



P- 2165



**DETECTION AND REMOVAL OF CRACKS IN
DIGITIZED PAINTING**

A PROJECT REPORT

Submitted by

BHARATHI KANNAN.K

71204205004

SELVA KUMAR.K.R

71204205047

RAMALINGAM.S

71204205305

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600025

APRIL 2008

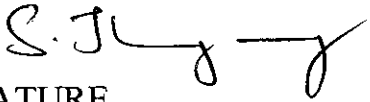


BONAFIDE CERTIFICATE

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this report “**DETECTION AND REMOVAL OF CRACKS IN DIGITIZED PAINTING**” is the bonafide work of “**Mr.BHARATHI KANNAN.K (71204205004), Mr. SELVA KUMAR.K.R (71204205047), Mr.RAMALINGAM.S (71204205305)**”who carried out the project work under my supervision.



SIGNATURE

Dr. S. Thangasamy

HEAD OF THE DEPARTMENT

Computer science and Engineering

Department of Computer Science
& Engineering,
Kumaraguru. College of Technology,
Coimbatore - 641006



SIGNATURE

Mr.E.A.Vimal

SUPERVISOR, Lecturer

Information Technology

Department of Information Technology,
Kumaraguru College of Technology,
Coimbatore - 641006

Submitted for viva-voice examination held on 23.04.08



Internal Examiner



External Examiner

DECLARATION

DECLARATION

We hereby declare that the project entitled “**DETECTION AND REMOVAL OF CRACKS IN DIGITIZED PAINTING**” is done by us and to the best of our knowledge a similar work has not been submitted to the **Anna University** or any other Institution, for fulfillment of the requirement of the course study.

This report is submitted on the partial fulfillment of the requirement for all awards of the **Degree of Bachelor of Information Technology of Anna University, Chennai.**

Place: Coimbatore

Date:


Bharathi Kannan.K


Selva Kumar.K.R


Mr.E.A.Vimal


Ramalingam.S

ACKNOWLEDGEMENT

ACKNOWLEDGEMENTS

We are extremely grateful to the **Principal, Dr. Joseph V. Thanikkal M.E., Ph.D., PDF. CEPIT.** For having given us a golden opportunity to embark on this project.

We would like to thank our project coordinator, **Mr. K. R. Baskaran, Asst Professor,** Department of Information Technology for his support during the course of our project.

We have immense pleasure in expressing our heartfelt thanks to our guide, **Mr.E.A.Vimal, Lecturer,** Department of Information Technology for his constant advice and support throughout this project.

We would like to express our sincere thanks to all the members of the faculty of **Department of Information Technology** for their support.

We thank many of our patient fellow students for listening about the problems we were tackling and helping us understand them more clearly by asking the right questions.

We thank all those who have been involved directly or indirectly in our project.

ABSTRACT

ABSTRACT

An integrated methodology for the detection and removal of cracks on digitized paintings is implemented in this project. The cracks are detected by threshold the output of the morphological top-hat transformation. Afterwards the thin dark brush strokes which have been misidentified as cracks are removed using either a median radial basis function neural network on hue and saturation data or a semi-automatic procedure based on region growing. Finally, crack filling using order statistics filters or controlled anisotropic diffusion is performed. The methodology has been shown to perform very well on digitized paintings suffering from cracks.

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ACKNOWLEDGEMENT	iii
	ABSTRACT	iv
1.	INTRODUCTION	
	1.1 GENERAL	1
	1.2 PROBLEM DEFINITION	4
2.	LITERATURE REVIEW	
	2.1 FEASIBILITY STUDY	5
	2.1.1 EXISTING SYSTEM	5
	2.1.2 PROPOSED SYSTEM	5
	2.2 HARDWARE REQUIREMENTS	7
	2.3 SOFTWARE REQUIREMENTS	7
	2.4 SOFTWARE OVERVIEW	8
3.	DETAILS OF THE METHODOLOGY EMPLOYED	
	3.1 DETECTION OF CRACKS	10
	3.2 SEPARATION OF THE BRUSH STROKES FROM THE CRACKS	13
	3.3 CRACK FILLING METHODS	16

4.	PERFORMANCE EVALUATION	
	4.1 Software Testing	17
	4.2 Functional Testing	18
	4.3 System Testing	18
5.	CONCLUSION	19
6.	FUTURE ENHANCEMENTS	21
7.	APPENDIX	
	7.1 SOURCE CODE	22
	7.2 SCREEN SHOTS	36
8.	REFERENCES	44

INTRODUCTION

1. INTRODUCTION

GENERAL

Many paintings especially old ones suffer from breaks in the substrate, the paint or the varnish. These patterns are usually called cracks or craquelure and can be caused by aging, drying, and mechanical factors. Age cracks can result from non uniform contraction in the canvas or wood-panel support of the painting which stresses the layers of the painting. Drying cracks are usually caused by the evaporation of volatile paint components and the consequent shrinkage of the paint. Finally, mechanical cracks result from painting deformations due to external causes, e.g., vibrations and impacts.

The appearance of cracks on paintings deteriorates the perceived image quality. However, one can use digital image processing techniques to detect and eliminate the cracks on digitized paintings. Such a “virtual” restoration can provide clues to art historians, museum curators and the general public on how the painting would look like in its initial state, i.e., without the cracks. Furthermore, it can be used as a nondestructive tool for the planning of the actual restoration. A system that is capable of tracking and interpolating cracks. The user should manually select a point on each crack to be restored. A method for the detection of cracks using multioriented Gabor filters. Crack detection and removal bears certain

similarities with methods proposed for the detection and removal of scratches and other artifacts from motion picture films. However, such methods rely on information obtained over several adjacent frames for both artifact detection and filling. Thus they are not directly applicable in the case of painting cracks. Other research areas that are closely related to crack removal include image inpainting which deals with the reconstruction of missing or damaged image areas by filling in information from the neighboring areas and disclosing, i.e., recovery of object parts that are hidden behind other objects within an image.

Methods developed in these areas assume that the regions where information has to be filled in are known. Different approaches for interpolating information in structured and textured image areas have been developed. The former are usually based on partial differential equations (PDEs) and on the calculus of variations whereas the latter rely on texture synthesis principles. A technique that decomposes the image to textured and structured areas and uses appropriate interpolation techniques depending on the area where the missing information. The results obtained by these techniques are very good. A methodology for the restoration of cracks on digitized paintings which adapts and integrates a number of image processing and analysis tool is proposed in this paper.

The methodology is an extension of the crack removal framework. The technique consists of the following stages:

- crack detection;
- Separation of the thin dark brush strokes, which have been misidentified as cracks;
- crack filling (interpolation).

A certain degree of user interaction most notably in the crack-detection stage is required for optimal results. User interaction is rather unavoidable since the large variations observed in the typology of cracks would lead any fully automatic algorithm to failure. However, all processing steps can be executed in real time and thus the user can instantly observe the effect of parameter tuning on the image under study and select in an intuitive way the values that achieve the optimal visual result. Needless to say, only subjective optimality criteria can be used in this case since no ground truth data are available. The opinion of restoration experts that inspected the virtually restored images were very positive. Two methods for the separation of the brush strokes which have been falsely identified as cracks. Methods for filling the cracks with image content from neighboring pixels.

PROBLEM DEFINITION

Many paintings especially old ones suffer from breaks in the substrate, the paint or the varnish. These patterns are usually called cracks or craquelure and can be caused by aging, drying, and mechanical factors. Age cracks can result from non uniform contraction in the canvas or wood-panel support of the painting which stresses the layers of the painting. Drying cracks are usually caused by the evaporation of volatile paint components and the consequent shrinkage of the paint. Finally, mechanical cracks result from painting deformations due to external causes, e.g., vibrations and impacts. The appearance of cracks on paintings deteriorates the perceived image quality.

- Problem for segmenting the lines and strokes of the under drawing.
- Cracks not only disturb the appearance of a painting.
- One major goal of the project is to identify the drawing tools used by the painter to create the under drawing from the appearance of the strokes in the infrared reflectogram.

LITERATURE REVIEW

2. LITERATURE REVIEW

2.1 FEASIBILITY STUDY

2.1.1 CURRENT STATUS OF THE PROBLEM

The existing methods for processing digital images are there which actually deal with enhancing the image picture quality, brightness, color etc. These factors can be degraded due to aging process. Such an image processing technique algorithm concentrates on improving those factor alone. There are not designed to analysis and improve in the cracks region. The cracks removal has to be rectified in the different manner. The principle applied to improve image color, brightness and other characteristic can not be used for crack detection and removal. This project concentrates on the digital image processing algorithm that deals only with crack detection and removal.

2.1.2 PROPOSED SYSTEM AND ADVANTAGES

The proposed system deals with digital image processing technique that detects and removes the cracks in the images. A system that is capable of tracking and interpolating cracks. The user should manually select a point on each crack to be restored. A method for the detection of cracks using multioriented Gabor filters. Crack detection and removal bears certain similarities with methods proposed for the detection and removal of scratches and other artifacts from motion picture films.

frames for both artifact detection and filling. Other research areas that are closely related to crack removal include image inpainting which deals with the reconstruction of missing or damaged image areas by filling in information from the neighboring areas and discarding. Methods developed in these areas assume that the regions where information has to be filled in are known. Different approaches for interpolating information in structured and textured image areas have been developed. The former are usually based on partial differential equations (PDEs) and on the calculus of variations whereas the latter rely on texture synthesis principles. A technique that decomposes the image to textured and structured areas and uses appropriate interpolation techniques depending on the area where the missing information is. A methodology for the restoration of cracks on digitized paintings which adapts and integrates a number of image processing and analysis tools is proposed in this paper. The technique consists of the following stages:

- There should be some method through which crack area in the digital image can be detected;
- Separation of the thin dark brush strokes which have been misidentified as cracks;
- Crack filling (interpolation).

A certain degree of user interaction most notably in the

2.2 HARDWARE REQUIREMENTS :(Minimum Requirements)

Processor	:	Intel Processor IV
RAM	:	256 MB
Hard disk	:	40 GB
CD drive	:	40 x Samsung
Floppy drive	:	1.44 MB
Monitor	:	15'' Samtron color
Keyboard	:	108 mercury keyboard
Mouse	:	Logitech mouse

2.3 SOFTWARE REQUIREMENTS:

Operating System – Windows XP/2000

Language used – VB .Net

2.4 SOFTWARE OVERVIEW

Microsoft Visual Basic .Net used as front end tool.

The reason for selecting Visual Basic dot Net as front end tool as follows:

- Visual Basic .Net has flexibility, allowing one or more language to interoperate to provide the solution. This Cross Language Compatibility allows to do project at faster rate.
- Visual Basic .Net has Common Language Runtime, that allows the entire component to converge into one intermediate format and then can interact.
- Visual Basic .Net has provided excellent security when your application is executed in the system.
- Visual Basic .Net has flexibility, allowing us to configure the working environment to best suit our individual style. We can choose between a single and multiple document interfaces, and we can adjust the size and positioning of the various IDE elements.
- Visual Basic .Net has Intelligence feature that make the coding easy and also dynamic help provides very less coding time.
- The working environment in Visual Basic .Net is often referred to as IDE because it integrates many different functions such as design, editing, compiling and debugging within a common

environment. In most traditional development tools each of separate programs with its own interface.

- The Visual Basic .Net language is quite powerful – if we can imagine a programming task and accomplished using Visual Basic .Net.
- After creating a Visual Basic .Net application, if we want to distribute it to others we can freely distribute any application to anyone who uses windows. We can distribute our applications on disk, CDs, across networks, over an intranet or the internet.
- Toolbars provide quick access to commonly used commands in the programming environment. We click a button on the toolbar once to carry out the action represented by that button. Additional toolbars for editing, form design and debugging can be toggled on or off from the toolbars command on view menu.
- Many parts of Visual Basic are context sensitive. Context sensitive means we can get help on these parts directly without having to go through the help menu.
- Visual Basic interprets our code as we enter it, catching and highlighting most syntax or spelling errors on the fly. It's almost like having an expert watching over our shoulder as we enter our code.

*DETAILS OF THE METHODOLOGY
EMPLOYED*

3. DETAILS OF THE METHODOLOGY EMPLOYED

3.1 DETECTION OF CRACKS

Cracks usually have low luminance and can be considered as local intensity minima with rather elongated structural characteristics. Therefore, a crack detector can be applied on the luminance component of an image and should be able to identify such minima. A crack-detection procedure top-hat transform is proposed in this paper. The top-hat transform is a grayscale morphological filter defined as follows:

$$Y(x) = f(x) - f_{nB}(x) \quad (1)$$

Where $f_{nB}(x)$ is the opening of the function $f(x)$ with the structuring set, defined as

$$nB = B \Phi B \Phi \dots \Phi B \text{ (n times)} \quad (2)$$

In the previous equation, Φ denotes the dilation operation. A square or a circle can be used as structuring element B . The final structuring set nB is evaluated only once using in the opening operation of (1). The opening f_{nB} of a function is a low-pass nonlinear filters that erases all peaks (local maxima) in which the structuring element nB cannot fit. Thus, the image $f - f_{nB}$ contains only those peaks and no background at all. Since cracks are local minima rather than local maxima, the top-hat transform should be applied on the negated luminance image. Alternatively, one can detect cracks by performing closing on the original image $f(x)$ with the structuring

set B and then subtracting from the result of closing $f_{nB}(x)$

$$Y(x) = f^{nB}(x) \text{ -- } f(x) \quad \text{----- (3)}$$

It can be easily shown that the result of (3) is identical to that of applying (1) on the negated image. Use of (3) does not require negation of $f(x)$ which grants it a small but not negligible computational advantage over (1).

In situations where the crack-like artifacts are of high luminance, as in the case of scratches on photographs, negation of the luminance component prior to the crack detection is not required, i.e., the crack detection procedure can be applied directly on the luminance image. The user can control the result of the crack-detection procedure by choosing appropriate values for the following parameters:

- The type of the structuring element B,
- The size of the structuring element B and the number n of dilations.

These parameters affect the size of the “final” structuring element nB and must be chosen according to the thickness of the cracks to be detected. It should be noted that these parameters are not very critical for the algorithm performance due to the threshold operation and also due to the existence of the brush-stroke/crack separation procedure, which is able to remove crack-like brush strokes that have been erroneously identified as cracks. The fact that all the results presented in this paper have been

indication that the above statement is indeed true. These parameters were the following:

- Structuring element type: square;
- Structuring element size: 3 X 3;
- Number n of dilations in (2) : 2.

The top-hat transform generates a grayscale output image $t(k, l)$ where pixels with a large grey value are potential crack or crack-like elements. Therefore, a threshold operation on $t(k, l)$ is required to separate cracks from the rest of the image. The threshold can be chosen by a trial and error procedure, i.e., by inspecting its effect on the resulting crack map. The low computational complexity of the threshold operation enables the user to view the crack-detection results in real time while changing the threshold value, e.g., by moving a slider. This fact makes interactive threshold selection very effective and intuitive. Alternatively, threshold selection can be done by inspecting the histogram of $t(k, l)$ for a lobe close to the maximum intensity value and assigning it a value that separates this lobe from the rest of the intensities. The result of the threshold is a binary image marking the possible crack locations. Instead of this global thresholding technique more complex threshold schemes, which use a spatially varying threshold can be used. Obviously, as the threshold value increases the number of image pixels that are identified as cracks decreases. Thus, certain cracks, especially in

can remain undetected. In principle, it is more preferable to select the threshold so that some cracks remain undetected than to choose a threshold that would result in the detection of all cracks but will also falsely identify as cracks and subsequently modify other image structures. The threshold (binary) output of the top-hat transform on the luminance component of an image containing cracks.

3.2 SEPARATION OF THE BRUSH STROKES FROM THE CRACKS

In some paintings, certain areas exist where brush strokes have almost the same thickness and luminance features as cracks. The hair of a person in a portrait could be such an area. Therefore, the top-hat transform might misclassify these dark brush strokes as cracks. Thus, in order to avoid any undesirable alterations to the original image, it is very important to separate these brush strokes from the actual cracks, before the implementation of the crack filling procedure. Two methods to achieve this goal are described in the following subsections.

A simple interactive approach for the separation of cracks from brush strokes is to apply a region growing algorithm on the threshold output of the top-hat transform, starting from pixels (seeds) on the actual cracks. The pixels are chosen by the user in an interactive mode. At least one seed per connected crack element should be chosen. Alternatively,

more convenient. The growth mechanism that was used implements the well-known grassfire algorithm that checks recursively for unclassified pixels with value 1 in the 8-neighborhood of each crack pixel. At the end of this procedure, the pixels in the binary image, which correspond to brush strokes that are not 8-connected to cracks will be removed. The above procedure can be used either in a stand-alone mode or applied on the output of the MRBF separation procedure described in the next section to eliminate any remaining brush strokes.

In our implementation, a MRBF network with two outputs was used. The first output represents the class of cracks while the second one the class of brush strokes. Input vectors were two-dimensional and consisted of the hue and saturation values of pixels identified as cracks by the top-hat transform. The number of clusters (hidden units) chosen for each class depends on the overlap between the populations of cracks and brush strokes. If there is a substantial overlap, the number should be increased, in order to reduce the classification error. In our implementation three hidden units have been incorporated. Training was carried out by presenting the network with hue and saturation values for pixels corresponding to cracks and crack-like brushstrokes. Data from 24 digitized portable religious icons from the Byzantine era were used for this purpose. The system trained using this specific training set can be considered to be

might result in somewhat suboptimal results. However, appropriately selected training sets can be used to train the system to separate cracks from brush strokes on paintings of different artistic styles or content. In order to select pixels corresponding to cracks and crack-like brush strokes the crack detection algorithm presented was applied on these images. Results were subsequently post processed by an expert using the semi-automatic approach.

The aim of this post processing step was twofold: to remove pixels that are neither cracks nor crack-like brush strokes and to separate cracks and crack-like brush strokes for the supervised step of the training procedure. In this supervised training step, the network was presented with these labeled inputs, i.e., pairs of hue-saturation values that corresponded to image pixels that have been identified as belonging to cracks and crack-like brush strokes. After the training session, the MRBF neural network was able to classify pixels identified as cracks by the top-hat transform to cracks or brush strokes. The trained network has been tested on 12 images from the training set and 15 images (of the same artistic style and era) that did not belong to the training set. Naturally, the performance of the cracks/brush-stroke separation procedure was judged only in a subjective manner (i.e., by visual inspection of the results), as ground truth data (i.e., brush stroke-free crack images) are not available. For this reason, two

after the application of the separation system and concluded that in the processed crack images the great majority of the brush strokes have been removed. A threshold top-hat transforms output containing many brush strokes. A great part of these brush strokes is separated by the MRFB

3.3 CRACK-FILLING METHODS

After identifying cracks and separating misclassified brush strokes, the final task is to restore the image using local image information to fill (interpolate) the cracks. Two classes of techniques, utilizing order statistics filtering and anisotropic diffusion are proposed for this purpose. Both are implemented on each RGB channel independently and affect only those pixels which belong to cracks. Therefore, provided that the identified crack pixels are indeed crack pixels, the filling procedure does not affect the “useful” content of the image. Image inpainting techniques can also be used for crack filling.

The performance of the crack filling methods presented below was judged by visual inspection of the results. Obviously, measuring the performance of these methods in an objective way is infeasible since ground truth data are not available. For the evaluation of the results, two restoration experts were asked to inspect several images restored using the various methods and comment based on their experience and quality of the filling results, whether the color used for filling was the correct one, whether

PERFORMANCE EVALUATION

4. PERFORMANCE EVALUATION

Testing is a critical element of software quality and assurance and represents the ultimate review of specification design and coding. It is a vital activity that has to be enforced in the development of any system. This could be done in parallel during all the phases of system development. The feedback received from these tests can be used for further enhancement of the system under consideration. The testing phase conducts test using the Software Requirement Specification as a reference and with the goal to see whether system satisfies the specified requirements.

Standard procedures have been followed in testing our system. Test cases are generated for each screen. These test cases will cover every possibility which could result in both positive and negative results. These test plans are maintained for any further testing done on the system.

- Software Testing
- Functional Testing
- System Testing

4.1 Software Testing:

Software Testing is the process of confirming the functionality and correctness of software by running it. Software testing is usually performed for one of two reasons:

- 1) Defect detection

The problem of applying software testing to defect detection is that software can only suggest the presence of flaws, not their absence (unless the testing is exhaustive). The problem of applying software testing to reliability estimation is that the input distribution used for selecting test cases may be flawed. It is common place to attempt to test as many of the syntactic features of the code as possible (within some set of resource constraints) are called *white box* software testing technique. Techniques that do not consider the code's structure when test cases are selected are called *black box technique*.

4.2 Functional Testing:

Functional testing is a testing process that is black box in nature. It is aimed at examine the overall functionality of the product. It usually includes testing of all the interfaces and should therefore involve the clients in the process.

4.3 System Testing:

System Testing should be the final stage of the testing process. This type of test involves examination of the whole computer system, all the software components, all the hard ware components and any interfaces.

The whole computer based system is checked not only for validity but also to meet the objectives.

CONCLUSION

5. CONCLUSION

In this paper, we have presented an integrated strategy for crack detection and filling in digitized paintings. Cracks are detected by using top-hat transform, whereas the thin dark brush strokes, which are misidentified as cracks, are separated either by an automatic technique (MRBF networks) or by a semi-automatic approach. Crack interpolation is performed by appropriately modified order statistics filters or controlled anisotropic diffusion. The methodology has been applied for the virtual restoration of images and was found very effective by restoration experts.

However, there are certain aspects of the proposed methodology that can be further improved. For example, the crack-detection stage is not very efficient in detecting cracks located on very dark image areas, since in these areas the intensity of crack pixels is very close to the intensity of the surrounding region. A possible solution to this shortcoming would be to apply the crack-detection algorithm locally on this area and select a low threshold value.

Another situation where the system (more particularly, the crack filling stage) does not perform as efficiently as expected is in the case of cracks that cross the border between regions of different color. In such situations, it might be the case that part of the crack in one area is filled with color from the other area, resulting in small spurs of color in the border

between the two regions. However, this phenomenon is rather seldom and furthermore, the extent of these erroneously filled areas is very small (2–3 pixels maximum). A possible solution would be to perform edge detection or segmentation on the image and confine the filling of cracks that cross edges or region borders to pixels from the corresponding region.

FUTURE ENHANCEMENT

6. FUTURE ENHANCEMENT

This project can be enhanced by

- Applying these algorithms to colored paintings.
- Instead of PGM format files, it can be implemented in other files like JPEG, BMP, etc.,
- Perfect crack detection can be performed automatically without selecting a particular area.

7. APPENDIX

7.1 SAMPLE CODE:

```
Public Class frmMDIMain
    Inherits System.Windows.Forms.Form
    Dim InputFilePath As String = ""

    Private Sub mnuImageOpenInputImage_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
mnuImageOpenInputImage.Click
        openDialog1.Filter = "PGM Images (*.pgm)|*.pgm"
        openDialog1.FileName = Application.StartupPath +
            "\images\in.pgm"
        If openDialog1.ShowDialog() =
            Windows.Forms.DialogResult.OK Then
            InputFilePath = openDialog1.FileName
            StatusBar1.Panels(0).Text = "InputFilePath: " +
                InputFilePath
            AddImage(openDialog1.FileName)
        End If
    End Sub

    Private Sub mnuImageOpenImage_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
mnuImageOpenImage.Click
        openDialog1.Filter = "PGM Images (*.pgm)|*.pgm"
        openDialog1.FileName = Application.StartupPath +
            "\images\in.pgm"
        If openDialog1.ShowDialog() =
            Windows.Forms.DialogResult.OK Then
            AddImage(openDialog1.FileName)
        End If
    End Sub

    Private Sub mnuCrackDetectionTopHatTransform_Click(ByVal
sender As System.Object, ByVal e As System.EventArgs)
Handles mnuCrackDetectionTopHatTransform.Click
        If InputFilePath <> "" Then
            Dim tCrackDetection As New CrackDetection
            tCrackDetection.setInFilePath(InputFilePath)
            tCrackDetection.setOutFilePath("out_3_tophat.pgm")
            tCrackDetection.TopHatTransform()
            TopHatFinished = True

            Call MsgBox("TopHat Transform, Finished.",
                MsgBoxStyle.Information Or MsgBoxStyle.OkOnly, strTitle)
        End If
    End Sub
```

```
Private Sub mnuCrackDetectionThreshold_Click(ByVal sender
As System.Object, ByVal e As System.EventArgs) Handles
mnuCrackDetectionThreshold.Click
```

```
    If TophatFinished = True Then
        Dim tfrmThreshold As New frmThreshold
        tfrmThreshold.MdiParent = tfrmMDIMain
        tfrmThreshold.Show()
    End If
```

```
End Sub
```

```
Private Sub mnuCrackRemovalOrderStatisticsFilter_Click
(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
mnuCrackRemovalOrderStatisticsFilter.Click
```

```
    ThresholdFinished = True
    If ThresholdFinished = True Then
        Dim tCrackRemoval As New CrackRemoval
        tCrackRemoval.setInFilePath(InputFilePath)
        tCrackRemoval.setOutFilePath("t.pgm")
        tCrackRemoval.ModifiedTrimmedMeanFilter(20)
        tCrackRemoval.setInFilePath("t.pgm")
        tCrackRemoval.setOutFilePath("out_6_removed.pgm")
        tCrackRemoval.RecursiveMeanFilter(5)
        Kill("t.pgm")
```

```
        Call MsgBox("Crack Removal, Finished.",
MsgBoxStyle.Information Or MsgBoxStyle.OkOnly, strTitle)
    End If
```

```
End Sub
```

```
Private Sub
mnuThinBrushStrokesRemovalRemoveThinBrushStrokes_Click
(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
mnuThinBrushStrokesRemovalRemoveThinBrushStrokes.Click
```

```
    'stop roi
    ROICaptureStarted = False
    StatusBar1.Panels(1).Text = ""
    'call grassfire algorithm
    If ThresholdFinished = True Then
```

```
        Dim tCrackDetection As New CrackDetection
        tCrackDetection.setInFilePath("out_4_threshold.pgm")
        tCrackDetection.setOutFilePath("out_5_cracks.pgm")
        tCrackDetection.GrassFire()
        RemoveThinBrushStrokesFinished = True
        Call MsgBox("RemoveThinBrushStrokes,
Finished.", MsgBoxStyle.Information Or MsgBoxStyle.OkOnly,
strTitle)
```

```
    End If
```

```
End Sub
```

Module modCrackDetection

```
Module modCrackDetection
Public Class CrackDetection
  Dim InFilePath As String, OutFilePath As String
  'constructor
  Public Sub New()
    InFilePath = ""
    OutFilePath = ""
  End Sub

  Public Function EdgeDetection(ByVal tIntensityDifference As
Integer) As PGM
  Dim imgIn As PGM = New PGM
  Dim imgOut As PGM = New PGM
  Dim r As Integer, c As Integer, inval As Integer,
outval As Integer
  'read input image
  imgIn.setFilePath(InFilePath)
  imgIn.readImage()
  'set output-image header
  imgOut.setFilePath(OutFilePath)
  imgOut.setMagicString("P5")
  imgOut.setComment("#edge image")
  imgOut.setDimension(imgIn.getCols(),
                      imgIn.getRows())
  imgOut.setMaxGray(imgIn.getMaxGray())
  'edgfinding algorithm
  For r = 0 To imgOut.getRows() - 1
    For c = 0 To imgOut.getCols() - 1
      inval = imgIn.getPixel(r, c) 'get current pixel
      Dim tival(2) As Integer
      Dim fval As Boolean
      Dim flag1 As Boolean, flag2 As Boolean
      tival(0) = imgIn.getPixel(r, c + 1) 'get right pixel
      tival(1) = imgIn.getPixel(r + 1, c) 'get down pixel
      flag1 = IIf(Math.Abs(inval - tival(0)) >
                  tIntensityDifference, True, False)
      flag2 = IIf(Math.Abs(inval - tival(1)) >
                  tIntensityDifference, True, False)
      fval = flag1 Or flag2
      outval = IIf(fval = True, 0, 255)
      imgOut.setPixel(r, c, outval)
    Next
  Next
  'write output image
  imgOut.writeImage()
  Return (imgOut)
End Function
End Class
```

'mean filter algorithm (low-pass filter)

Public Function MeanFilter(ByVal tIterations As Integer) As PGM

```
Dim imgIn As PGM = New PGM
Dim imgOut As PGM = New PGM
Dim r As Integer, c As Integer
Dim inval As Integer, outval As Integer
'read input image
imgIn.setFilePath(InFilePath)
imgIn.readImage()
'set output-image header
imgOut.setFilePath(OutFilePath)
imgOut.setMagicString("P5")
imgOut.setComment("#smoothed image")
imgOut.setDimension(imgIn.getCols(),
                    imgIn.getRows())
imgOut.setMaxGray(imgIn.getMaxGray())
'smoothing algorithm (mean)
Dim t As Integer
For t = 1 To tIterations
  For r = 0 To imgOut.getRows() - 1
    For c = 0 To imgOut.getCols() - 1
      inval = imgIn.getPixel(r, c) 'get current pixel
      'get neighbourhood pixel intensity values
      Dim neighbour(8) As Integer
      neighbour(0) = imgIn.getNeighbor(r, c,
                                       Directions.NW)
      neighbour(1) = imgIn.getNeighbor(r, c,
                                       Directions.W)
      neighbour(2) = imgIn.getNeighbor(r, c,
                                       Directions.SW)
      neighbour(3) = imgIn.getNeighbor(r, c,
                                       Directions.S)
      neighbour(4) = imgIn.getNeighbor(r, c,
                                       Directions.SE)
      neighbour(5) = imgIn.getNeighbor(r, c,
                                       Directions.E)
      neighbour(6) = imgIn.getNeighbor(r, c,
                                       Directions.NE)
      neighbour(7) = imgIn.getNeighbor(r, c,
                                       Directions.N)

      'calculate new intensity value
      Dim tsum As Integer = inval
      Dim k As Integer
      For k = 0 To 7
        tsum = tsum + neighbour(k) 'calc sum
      Next
      outval = tsum / 9 'calc mean
      imgOut.setPixel(r, c, outval)
```

```
Next
```

```
'set outputimage as input to next iteration (if not last iteration)
```

```
    If t <> tIterations Then
```

```
        Dim r1 As Integer
```

```
        For r1 = 0 To imgIn.getRows() - 1
```

```
            For c = 0 To imgIn.getCols() - 1
```

```
                imgIn.setPixel(r1, c, imgOut.getPixel(r1, c))
```

```
            Next
```

```
        Next
```

```
    End If
```

```
Next
```

```
'write output image
```

```
imgOut.writeImage()
```

```
Return (imgOut)
```

```
End Function
```

```
'thresholding
```

```
Public Function Thresholding(ByVal tIntensityThreshold As Integer) As PGM
```

```
    Dim imgIn As PGM = New PGM
```

```
    Dim imgOut As PGM = New PGM
```

```
    Dim r As Integer, c As Integer, inval As Integer,  
        outval As Integer
```

```
    'read input image
```

```
imgIn.setFilePath(InFilePath)
```

```
imgIn.readImage()
```

```
    'set output-image header
```

```
imgOut.setFilePath(OutFilePath)
```

```
imgOut.setMagicString("P5")
```

```
imgOut.setComment("#thresholded image")
```

```
imgOut.setDimension(imgIn.getCols(), imgIn.getRows())
```

```
imgOut.setMaxGray(imgIn.getMaxGray())
```

```
    'binary thresholding algorithm
```

```
    For r = 0 To imgOut.getRows() - 1
```

```
        For c = 0 To imgOut.getCols() - 1
```

```
            inval = imgIn.getPixel(r, c) 'get current pixel
```

```
            outval = IIf(inval >= tIntensityThreshold, 0, 255)
```

```
            imgOut.setPixel(r, c, outval)
```

```
        Next
```

```
    Next
```

```
    'write output image
```

```
imgOut.writeImage()
```

```
Return (imgOut)
```

```
End Function
```

```
'erosion
```

```
Public Function TopHatTransform() As PGM
```

```
    Dim imgIn As PGM = New PGM
```

```

Dim imgDilation As PGM = New PGM
Dim imgTophat As PGM = New PGM
Dim r As Integer, c As Integer
Dim inval As Integer, outval As Integer
'read input image
imgIn.setFilePath(InFilePath)
imgIn.readImage()
'set erosion-image header
imgErosion.setFilePath("out_1_erosion.pgm")
imgErosion.setMagicString("P5")
imgErosion.setComment("#eroded image")
imgErosion.setDimension(imgIn.getCols(),
                        imgIn.getRows())
imgErosion.setMaxGray(imgIn.getMaxGray())
'set dilation-image header
imgDilation.setFilePath("out_2_dilation.pgm")
imgDilation.setMagicString("P5")
imgDilation.setComment("#dilated image")
imgDilation.setDimension(imgIn.getCols(),
                        imgIn.getRows())
imgDilation.setMaxGray(imgIn.getMaxGray())
'set tophat-image header
imgTophat.setFilePath(OutFilePath)
imgTophat.setMagicString("P5")
imgTophat.setComment("#tophat-transformed image")
imgTophat.setDimension(imgIn.getCols(),
                        imgIn.getRows())
imgTophat.setMaxGray(imgIn.getMaxGray())
'erosion algorithm
For r = 0 To imgErosion.getRows() - 1
  For c = 0 To imgErosion.getCols() - 1
    inval = imgIn.getPixel(r, c)
    'get neighborhood pixel intensity values
    Dim neighbor(8) As Integer
    neighbor(0) = imgIn.getNeighbor(r, c, Directions.NW)
    neighbor(1) = imgIn.getNeighbor(r, c, Directions.W)
    neighbor(2) = imgIn.getNeighbor(r, c, Directions.SW)
    neighbor(3) = imgIn.getNeighbor(r, c, Directions.S)
    neighbor(4) = imgIn.getNeighbor(r, c, Directions.SE)
    neighbor(5) = imgIn.getNeighbor(r, c, Directions.E)
    neighbor(6) = imgIn.getNeighbor(r, c, Directions.NE)
    neighbor(7) = imgIn.getNeighbor(r, c, Directions.N)
    Dim flag As Boolean = True, t As Integer
    For t = 0 To 7
      If Math.Abs(neighbor(t) - inval) > 5 Then
        flag = False
      Exit For
    End If
  Next
  outval = IIf(flag, inval, 255)

```

```

Next
Next
'dilation algorithm (grayscale opening)
For r = 0 To imgErosion.getRows() - 1
  For c = 0 To imgErosion.getCols() - 1
    inval = imgErosion.getPixel(r, c)
    imgDilation.setPixel(r - 1, c - 1, inval)
    imgDilation.setPixel(r - 1, c, inval)
    imgDilation.setPixel(r - 1, c + 1, inval)
    imgDilation.setPixel(r, c - 1, inval)
    imgDilation.setPixel(r, c, inval)
    imgDilation.setPixel(r, c + 1, inval)
    imgDilation.setPixel(r + 1, c - 1, inval)
    imgDilation.setPixel(r + 1, c, inval)
    imgDilation.setPixel(r + 1, c + 1, inval)
  Next
Next
'tophat transform (originalImage - grayscaleOpenedImage)
Dim inval1 As Integer
For r = 0 To imgIn.getRows() - 1
  For c = 0 To imgIn.getCols() - 1
    inval = imgIn.getPixel(r, c) 'pixel from originalImage
    inval1 = imgDilation.getPixel(r, c) 'pixel from
                                   grayscaleOpenedImage

    outval = Math.Abs(inval - inval1)
    imgTophat.setPixel(r, c, outval)
  Next
Next
imgErosion.writeImage()
imgDilation.writeImage()
imgTophat.writeImage()
Return (imgErosion)
Return (imgDilation)
Return (imgTophat)
End Function

```

```

'grassfire algorithm (removal of thin brush strokes)
Public Function GrassFire() As PGM
  Dim imgIn As PGM = New PGM
  Dim imgOut As PGM = New PGM
  Dim r As Integer, c As Integer
  Dim inval As Integer
  'read input image
  imgIn.setFilePath(InFilePath)
  imgIn.readImage()
  'set output-image header
  imgOut.setFilePath(OutFilePath)
  imgOut.setMagicString("P5")
  imgOut.setComment("#cracks image")
  imgOut.setDimension(imgIn.getCols(), imgIn.getRows())

```

```

'grass algorithm
'set background as white
For r = 0 To imgOut.getRows() - 1
  For c = 0 To imgOut.getCols() - 1
    imgOut.setPixel(r, c, 255)
  Next
Next
Dim i As Integer
For i = 0 To nROI - 1
'turn roi pixel of output-image into black
Dim tX As Integer, tY As Integer
  tX = ROIX(i)
  tY = ROIY(i)
  imgOut.setPixel(tY, tX, 0)
'perform the algorithm
Dim nIterations As Integer
Dim nPrevHits As Integer
While True
  Dim nHits As Integer
  nHits = 0
  For r = 0 To imgOut.getRows() - 1
    For c = 0 To imgOut.getCols() - 1
      'get roi pixel, if any
      inval = imgOut.getPixel(r, c)
      If inval = 0 Then
        'check imgin's 8 neighbors
        Dim neighbor(8) As Integer
        neighbor(0) = imgIn.getNeighbor(r, c, Directions.NW)
        neighbor(1) = imgIn.getNeighbor(r, c, Directions.W)
        neighbor(2) = imgIn.getNeighbor(r, c, Directions.SW)
        neighbor(3) = imgIn.getNeighbor(r, c, Directions.S)
        neighbor(4) = imgIn.getNeighbor(r, c, Directions.SE)
        neighbor(5) = imgIn.getNeighbor(r, c, Directions.E)
        neighbor(6) = imgIn.getNeighbor(r, c, Directions.NE)
        neighbor(7) = imgIn.getNeighbor(r, c, Directions.N)
        Dim t As Integer, tCount As Integer
        tCount = 0
        For t = 0 To 7
          If neighbor(t) = 0 Then
            tCount += 1
          End If
        Next
        'if all neighbors are black
        If tCount = 8 Then
          nHits += 1
        'set all neighbors of current output pixel to black
        imgOut.setPixel(r - 1, c - 1, inval)
        imgOut.setPixel(r - 1, c, inval)
        imgOut.setPixel(r - 1, c + 1, inval)
        imgOut.setPixel(r, c - 1, inval)

```



```

        imgOut.setPixel(r, c + 1, inval)
        imgOut.setPixel(r + 1, c - 1, inval)
        imgOut.setPixel(r + 1, c, inval)
        imgOut.setPixel(r + 1, c + 1, inval)
    End If
End If
Next
Next
nIterations += 1
If nIterations = 1 Then
    nPrevHits = nHits
Else
    If nPrevHits = nHits Then
        Exit While
    End If
    nPrevHits = nHits
End If
End While
Next
'write output image
imgOut.writeImage()
Return (imgOut)
End Function
End Class
End Module

```

Module modCrackRemoval

```

Module modCrackRemoval
Public Class CrackRemoval
    Dim InFilePath As String, OutFilePath As String
    'constructor
    Public Sub New()
        InFilePath = ""
        OutFilePath = ""
    End Sub
    Public Function RecursiveMeanFilter(PvVal tIterations As
Integer) As PGM
        Dim imgIn As PGM = New PGM
        Dim imgOut As PGM = New PGM
        'read input image
        imgIn.setFilePath(InFilePath)
        imgIn.readImage()
        'set output-image header
        imgOut.setFilePath(OutFilePath)
        imgOut.setMagicString("P5")
        imgOut.setComment("#mean-filtered image")
        imgOut.setDimension(imgIn.getCols(), imgIn.getRows())
        imgOut.setMaxGray(imgIn.getMaxGray())

```

```

'mean filter algorithm
Dim r As Integer, c As Integer
Dim inval As Integer, outval As Integer
Dim t As Integer
For t = 1 To tIterations
  For r = 0 To imgOut.getRows() - 1
    For c = 0 To imgOut.getCols() - 1
      inval = imgIn.getPixel(r, c) 'get current pixel
      'get neighborhood pixel intensity values
      Dim neighbor(8) As Integer
      neighbor(0) = imgIn.getNeighbor(r, c, Directions.NW)
      neighbor(1) = imgIn.getNeighbor(r, c, Directions.W)
      neighbor(2) = imgIn.getNeighbor(r, c, Directions.SW)
      neighbor(3) = imgIn.getNeighbor(r, c, Directions.S)
      neighbor(4) = imgIn.getNeighbor(r, c, Directions.SE)
      neighbor(5) = imgIn.getNeighbor(r, c, Directions.E)
      neighbor(6) = imgIn.getNeighbor(r, c, Directions.NE)
      neighbor(7) = imgIn.getNeighbor(r, c, Directions.N)
      'compute new intensity value as mean of neighborhood
      Dim tsum As Integer = inval
      Dim k As Integer
      For k = 0 To 7
        tsum = tsum + neighbor(k) 'calc sum
      Next
      outval = tsum / 9
      imgOut.setPixel(r, c, outval)
    Next
  Next
  'set outputimage as input to next iteration (if not
last iteration)
  If t <> tIterations Then
    Dim r1 As Integer
    For r1 = 0 To imgIn.getRows() - 1
      For c = 0 To imgIn.getCols() - 1
        imgIn.setPixel(r1, c, imgOut.getPixel(r1, c))
      Next
    Next
  End If
Next
'write output image
imgOut.writeImage()
Return (imgOut)
End Function
End Class
End Module

```

Module modPGM

Module modPGM

Public Enum Directions

NW = 0

N

NE

E

SE

S

SW

W

End Enum

'pgmimage

Public Class PGM

Dim tFilePath As String

'pgm imageheader

Dim MagicString As String

Dim Comment As String

Dim Cols As Integer, Rows As Integer, MaxGray As Integer

'pgm imagedata

Dim Pixels(,) As Integer

'constructor

Public Sub New()

tFilePath = ""

MagicString = ""

Comment = ""

Cols = 0

Rows = 0

MaxGray = 0

End Sub

Public Function getPixel(ByVal tr As Integer, ByVal tc
As Integer) As Integer

Dim tIntensity As Integer

If tr >= 0 And tr <= Rows - 1 And tc >= 0 And tc <=
Cols - 1 Then

tIntensity = Pixels(tr, tc)

End If

Return (tIntensity)

End Function

Public Function getNeighbor(ByVal tr As Integer, ByVal
tc As Integer, ByVal direction As Integer) As Integer

Dim pval As Integer = 0

If direction = Directions.NW Then

pval = getPixel(tr - 1, tc - 1)

ElseIf direction = Directions.W Then

pval = getPixel(tr, tc - 1)

ElseIf direction = Directions.SW Then

```

    ElseIf direction = Directions.S Then
        pval = getPixel(tr + 1, tc)
    ElseIf direction = Directions.SE Then
        pval = getPixel(tr + 1, tc + 1)
    ElseIf direction = Directions.E Then
        pval = getPixel(tr, tc + 1)
    ElseIf direction = Directions.NE Then
        pval = getPixel(tr - 1, tc + 1)
    ElseIf direction = Directions.N Then
        pval = getPixel(tr - 1, tc)
    End If
    Return (pval)
End Function
'methods
Public Function readImage()
    Try
        Dim fin As IO.FileStream = New
            IO.FileStream(tFilePath, IO.FileMode.Open)
        Dim tByte As Byte, tStr As String
    'read magicstring
        tByte = fin.ReadByte()
        MagicString = Chr(tByte)
        tByte = fin.ReadByte()
        MagicString = MagicString + Chr(tByte)
    'read comment
        tByte = fin.ReadByte() 'read vbCr
        tByte = fin.ReadByte() 'read '#'
        If tByte = 10 Then tByte = fin.ReadByte()
        Do While Chr(tByte) = "#" 'read comment
            tStr = Chr(tByte)
            Do While Chr(tByte) <> vbLf
                tByte = fin.ReadByte()
                tStr = tStr + Chr(tByte)
            Loop
            tByte = fin.ReadByte() 'read next '#'
        Loop
        If tStr <> "" Then
            Comment = Left(tStr, Len(tStr) - 1)
        End If
    'read cols
        tByte = 1
        tStr = ""
        fin.Seek(-1, IO.SeekOrigin.Current)
        tByte = fin.ReadByte()
        tStr = Chr(tByte)
        Do While Chr(tByte) <> " " And Chr(tByte) <> vbLf
            tByte = fin.ReadByte()
            tStr = tStr + Chr(tByte)
        Loop
        Cols = Int(tStr)

```

```

tByte = 1
tStr = ""
tByte = fin.ReadByte()
Do While Chr(tByte) <> " " And Chr(tByte) <> vbCrLf
    tStr = tStr + Chr(tByte)
    tByte = fin.ReadByte()
Loop
Rows = Int(tStr)
'read maxgray
tByte = 1
tStr = ""
tByte = fin.ReadByte()
Do While Chr(tByte) <> " " And Chr(tByte) <> vbCrLf
    tStr = tStr + Chr(tByte)
    tByte = fin.ReadByte()
Loop
MaxGray = Int(tStr)
'read imagedata
ReDim Pixels(Rows, Cols)
Dim x As Integer, y As Integer
For y = 0 To Rows - 1
    For x = 0 To Cols - 1
        tByte = fin.ReadByte()
        Pixels(y, x) = tByte
    Next
Next
fin.Close()
Catch err As Exception
    ShowError(err)
End Try
End Function

```

```

Public Function writeImage()
    Try
        Dim fout As IO.FileStream = New IO.FileStream(tFilePath,
            IO.FileMode.Create)
        Dim tByte As Byte, tStr As String, i As Integer, j As Integer
        'write image header
        'write magicstring
        tStr = "P5" + vbCrLf
        For i = 0 To Len(tStr) - 1
            tByte = Asc(tStr.Chars(i))
            fout.WriteByte(tByte)
        Next
        'write comment
        If Comment <> "" Then
            tStr = Comment + vbCrLf
            For i = 0 To Len(tStr) - 1
                tByte = Asc(tStr.Chars(i))
                fout.WriteByte(tByte)
            Next
        End If
    End Try
End Function

```

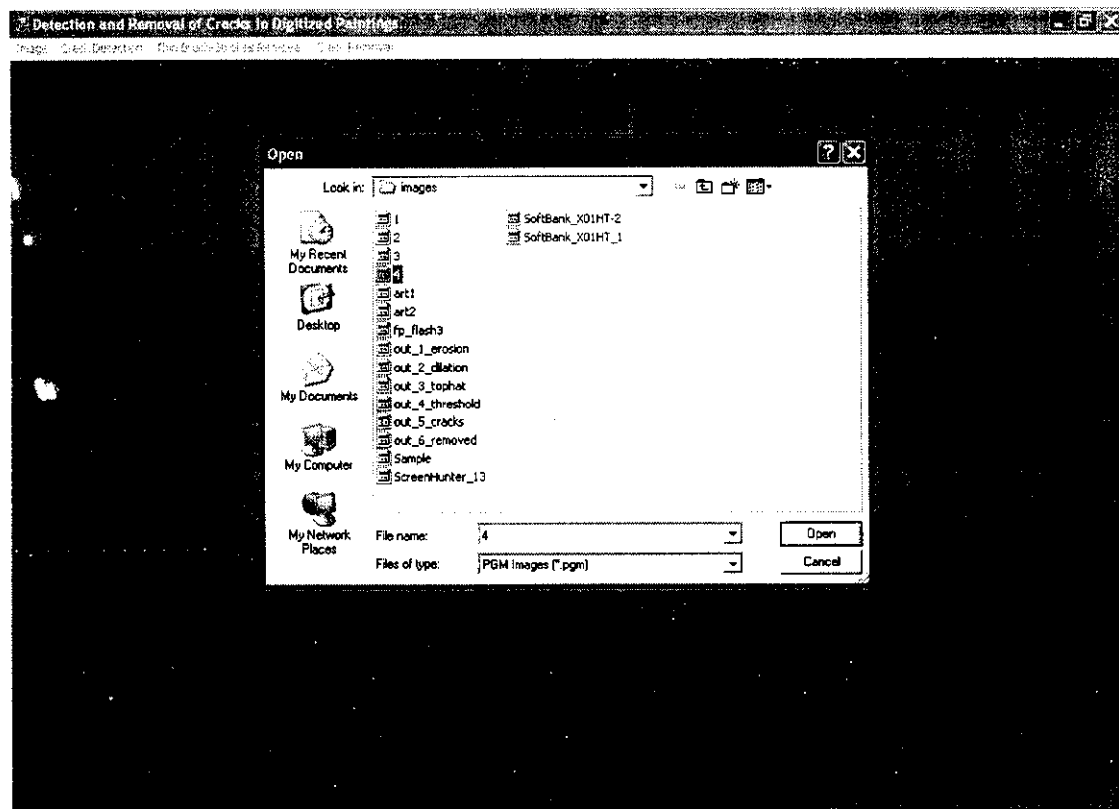
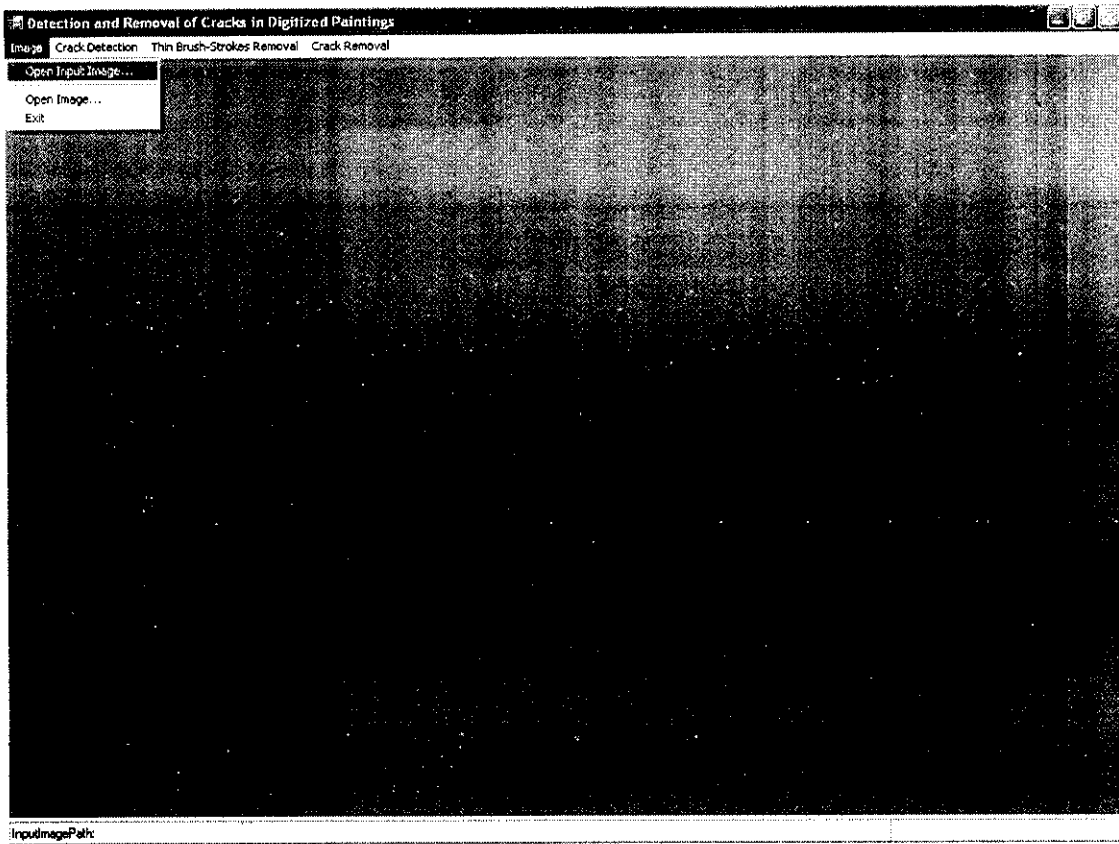
```

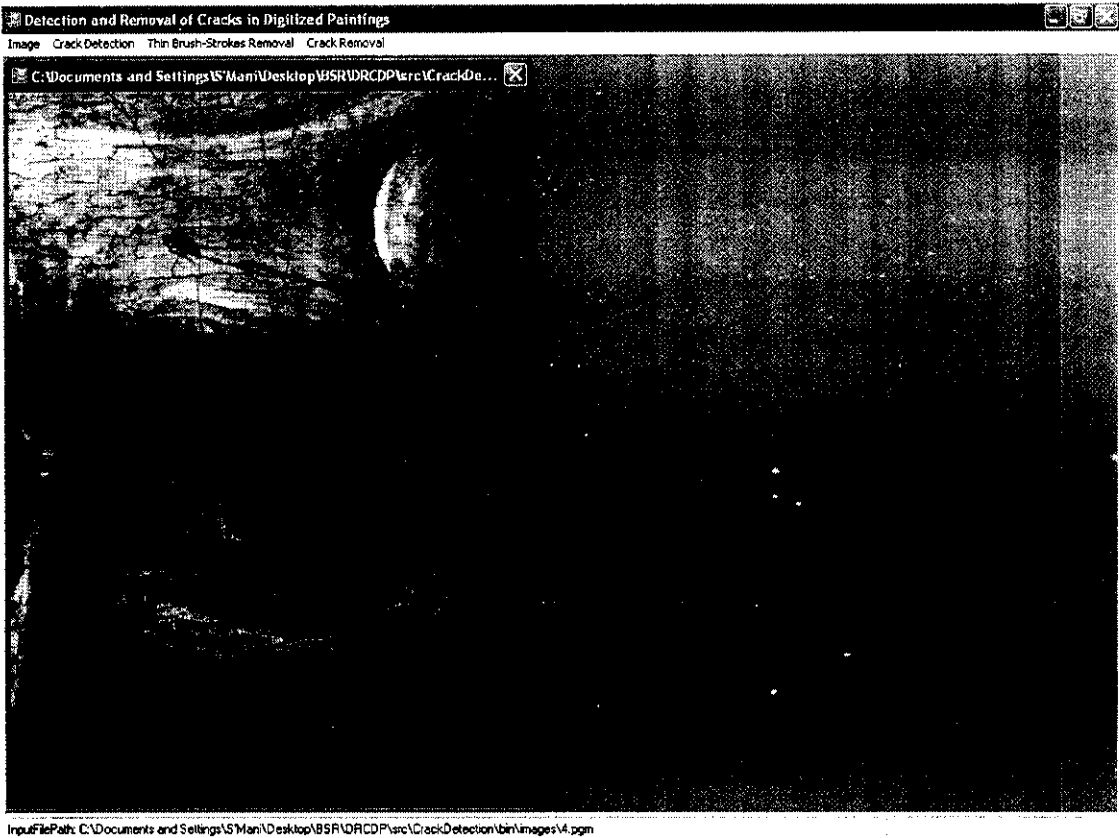
        End If
'write cols
tStr = Trim(Str(Cols)) + " "
For i = 0 To Len(tStr) - 1
    tByte = Asc(tStr.Chars(i))
    fout.WriteByte(tByte)
Next
'write rows
tStr = Trim(Str(Rows)) + vbCrLf
For i = 0 To Len(tStr) - 1
    tByte = Asc(tStr.Chars(i))
    fout.WriteByte(tByte)
Next
'write maxgray
tStr = Trim(Str(MaxGray)) + vbCrLf
For i = 0 To Len(tStr) - 1
    tByte = Asc(tStr.Chars(i))
    fout.WriteByte(tByte)
Next
For i = 0 To Rows - 1
    For j = 0 To Cols - 1
        tByte = Pixels(i, j)
        fout.WriteByte(tByte)
    Next
Next
fout.Close()
Catch err As Exception
    ShowError(err)
End Try
End Function
End Class
End Module

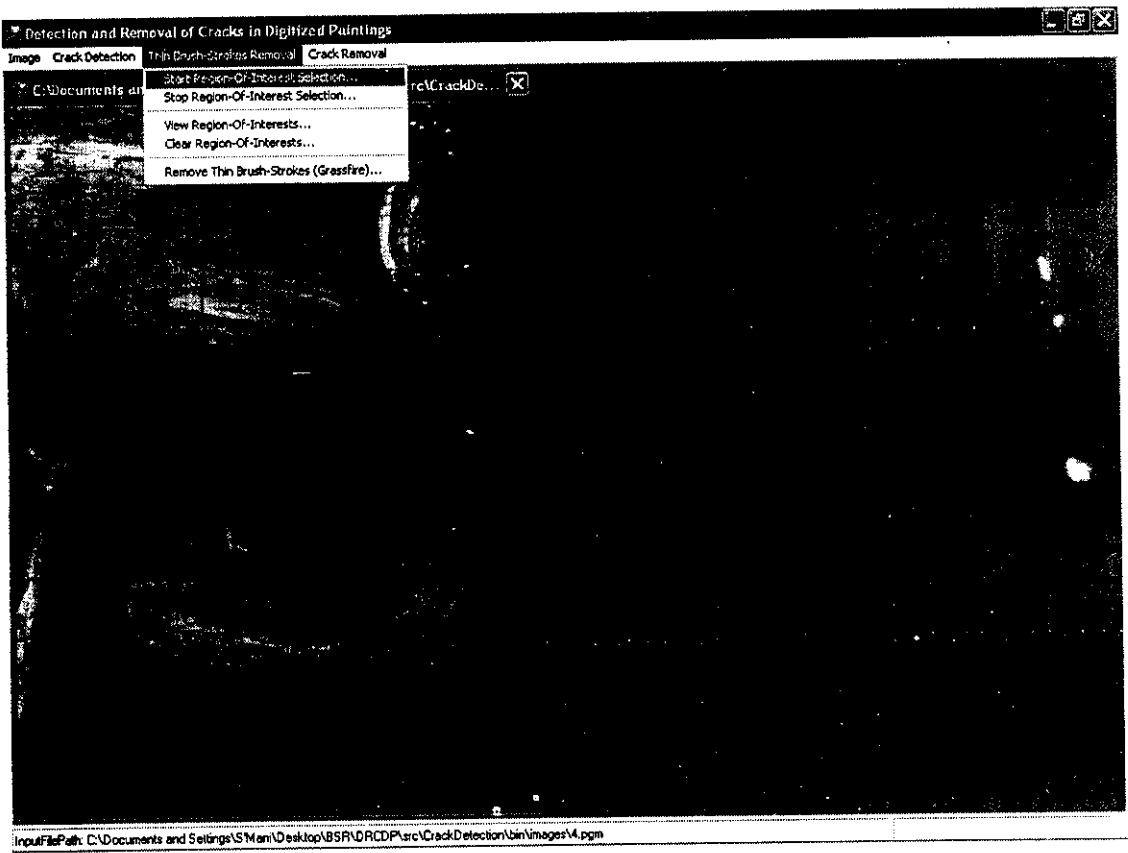
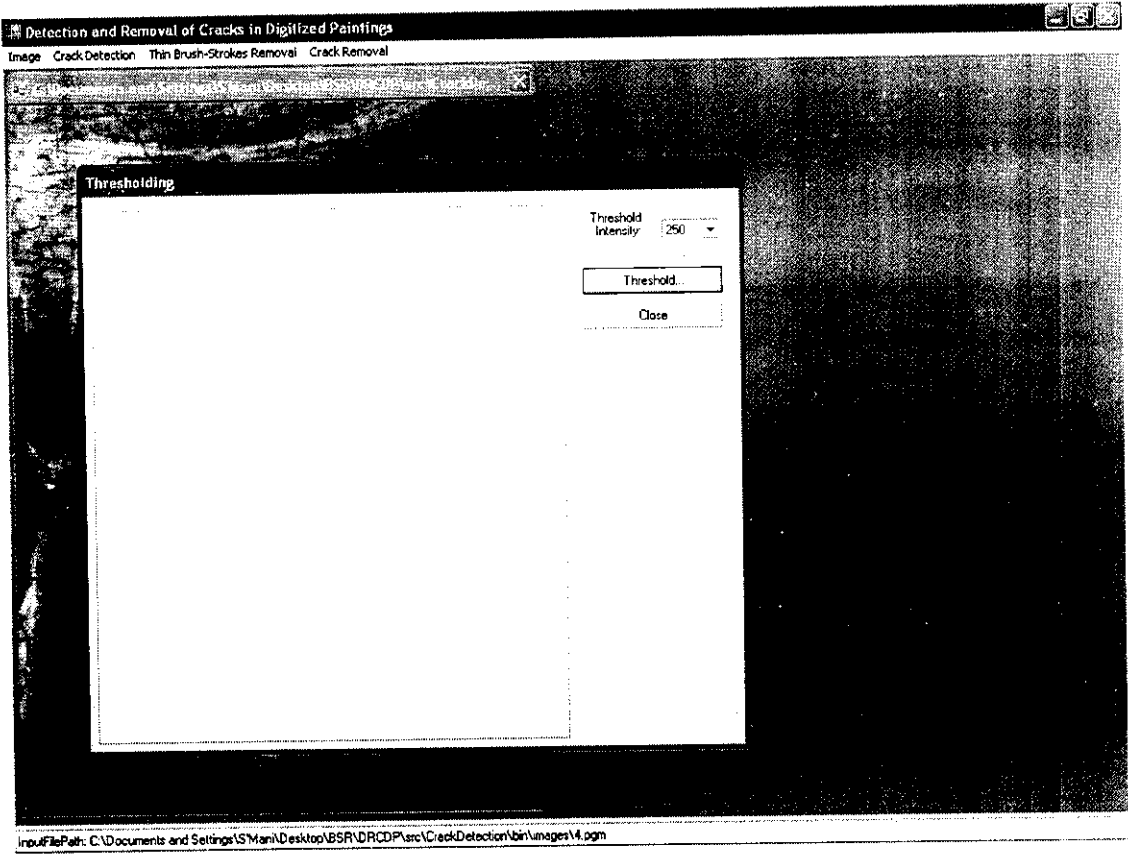
```

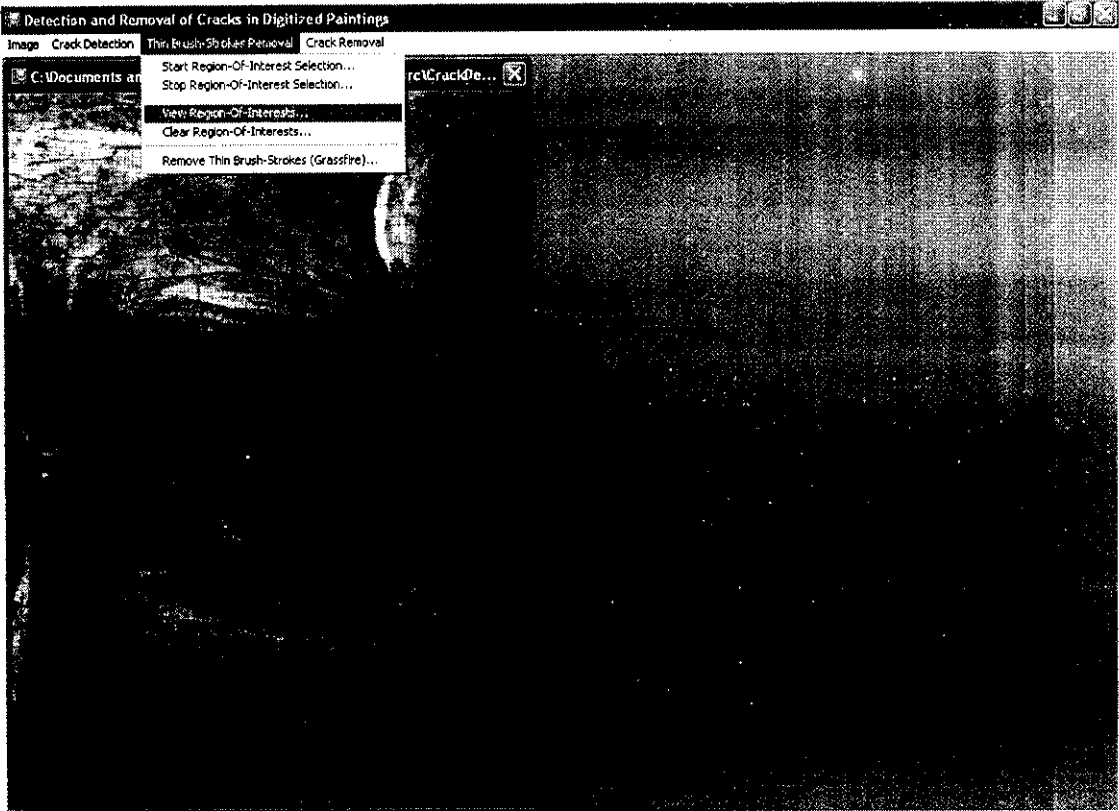
APPENDIX 2-SCREEN SHOTS

7.2 OUTPUT SCREENS:

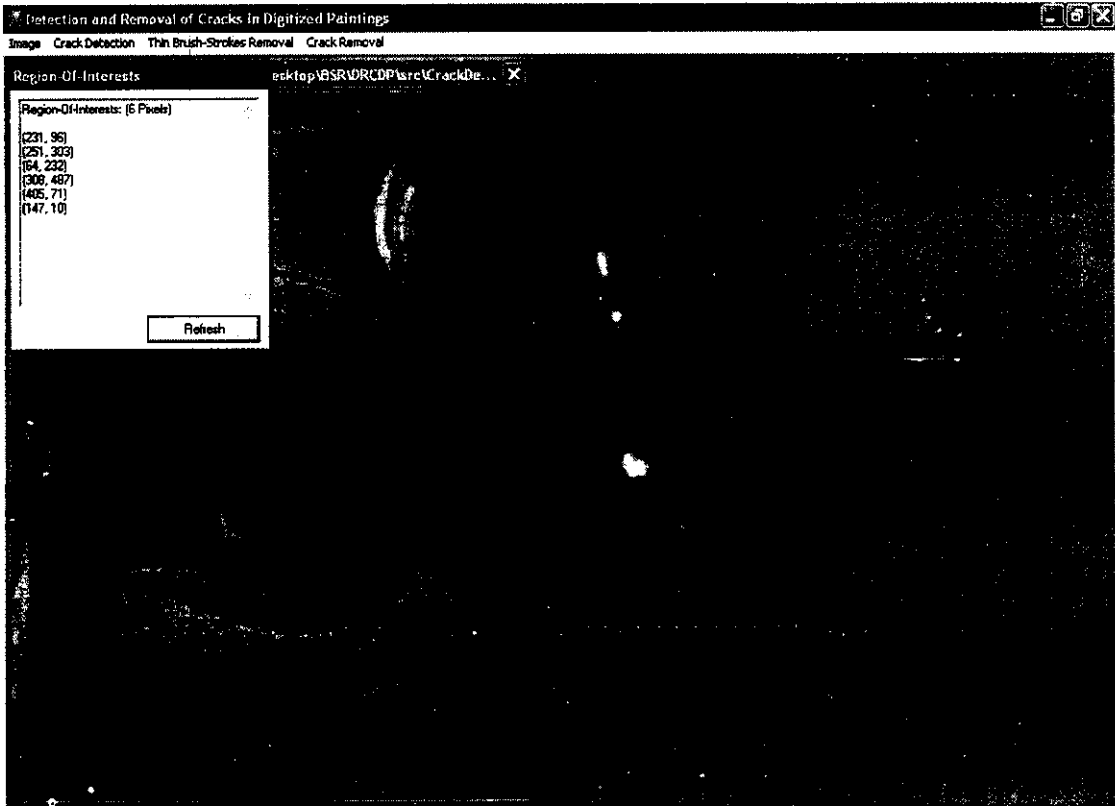








InputFilePath: C:\Documents and Settings\Mani\Desktop\BSR\DRCDP\src\CrackDetection\bin\Images\4.pgm



InputFilePath: C:\Documents and Settings\Mani\Desktop\BSR\DRCDP\src\CrackDetection\bin\Images\4.pgm

Detection and Removal of Cracks in Digitized Paintings

RemoveThinBrushStrokes, Finished.

OK

InputFilePath: C:\Documents and Settings\S'Mani\Desktop\BSR\DRCDP\src\CrackDetection\bin\images\4.pgm

Detection and Removal of Cracks in Digitized Paintings

Image Crack Detection Thin Brush-Strokes Removal Crack Removal

C:\Documents and Settings\S'Mani\Desktop\Digital_Strokeless_Filter... e...

InputFilePath: C:\Documents and Settings\S'Mani\Desktop\BSR\DRCDP\src\CrackDetection\bin\images\4.pgm

Detection and Removal of Cracks in Digitized Paintings



Crack Removal, Finished.

OK

InputFilePath: C:\Documents and Settings\Mani\Desktop\BSR\DRCDP\src\CrackDetection\bin\images\4.pgm

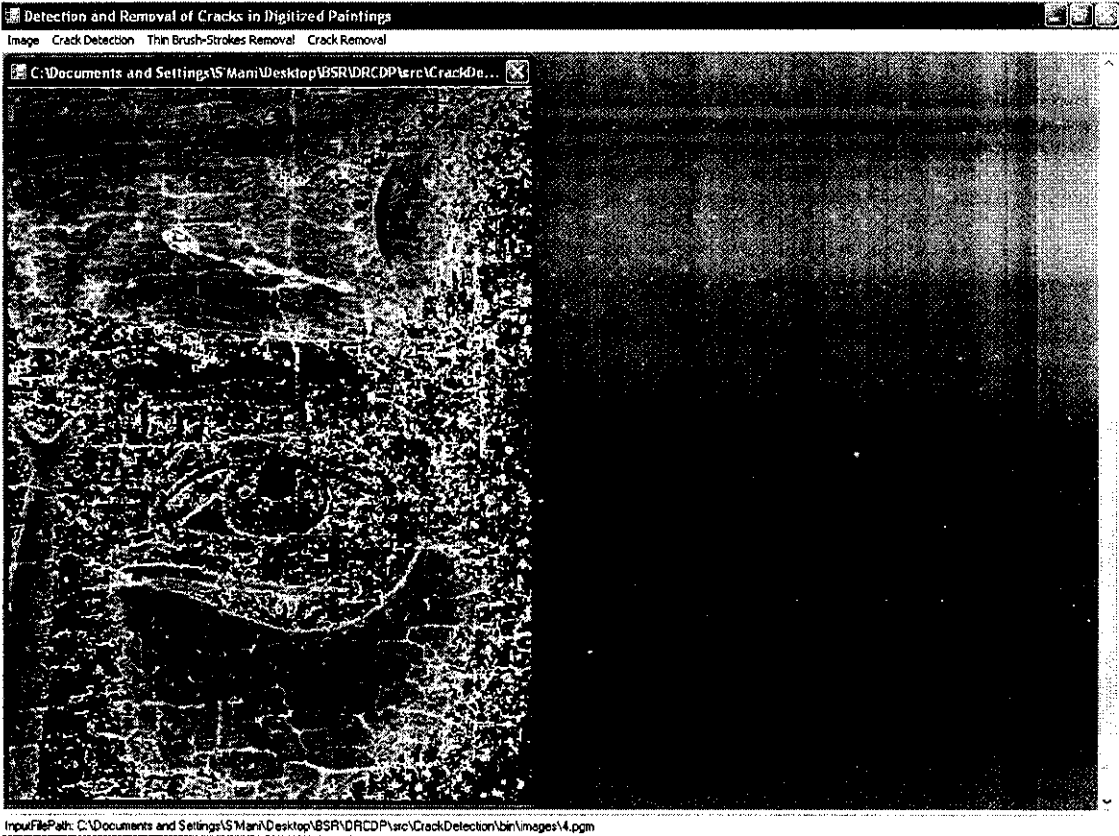
Detection and Removal of Cracks in Digitized Paintings

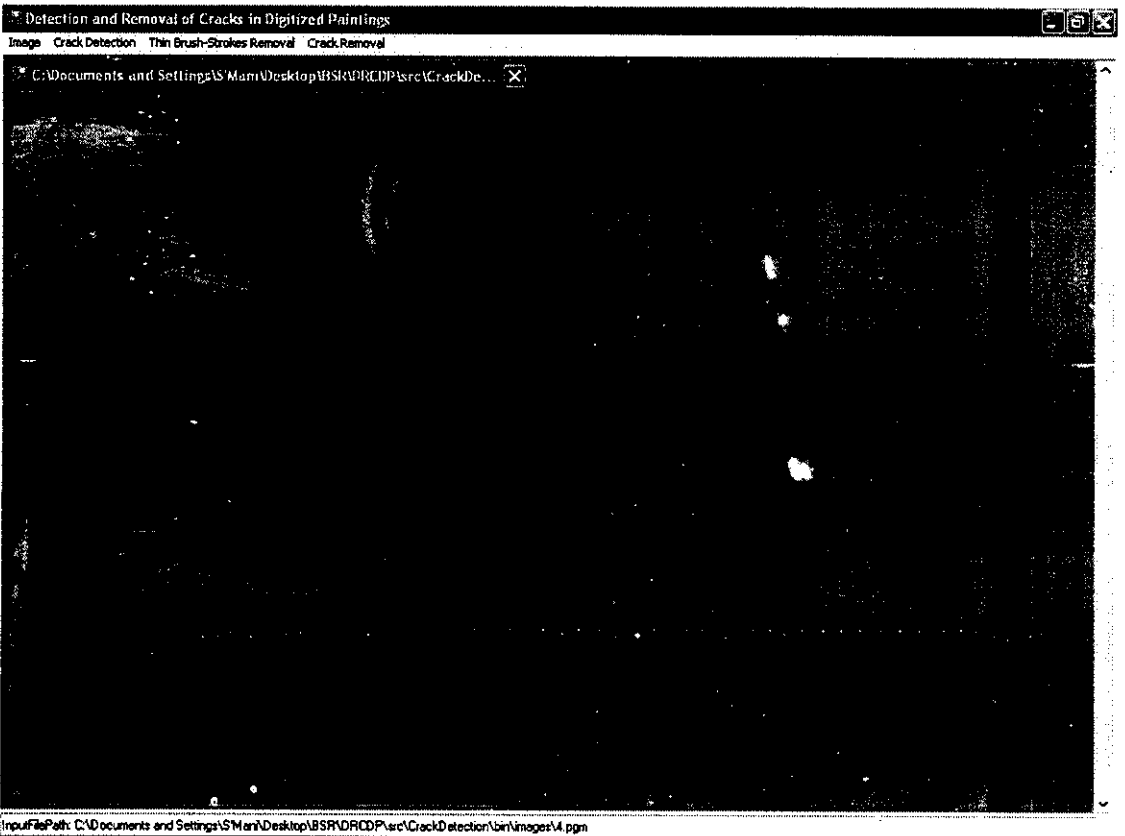
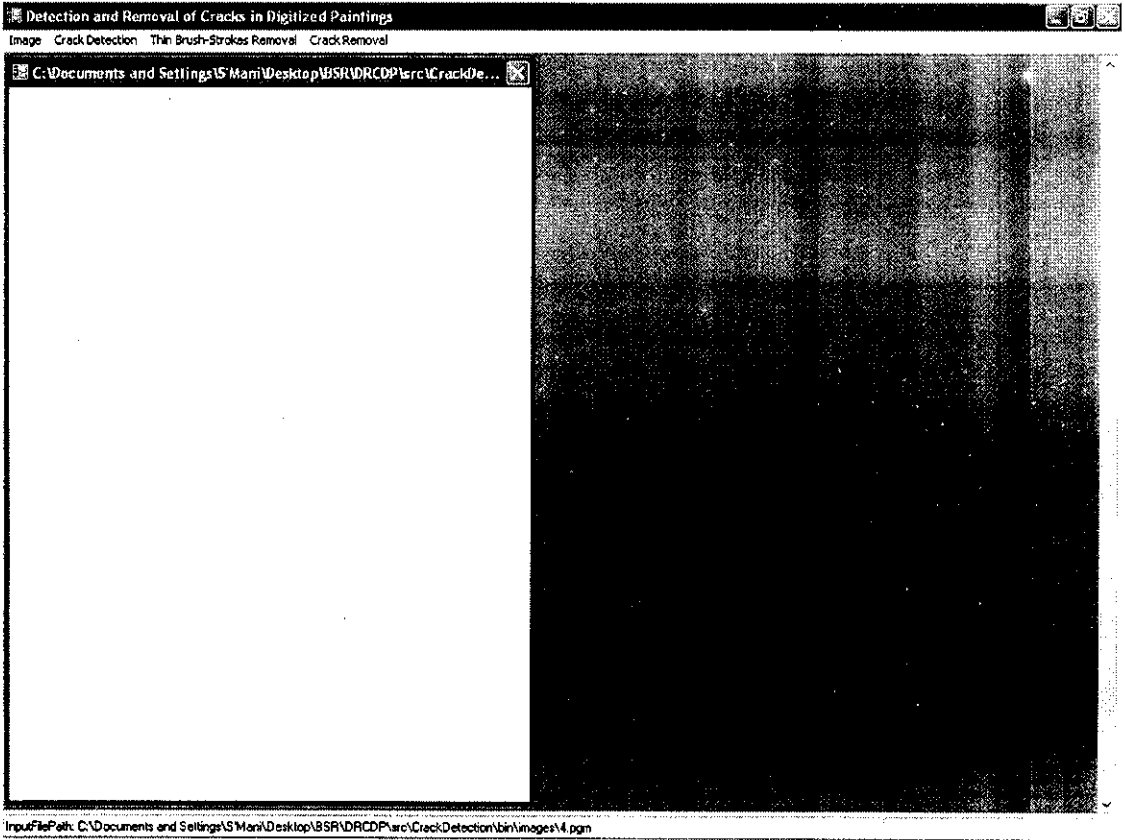
Image Crack Detection Thin Brush-Strokes Removal Crack Removal

C:\Documents and Settings\Mani\Desktop\BSR\DRCDP\src\CrackDe...



InputFilePath: C:\Documents and Settings\Mani\Desktop\BSR\DRCDP\src\CrackDetection\bin\images\4.pgm





REFERENCES

8. REFERENCES

- [1] F. S. Abas and K. Martinez. Classification of painting cracks For content-based analysis. In *Proceedings of IS&T/SPIE's 15th Annual Symposium on Electronic Imaging: Machine Vision Applications in Industrial Inspection XI*, 2003.
- [2] J. V. A. de Boer. *Infrared Reflectography. - A Contribution to The Examination of Earlier European Paintings*. PhD thesis, Univ. Amsterdam, 1970.
- [3] P. de Willigen. A mathematical study on craquelure and other mechanical damage in paintings. Technical report, Delft University of Technology, Faculty of Information Technology and Systems, Department of Mathematics and Computer Science, 1999.
- [4] I. Giakoumis and I. Pitas. Digital restoration of painting cracks. In *Proceedings of the IEEE Int. Symposium on Circuits and Systems (ISCAS '98)*, 1998.
- [5] L. Joyeux, O. Buisson, B. Besserer, and S. Boukir. Detection and removal of line scratches in motion picture films. In *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition* , 1999.
- [6] F. Mairinger. *Strahlenuntersuchung an Kunstwerken*. E.A. Seemann, Berlin, 2003.
- [7] J. Serra. Les treillis visqueux. Technical Report N-51/99/MM, CMM, Ecole des Mines de Paris, 1999.
- [8] P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 1999.