagnosis of skin melanoma.

In this paper, a texture analysis and a colour analysis are developed. The texture analysis is based on the implementation of a Split and Merge algorithm, and the colour analysis uses a k-means clustering approach. Then the two approaches are merged together in order to obtain an image issued from a colour-texture segmentation. Because this last step is not working in the program, that could be interesting to continue it in a further research project.

7 **References**

- [1] – *A new structure for the implementation of unsupervised texture segmentation* – Padmapriya N, Pradeep P.P, Whelan P.F – Dublin City University – ISCC 2002 june 25-26
- [2] – *Computer and Machine Vision Lecture notes* – Whelan P.F – Dublin City University
- [3] – *Segmentation of skin cancer images : A report* – Padmapriya Nammalwar – Dublin City University - 15/01/2004
- [4] – *Measuring border irregularity and shape of cutaneous melanocytic lesions* – Tim Kam Lee – The University of British Columbia – 1980_1983
- [5] – *Towards a computer-aided diagnosis system for pigmented skin lesions* – Philippe Schmid-Saugeon, Joël Guillod, Jean-Philippe Thiran – Signal Processing Institute ITS, Swiss Federal Institute of technology EPFL, Dermatology Department Lausanne university Hospital CHUV – 17/11/2000
- [6] – *Colour segmentation for the analysis of pigmented skin lesions* – Philippe Shmid and S. Fisher – Signal Processing Laboratory, Swiss Federal institute of Technology.
- [7] – *Initial Results of Automated Melanoma Recognition* – H. Ganster, M. Gelautz, A. Pinz – Institute for Computer Graphics, Technical University Graz Muenzgrabenstrasse 11 – M. Blinder, H. Pehamberger - Department of Dermatology, University of Vienna Medical School – M. bammer, J. Krocza – Austrian Research Center Seibersdorf.
- [8] - <http://www-2.cs.cmu.edu/afs/cs/project/cil/ftp/html/vision.html>
- [9] – *A new Algorithm for Border Description of Polarized Light Surface Microscopic Images of Pigmented skin Lesion* – Costantino Grana, Giovanni Pellacani, Rita Cucchiara, Stefania Seidenari – IEEE Transactions on medical imaging, vol. 22 n°8 August 2003.
- [10] – *Irregularity index : A new border irregularity measure for cutaneous melanocytis lesions* – Tim K. Lee, David I. McLean, M. Stella Atkins – Cancer Control Research Program, BC cancer agency, Vancouver, BC, Canada – 18/04/2002


```

{
R[j]=file_in[j];
B[j]=file_in[j];
G[j]=file_in[j];
}

picture=tga_load_image(strcat(file_in,".tga"));
in=alloc_image(picture->width,picture->height);
in->pels=picture->bpels;
pgm_write_image(in,strcat(B,"blue.pgm"));
in->pels=picture->rpels;
pgm_write_image(in,strcat(R,"red.pgm"));
in->pels=picture->gpels;
pgm_write_image(in,strcat(G,"green.pgm"));
printf("\nthe transformation has been done, your three components are %s, %s and %s\n\n",B,R,G);
choice=-1;
break;

```

case 2 :

```

printf("\nYou choose to do a texture analysis : \n\n");
printf("Enter the name of the original image : ");
scanf("%s",&file_in);

```

```

for (j=0;j<100;j++)
{
R[j]=file_in[j];
B[j]=file_in[j];
G[j]=file_in[j];
}

```

//load the image and extract the red, green and blue components and create 3 pictures

```

picture=tga_load_image(strcat(file_in,".tga"));
in=alloc_image(picture->width,picture->height);
in->pels=picture->bpels;
pgm_write_image(in,strcat(B,"blue.pgm"));
in->pels=picture->rpels;
pgm_write_image(in,strcat(R,"red.pgm"));
in->pels=picture->gpels;
pgm_write_image(in,strcat(G,"green.pgm"));

```

//load the new red, green and blue pgm images

```

blue=pgm_load_image(B);
red=pgm_load_image(R);
green=pgm_load_image(G);

```

//get the size of the image

```

w = get_image_width(red);
h = get_image_height(red);

```

//create the format of the mean image

```

meanrgb=alloc_image(w,h);

```

//extract the pixel value of the r,g,b planes

```

pixelred = get_image_pels(red);
pixelgreen = get_image_pels(green);
pixelblue = get_image_pels(blue);
pixelmean = get_image_pels(meanrgb);

```

//this part of the program is used for calculating the mean values of RGB plane

```

for(y = 0; y < h; y++)
for(x = 0; x < w; x++, pixelred++, pixelblue++, pixelgreen++, pixelmean++)
(double)*pixelmean=((double)(((double)*pixelred+(double)*pixelgreen+(double)*pixelblue)/3);

```

```

red=alloc_image(512,512);
taf2(meanrgb,red);

```

```

    choice=-1;
    getchar();
    break;

    case 3 :
    printf("\nYou choose to do a colour analysis : \n\n");
    printf("Enter the name of the original image : ");
    scanf("%s",&file_in);

for (j=0;j<100;j++)
    {
    R[j]=file_in[j];
    B[j]=file_in[j];
    G[j]=file_in[j];
    }

//load the image and extract the red, green and blue components and create 3 pictures
picture=tga_load_image(strcat(file_in,".tga"));
in=alloc_image(picture->width,picture->height);
in->pels=picture->bpels;
pgm_write_image(in,strcat(B,"blue.pgm"));
in->pels=picture->rpels;
pgm_write_image(in,strcat(R,"red.pgm"));
in->pels=picture->gpels;
pgm_write_image(in,strcat(G,"green.pgm"));

//save the size of the tga picture
w=picture->width;
h=picture->height;

//load the new red, green and blue pgm images
blue=pgm_load_image(B);
red=pgm_load_image(R);
green=pgm_load_image(G);
kmclustered=alloc_image(w,h);

//call the kmeanclustering function
kmeanclustering(red, green, blue, w, h, kmclustered);

    choice=-1;
    getchar();
    break;

    case 4:
    printf("\nYou choose to do a texture analysis : \n\n");
    printf("Enter the name of the original image : ");
    scanf("%s",&file_in);

    for (j=0;j<100;j++)
    {
    R[j]=file_in[j];
    B[j]=file_in[j];
    G[j]=file_in[j];
    }

//load the image and extract the red, green and blue components and create 3 pictures
picture=tga_load_image(strcat(file_in,".tga"));
in=alloc_image(picture->width,picture->height);
in->pels=picture->bpels;
pgm_write_image(in,strcat(B,"blue.pgm"));
in->pels=picture->rpels;
pgm_write_image(in,strcat(R,"red.pgm"));
in->pels=picture->gpels;
pgm_write_image(in,strcat(G,"green.pgm"));

//load the new red, green and blue pgm images
blue=pgm_load_image(B);

```

```

red=pgm_load_image(R);
green=pgm_load_image(G);

//get the size of the image
w = get_image_width(red);
h = get_image_height(red);

//create the format of the mean image
meanrgb=alloc_image(w,h);

//extract the pixel value of the r,g,b planes
pixelred = get_image_pels(red);
pixelgreen = get_image_pels(green);
pixelblue = get_image_pels(blue);

kmclustered=alloc_image(w,h);

//call the kmeanclustering function
kmeanclustering(red, green, blue, w, h, kmclustered);

pixelmean = get_image_pels(meanrgb);

//this part of the program is used for calculating the mean values of RGB plane
for(y = 0; y < h; y++)
  for(x = 0; x < w; x++, pixelred++, pixelblue++, pixelgreen++, pixelmean++)
    (double)*pixelmean=((double)(((double)*pixelred+(double)*pixelgreen+(double)*pixelblue)/3);

red=alloc_image(512,512);
taf2(meanrgb,red);

    choice=-1;
    getchar();
    break;

    case 5:
    break;

    default:
    printf("that's net good");
    break;
    }
    }
    while ((choice<1) || (choice>8));
}

```

8.1.2 Skinfeature_lib1.c

8.1.2.1 Texture analysis source code

```

//structural class
class clustStr
{
public:
    image *splited[4];
    image *split;
    //type test the uniformity
    //type=0 non-uniformity
    //type=1 uniformity
    //type=-1 not tested
    int type;
    bool exist;

//constructor
public: clustStr(image *imgin)
    {

```

```

        split=clone_image(imgin);
        type=-1;
        exist=true;
    };

//constructor
public: clustStr(image *imgin,int tmp)
    {
        create_diff_imagebis(imgin);
        type=-1;
        exist=true;
    };

//constructor
public: clustStr()
    {
        type=-1;
        exist=false;
    };

//constructor
public: clustStr(int ttmp)
    {
        type=ttmp;
        exist=false;
    };

//this function is used to split the part of the image in case the class has been
//initialised (exist condition)and the block considered is non uniform
public: int step()
    {
        if(exist)
        {
            if(!isUniform(split))
            {
                splited[0]=alloc_image(split->width/2,split->height/2);
                splited[1]=alloc_image(split->width/2,split->height/2);
                splited[2]=alloc_image(split->width/2,split->height/2);
                splited[3]=alloc_image(split->width/2,split->height/2);
                coup4(split,splited[0],splited[1],splited[2],splited[3]);
                type=0;
            }
            else
                type=1;
        }
        else
            type=-1;
        return type;
    };

private:

//calculates the lbp values
int lbp_block(int in[])
{
    int weight[9];
    int ws=0;
    int i;

    weight[0]=1; weight[1]=2; weight[2]=4; weight[3]=8; weight[4]=0;
    weight[5]=16; weight[6]=32; weight[7]=64; weight[8]=128;

    for (i=0;i<9;i++)
    {
        if (in[i]>=in[4]) ws+=weight[i];
    }
    return ws;
};

```

```
//create the LBP neighbourhood and create the LBP image
```

```
void lbp_img(image *imgin,image *imgout)
{
    int ws=0;
    int x,y,i;
    int buf[9];

    for (x=0;x<imgin->width;x++)
        for (y=0;y<imgin->height;y++)
            {
                for (i=0;i<9;i++) buf[i]=0;
                if ((x>0)&&(x<imgin->width-1))
                    if ((y>0)&&(y<imgin->height-1))
                        {
                            buf[0]=imgin->pels[(y-1)*imgin->width+(x-1)];
                            buf[1]=imgin->pels[(y )*imgin->width+(x-1)];
                            buf[2]=imgin->pels[(y+1)*imgin->width+(x-1)];
                            buf[3]=imgin->pels[(y-1)*imgin->width+(x )];
                            buf[4]=imgin->pels[(y )*imgin->width+(x )];
                            buf[5]=imgin->pels[(y+1)*imgin->width+(x )];
                            buf[6]=imgin->pels[(y-1)*imgin->width+(x+1)];
                            buf[7]=imgin->pels[(y )*imgin->width+(x+1)];
                            buf[8]=imgin->pels[(y+1)*imgin->width+(x+1)];
                        }
                imgout->pels[y*imgin->width+x]=lbp_block(buf);
            }
};
```

```
//this function split the image into four images in case of non uniformity
```

```
void coup4(image *imgin,image *out1,image *out2,image *out3,image *out4)
{
    int x,y;

    for(x=0;x<imgin->width/2;x++)
        for(y=0;y<imgin->height/2;y++)
            {
                out1->pels[(y )*imgin->width/2+(x )]=imgin->pels[(y )*imgin->width+(x )];
            }

    for(x=imgin->width/2;x<imgin->width;x++)
        for(y=0;y<imgin->height/2;y++)
            {
                out2->pels[(y )*imgin->width/2+(x -imgin->width/2 )]=imgin->pels[(y )*imgin->width+(x )];
            }

    for(x=0;x<imgin->width/2;x++)
        for(y=imgin->height/2;y<imgin->height;y++)
            {
                out3->pels[(y-imgin->height/2 )*imgin->width/2+(x )]=imgin->pels[(y )*imgin->width+(x )];
            }

    for(x=imgin->width/2;x<imgin->width;x++)
        for(y=imgin->height/2;y<imgin->height;y++)
            {
                out4->pels[(y -imgin->height/2 )*imgin->width/2+(x -imgin->width/2 )]=imgin->pels[(y )*imgin->
width+(x)];
            }
};
```

```
//calculation of histograms used in the G calculation
```

```
void histog(image *in, double histo[])
{
    int i,pixcnt;

    for(i=0;i<256;i++)
```

```

    {
        histo[i]=0;
    }

    pixcnt=in->height*in->width;

    for(i=0;i<pixcnt;i++)
    {
        histo[(int)in->pels[i]]+=1;
    }

    for(i=0;i<256;i++)
    {
        histo[i]/=pixcnt;
    }
};

//calculation of the G value between two adjacent regions
double calcG(double histom[],double histos[])
{
    double tmp=0;
    int i;
    for(i=0;i<255;i++)
    {
        if((histom[i]!=0)&&(histos[i]!=0)) tmp+=histos[i]*log(histos[i]/histom[i]);
    }
    tmp*=2;
    return tmp;
};

```

//This function implements the global splitting algorithm and deduce the uniformity of the textures within the image

```

bool isUniform(image *in)
{
    int i,j;
    double Hist1[256],Hist2[256],Hist3[256],Hist4[256],G[6],gmin,gmax,R;
    image *o1,*o2,*o3,*o4;

    o1=alloc_image(in->width/2,in->height/2);
    o2=alloc_image(in->width/2,in->height/2);
    o3=alloc_image(in->width/2,in->height/2);
    o4=alloc_image(in->width/2,in->height/2);

    coup4(in,o1,o2,o3,o4);

    histog(o1,Hist1);
    histog(o2,Hist2);
    histog(o3,Hist3);
    histog(o4,Hist4);

    G[0]=calcG(Hist1,Hist2);
    G[1]=calcG(Hist1,Hist3);
    G[2]=calcG(Hist1,Hist4);
    G[3]=calcG(Hist2,Hist3);
    G[4]=calcG(Hist2,Hist4);
    G[5]=calcG(Hist3,Hist4);

    gmin=G[0];
    gmax=G[0];

    for (i=0;i<6;i++)
    {
        if (G[i]>gmax) gmax=G[i];
        if (G[i]<gmin) gmin=G[i];
    }
}

```



```

//printf("G1:%f G2:%f G3:%f gmin:%f gmax:%f\n",G[0],G[1],G[2],gmin,gmax);

if ((gmax>=2*gmin)&&(((gmax*gmax)>0)||((gmin*gmin)>0))) // <=> R>1.2 ||(gmax==gmin)
    return false;
else
    return true;
};

void create_diff_imagebis(image *imgin)
{
    int tmpx,tmpy,x,y;

    split=alloc_image(512,512);
    image *imgtmp=alloc_image(imgin->width,imgin->height);
    lbp_img(imgin,imgtmp);

    tmpx=(512-imgtmp->width)/2;
    tmpy=(512-imgtmp->height)/2;

    for (x=0;x<imgtmp->width;x++)
        for (y=0;y<imgtmp->height;y++)
            {
                split->pels[(y+tmpy)*512+tmpx+x] = imgtmp->pels[y*imgtmp->width+x];
            }
};
};

void taf(image **in, clustStr Cout[])
{
    clustStr *tmp1,*tmp2,*tmp3,*tmp4;
    {
        tmp1=new clustStr(in[0]);
        tmp2=new clustStr(in[1]);
        tmp3=new clustStr(in[2]);
        tmp4=new clustStr(in[3]);
        Cout[0]=*tmp1;
        Cout[1]=*tmp2;
        Cout[2]=*tmp3;
        Cout[3]=*tmp4;
    }
}

int rety(int i,int j,int k,int l,int m)
{
    int tmpy=0;

    if (i>=2) tmpy+=256;
    if (j>=2) tmpy+=128;
    if (k>=2) tmpy+=64;
    if (l>=2) tmpy+=32;
    if (m>=2) tmpy+=16;

    if (tmpy>0) return tmpy-1; else return tmpy;
}

int retx(int i,int j,int k,int l,int m)
{
    int tmpx=0;
    if ((i==1)||i==3) tmpx+=256;
    if ((j==1)||j==3) tmpx+=128;
    if ((k==1)||k==3) tmpx+=64;
    if ((l==1)||l==3) tmpx+=32;
    if ((m==1)||m==3) tmpx+=16;
    if (tmpx>0) return tmpx-1; else return tmpx;
}

```

```
//function used for tracing white boundaries between regions in case of non uniformity
```

```
void draw_white(image *in,int pos,int startx,int starty,int leng)
{
    int x,y,tmpx,tmpy;
    if (pos==0) {tmpx=0; tmpy=0;}
    if (pos==1) {tmpx=leng; tmpy=0;}
    if (pos==2) {tmpx=0; tmpy=leng;}
    if (pos==3) {tmpx=leng; tmpy=leng;}

    for (x=startx+tmpx;x<startx+tmpx+leng;x++)
    {
        in->pels[(starty+tmpy)*in->width+x]=255;
        in->pels[(starty+tmpy+leng-1)*in->width+x]=255;
    }

    for (y=starty+tmpy;y<starty+tmpy+leng;y++)
    {
        in->pels[y*in->width+startx+tmpx]=255;
        in->pels[y*in->width+startx+tmpx+leng-1]=255;
    }
}
```

```
//function used to realised the iterative splitting until a stopping condition is reached
```

```
void taf2(image *in,image *out)
{
    clustStr *ccc=new clustStr(in,0);
    out=clone_image(ccc->split);
    pgm_write_image(ccc->split,"LBP.pgm");

    int i,j,k,l,m,n;

    //stocks the different images issued from the splitting procedure
    clustStr c1[4];
    clustStr c2[4][4];
    clustStr c3[4][4][4];
    clustStr c4[4][4][4][4];
    clustStr c5[4][4][4][4][4];

    if(ccc->step()==0) taf(ccc->splited,c1);

    for(j=0;j<4;j++)
    {
        if((c1[j].step()==0))
        {
            {
                taf(c1[j].splited,c2[j]);
            }
        }
    }

    for(k=0;k<4;k++)
    {
        for(j=0;j<4;j++)
        {
            if((c2[k][j].step()==0))
            {
                {
                    taf(c2[k][j].splited,c3[k][j]);
                }
            }
        }
    }
}
```

```

for(l=0;l<4;l++)
{
    for(k=0;k<4;k++)
    {
        for(j=0;j<4;j++)
        {
            if(c3[l][k][j].step()==0)
            {
                {
                    taf(c3[l][k][j].splited,c4[l][k][j]);
                }
            }
        }
    }
}

for(m=0;m<4;m++)
{
    for(l=0;l<4;l++)
    {
        for(k=0;k<4;k++)
        {
            for(j=0;j<4;j++)
            {
                if(c4[m][l][k][j].step()==0)
                {
                    {
                        taf(c4[m][l][k][j].splited,c5[m][l][k][j]);
                    }
                }
            }
        }
    }
}

for(i=0;i<4;i++)
{
    if (c1[i].type==0)
    {
        draw_white(out,i,0,0,256);
    }
}

for(j=0;j<4;j++)
    for(i=0;i<4;i++)
    {
        if (c2[j][i].type==0)
        {
            draw_white(out,i,retx(j,-1,-1,-1),rety(j,-1,-1,-1),128);
        }
    }

for(k=0;k<4;k++)
    for(j=0;j<4;j++)
        for(i=0;i<4;i++)
            {
                if (c3[k][j][i].type==0) draw_white(out,i,retx(k,j,-1,-1),rety(k,j,-1,-1),64);
            }

for(l=0;l<4;l++)
    for(k=0;k<4;k++)
        for(j=0;j<4;j++)
            for(i=0;i<4;i++)
                {
                    if(c4[l][k][j][i].type==0) draw_white(out,i,retx(l,k,j,-1,-1),rety(l,k,j,-1,-1),32);
                }

```

```

    for(m=0;m<4;m++)
        for(l=0;l<4;l++)
            for(k=0;k<4;k++)
                for(j=0;j<4;j++)
                    for(i=0;i<4;i++)
                        {
                            if (c5[m][l][k][j][i].step()==0)
                                draw_white(out,i,retx(m,l,k,j,-1),rety(m,l,k,j,-1),16);
                        }

    pgm_write_image(out,"SPLIT.pgm");

}

```

8.1.2.2 K-mean clustering source code

```

void kmeanclustering(image *r, image *g, image *b, int w, int h, image *kmean)
{

int i=0,k=0,j=0,l=0,m=0,n=0, Klust=0,test=0,size=w*h;
double cmp1=0,cmp2=0,cmp3=0,cmp4=0,cmp5=0,cmp6=0,cmp7=0,cmp8=0,cmp9=0;
double cent1X=0,cent2X=0,cent3X=0,cent4X=0,cent5X=0,cent6X=0,cent7X=0,cent8X=0,cent9X=0;
double cent1Y=0,cent2Y=0,cent3Y=0,cent4Y=0,cent5Y=0,cent6Y=0,cent7Y=0,cent8Y=0,cent9Y=0;
double xyzval[3],rgbval[3], Xk[9], Yk[9], dist[9], min=0, Ykpre[9], Xkpre[9];
image *x, *y, *z, *klust, *preklust;
pel *pixelred, *pixelblue, *pixelgreen;
FILE *fichier, *fichier2;

//initialisations
x=clone_image(r);
y=clone_image(r);
z=clone_image(r);
klust=clone_image(r);
preklust=clone_image(r);

for(i=0;i<size;i++)
{
preklust->pels[i]=0;
klust->pels[i]=0;
}

for(n=0;n<9;n++)
{
Xk[n]=0;
Yk[n]=0;
Xkpre[n]=0;
Ykpre[n]=0;
dist[n]=0;
}

printf("%i * %i\n\n",h,w);

//get the pixels of the images
pixelred = get_image_pels(r);
pixelgreen = get_image_pels(g);
pixelblue = get_image_pels(b);

//open a csv file which will stock the x,y,z values
fichier=fopen("xyz.csv","w");

for (i=0;i<size;i++,pixelred++,pixelgreen++,pixelblue++)
{
    rgbval[0]=(double)*pixelred;
    rgbval[1]=(double)*pixelgreen;
    rgbval[2]=(double)*pixelblue;
}
}

```

```

[ X ] [ 2.36 -0.515 0.0052 ] [ R ]
[ Y ] = [ -0.89 1.42 -0.014 ] * [ G ]
[ Z ] [ -0.46 0.88 1.009 ] [ B ] */

xyzval[0]=2.36*rgbval[0] - 0.515*rgbval[1] + 0.0052*rgbval[2];
xyzval[1]=-0.89*rgbval[0] + 1.42*rgbval[1] - 0.014*rgbval[2];
xyzval[2]=-0.46*rgbval[0] + 0.88*rgbval[1] + 1.009*rgbval[2];

//k=X+Y+Z
k=(xyzval[0]+xyzval[1]+xyzval[2]);

if(k!=0)
{
x->pels[i]=xyzval[0]/k;
y->pels[i]=xyzval[1]/k;
z->pels[i]=xyzval[2]/k;
}

else
{
x->pels[i]=x->pels[i-1];
y->pels[i]=y->pels[i-1];
z->pels[i]=z->pels[i-1];
}

//write the values in a csv file
fprintf(fichier,"%f;%f;%f\n",x->pels[i],y->pels[i],z->pels[i]);
}

//close the csv file
fclose(fichier);

//initialisation of the cluster centroïdes
Xk[0]=x->pels[(int)((w*h/4)-3*w/4)];
Xk[1]=x->pels[(int)((w*h/4)-w/4)];
Xk[2]=x->pels[(int)((w*h/3)-2*w/3)];
Xk[3]=x->pels[(int)((w*h/3)-w/3)];
Xk[4]=x->pels[(int)((w*h/2)-w/2)];
Xk[5]=x->pels[(int)((w*2*h/3)-2*w/3)];
Xk[6]=x->pels[(int)((w*2*h/3)-w/3)];
Xk[7]=x->pels[(int)((w*3*h/4)-3*w/4)];
Xk[8]=x->pels[(int)((w*3*h/4)-w/4)];

Yk[0]=y->pels[(int)((w*h/4)-3*w/4)];
Yk[1]=y->pels[(int)((w*h/4)-w/4)];
Yk[2]=y->pels[(int)((w*h/3)-2*w/3)];
Yk[3]=y->pels[(int)((w*h/3)-w/3)];
Yk[4]=y->pels[(int)((w*h/2)-w/2)];
Yk[5]=y->pels[(int)((w*2*h/3)-2*w/3)];
Yk[6]=y->pels[(int)((w*2*h/3)-w/3)];
Yk[7]=y->pels[(int)((w*3*h/4)-3*w/4)];
Yk[8]=y->pels[(int)((w*3*h/4)-w/4)];

for(l=0;l<9;l++)
printf("%f _ %f\n",Xk[l],Yk[l]);

do
{
cent1X=0;cent1Y=0;cent2X=0;cent2Y=0;cent3X=0;cent3Y=0;cent4X=0;cent4Y=0;cent5X=0;
cent5Y=0;cent6X=0;cent6Y=0;cent7X=0;cent7Y=0;cent8X=0;cent8Y=0;cent9X=0;cent9Y=0;

cmp1=0;cmp2=0;cmp3=0;cmp4=0;cmp5=0;cmp6=0;cmp7=0;cmp8=0;cmp9=0;test=0;

//find the cluster corresponding to each pixel
for(j=0;j<size;j++)
{

```

```

//calculation of the 9 distances between the pixel and the 9 clusters
for(l=0;l<9;l++)
{
dist[l]=sqrt(((Xk[l]-x->pels[j])*(Xk[l]-x->pels[j]))+((Yk[l]-y->pels[j])*(Yk[l]-y->pels[j]]));
}

Klust=0;
min=dist[0];

    for(i=0;i<9;i++)
    {
        if(dist[i]<=min)
        {
            min=dist[i];
            Klust=i+1;
        }
    }

    klust->pels[j]=Klust;
}
//modification of the centroïdes
for(l=0;l<size;l++)
{
if(klust->pels[l]==1) {cent1X=cent1X+x->pels[l];cent1Y=cent1Y+y->pels[l]; cmp1=cmp1+1;}
if(klust->pels[l]==2) {cent2X=cent2X+x->pels[l];cent2Y=cent2Y+y->pels[l]; cmp2=cmp2+1;}
if(klust->pels[l]==3) {cent3X=cent3X+x->pels[l];cent3Y=cent3Y+y->pels[l]; cmp3=cmp3+1;}
if(klust->pels[l]==4) {cent4X=cent4X+x->pels[l];cent4Y=cent4Y+y->pels[l]; cmp4=cmp4+1;}
if(klust->pels[l]==5) {cent5X=cent5X+x->pels[l];cent5Y=cent5Y+y->pels[l]; cmp5=cmp5+1;}
if(klust->pels[l]==6) {cent6X=cent6X+x->pels[l];cent6Y=cent6Y+y->pels[l]; cmp6=cmp6+1;}
if(klust->pels[l]==7) {cent7X=cent7X+x->pels[l];cent7Y=cent7Y+y->pels[l]; cmp7=cmp7+1;}
if(klust->pels[l]==8) {cent8X=cent8X+x->pels[l];cent8Y=cent8Y+y->pels[l]; cmp8=cmp8+1;}
if(klust->pels[l]==9) {cent9X=cent9X+x->pels[l];cent9Y=cent9Y+y->pels[l]; cmp9=cmp9+1;}
}

//calculation of the new clusters centroides
//in case the cmpi is null i.e two clusters corresponds to the same coordinates

if(cmp1!=0) {Xk[0]=cent1X/cmp1; Yk[0]=cent1Y/cmp1;}
else {Xk[0]=40; Yk[0]=40;}

if(cmp2!=0) {Xk[1]=cent2X/cmp2; Yk[1]=cent2Y/cmp2;}
else {Xk[1]=40; Yk[1]=40;}

if(cmp3!=0) {Xk[2]=cent3X/cmp3; Yk[2]=cent3Y/cmp3;}
else {Xk[2]=40; Yk[2]=40;}

if(cmp4!=0) {Xk[3]=cent4X/cmp4; Yk[3]=cent4Y/cmp4;}
else {Xk[3]=40; Yk[3]=40;}

if(cmp5!=0) {Xk[4]=cent5X/cmp5; Yk[4]=cent5Y/cmp5;}
else {Xk[4]=40; Yk[4]=40;}

if(cmp6!=0) {Xk[5]=cent6X/cmp6; Yk[5]=cent6Y/cmp6;}
else {Xk[5]=40; Yk[5]=40;}

if(cmp7!=0) {Xk[6]=cent7X/cmp7; Yk[6]=cent7Y/cmp7;}
else {Xk[6]=40; Yk[6]=40;}

if(cmp8!=0) {Xk[7]=cent8X/cmp8; Yk[7]=cent8Y/cmp8;}
else {Xk[7]=40; Yk[7]=40;}

if(cmp9!=0) {Xk[8]=cent9X/cmp9; Yk[8]=cent9Y/cmp9;}
else {Xk[8]=40; Yk[8]=40;}

//if there is no change between two iterations
if(Xkpre[0]==Xk[0] && Ykpre[0]==Yk[0] && Xkpre[1]==Xk[1] && Ykpre[1]==Yk[1] && Xkpre[2]==Xk[2] &&
Ykpre[2]==Yk[2] && Xkpre[3]==Xk[3] && Ykpre[3]==Yk[3] && Xkpre[4]==Xk[4] && Ykpre[4]==Yk[4] &&

```

```

Xkpre[5]==Xk[5] && Ykpre[5]==Yk[5] && Xkpre[6]==Xk[6] && Ykpre[6]==Yk[6] && Xkpre[7]==Xk[7] &&
Ykpre[7]==Yk[7] && Xkpre[8]==Xk[8] && Ykpre[8]==Yk[8])
    test=1;

        for(m=0;m<9;m++)
        {
            Xkpre[m]=Xk[m];
            Ykpre[m]=Yk[m];
        }

    while(test!=1);

fichier2=fopen("centroides.csv","w");
for(l=0;l<9;l++)
{
    fprintf(fichier2,"%f;%f;%f\n",Xk[l],Yk[l]);
    printf("\nX%i %f Y%i %f",l+1,Xk[l],l+1,Yk[l]);
}
fclose(fichier2);

printf("\n\n");

//define the corresponding color for each cluster
for(i=0;i<size;i++)
{
    if(klust->pels[i]==1)kmean->pels[i]=255;
    if(klust->pels[i]==2)kmean->pels[i]=224;
    if(klust->pels[i]==3)kmean->pels[i]=192;
    if(klust->pels[i]==4)kmean->pels[i]=160;
    if(klust->pels[i]==5)kmean->pels[i]=128;
    if(klust->pels[i]==6)kmean->pels[i]=96;
    if(klust->pels[i]==7)kmean->pels[i]=64;
    if(klust->pels[i]==8)kmean->pels[i]=32;
    if(klust->pels[i]==9)kmean->pels[i]=0;
}

//create the k-mean clusterised image
pgm_write_image(kmean,"K-MEAN.pgm");
}

```