〔非公開〕

TR - C - 0023

Relative Order Determination in Ambigus Moire Pictures

Surface Curvatures Computation in Moire Pictures

オリビエ・クリング　　肥塚　隆
OLIVIER KLING　　TAKASHI KOEZUKA

秋山　健二　　ダニエル・リー　　小林　幸雄
KENJI AKIYAMA　　DANIEL LEE　　YUKIO KOBAYASHI

1989. 1. 6.

ＡＴＲ通信システム研究所

Relative Order Determination in Ambiguous Moire Pictures

Surface Curvatures Computation In Moire Pictures

Internship Report

Olivier KLING

Advanced Telecommunications Research Institute

Osaka - JAPON

20 mars 1988 - 22 novembre 1988

Abstract

The research performed during this internship has been concerned with two main problems with the moire range acquisition system. They are : ambiguous moire pictures interpretation and surface curvatures computation from moire 3D data.

The current moire system can only take pictures of simple objects with no great depth variations. Especially when there are occluding edges in the object, the extracted contours will give incorrect relative orders leading to a distorted or incorrect wire frame model reconstruction. An algorithm has been developed an algorithm to detect inconsistencies in moire patterns using a graph representation of the pattern. That same representation is also used to compute more efficiently the relative orders of the moire fringes. An algorithm has been implemented to correct the inconsistencies of the pattern (such as occluding edges) : it uses a special kind of pixels called "wrong points" which are to be found in high density on the location of pattern inconsistencies. Providing the noise is not too important, it is shown that this algorithm allows a very precise detection of the location of the occluding edge and an accurate computation of relative orders.

The last part of this report deals with the surface curvatures computation. we present a novel surface curvature computation scheme that directly computes the surface curvatures (principal curvatures, Gaussian and mean curvatures) from the equidistance contours without any explicit computations or implicit estimates of partial derivatives. We show how the special nature of the equidistance contours, specifically, the dense information of the surface curves in the 2D contour plane, turns into an advantage for the computation of the surface curvatures. The approach is based on using simple geometric construction to obtain the normal sections and using osculating circles to obtain normal curvatures. It is also general and can be extended to any dense range image data. We show in details how this computation is formulated and give an analysis on the error bounds of the computation steps showing that the method is stable. Computation results on real equidistance range contours are also shown.

ACKNOWLEDGEMENTS

I would like to thank Koichi Yamashita, President of ATR Communication Systems Research Laboratories, for inviting me to his laboratory; Dr. Kobayashi, Head of the Artificial Intelligence Department, who welcomed me in his department and supported my research ; Mr Akiyama, senior researcher, and Mr Koezuka, researcher, who directed my research and gave me constant support and advice.

I would also like to thank Dr. Hisao Kuwabara for taking care of all administrative formalities before and during our stay in Japan, and also Mr Isao Ito, who took care of all our material problems.

I am also greatly indebted to Daniel Lee, supervisor, for providing excellent guidance with my research, helping me in the formulation of this report and correcting it; Mrs Tanaka, visiting researcher, with whom I had many fruitful discussions and who gave me good programming advice and also helped in the correction of this report; Mr Nishino, researcher, for his help and advice, and Mr Ohuchi and Ura, programmers, for the patient and useful help they always gave me about the moire system software.

I would also like to extend this acknowledgment to all the employees of ATR Communication Systems Laboratories and especially to the administrative staff who, through their constant kindness, helped me acquaint myself rapidly with the Japanese way of life.

Table of Contents

Chapter I

Introduction : The Experimental System

I. CONTENTS OF THIS REPORT

I.1 INTRODUCTION

This internship was mostly focused on the moire range acquisition system and its improvement. Two main problems were studied : determination of relative orders in ambiguous moire pictures and surface curvatures computation from moire 3D data.

I.1.1 The First Problem

The current moire system can only take pictures of simple objects with no great depth variations. Especially if there are occluding edges in the object, the extracted contours will give incorrect relative orders leading to a distorted or incorrect wire frame model reconstruction. Thus the problem to be solved was the following :

> Is it possible to compute correct relative orders from a moire pattern containing occluding edges ?

The answer is "yes" : it is possible to detect occluding edges and some other deteriorations of the fringe pattern, and to correct them so as to compute accurate relative orders, providing there is enough information in the picture.

For that purpose a new algorithm for relative order computation was developed : this the subject of chapter II.

The detection of moire pattern inconsistencies and the moire pattern correction algorithms are the subject of chapter III.

I.1.2 The Second Problem

We want to use the moire pictures for object recognition and 3D data bases building. For that purpose we need to segment the surface in order to extract its most stringent features. However the moire pictures provide us with view-dependent 3D data : these pictures are relative to the view point from where the picture has been taken. That is not siutatble for surface analysis : we need to extract the intrinsic properties of the surface, that is those which are invarient with respect to the view point. The most

important of these intrinsic properties are the principal curvatures from which other important properties such as Gaussian and mean curvatures are derived.

The problem can then be formulated as :

> Is it possible to easily extract intrinsic features such as Gaussian curvature directly from the view-dependant moire data without resorting to surface fitting techniques (which are too computational intensive and sometimes inaccurate) ?

Here again the answer is "yes" : an algorithm was developed which allows surface curvatures computation in a simple fashion without computing second order derivatives.

I.2 THIS CHAPTER

The main purpose of this introduction chapter is to provide enough information on the moire system to explain the following chapters.

I.3 CHAPTER II

In this chapter a new algorithm for relative orders computation is described : a graph representation of the moire pattern is computed. Using a depth-first search recursive algorithm, the relative orders are computed. The picture and the graph need to be filtered in order to get good results.

I.4 CHAPTER III

In this chapter two algorithms are described. The first of them is designed to detect the inconsistencies into the moire pattern graph. If inconsistencies are detected the picture needs to be corrected : the second algorithm is designed to precisely locate into the picture the occluding edge and other distortions of the moire pattern : this approximation of the occluding edge is done by searching for a new feature called *wrong points* in the moire picture : these points are present in high density around places where the pattern is distorted. These defects are then corrected and the relative order determination algorithm can be used safely.

This algorithm has been tested on moire picture of a tea cup and on simple polyhedron scenes where it showed good performance. The time needed to check the picture, correct it and generate the relative orders and the coded contours is for example 3 minutes of CPU time on a VAX 8600 for the tea cup of figure III-1.

I.5 CHAPTER IV

This chapter describes the implemented algorithm to extract from a moire picture the principal curvatures and directions, the Gaussian and the mean curvature.

Some error bounds on the curvatures have been found which shows that the error is bounded by the moire system physical resolution or under control.

II. THE MOIRE SYSTEM

II.1 INTRODUCTION

The moire topography system has been developed in order to automatically acquire the 3D shape data of an object. The ultimate purpose is to develop the basic techniques for feature extraction, 3D recognition and 3D databases acquisition.

The moire topography allows you to acquire with a single shot all 3D data of an object view. A regular grating is projected on the object and a picture is taken by a CCD camera. After extracting the periodic components of the grating, a digital grating is superimposed on the picture and a low-pass filter is applied allowing moire fringes only to be collected. We obtain a picture with fringes which represent slices of the object at different depth levels from the camera : all 3D information lies within these fringes.

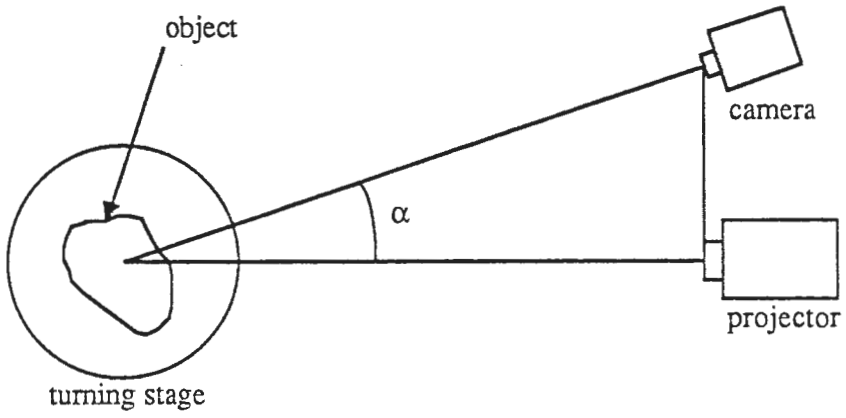


Fig. I-1 : the experimental system.

II.2 TAKING THE PICTURE

II.2.1 Moire Fringes Generation

As seen above, a regular grating is projected on the object. An image of the object is taken by a camera at an angle α with respect to the projector : the picture shows the deformed grating.

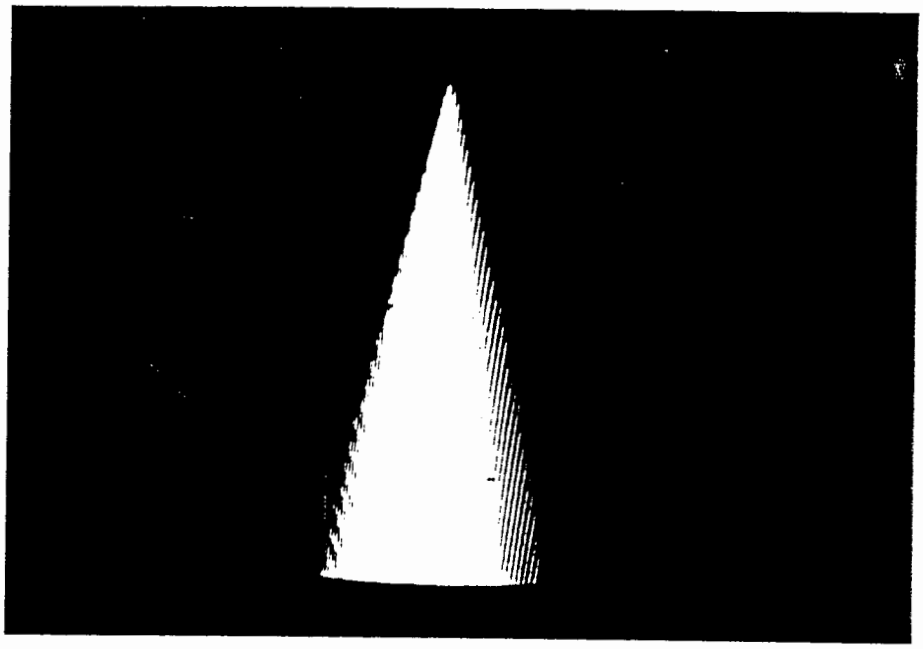


Fig. I-2 : a picture taken by the camera of a deformed grating pattern.

That grating pattern is then multiplied with a digital grating by a graphic processor :

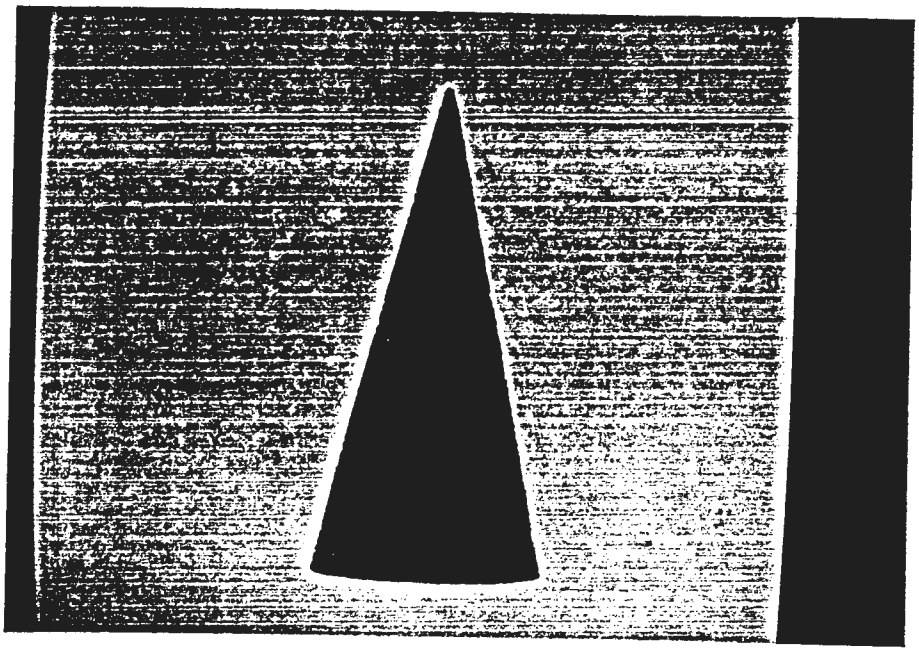


Fig. I-3 : generation of the moire pattern from the deformed grating and the digital grating.

After multiplication a low-pass filter is used to get rid of both digital and deformed gratings. During filtering the fringes are left unchanged in the picture as they belong to the low-frequency part of the picture

II.2.2 Result

The picture is then thresholded : high intensity pixels become red and low intensity pixels become black, leaving only red fringes or black fringes.

Each fringe contour is the countour of a slice of the object at a certain distance from the camera.

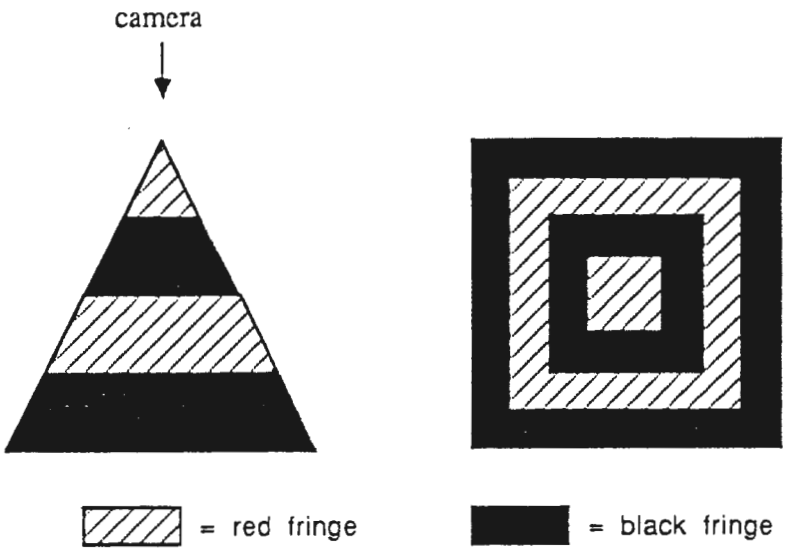


Fig. I-4 : a moire picture of a small pyramid.

II.2.3 Problems

As can easily been seen from figure I-4, it is not possible to detect if an object is concave or convex from the fringe pattern on the right. Below is another way of interpreting the moire pattern of figure I-4.

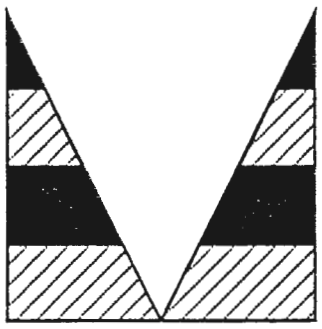


Fig. I-5 : another possible interpretation of moire pattern of figure I4.

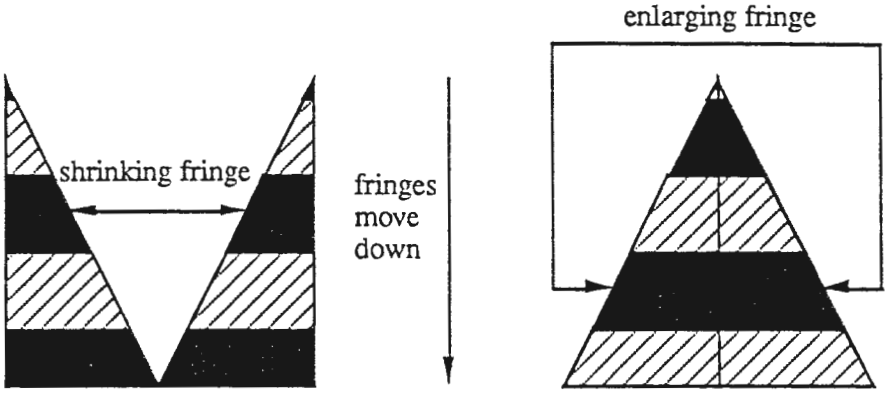
The moire pattern does not contain any information about the convexity of an object surface.

II.3 CONCAVITY AND CONVEXITY DISCRIMINATION

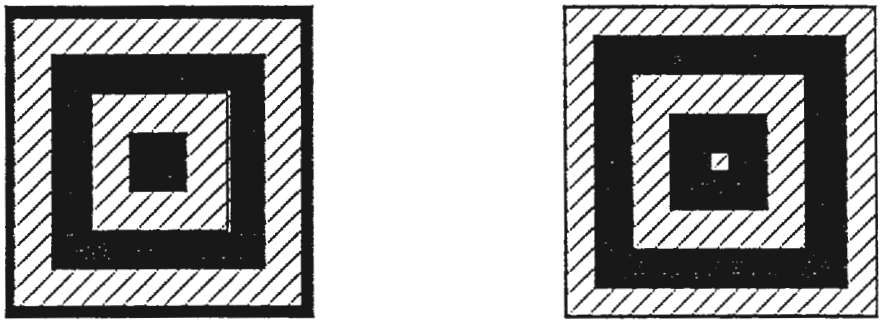
II.3.1 Principle

If the projector grating or the digital grating is shifted horizontally, the moire fringes will also be shifted on the object surface : they will all move in the same direction (toward or from the camera). We use an angle to measure the shift : 2 π means the shift is one period long, and thus nothing changes

Let us say for example that all fringes move up (= toward the camera) when the grating is shifted by an angle β. The fringes on a convex part will then shrink while those on a concave part will be enlarged : the way the fringe will move is then a good signature of its convexity properties. Here is the evolution of fringes for the two previous models from figure I-4 and I-5 :



corresponding fringe patterns



shrinking fringes enlarging fringes

Fig. I-6 : modification of fringe pattern according to convexity of surface.

Moire fringes are very sensitive to phase shift. So a small amount of shift ($< \pi$) is enough to obtain the phase shifted fringes.

II.3.2 Realization

As seen previously we chose a small angle of shifting : $\frac{\pi}{2}$ (see [Koezuka et al. 87]).

We take one picture of the deformed grating on the object and decode it twice using a digital grating and a phase-shifted grating : we get two moire pictures of red and black fringes.

We then superimpose the two pictures : the resulting picture is computed according these rules :

original fringe	shifted fringe	resulting fringe
red	red	blue
black	red	green
red	black	red
black	black	black

Below is a figure showing the results of the computation :

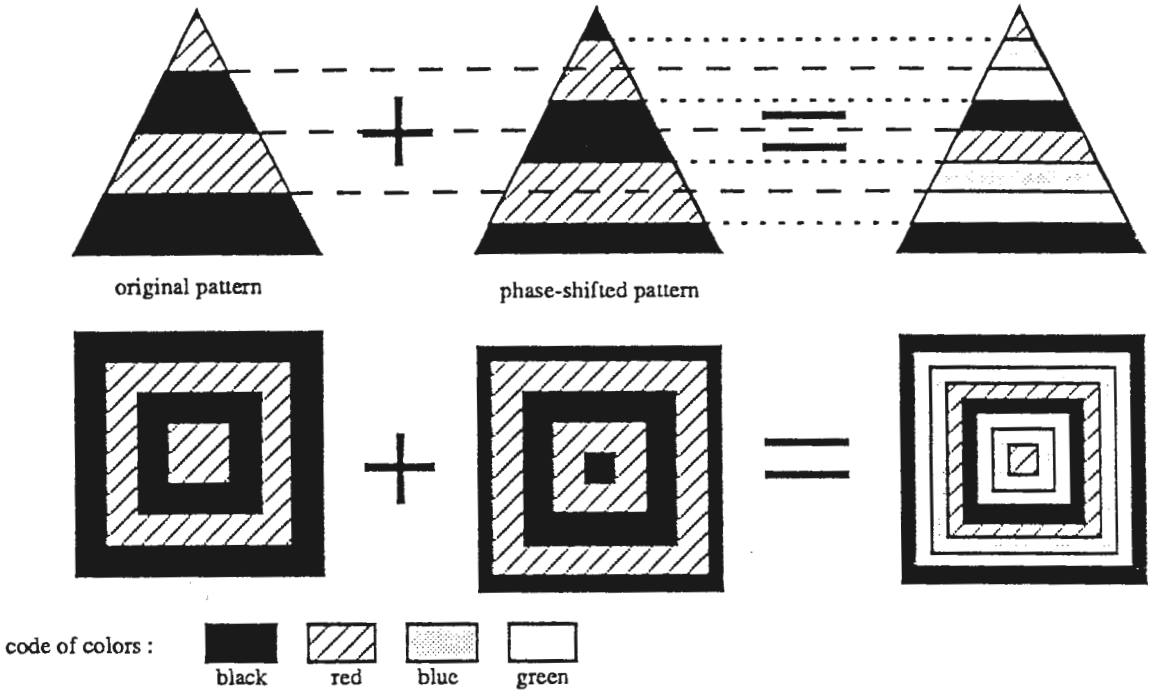


Fig. I-7 : generation of green and blue fringes by phase shifting of 1/4 th of period.

It can clearly be seen on the picture that fringe colors are ordered in a precise fashion : for example a red fringe is always surrounded by blue and black fringes and it will always be farther from the camera than its black neighbors and closer to it than its blue neighbors. We can then recognize convex from concave surfaces.

II.4 INTERPRETING THE DATA

The phase shifting lifts the ambiguity between convex and concave surfaces. The following figure shows the fringe pattern for the concave surface of figure I-5 : it is very

different from the convex surface pattern and makes apparent the concavity of the surface.

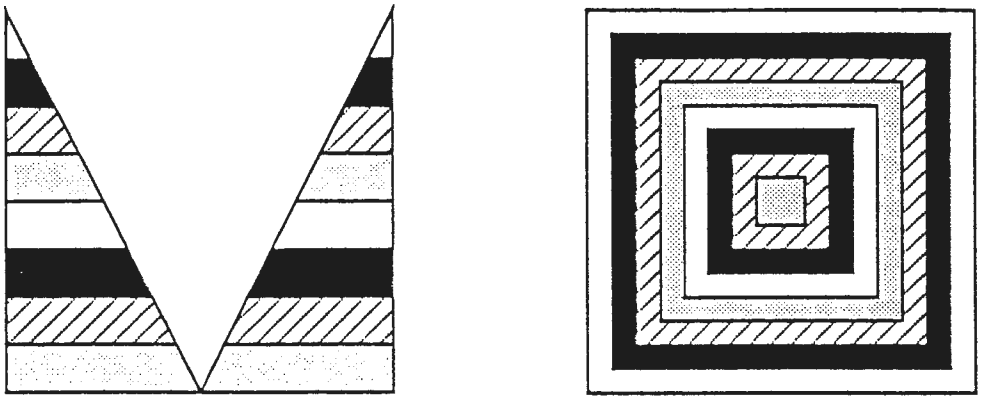


Fig. I-8 : concave surface fringe pattern : the order of color (opposite to convex pattern) allows a concavity discrimination.

The relative position of all fringes toward its neighbors is uniquely defined : we can order adjacent fringes by determining which one is closest to the camera. Taking one fringe as reference and giving it the level 0, we can give a label to all fringes in the picture : this label is called the relative order because it depends on the level you give to the reference fringe and on that fringe also.

After computing the relative orders we want to get the real distance from each fringe to the camera. It is enough to know the distance of only one fringe to the camera : the absolute order of that fringe can be determined using the moire formula (I-1) given below :

$$(I\text{-}1) \qquad \boxed{Z_N = a' + \frac{a'\,l}{p\,S\,N}}$$

Where : a' = focal length of camera

l = distance between camera and projector

p = CCD pixel size (currently 0.0068 mm)

S = sampling rate : period of the digital grating : 2 or 6 pixels

That fringe is then taken as reference fringe, the relative orders are determined relatively to that fringe and its absolute order : these orders are then absolute orders. Used in the same formula they give the exact depth of each fringe

Chapter II

Relative Order Determination

I PURPOSE OF THE STUDY

The existing algorithm gives wrong results if applied to some specific moire patterns (occluding edges, noisy pictures). A new algorithm has been designed to compute relative order more quickly and more effectively, taking advantage of the information gathered for moire pattern consistency checking (see chapter III).

In the whole study we will deal only with "connex" moire patterns: e.g., the pattern cannot be separated completely by background pixels into two independent sets of fringes.

A graph is computed from the labelled picture according to the same rules used in [Cline et al. 84]. It is then used to compute the relative orders.

II ALGORITHM

II.1 LABELLING

The labelling algorithm is similar to the one already implemented in the moire system. The old algorithm uses different kinds of labels for each of the four basic colors (red, black, green and blue). This algorithm gives all fringes a label which is independent from their colors.

The picture is segmented in "fringes" : *a fringe consists of a set of pixels of same color which are connected together by a neighborhood of 4 or 8 pixels*. Two algorithms have been implemented : one using the 4-pixel neighborhood and the other using the 8-pixel neighborhood.

The picture is scanned horizontally. At a pixel P the neighbors which have already been labelled are scanned : these are named in the picture below A, B, C and D.

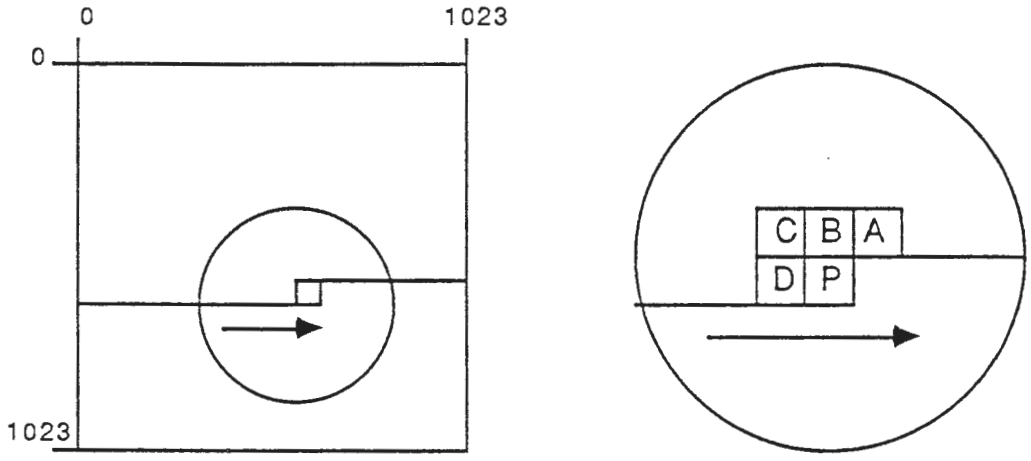


Fig II-1 : a) scanning of the picture, b) when labelling pixel P the neighbors are : A, B, C, and D pixels.

According to their color and label the pixel P is given a label and if necessary some fringes are merged and renumbered.

Once the picture is labelled we have some more information on the moire pattern : the picture is segmented into a set of fringes. For each fringe we know its size in pixels, its color and we have the exact coordinates of one pixel belonging to that fringe. We can even access all pixels belonging to a certain fringe using a recursive algorithm such as this one :

```
visit(x , y , fringe_label)
{
      /* mark (x , y) pixel as visited */

      work[y][x] = 1 ;


      /* visit the other pixels of neighborhood */

      for( all pixels (x' , y') of (x , y) neighborhood )
         {
         if ( label[y'][x'] == fringe_label && work[y][x] == 0 )
            {
            visit(x' , y' , fringe_label) ;
            }
         }
}
```

In this function "label" is the 1024 by 1024 array representing the labelled image : label[y][x] gives the label of pixel (x , y). "work" is an array of same size initialized to 0 and used to record the visited points.

II.2 GRAPH STRUCTURE

We now want to gather some information on the relations between fringes : which fringe is in contact with which fringe ? ; what is the size of the area of contact between two fringes ?

Two fringes will be said adjacent if they share a common edge. In the picture below we can see the following relations between fringes :

- fringe 1 is adjacent to fringe 2.

- fringe 2 is adjacent to fringe 1, 3 and 6.

- fringe 3 is adjacent to fringe 2 and 4.

- fringe 4 is adjacent to fringe 3 and 5.

- fringe 5 is adjacent to fringe 4.

- fringe 6 is adjacent to fringe 7 and 2.

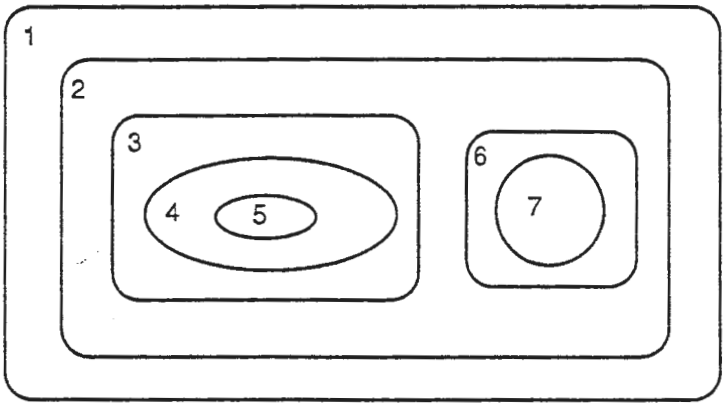


Fig. II-2 : a simple moire pattern.

From the labelled picture a graph is generated. There are two features :

nodes : they represent a fringe.

links : a link is established between fringes A and B when they are adjacent. The weight of the link is proportional to the length of the common edge between the two fringes.

The graph of moire pattern of Fig. II-2 is shown below.

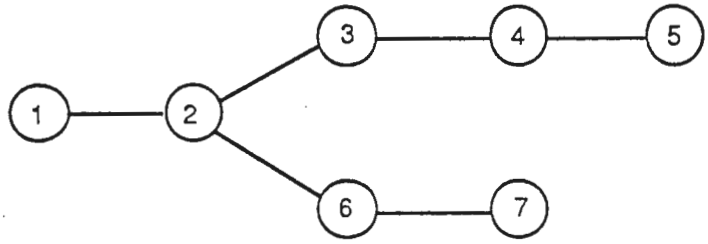


Fig. II-3 : the graph derived from Fig.II-2 moire pattern : each link has a weight proportional to the area of contact between the fringes.

But the fringes represent slices of the object at different depth-levels. The links are thus oriented according to the depth of the fringes. So far we don't know the exact depth of the fringes ; but we can easily determine the depth difference (or relative depth) of two adjacent fringes. We only have to consider the colors of the fringes. The figure below shows the relative depth scale deduced from the color of the fringes (see chapter I).

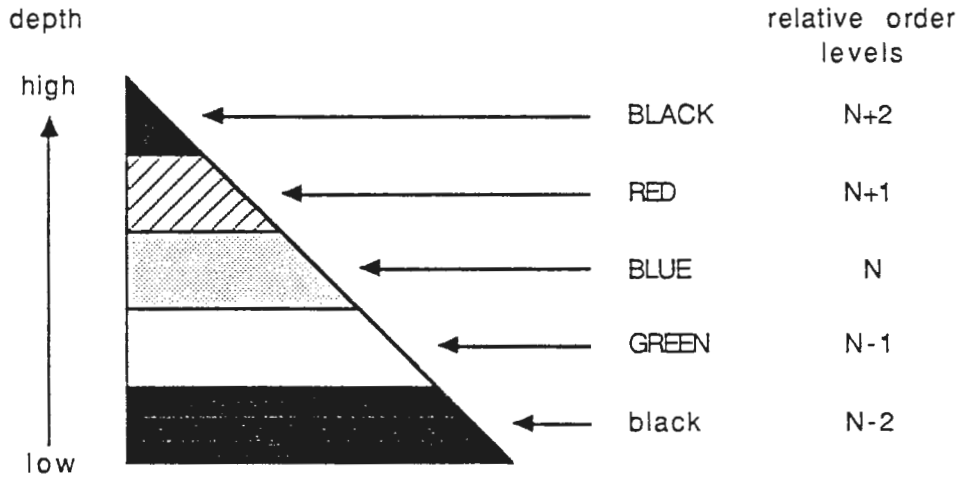


Fig. II-4 : relative depth scale according to colors.

An example of the simple moire pattern colors (see up) is given here :

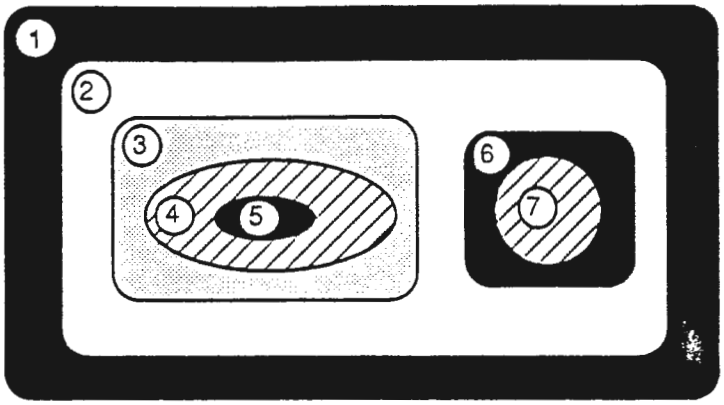


Fig. II-5 : a simple moire pattern with indication of depth levels.

Adding the depth relations in the graph we get the following :

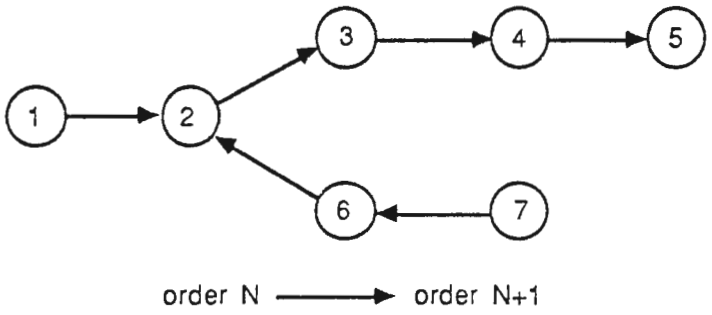


Fig II-6 : the complete directed graph of the simple moire pattern from Fig. I-5.

Here is the C-structure used to represent a fringe in the implemented program :

```
struct fringe {
        short number ;          /* label of the fringe                          */
        short color ;           /* color of the fringe                          */
        short level ;           /* relative order of the fringe                 */
        short visited ;         /* visit flag used in graph search              */
        short nbr_high ;        /* nbr. of links to higher fringes              */
        struct fringe *higher[20] ; /* array of pointers to other fringes       */
        short nbr_low ;         /* nbr. of links to lower fringes               */
        struct fringe *lower[20] ;  /* array of pointers to other fringes       */
        } ;
```

II.3 RELATIVE ORDER DETERMINATION

We call the graph "consistent" if it is possible to give to each node a relative order which is not conflicting with the links. For example the graph of figure II-6 is consistent : here is a set of relative orders which shows it :

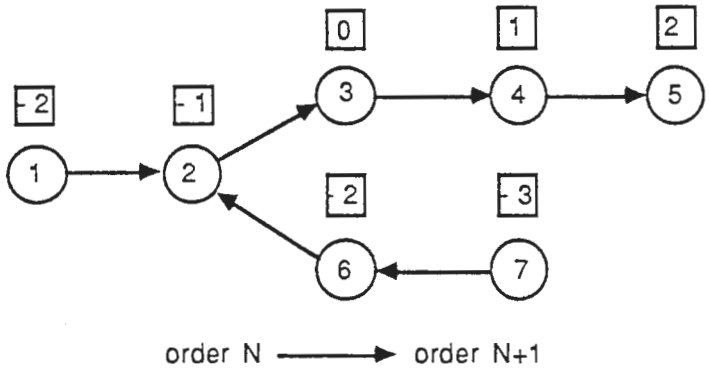


Fig. II-7 : an example of relative orders on a consistent graph.

A classical depth-first search of the graph is made:in the classical algorithm the search of the graph goes trough each node once only. The computation cost grows linearly with the number of links (for details see [3]). The implemented algorithm recursively visits the graph in the same fashion : the function also gives a relative order N to a node when entering it. It then visits all lower unvisited nodes giving them the order N-1 and all upper unvisited nodes giving the order N+1.

```
visit_fringe( s , level )
s is a pointer to a fringe node structure ;
level is the relative order of s
{

if ( s->visited == 1 )        /*  if this fringe has already been visited        */
      {
      if ( s->level != level )/* if previous order is different from "level"    */
         {
         /* there is something wrong in the picture : record fringe            */
         /* number and level difference */

         exception++ ;
         strange_fringe[exception][0] = s->number ;
         strange_fringe[exception][1] = s->level ;
         strange_fringe[exception][2] = level ;
         }
      return ;
      }
else                          /* this fringe hasn't been visited yet            */
      {
      s->visited = 1   ;      /* this fringe is being visited                   */
      s->level = level ;      /* record relative order                          */

      for( all lower fringes pointers low_s )  /* visit all lower fringes       */
         {
         visit_fringe( low_s , level-1 ) ;
         }

      for( higher fringes pointers high_s )    /* visit all higher fringes      */
         {
         visit_fringe( high_s , level+1 ) ;
         }

      return ;
      }

}
```

If a node has already been visited the function also controls if its relative order is coherent with that of the present node. If not the graph is declared "inconsistent" and the number and location of detected inconsistencies are recorded. Below is an example of inconsistent graph. It is impossible to find a set of relative orders consistent with the links of the graph.

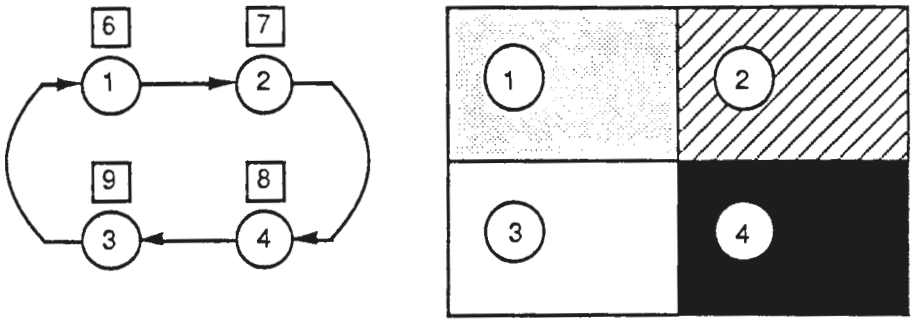


Fig. II-8 : left : inconsistent graph of the right moire pattern.

In figure II-8 the set of relative orders { 6 , 7, 8 , 9 } is not consistent with the left link which "jumps" from order 9 to order 6.

The implemented algorithm detects those inconsistencies.

II.4 FIRST RESULTS

For moire patterns the graph of which is not inconsistent, the algorithm successfully determines a set of relative orders.

But in some pictures the fringe pattern degenerates : the surface of the object is too steep or too dark preventing an accurate computation of the pattern. At those locations the fringes disappear in the background noise : a random distribution of small dots of 4 colors. Lots of inconsistencies appear in the moire pattern and in the graph as can be seen in the following picture :

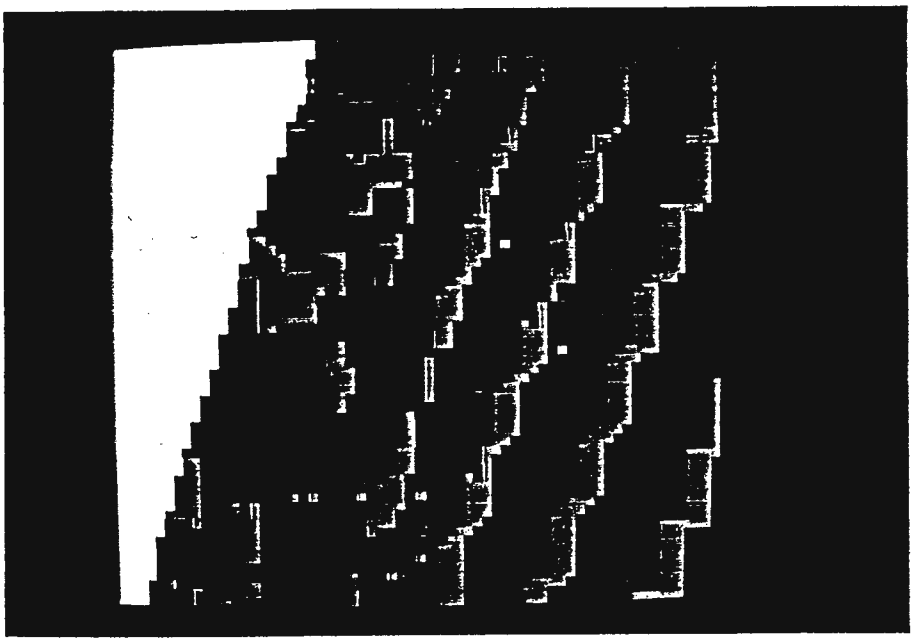


Fig. II-9 : an example of degenerated moire pattern on a steep surface.

It is then impossible to determine the relative orders from such data: it has to be filtered first.

III FILTERING

III.1 PROBLEM

As seen from above some noise can appear in the moire pictures. This noise consists of a random pattern of small fringes. This pattern leads to a graph representation being inconsistent and containing unnecessary data. The picture and the graph have to be filtered before relative order can be computed.

III.2 FILTERING : SIZE OF THE FRINGES CRITERIA

This first technique is a filtering of the moire pattern itself : the original picture is modified to eliminate unnecessary data. One way to get rid of the noise in the picture is to delete all fringes the size of which is less than a given threshold.

The implemented algorithm checks the size of all fringes in the picture and deletes those which are too small from the data structure. A recursive function is used to go through these fringes and turn their pixels back to yellow (= background color).

This method is efficient in suppressing the noise in dark parts of the object: there the moire pattern is a random pattern of small fringes.

III.3 FILTERING : SIZE OF THE GRAPH-LINKS CRITERIA

Depending on the orientation of the object surface the noise cannot be substantially reduced by the previous filtering at some angles. Here is an example of such moire pattern :

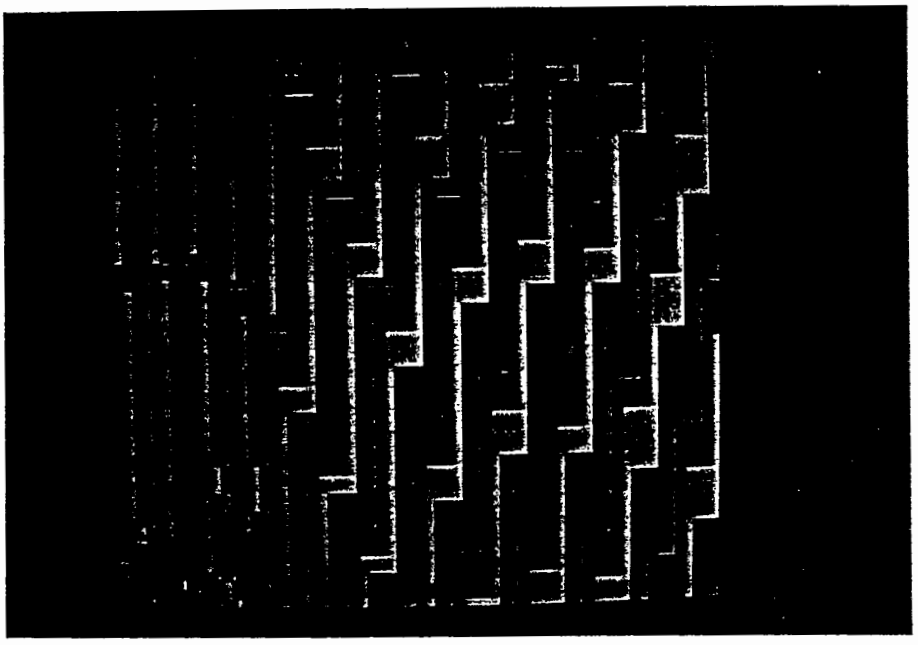


Fig. II-10 : noise on a steep surface : long fringes with small holes.

The pattern consists of long fringes disconnected from times to times. The links which appear in those holes are of two kinds :

"wrong" links : these are links between two fringes of non-compatible colors (e.g. , the relative order difference between these fringes is more than 1).

small links : the surface of contact between fringes through these holes is small.

So when computing the graph representation only the biggest links are used. Under a threshold value the links are considered as non-significant and are deleted. Furthermore all "wrong" links are ignored.

This second filtering technique is applied to the graph: the graph is modified but not the moire pattern : there is no loss of information and it is always possible to come back to the original data.

IV RESULT

IV.1 LOSS OF RESOLUTION

In order to efficiently filter the moire pattern and the graph the thresholds for both filtering technique have to be set at a high level.

If the thresholds for the size of fringes and links are set at about 30 pixels we will be able to "see" only the fringes which are 30 pixels or bigger even if the physical resolution of the experimental system is better than this.

IV.2 RESULTS

This algorithm successfully computed the relative orders of various moire patterns providing the threshold levels where appropriately set. The thresholds used in the computation were taken twice as big as the noise fringe size, that is between 30 and 50 pixels.

Chapter III

Occluding Edges Detection and Correction

I. PROBLEM

I.1 WHAT ARE OCCLUDING EDGES ?

Occluding edges appear on non convex objects when one part of the object is hiding another part : in the picture of the tea cup below there is an occluding edge at the boundary between the body and the handle.

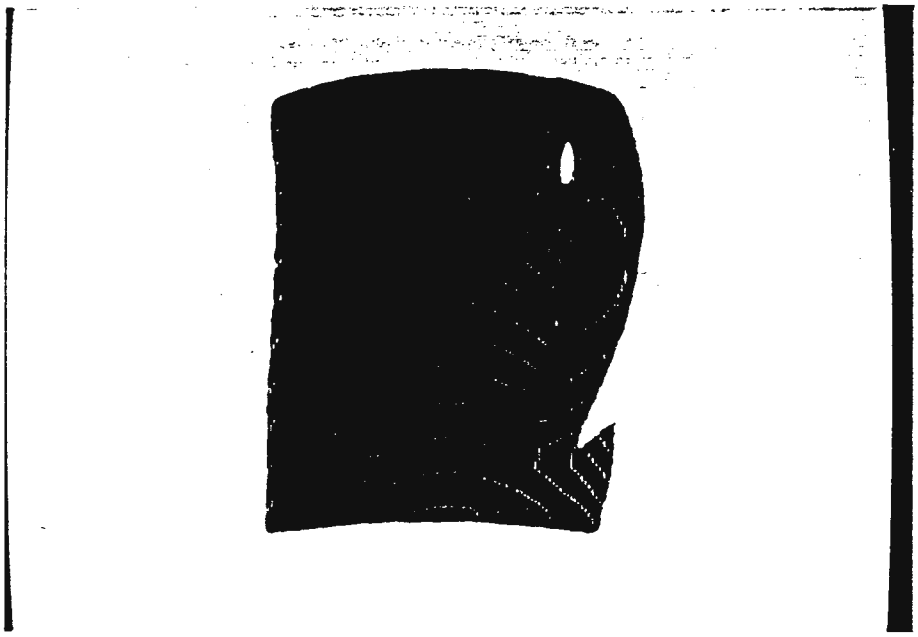


Fig. III-1 : a moire picture of a tea cup with an occluding edge.

I.2 LIMITATIONS OF MOIRE TOPOGRAPHY

Using moire photography one can access the relative depth of an object. But as this information is relative, there are cases where a moire picture is completely ambiguous, preventing one from getting any useful information from it. An easy example is that of two parallel planes separated by a distance equivalent to one order. The fringe pattern will be strictly identical on both planes and continuous from one to the other.

This is what shows the picture below : a dodecahedron is behind an octahedron. The front sides of both objects have exactly the same direction, preventing any accurate segmentation of the surface.

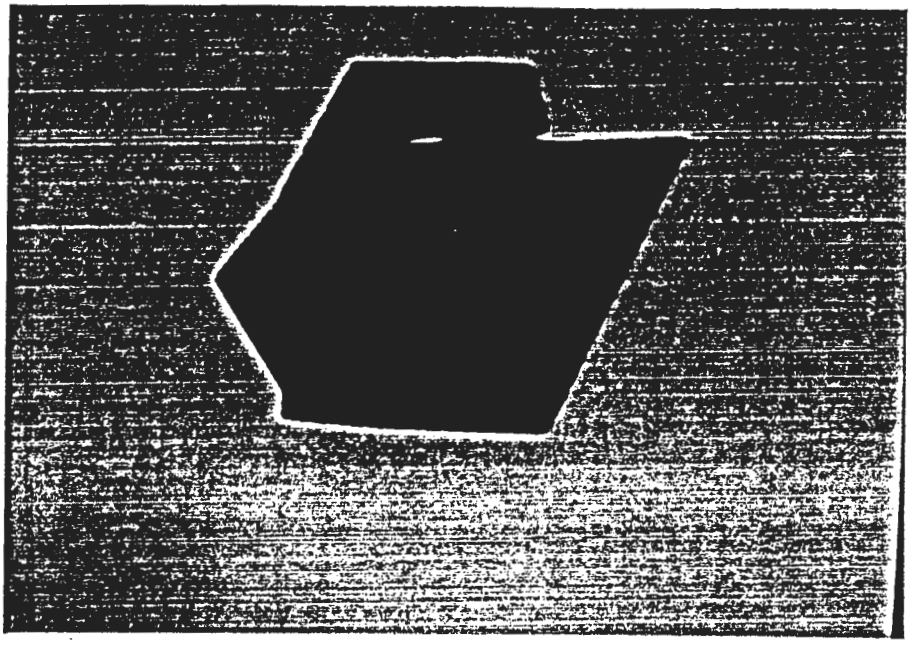


Fig. III-2 : an ambiguous moire picture of a dodecahedron hidden by an octahedron.

II. OBJECTIVE

II.1 OCCLUDING EDGES IN MOIRE PICTURES

An occluding edge will be easy or difficult to detect in a moire picture depending on two factors : the direction of the two surfaces on each side of the edge and the spacing between them.

The closer the directions of the planes are, the more difficult it will be to see the occluding edge: the patterns will be exactly the same on both surfaces. But we will also hardly detect it if the distance along the occluding edge is constant and equal to an integer number of orders : the color of both surfaces will be the same and the edge will not be detectable. The two surfaces seem to merge into one big continuous surface.

This generates very ambiguous pictures such as in figure III-2 : it is not possible to find the real shape of the object without collecting additional data.

II.2 OBJECTIVE OF IMPLEMENTED ALGORITHM

In order to get a picture such as figure III-2 the angle and position of objects have to be finely adjusted. A small variation of the parameters clearly reveals the edge : in the following picture (Fig. III-3), the octahedron has been slightly translated toward the camera. The edge is much more easily detected. Figure III-4 shows the octahedron rotated by an angle of 30 degree: the edge is also much clearer.

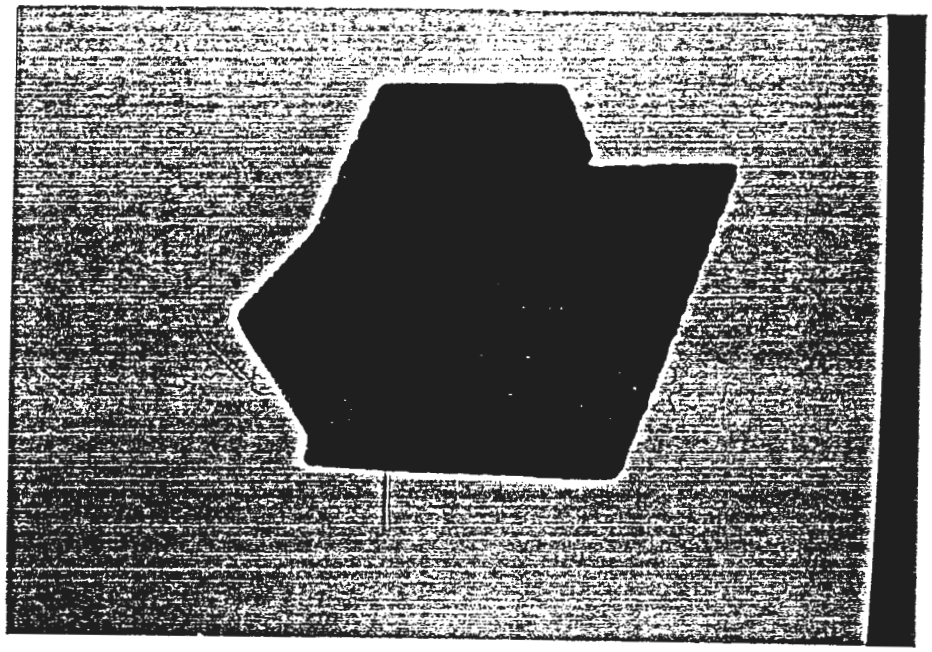


Fig. III-3 : translated octahedron from Fig. III-2.

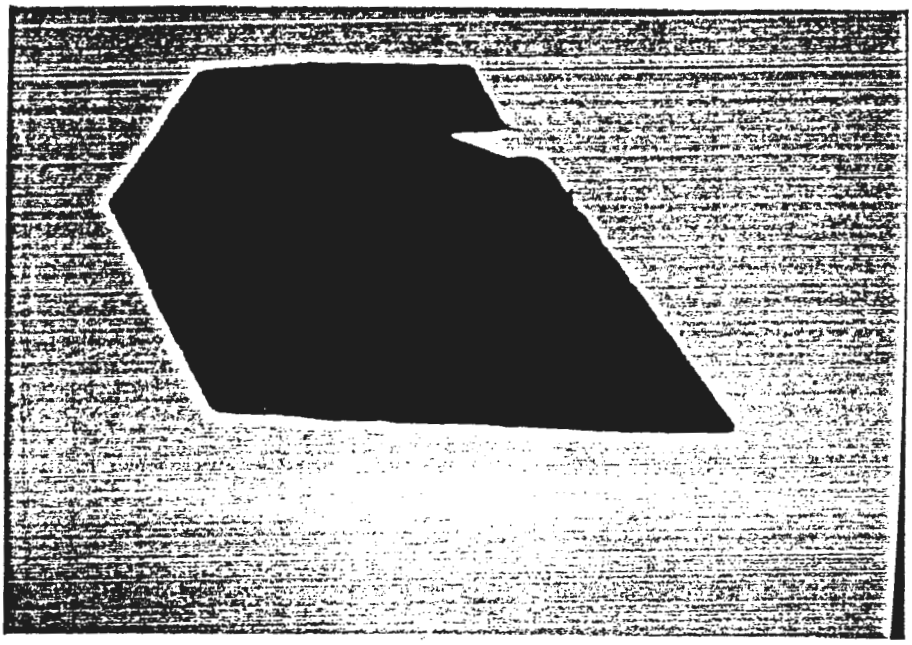


Fig. III-4 : rotated octahedron from Fig. III-2.

What usually happens is that, along the edge, the two patterns are sometimes connected and sometimes disconnected : these are the cases the implemented algorithm can deal with.

The first part of the algorithm is the occluding edge detection : this can be easily done by a graph consistency check. When there is an occluding edge in the picture, the graph is no longer consistent.

But in order to be able to determine accurately the set of relative orders the algorithm needs to locate precisely the edge in the picture. It can do so if the picture is not too ambiguous : if the moire patterns on each side of the edge are very similar (such as in figure III-2) the algorithm won't be able to detect the edge. But if the patterns are a little different the detection becomes easy. The second part of the algorithm is designed to locate the edge as accurately as possible and to use this information to correct the moire pattern and to compute the good relative orders.

III. DETECTION OF OCCLUDING EDGES

III.1 OCCLUDING EDGES MODIFY THE GRAPH REPRESENTATION

The occluding edge in the picture is detected by checking the graph consistency : the fringe pattern is disturbed by the edge and becomes inconsistent.

When two different moire patterns meet at an occluding edge some fringes terminate abruptly and some merge with a fringe of the other pattern : this is shown in the figure below.

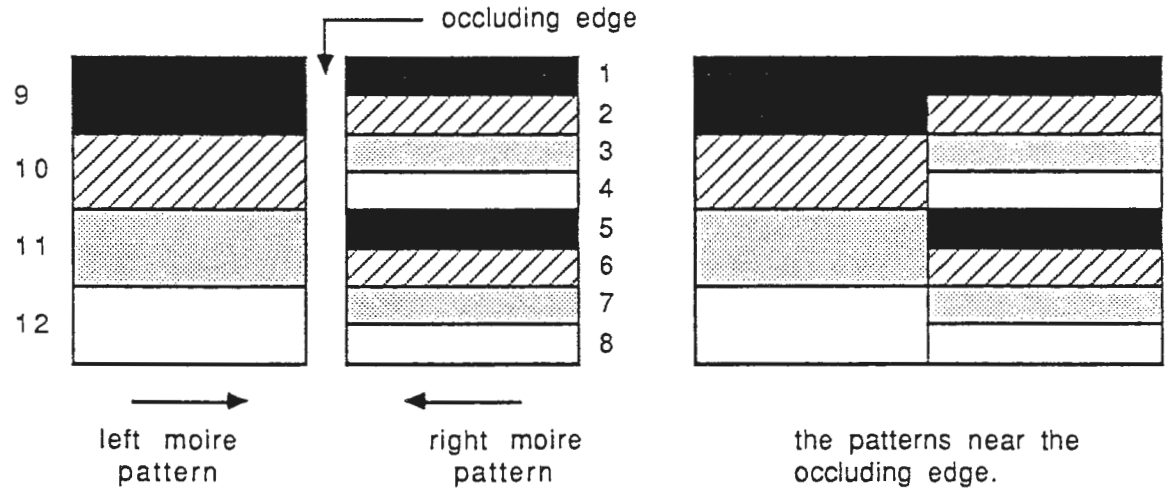


Fig. III-5 : at the occluding edge fringes from both sides of moire pattern merge together.

The fringes 1 and 9 merge and make only one fringe although they might be of different relative depth.

If we try to determine the relative orders for that pattern we get the following graph :

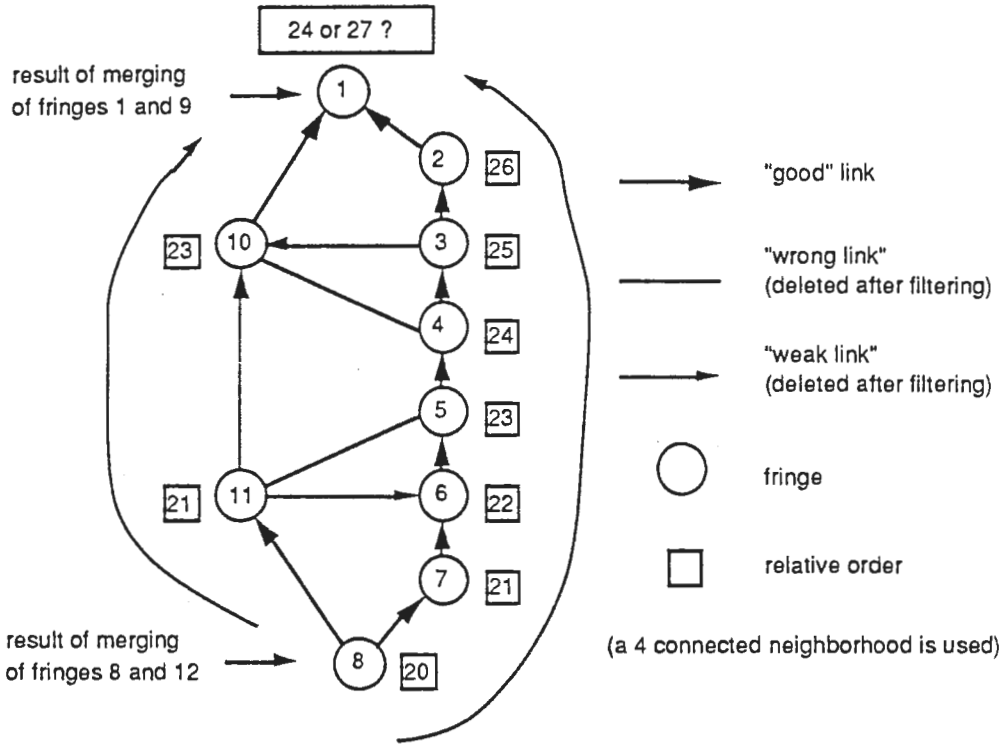


Fig. III-6 : graph of fringe pattern from figure III-5 : accurate relative order computation is impossible. The graph is inconsistent.

Instead of having on each side of the occluding edge two independent sets of fringes we now see some fringes of the same color but of different relative order merge together.

Reflecting the change of the moire pattern, the graph also changes : some nodes of different relative depth merge and thus inconsistencies appear.

The graph is no longer consistent. Note that after detecting the presence of the occluding edge we have no idea of where it lies in the picture : we know only a set of fringes which lie close to it. But as we don't use any representation of the topology of the fringe it is not possible to locate the occluding edge by only using the graph

information. It is precisely that information which we need to correct the picture. Another algorithm is needed for that purpose.

III.2 CONCLUSION

There is a one to one correspondence between the graph consistency and the fringe pattern consistency. This inconsistency of the graph comes either from noise (degenerated fringes or bad contouring) or from occluding edges.

Provided that the size of the fringes of the occluding edge is much bigger than the filtering threshold, the only inconsistencies left in the moire pattern come from the occluding edge. We will assume from now on that all pictures used have been filtered or taken with enough care so that the fringe pattern is noiseless : thus checking the graph consistency is enough for detecting occluding edges.

IV. CORRECTION OF THE MOIRE PICTURE

IV.1 PURPOSE AND PRINCIPLE OF FRINGE PATTERN "CORRECTION"

We want to be able to determine the set of relative order of a moire pattern even when there are occluding edges inside. We will assume that the moire patterns from both sides of the edge are different enough so we can localize the edge easily.

We will first approximate as closely as possible the occluding edge with a set of segments and then use this approximation to separate the two sets of fringes along the computed edge by splitting in two the merged fringes. This corresponds to the splitting of the nodes in the graph.

When all fringes have been corrected this way the graph should be consistent and we should be able to determine the relative order of fringes.

IV.2 DISCONTINUITIES OF FRINGE PATTERNS NEAR OCCLUDING EDGES : WRONG POINTS DETECTION

On both sides of an occluding edge the frequencies of the fringe pattern are different : the two regions of different frequencies join along the edge ; the pattern is strongly distorted along that line ; some fringes disappear, some are bent, some merge as seen in figure III-5.

If the occluding edge is long enough, necessarily, two fringes will come into contact and create an illegal link. We search the picture for pixels involved in these links: we call them "wrong pixels" : a pixel is classified as *wrong point* when there is in its neighborhood, a pixel of forbidden color (for example a black pixel in the neighborhood of a blue pixel). These wrong points are always present where pattern inconsistencies occur: and can be used to detect a wide range of inconsistencies in the picture : on a distorted and noisy part of the pattern, the distribution of wrong points will be 2-dimensionnal (see paragraph VI). Near an occluding edge the distribution will be 1-dimensionnal.

The next figure shows how the wrong points are distributed along a simple occluding edge.

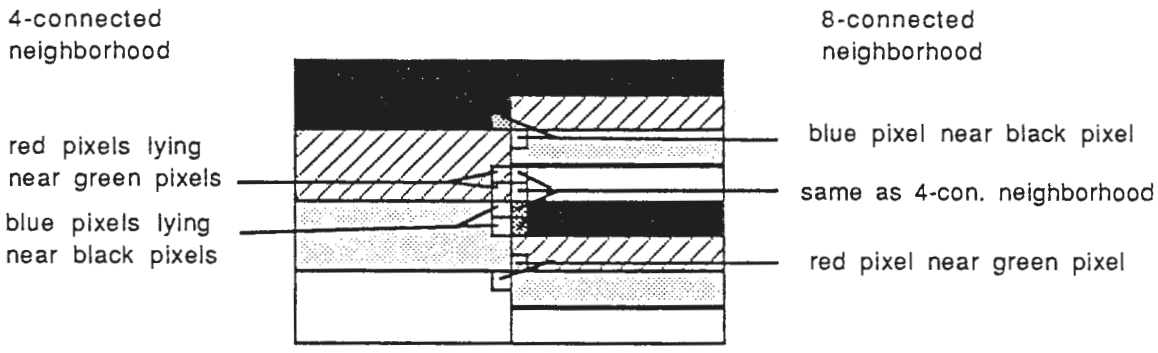


Fig. III-7 : wrong points along a simple occluding edge pattern.

For detecting wrong points we can either use a 4 or a 8-connected neighborhood. The latter has been chosen for two reasons : all pixels which are wrong points for the 4-connected neighborhood are also wrong points for the 8-connected. The reverse is not true. The 8-connected neighborhood provides us with more data and thus with more information on the moire pattern. Second the 8-connected neighborhood allows us to detect inconsistent moire patterns such as the one from figure II-8. Using a 4-connected neighborhood wouldn't allow us to detect wrong points in these patterns.

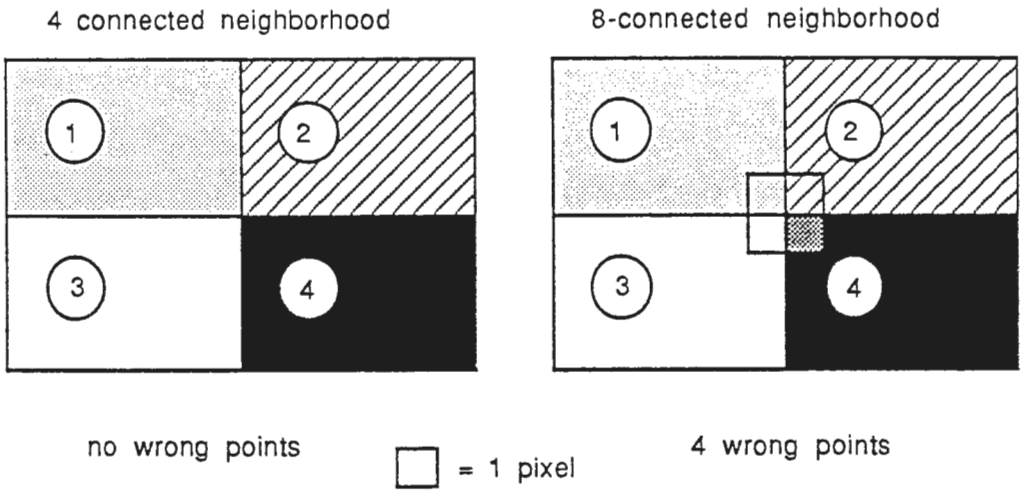


Fig. III-8 : detection of wrong points in inconsistent moire pattern for 4 and 8-connected neighborhood.

So these "wrong" pixels are a good signature of the occluding edge. Provided that the occluding edge can be detected these wrong points always appear along it : they are thus a good "signature" of the edge. If the patterns are very different on each side of the edge, the density in wrong points will be high. It will be much lower if the patterns are similar.

It must be noted that these wrong points are also to be found in noisy parts of the picture. In order to get rid of these unnecessary points we first filter the moire picture and then look for wrong points : thus most of the points we find are closely related to the

presence of the occluding edge. The next figure shows an example of a search for wrong points in a moire picture of a tea cup.

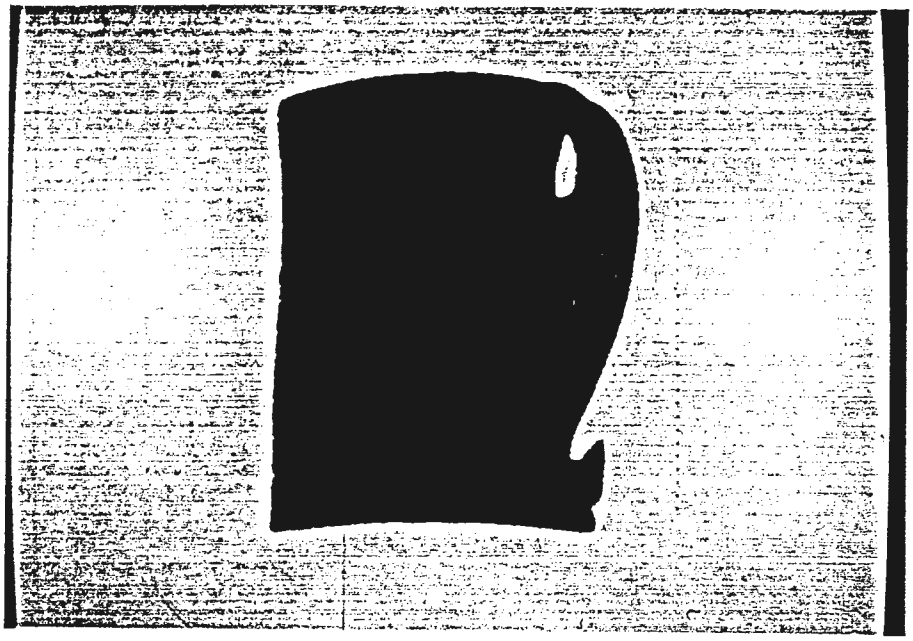


Fig. III-9 : original moire pattern of a tea cup (sampling pitch = 2)

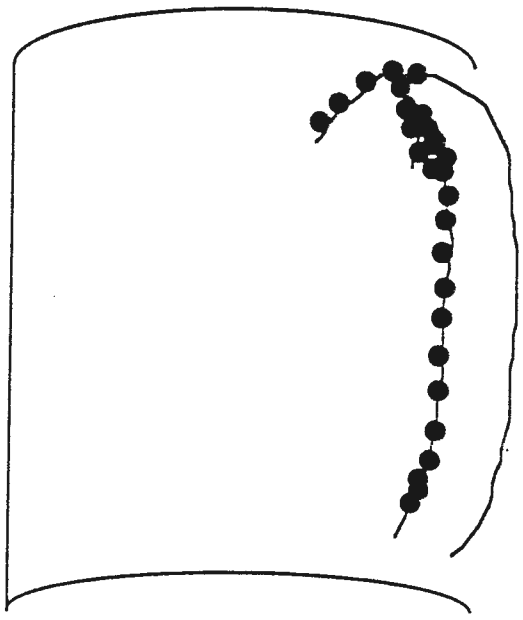


Fig. III-10 : detection of wrong points for Fig. II-9 moire pattern.

IV.3 LINKING "WRONG POINTS" APPROXIMATES THE OCCLUDING EDGE

A simple algorithm is then used to link the wrong points : we assume the set of segments computed at that step will be a good approximation of the occluding edge.

The algorithm links the closest points and makes sure no loop is created (in order not to isolate a particular part of the image).

After linking wrong points in figure III-10 we get :

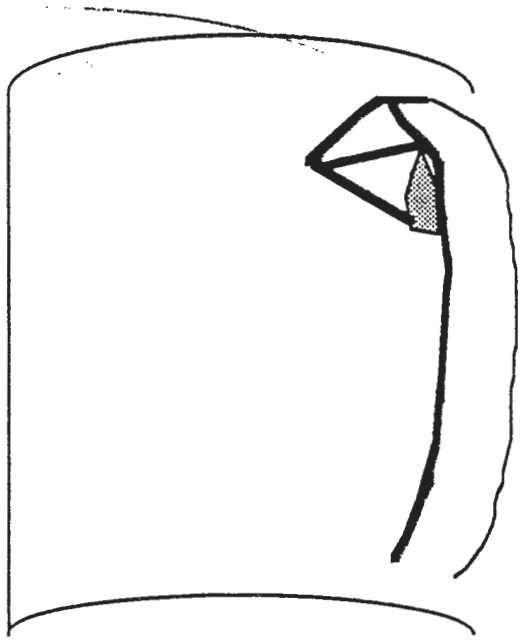


Fig. III-11 : after linking all wrong points which are closer than 100 pixels.

After the linking we get a good approximation of the occluding edge plus a certain number of extra segments.

IV.4 CORRECTION OF THE PICTURE

The segments approximating the edge are then inserted into the moire pattern beginning with the smallest ones.

When a segment is added to the picture some fringes are split into several parts : the old fringe is erased in the data structure and the new fringes are created. They are labelled in the picture, their links are calculated and the new graph is computed. The consistency is then checked. If the moire pattern isn't yet consistent more segments are added. If it is consistent the algorithm stops adding segments : the next figure shows the results of the computation for the tea cup at different resolutions.

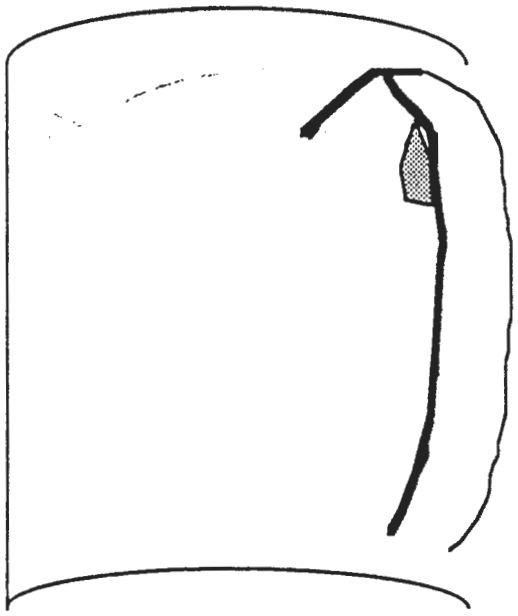


Fig. III-12 : set of segments inserted in the moire picture for correcting it.

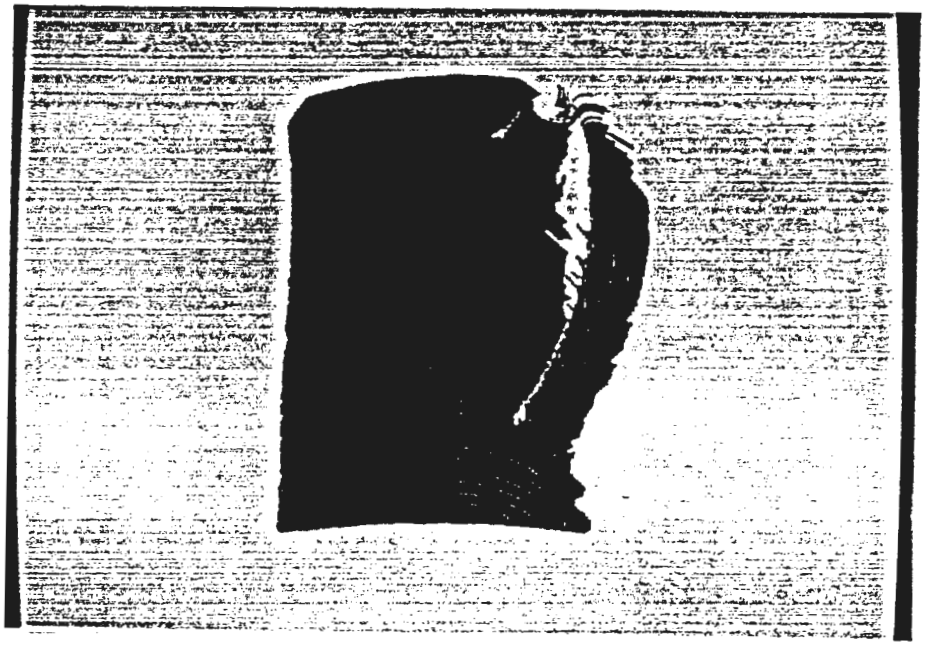


Fig. III-13 : tea cup moire pattern after correction. (sampling pitch=6)

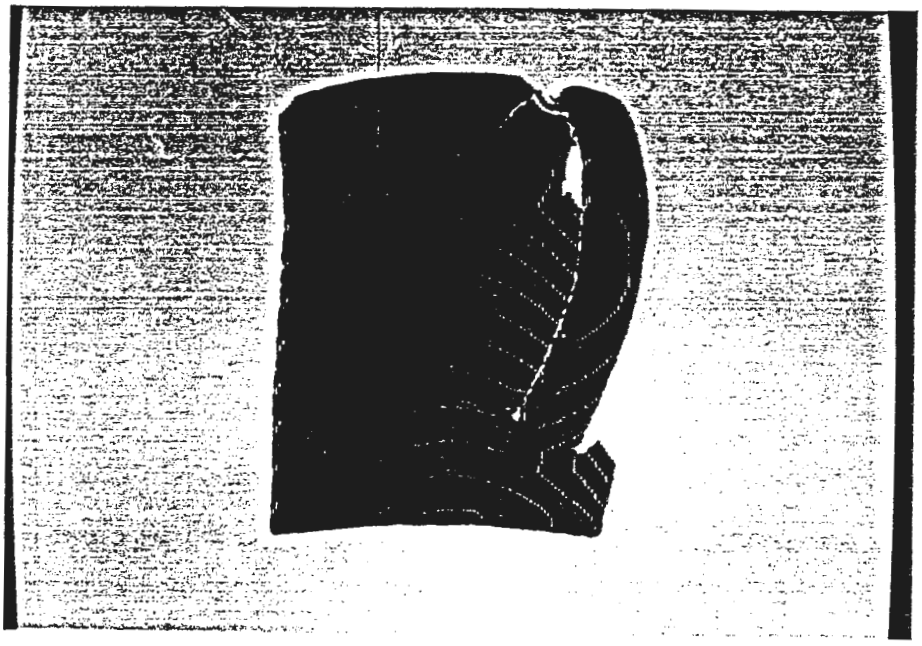


Fig. III-14 : tea cup after correction (sampling pitch=2).

IV.5 RESULTS AND CONCLUSION

As far as I know there is no algorithm for the detection of occluding edges in a moire picture. This algorithm can compute relative orders and moire patterns in pictures where usual algorithms are not effective. Provided that the size of the noisy fringes is below the resolution needed for the picture, an accurate set of relative orders can be computed.

The algorithm has been tested on the tea cup at different angles and on simple polyhedra scenes. The picture below shows an example of computation for an octahedron on a cube.

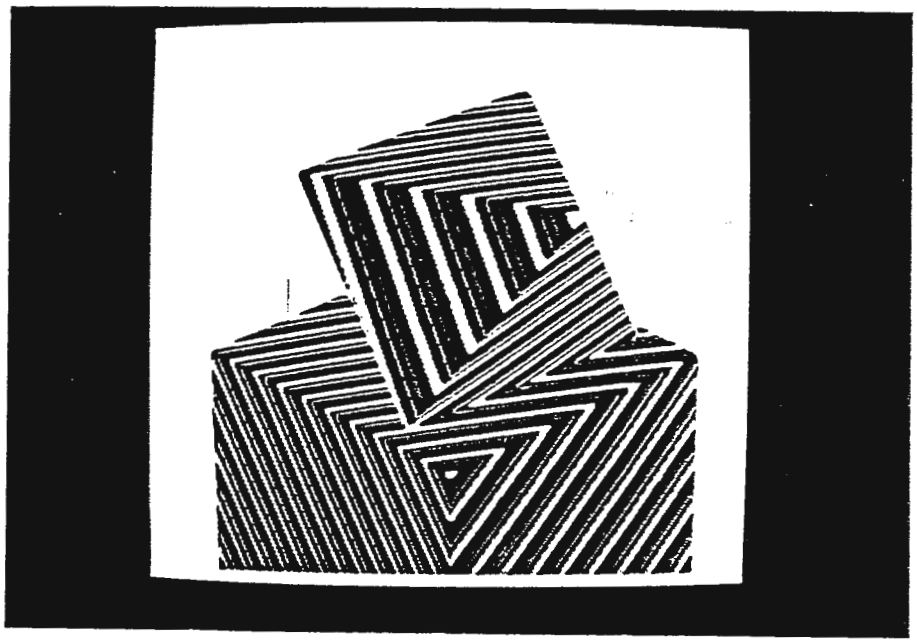


Fig. III-15 : an ambiguous picture of a simple polyhedron scene after correction

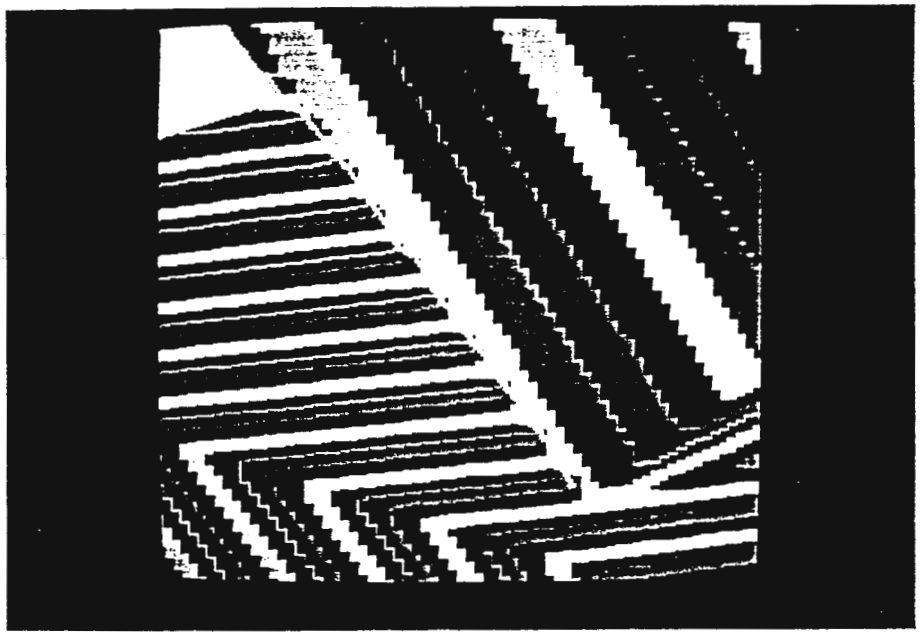


Fig. III-16 : a closer look at the corrected picture of Fig. III-15

Now the moire contour is computed from the tea-cup of figure III-9 : the first figure shows the results of the conventional algorithm.

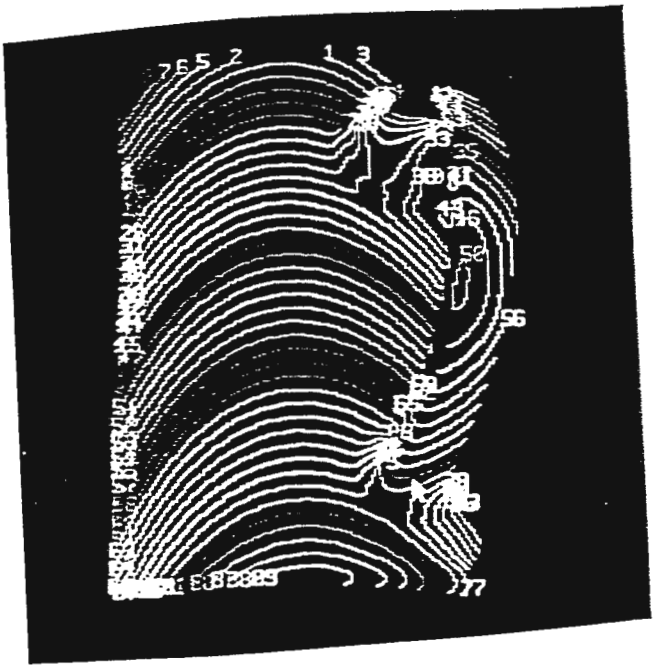


Fig. III-17 : the moire contour computed with the conventional algorithm shows an error in the handle.

It is easy to see that there is a jump in the relative orders at the base of the handle : the gap is about 9 orders wide.

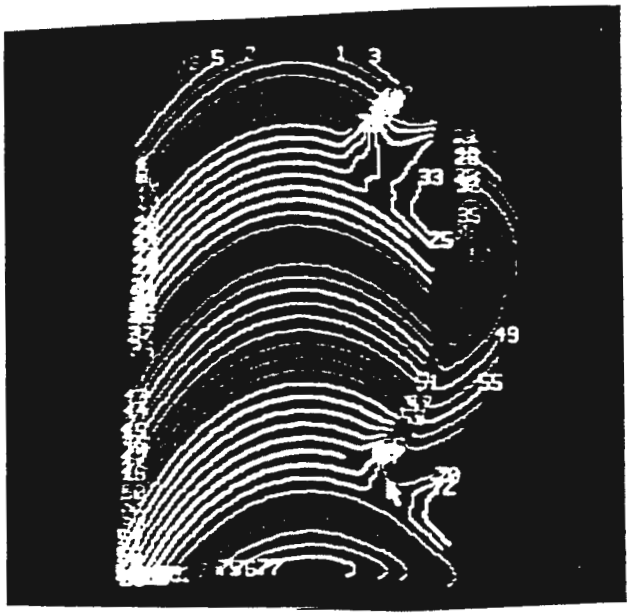


Fig. III-18 : results of our algorithm : the handle of the tea cup is given good relative orders.

Our algorithm has performed well : the relative orders of the tea cup are accurately computed.

The correction and computation of picture III-1 took about 3 minutes of CPU time on a VAX 8600, while that of picture III-9 took 8 minutes.

The second is more complex than the first one (three times as many fringes and many more wrong points), but that does not account for all the time difference. We believe this difference is mainly due to the lack of optimization of our implementation : a great amount of time is spent updating a 2D array (named "contact") containing all link weights : contact[i][j] = "weight of link between fringe i and fringe j"

The cost of that search is estimated to be $O(n^2)$ where n is the number of fringes. When n grows too much that search slows the algorithm down. For the second picture (Fig. III-9) there are about 1000 fringes at the beginning of the computation, before filtering.

A solution would be to record the weight of links directly where it is needed, that is within the C structure representing a fringe (see chapter II end of paragraph II.2.). We did not have time so far to implement this.

V. SOME FURTHER POSSIBLE IMPROVEMENTS

V.1 LIMITATIONS OF IMPLEMENTED ALGORITHM

It is quite easy to detect singularities in a fringe pattern that needs repair. After analyzing the graph representation we can find the most suspect fringes (those that are very likely to be the result of an illegal merging). But once one of those fringes is found, it is much more difficult to find the point where the fringe has to be split.

That algorithm makes no use of the information lying in the graph to correct the image : it only uses the "wrong" points data, then computes a set of segments and inserts as many as necessary in the picture. This is not sufficient for images which are very noisy.

V.2 ANOTHER POSSIBLE STRATEGY

So far the model of a fringe is quite rough : a node with links to other nodes. By using a more sophisticated model of fringe (including the locations of wrong points for example) it should be possible to try to correct only the most suspicious fringes (determined after analysis of the graph). The efficiency of the program would be improved.

It would also be interesting to implement some "feedback" in the algorithm. So far the program adds segments until it gets the "consistent" answer from the graph. If the

picture is noisy many segments inserted into the picture are not useful and don't lead to an improvement of the graph. The program should then undo those changes.

VI. ANNEX : USE OF "WRONG POINTS" FOR AUTOMATIC CONTOURING

VI.1 PROBLEM

When a moire picture of an object is taken a mask is computed after low-pass filtering and thresholding of the image. That mask is applied on the picture after the fringe pattern has been generated (yellow part of moire pictures): only the significant parts of the moire pattern will appear.

The criteria for masking a pixel or not is its intensity after low pass filtering. That criteria is a little artificial : some dark parts of the object can provide a nice moire pattern, but be masked because of their low intensity ; whereas some bright and steep areas with a deteriorated pattern will not be masked. The Fig. III-19 shows such a case : if the treshold level is set low, there is too much noise in the picture. But if it is set high enough to mask the noisy parts, a great part of the clean pattern disappears.

Fig. III-19 : a difficult mask computation problem.

IV.2 PROPOSED ALGORITHM

The proposed algorithm would be to use the wrong points to approximate the parts of the picture where the pattern is too bad to be of any use. This information would then be used to mask these parts.

As seen previously the wrong point density is a very good indication of how good the moire pattern is: if it is distorted or noisy, lots of wrong points appear.

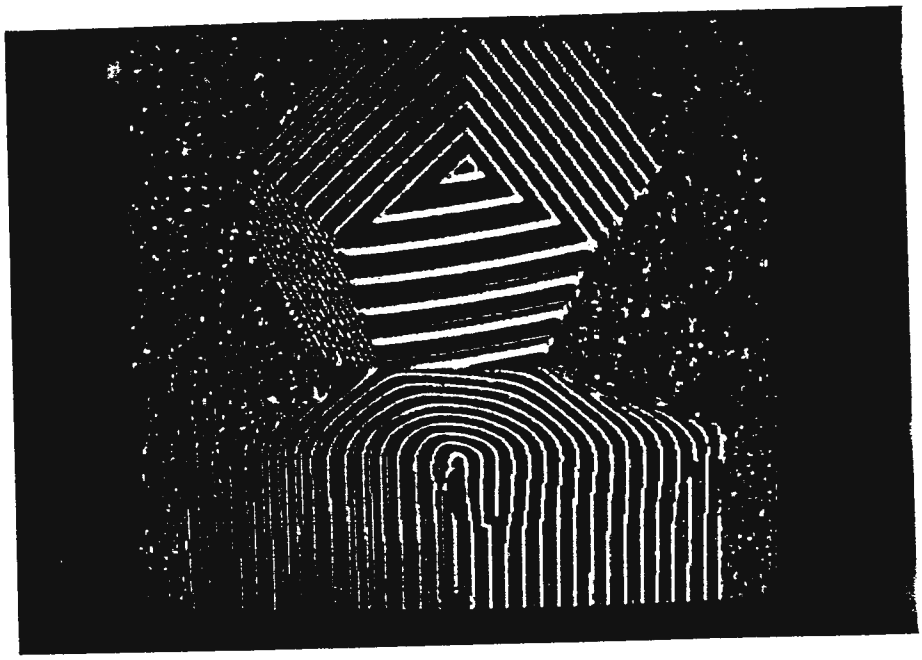


Fig. III-20 : the picture of Fig.III-19 before masking : the fringe pattern quality can clearly be appreciated.

On that picture there are two similar kinds of noise : one is the background noise and has been almost completely masked by the coventionnal algorithm. The second is the very bad moire pattern on the lower left face of the dodecahedron : this has not been masked.

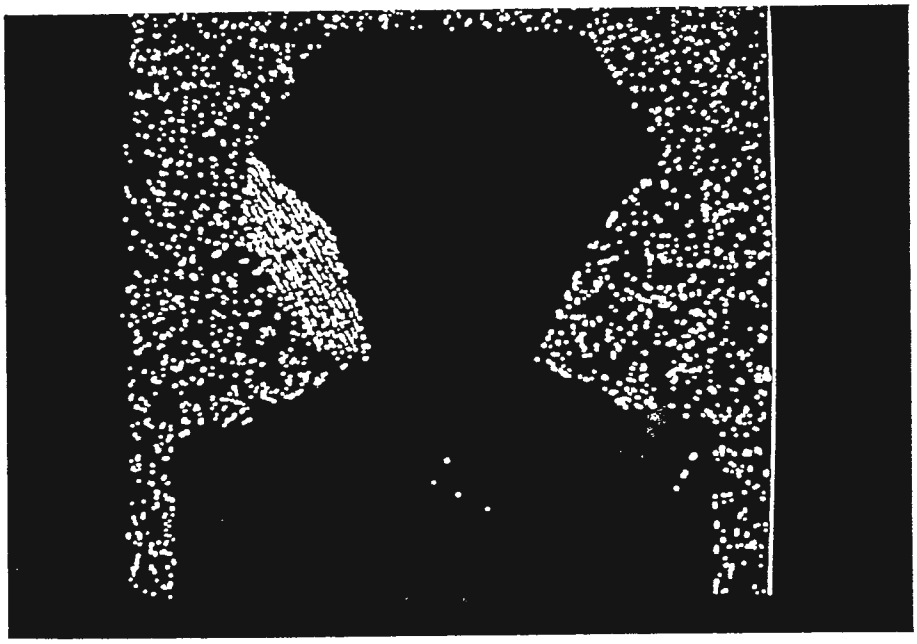


Fig. III-21 : the wrong points detected for the picture of fgure III-20 (with an 8-connected neighborhood).

For that last picture all parts that should be masked appear clearly. For the algorithm we would suggest to keep, as a first step, the same mask computation algorithm, but with a low threshold so as to be sure not to mask any good part of the pattern : most of the background would be masked but the object would be left intact. We would then search for all wrong points left in the picture.

We could then use a triangulation technique to compute the mask to be applied on the pattern. That algorithm would be precise but might be time consuming. Another solution would be to replace each wrong point in the picture with a disk or a square of small size (enlarging each pixel). If that size is appropriately chosen a good contour could be obtained.

Using the wrong points for accurate contouring of the object has the advantage of eliminating in the picture only the information we do not need : it should not modify the moire patterns that can be useful.

Chapter IV

Surface Curvatures Computation

I. PURPOSE OF THE STUDY

I.1 EXTRACTION OF SPECIFIC 3D SURFACES FROM A MOIRE PICTURE FOR 3D OBJECT RECONSTRUCTION

The moire photography system has been conceived as a 3D object acquisition system. Once several pictures of the object have been taken we want to build an accurate representation of its shape. We want this representation to be as concise and as stringent as possible.

In order to achieve these opposite goals we want to extract from object shape the lines of surface intersections : their content in information is very high and they should allow us to segment the surface and to more easily represent the surface.

I.2 SURFACE SEGMENTATION

We want to segment the surface using intrinsic and extrinsic properties. The most important intrinsic feature of a surface is its Gaussian curvature : it should allow us to extract parabolic lines (lines where one of the curvatures becomes zero), umbilic surfaces (e.g. spheres) and planes.The extrinsic features are the local minima and maxima of the picture, the 2D curvatures of the fringes.

The first step toward surface segmentation is then to be able to compute the 3D curvatures all over the surface. The other features are easy to compute using classical techniques. The 3D curvatures computation cannot be done using classical techniques in a moire system as the 3D data from moire pictures do not give us a dense map.

I.2. CURVATURES COMPUTATION : BASIC IDEA

The important starting point is that an equidistance contour, Γ , is a 2-D surface curve lying on the surface of the 3D object. Since the data is dense in this 2D plane, given any point M lying on Γ, we can obtain a very accurate estimate of the tangent, $\vec{t}$, principal normal, $\vec{n_\Gamma}$, and the curvature, k , to the curve at M . So, one of the 3 axis of the *Frenet trihedron* of Γ at M is fixed.

Next, we go to the contours lying immediately above and below this plane. Since these contours are equidistance contours (the so-called level sets), we can parallel transport the tangent at $\vec{t}$, to the neighborhood contours, say, to points P_1 and P_2 . Then we can obtain a good estimate of the surface normal, $\vec{n}$, at M by using osculating circle to the points (P_1 , M , P_2) . The tangent plane, T(M), and the *normal section* that is orthogonal to $\vec{t}$ are thus fixed. (See Fig. IV-1 and Fig. IV-2)

Notice that the normal curvature at M in this direction, which we denote by k_0 , can immediately be obtained from the osculating circle by a simple formula which we will derive.

Once the normal $\vec{n}$, is fixed, then the normal curvature of Γ, at M , denoted by k_{90} , is also immediately available from *Meusnier Theorem* , [do Carmo 76, p.142] which states that:

Meusnier Theorem : All curves lying on a surface S and having at a given point $p \in S$ the same tangent line have at this point the same normal curvatures.

Thus we have : $k_{90} = k \cos \theta$ where θ is the angle between the surface normal $\vec{n}$ and the principle normal $\vec{n_\Gamma}$ of Γ at M .

Here the *mean curvature* is immediately available because it is known that the sum of the normal curvatures in two orthogonal direction is always 2H (2 times the mean curvature).

Thus we have : $H = \frac{1}{2}(k_0 + k_{90})$

Next to obtain the *principal curvatures* and *Gaussian curvature*, we make use of the *Euler's Formula* , which states that:

Euler's Formula :

$$k_0 = k_1 \cos^2 \alpha + k_2 \sin^2 \alpha ,$$

where k_0 is the normal curvature of a surface curve along a given direction on T(M), and (k_1 , k_2) are the *principal curvatures*, and α is the angle between the first principal direction and the tangent of the surface curve.

Since we already have a good estimate of (k_0 , k_{90}), the remaining three unknowns, (k_1 , k_2 , α) can be deduced from *Euler's Formula* if we can also obtain good estimates of (k_{45} , k_{135}), the normal curvatures of the 45 and 135 degree normal sections.

To obtain the 45 and 135 degree normal sections, we again make use of the equidistance (or level sets) properties of the equidistance contours to locate the intersection points between these normal sections and the surface. Once these intersection points are obtained, the respective normal curvatures are estimated using their respective osculating circles.

Thus we see that the computation of the normal curvatures, from k_0 to k_{90} , k_{45} and k_{135}, are carried out in a step by step fashion, where the errors can all be controlled. The final step the involves solving a quadratic equation is also stable. The only complicated manipulation is on the location of the 45 and 135 degrees normal sections which we show below in details how it is done.

We first show below an outline of the computation steps followed by some more details of the intricate parts of the algorithm.

II. ALGORITHM

II.1 CHARACTERISTICS OF THE 3D MOIRE DATA

The 3D moire data is different from the usual 3D data used in surface segmentation : it consists of a set of fringes which represent slices of the object at different heights. The density of the data is much higher along a fringe than usual dense maps. But the spacing between two fringes at different levels is not dense. Thus the data is dense in 2D but sparse along the Z axis.

In this chapter, we show how to take advantage of the great amount of data in the fringes to segment more efficiently the surface. This should overcome the lack of data in the Z direction. This structure of data allows us to segment the surface without having to approximate the surface with patches : the computation should then be less intensive.

II.2 COMPUTATION OF CURVATURES : USING OSCULATING CIRCLE

II.2.1 What is the Osculating Circle Approximation ?

As shown in Fig. IV-1, the objective is to approximate the surface at a point M within a normal plane P. The normal plane intersects with the surface along a curve. This curve is of course included in plane P and M lies on it : we assume that on each side of M, the curve intersects a fringe at points P_1 and P_2.

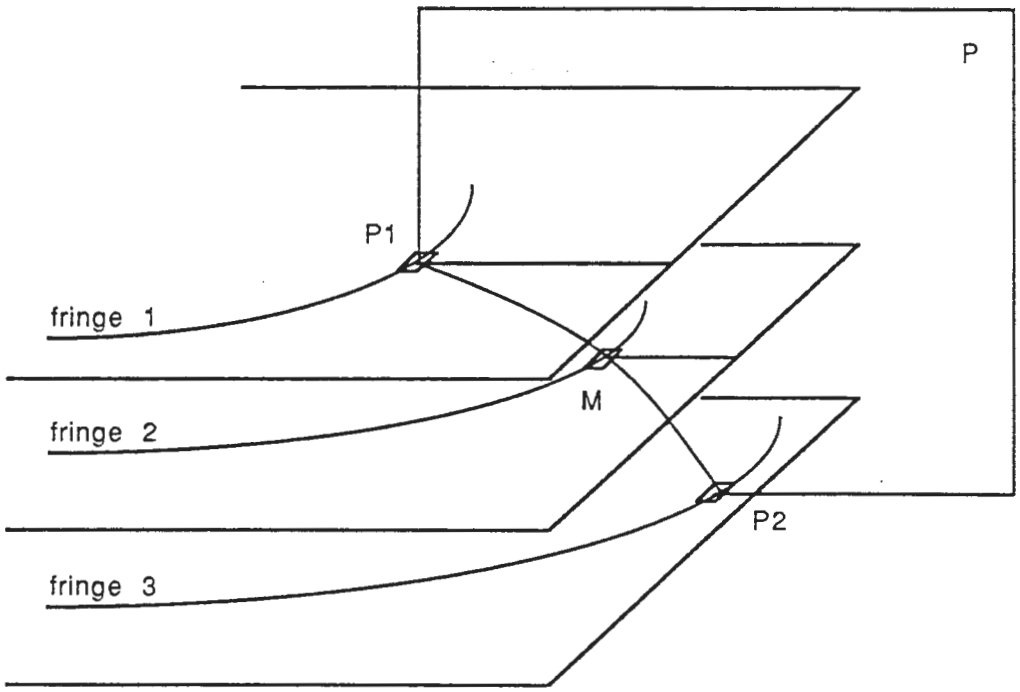


Fig. IV-1 : intersection of normal plane P at point M with object surface : P_1 and P_2 are points of intersection curve which belong to fringes.

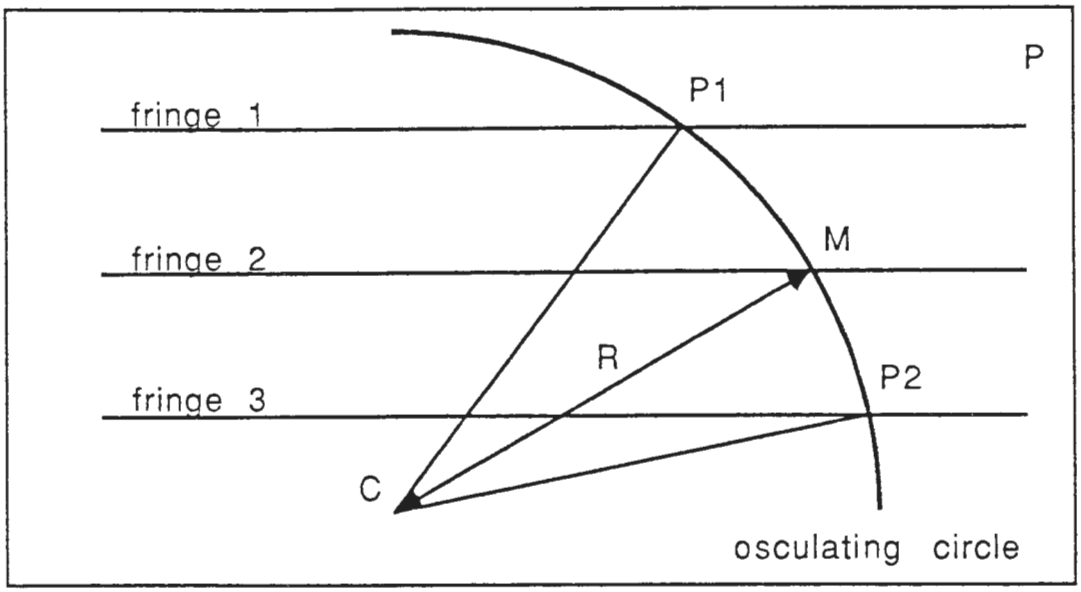


Fig. IV-2 : view of plane P : osculating circle of center C and radius R defined by the points P_1, P_2 and M.

These three points lie in plane P and we approximate the curve at point M with the circle C defined by these three points. If P_1, P_2 and M lie close enough, the center of circle C is close to the center of curvature and the distance CM is close to the radius of curvature of the curve at point M.

II.2.2 A Quick Look at the Algorithm

a) Computing the Tangent :

The tangent to the curve is computed first

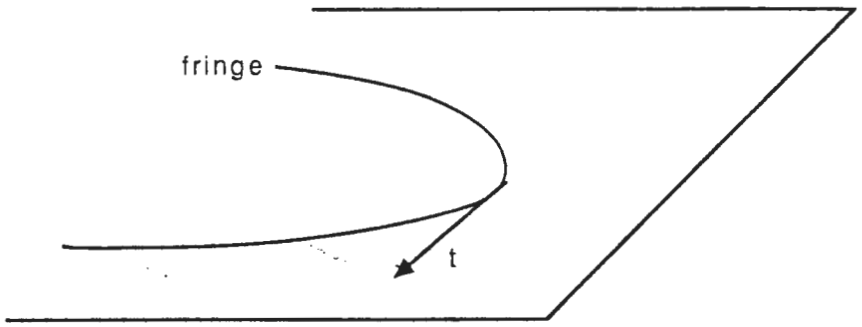


Fig. IV-3 : tangent to the fringe.

We get :

$$\vec{t} = \begin{bmatrix} t_1 \\ t_2 \\ 0 \end{bmatrix} \qquad \text{with } \| \vec{t} \| = 1.$$

b) Computing the Normal :

The normal is then computed :the picture is scanned orthogonally to the tangent, two cross-points P_1 and P_2 are found.

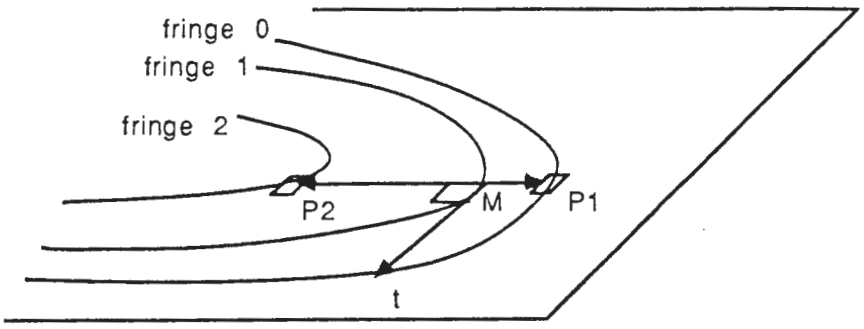


Fig. IV-4 : normal computation : finding the cross-points.

The direction of the normal is given by the vector CM where C is the cross-point of the bisecting lines of segments MP_1 and MP2, as given in Fig. IV-5.

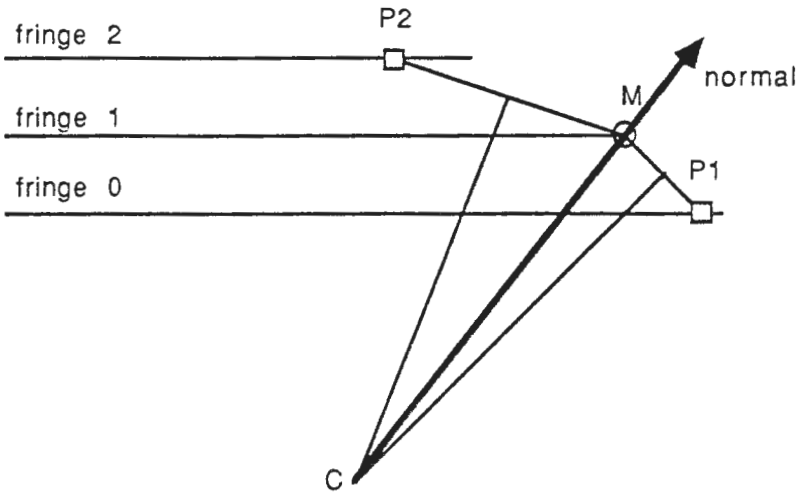


Fig. IV-5 : normal computation : computing the normal using the osculating circle approximation.

We get :

$$\vec{n} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} \qquad \text{with } \| \vec{n} \| = 1.$$

For computing the normal we need to compute the distance CM. Thus the normal curvature $k_0 = \frac{1}{CM}$ is computed at the same time.

c) Completing the Orthogonal Basis :

The last vector of the orthonormal basis is computed :

$$\vec{b} = \vec{t} \otimes \vec{n} = \begin{bmatrix} t_2 n_3 \\ - t_1 n_3 \\ t_1 n_2 - t_2 n_1 \end{bmatrix}$$

d) Computing k_{90} :

The k_{90} curvature is computed in a special way in order to reduce the error made using the algorithm for k_{45} and k_{135} (see II.4.2).

The 2D curvature of the fringe at point M is computed and then projected on the normal. This projection is the normal curvature at point M for the normal section at 90 degree (from Meunier theorem)

e) Computing Normal Plane Coordinates :

As seen previously, k_0 and k_{90} have been computed. We now need to compute k_{45} and k_{135}. These are done using the osculating circle approximation in the normal sections P_{45} and P_{135} : the intersection of these sections are computed and the osculating circle approximation is used to compute the curvatures. Below are the definitions of the planes used in that computation :

- P_{45} is the plane normal to $\vec{b} - \vec{t}$.

- P_{135} is the plane normal to $\vec{b} + \vec{t}$.

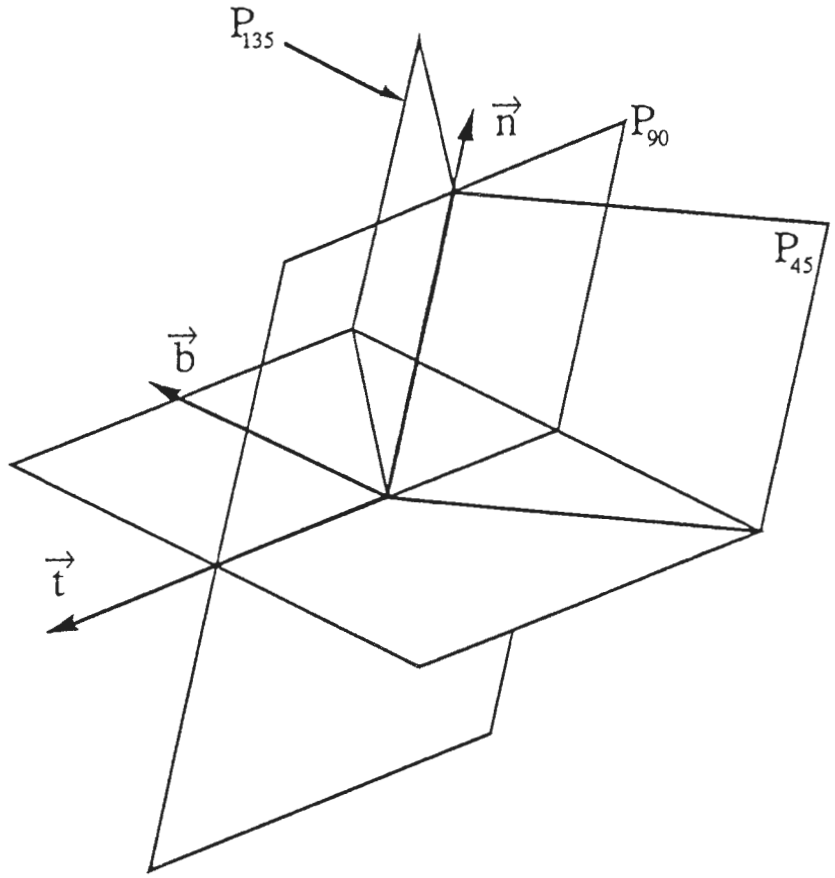


Fig. IV-6 : the planes used in the curvature computation.

f) Finding Intersection Points Between Plane and Fringes :

For each normal plane, we compute the intersection of the normal plane with fringes one level over and one level under that of point M.

Let us find the intersection of a normal plane with a fringe one level under M. The normal plane intersects the fringe plane along a line D : its direction is given by the formula :

$$\vec{d} = \begin{bmatrix} a_2 \\ -a_1 \\ 0 \end{bmatrix} \quad \text{where } \vec{A} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \text{ is the vector orthogonal to the normal plane.}$$

The D line goes through the point M' , projection of M following the direction of normal $\vec{n}$. The intersection of D with the fringe gives us the points we want.

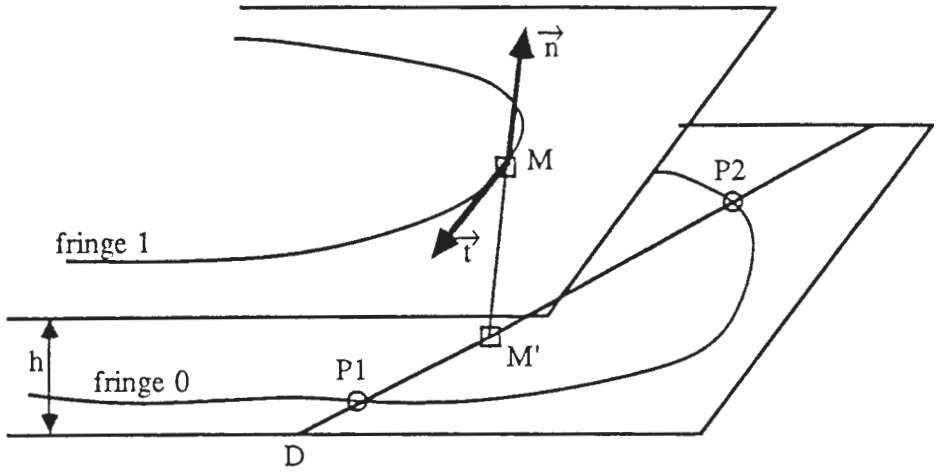


Fig. IV-7 : scanning along line D and searching for points P_1 and P_2.

This computation is done three times : for the plane of fringe 1, and for the planes just under and just above it. Each time from 0 to 2 cross-points are found. The two closest points to M (one on each side of the normal) are selected and given the names P_1 and P_2.

g) Computing The Curvature :

From the 3D coordinates of P_1, P_2 and M we compute the radius of curvature, using the following figure (in this example P_1 has been found on the same level as M and P_2 one level under) :

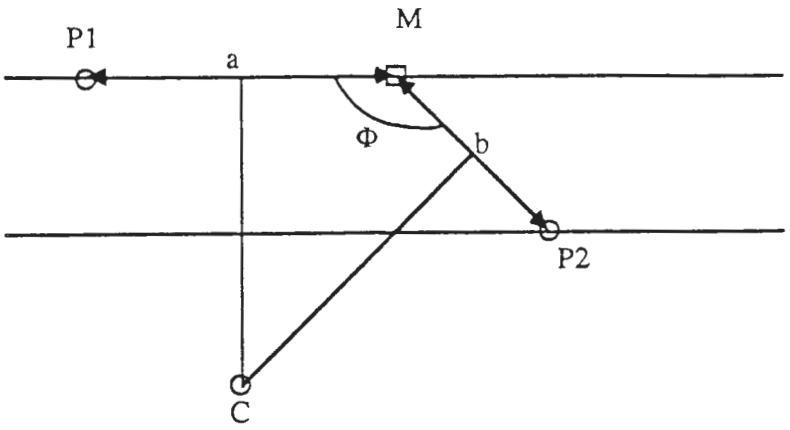


Fig. IV-8 : computing the osculating circle from the points P_1, P_2 and M.

The curvature is given by the formula [see Appendix I] :

$$k = \frac{2 \sin \Phi}{\sqrt{a^2 + b^2 - 2\,a\,b \cos \Phi}}$$

h) Computing the Intrinsic Features :

Four normal curvatures are computed : at 0, 45, 90 and 135 degree. This data is sufficient for calculating the two principal curvatures K1 and K2, the Gaussian and the mean curvature K and H, and the principal directions $\vec{e_1}$ and $\vec{e_2}$ [Fan and al. 83].

II.3 THE IMPLEMENTED ALGORITHM IN DETAILS

II.3.1 The Tangent Computation

The fringes are not smooth enough to allow a direct computation of first order derivative. So we had to approximate them with beta-splines in order to get an accurate computation of tangent.

II.3.2 The Normal Computation

We want to compute the coordinates n_1, n_2, n_3 of $\vec{n}$ and the k_0 curvature. We already know that $\| \vec{n} \| = 1$ and that $\vec{n} \perp \vec{t}$. This gives us the following equations :

$$n_1^2 + n_2^2 + n_3^2 = 1$$

and

$$\begin{cases} n_1 = - \lambda\ t_2 \\ n_2 = \lambda\ t_1 \\ n_3 = n_3 \end{cases}$$ with : $\lambda = \cos \theta$ where θ is the oriented angle from $\vec{t}^{\perp}$ to $\vec{n}$.

These two equations can be rewritten as :

$$\begin{cases} n_1 = - \cos \theta\ t_2 \\ n_2 = \cos \theta\ t_1 \\ n_3 = \sin \theta \end{cases}$$

We already know t_1 and t_2 and thus have only to compute θ . The normal plane contains the Z axis and the othogonal vector to the tangent : these two vectors will be taken as a basis.

For computing sin θ and cos θ we will have to consider four cases according to the sign of curvature and of inner product $\overrightarrow{MP2} \cdot \vec{t}^{\perp}$. The following figure shows the case where both are positive.

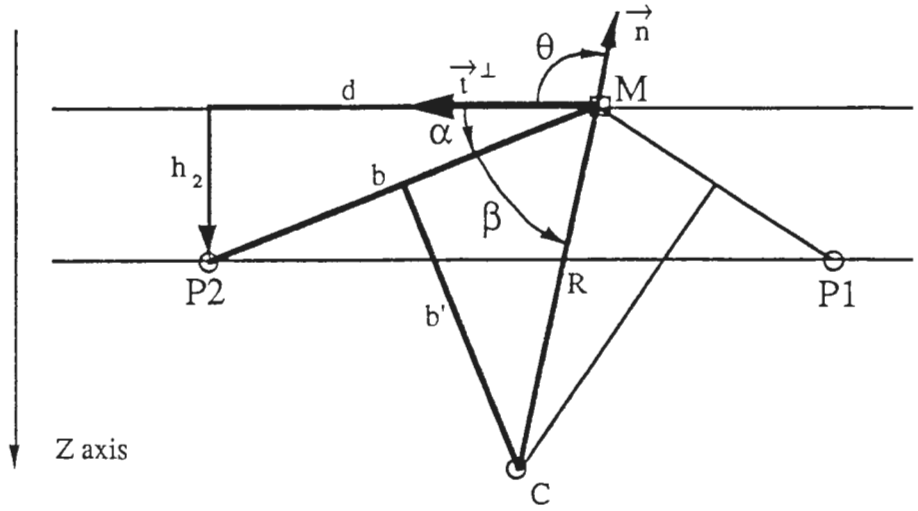


Fig. IV-9 : the computation of normal azimuth in the case of positive curvature and inner product.

We have a relationship between θ , α and β (here : $\alpha + \beta - \theta = \pi$). Using it we compute sin θ and cos θ with :

(1) $\cos \alpha + \beta = \cos \alpha \cos \beta - \sin \alpha \sin \beta$

and

(2) $\sin \alpha + \beta = \cos \alpha \sin \beta + \cos \beta \sin \alpha$

but we have the following :

$$|\cos \alpha| = \frac{d}{b}$$

$$|\sin \alpha| = \frac{|h_2|}{b}$$

$$|\cos \beta| = \frac{b/2}{R}$$

$$|\sin \beta| = \frac{b'}{R}$$

$$b' = \sqrt{R^2 - \frac{b^2}{4}}$$

Remark : b, b' and d are distances (always positive) while h_2 is the z-coordinate of $\overrightarrow{MP2}$ vector.

The radius of curveture R is computed using the osculating circle approximation (see II.3.5).

The result of the caculus for all cases can be summed up in the following formulas [for details see Appendix II] :

$$(3)\ \cos\theta = \frac{\text{sign}(\overrightarrow{MP2} \cdot \vec{t}^{\perp})}{R}\left(- \text{sign}(k)\frac{d}{2} + h_2\sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}}\right)$$

$$(4)\ \sin\theta = -\frac{1}{R}\left(- \text{sign}(k)\frac{h_2}{2} - d\sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}}\right)$$

Using $\begin{cases} n_1 = -\cos\theta\ t_2 \\ n_2 = \cos\theta\ t_1 \\ n_3 = \sin\theta \end{cases}$ we easily get the normal coordinates.

At the same time we have computed the normal curvature at 0 degree : $k_0 = \frac{1}{R}$

This method is much more accurate than the method that would consist in finding the intersection C of the two bisecting lines of MP1 and MP2 and computing : $\vec{n} = \frac{1}{\| \overrightarrow{CM} \|}\overrightarrow{CM}$.

II.3.2 The k_{90} Normal Curvature Computation

The 90 degree curvature of the normal section is computed from the 2D curvature : the cross-point computation for that angle is not accurate enough (see II.4.2.c).

As the normal section curve at 90 degree and the fringe have the same tangent at point M, we can use the Meusnier theorem (see [do Carmo 76] p142) : the normal curvature k_{90} is equal to the projection of k (2D curvature of the firnge) on the surface normal :

$$\boxed{K_{90} = K (\vec{n_\Gamma} . \vec{n})}$$ where $\vec{n_\Gamma}$ is the 2D principal normal of the fringe at point M.

II.3.3 The Normal Planes Coordinates

We have already computed the normal curvatures at 0 and 90 degrees. We now need to compute the curvatures at 45 and 135 degree from an arbitrary reference angle. In order to simplify the computation of the equations of the normal sections we measure the angle in the ($\vec{b}$, $\vec{t}$) basis.

The equations of the planes are then more easily computed :

- P_{45} is the plane normal to $\vec{b} - \vec{t}$.

- P_{135} is the plane normal to $\vec{b} + \vec{t}$.

II.3.4 Finding the Intersection Between a Given Normal Plane and a Fringe.

Let n be the order of point M. We compute the intersection between normal plane P and fringes at levels n, n+1, and n-1. For each of these three planes the equation of line D is computed : its direction (identical for the three planes) and a point belonging to it (M, M_1 or M_{-1}). M_1 and M_{-1} are respectively the projections of point M along the normal on planes P_{n+1} and P_{n-1}: we get the three lines D_n , D_{n-1} and D_{n+1}.

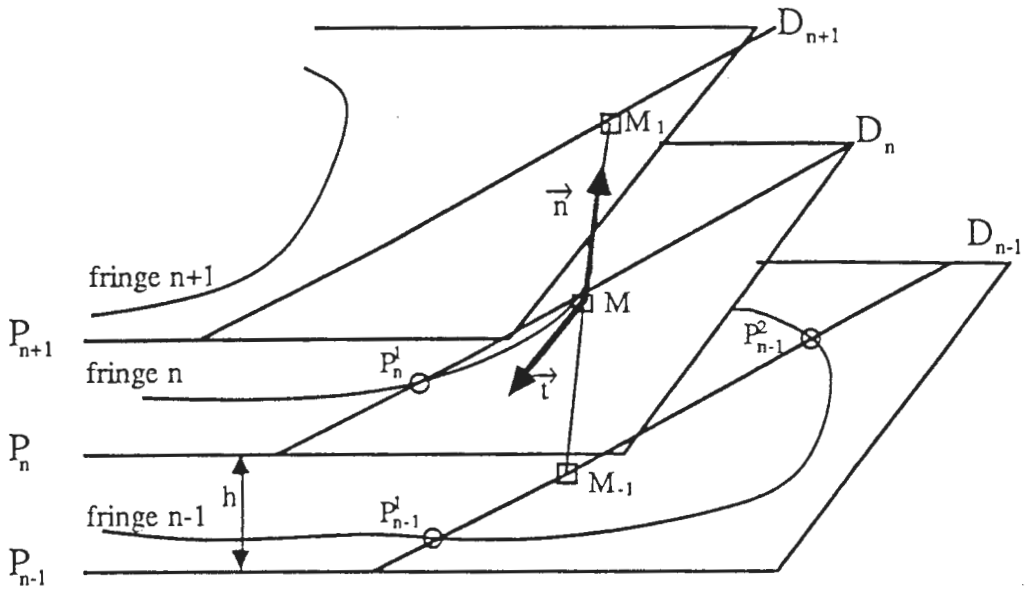


Fig. IV-10 : intersection points of normal plane and fringes of level n, n-1 and n+1.

The results are :

$$P \cap P_{n+1} = \varnothing$$

$$P \cap P_n = \{ P_n^1 \}$$

$$P \cap P_n = \{ P_{n-1}^1 , P_{n-1}^2 \}$$

The line $\Delta = (M , \vec{n})$ separates the plane P into two parts. In each of these part the closest point to M is chosen among the P_j^i : they are named P_1 and P_2.

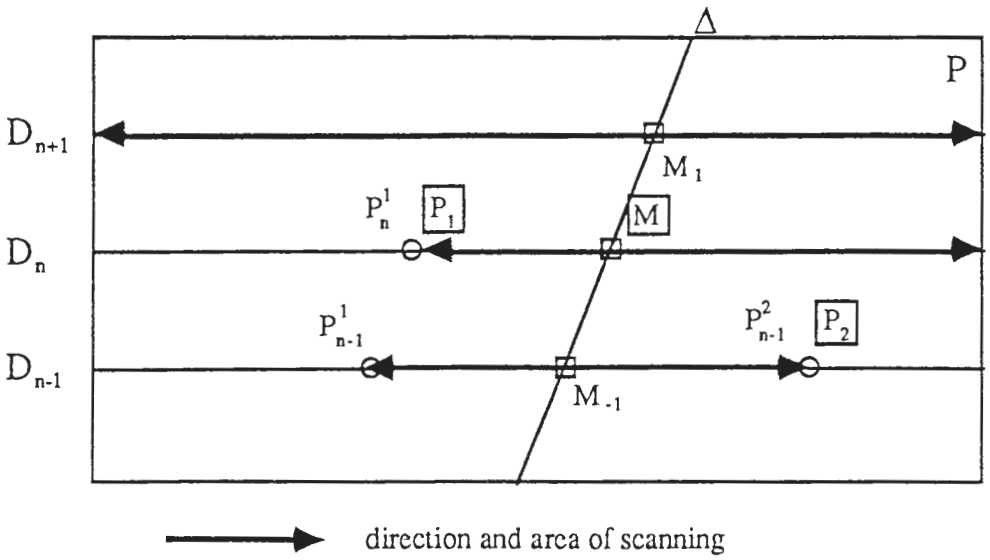


Fig. IV-11 : view of the normal plane P : Δ line, D lines, intersection points, P_1 and P_2.

Fig. IV-11 shows the distribution of the intersection points in the plane P, the line Δ and the points P_1 and P_2. In this case we have for example :

$$P_1 = P_n^1 \text{ and } P_2 = P_{n-1}^2 .$$

II.3.5 Computing the Curvature.

The curvature in normal plane P is computed using the formula stated before (see Fig. IV-8). This formula comes from the osculating circle approximation.

$$k = \frac{2 \sin \Phi}{\sqrt{d_1^2 + d_2^2 - 2\, d_1\, d_2 \cos \Phi}}$$ with d_1 = MP1 and d_2 = MP2.

II.3.6 Computing Intrinsic Features

a) The Principal Curvatures :

Once the four normal curvatures k_0 , k_{45} , k_{90} and k_{135} have been computed, The Gaussian curvature is calculated.

We have the following relationships between these curvatures and the principal curvatures k_1 and k_2 (see [3]) :

$$(1) \qquad S = k_1 + k_2 = k_0 + k_{90}$$

$$(2) \qquad S = k_1 + k_2 = k_{45} + k_{135}$$

In the article [3], the mean curvature S is defined as :

$$(3) \qquad S = \frac{1}{2}(k_0 + k_{45} + k_{90} + k_{135})$$

This formula does not advantage any of the computed curvatures. But as can be seen in the error bound chapter, the k_{90} curvature computation is more unprecise than the others. For that reason only the formula (2) is used in the implemented algorithm to find the sum of k_1 and k_2.

Using this other formula :

$$4 (k_0 k_{90} + k_{45} k_{135}) = 4 \; k_1 k_2 + (k_1 + k_2)^2$$

with $P = k_1 k_2$ we get :

$$4 (k_0 k_{90} + k_{45} k_{135}) = 4 P + S^2$$

that is :

$$P = 4 (k_0 k_{90} + k_{45} k_{135}) - S^2$$

k_1 and k_2 are then computed by solving the second degree equation :

$$X^2 - S X + P = 0$$

By convention we have : $| k_1 | > | k_2 |$.

b) <u>The Principal Directions :</u>

The basis of the tangent plane is ($\vec{b}$, $\vec{t}$). The angle between $\vec{b}$ and the principal direction of curvature k_1 is named α.

The following equation gives two values for α : α and $\pi - \alpha$

$$k_0 = k_1 + (k_2 - k_1) \sin^2 \alpha$$

This second equation also gives two values for α : α and $\frac{\pi}{2} - \alpha$:

$$2 k_{45} = k_1 + k_2 + (k_1 - k_2) \sin 2\alpha$$

Comparing those two sets of solution we find the unique value for α.

We then compute the principal vector coordinates :

$$\vec{e_1} = \cos \alpha \ \vec{b} + \sin \alpha \ \vec{t}$$

$$\vec{e_2} = - \sin \alpha \ \vec{b} + \cos \alpha \ \vec{t}$$

II.4 ERROR BOUNDS

II.4.1 introduction

The error bound study has been done on the accuracy of two points of the computation : the P_1 and P_2 computation and the curvature computation.

The P_1 and P_2 computation accuracy mainly depends on the tangent computation accuracy : the location of P_1 and P_2 is quite sensitive to a small change of $\vec{t}$. That is why we thought it was worth using a beta-spline approximation to compute the tangent.

The curvature accuracy itself depends on the accuracy of the measures of MP1 and MP2 and on that of the angle Φ .

II.4.2 Error on the Cross-points

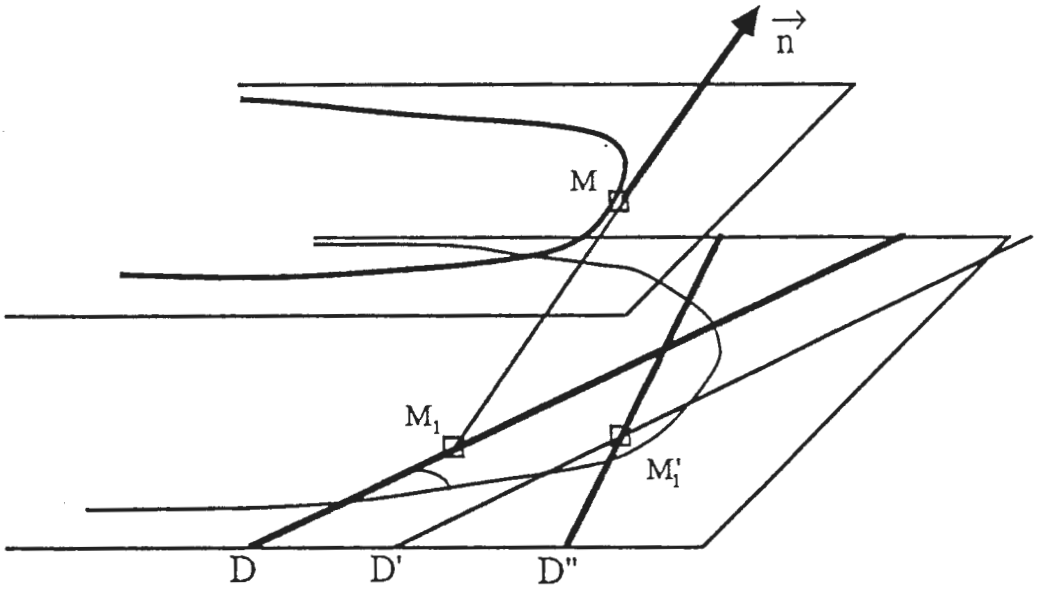


Fig. IV-12 : error in the computation of cross-points.

The error on the cross-points determination comes from a wrong computation of the line D. We will decompose that error into a translation error (D → D')and a rotation error (D' → D").

a) Translation Error (D → D') :

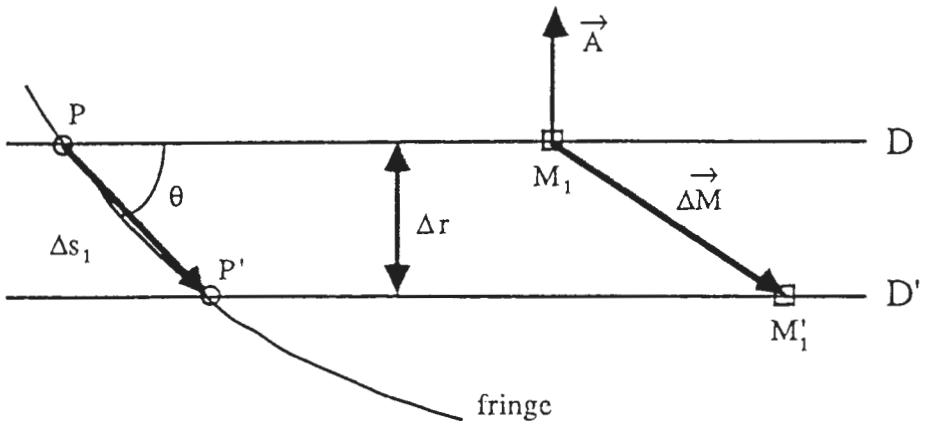


Fig. IV-13 : translation error in the plane of the fringe.

D : intersection line of normal plane and fringe plane.

$\vec{A}$: orthogonal vector to line D.

M_1 : projection of M on fringe plane.

θ : angle between fringe tangent and D line.

P : intersection point between D line and fringe.

D' , M'_1 , P' : computed data.

Δs_1 : error of arclength coordinate of P.

$\vec{\Delta M}$: error of M_1 computation.

Δr : distance between D and D'.

We have then :

$$\Delta s_1 = \frac{\Delta r}{\sin \theta} = \frac{\vec{\Delta M} \cdot \vec{A}}{\sin \theta}$$

b) Rotation Error (D' → D'') :

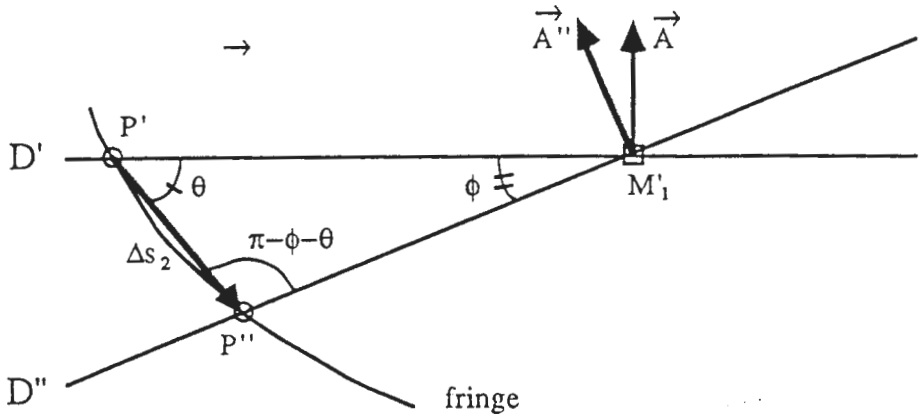


Fig. IV-14 : rotation error diagram : $\vec{A} \rightarrow \vec{A}''$.

We suppose here that $\| \vec{A} \| = 1$.

$$\Delta s_2 = \frac{M'_1P' \sin \phi}{\sin \phi + \theta} = M'_1P' \frac{\sin \phi}{\sin \phi \cos \theta + \sin \theta \cos \phi}$$

We have : $\sin \phi \cong \| \vec{\Delta A} \|$ (providing ϕ is small)

$\cos \phi \cong 1$

And thus :

$$\Delta s_2 \cong MP' \frac{\| \vec{\Delta A} \|}{\cos \theta \ \| \vec{\Delta A} \| + \sin \theta}$$

But $\| \vec{\Delta A} \|$ is first order so we can replace MP' by MP:

$$\boxed{\Delta s_2 \cong MP \frac{\| \vec{\Delta A} \|}{\cos \theta \ \| \vec{\Delta A} \| + \sin \theta}}$$

c) Total Error on Δs :

The total error on the arclength of cross-point P is :

$$\Delta s = \Delta s_1 + \Delta s_2 = \frac{\Delta r}{\sin\theta} = \frac{\Delta \vec{M} \cdot \vec{A}}{\sin\theta} + MP \frac{\| \Delta \vec{A} \|}{\cos\theta \, \| \Delta \vec{A} \| + \sin\theta}$$

This result shows that a very big computing error can occur if the angle between the fringe and line D is small. This error would occur mainly when trying to compute the curvature at 90 degree (normal plane tangent Of the fringe) : at this angle θ takes its smallest value. The cross-points with the fringe may not be accurately computed. But as seen previously, the 90 degree curvature is computed using a different algorithm (from 2D curvature) : this problem is avoided.

The computed curvature k_0 should be the more precise, followed equally by k_{45} and k_{135}.

II.4.3 Error on Φ Angle

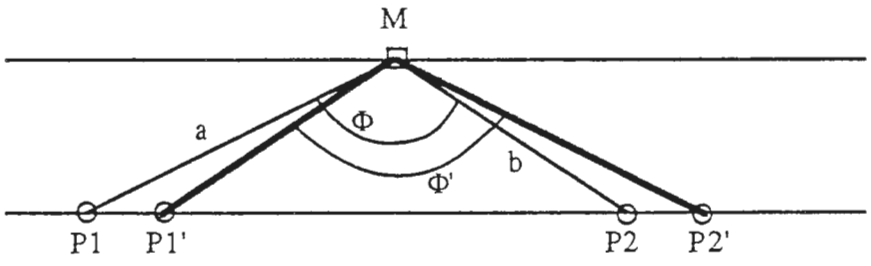


Fig. IV-15 : error on curvature computation.

We compute the curvature using the formula (3) :

$$k = \frac{2 \sin \Phi}{\sqrt{d_1^2 + d_2^2 - 2\, d_1\, d_2 \cos \Phi}}$$ with d_1 = MP1 and d_2 = MP2.

If we differentiate it, we get :

$$\frac{\delta k}{k} = - \frac{(d_1 - d_2 \cos \Phi)}{d_1^2 + d_2^2 - 2\, d_1\, d_2 \cos \Phi} \delta d_1$$

$$- \frac{(d_2 - d_1 \cos \Phi)}{d_1^2 + d_2^2 - 2\, d_1\, d_2 \cos \Phi} \delta d_2$$

$$+ \left(\frac{\cos \Phi}{\sin \Phi} - \frac{2\, d_1\, d_2 \sin \Phi}{d_1^2 + d_2^2 - 2\, d_1\, d_2 \cos \Phi} \right) \delta \Phi$$

In the last term there is an arc-tangent in Φ that might grow infinite if Φ is not bounded. But the moire system has a limited angular resolution : all planes inclined by more than about 60 degree will not appear in the pictures : Φ will then always be bigger than 120 degree. So around 0 degree the relative error is bounded.

The same problem arises when Φ gets close to 180 degree : the relative error cannot be bounded : this is normal as this happens when the curvature becomes close to 0. The relative error calculation has no meaning here.

III. COMPUTATION RESULTS

We present some examples of computation results of our surface curvatures computation scheme on real equidistance contour maps containing a single simple object.

Real equidistance range contours used in these examples were acquired with a moire interferometry system [Koezuka, et al 1988]. Equidistance range contours were

presented in the form of ``eight-connected chain codes", each of which is associated with an integer value, called an absolute order, indicating the depth values from a reference plane. The spacing in depth direction between two adjacent contours was measured as 7 mm in these examples.

Computations were performed on range images of a single object of simple shape. The first example is for a hemispherical object, where the direction of the surface normal at every point along a contour is equivalent to that of a distance vector from the center of the sphere to that point. The curvature at every direction is equal to 1/R, where R is the radius of the sphere. The result from the hemispherical contours is shown in Fig. IV-16. Values for the surface normals and the Gaussian curvatures are obtained accurately. As shown in Fig. IV-16, surface normals computed along contours are shown in a needle map representation. Gaussian curvatures computed at points, marked from 1 through 12, along the fourth contour are displayed as a graph where the x-axis is set to be the arc-length from a starting point (marked by 1) and the y-axis as the approximated values of the Gaussian curvature.

In the second example, computations were performed on a cylindrical object in which the Gaussian curvature along every contour is equal to zero. Fig. IV-17 shows the result of a scene containing a coffee cup that is viewed with the bottom at low angle. Both the surface normals and the Gaussian curvatures at every point were computed correctly. The graph shows the values of the Gaussian curvatures along the 20-th contour.

Further examples on other surface features such as local extrema and zero-crossing points of Gaussian curvature are reported elsewhere [Tanaka, et al 1988].

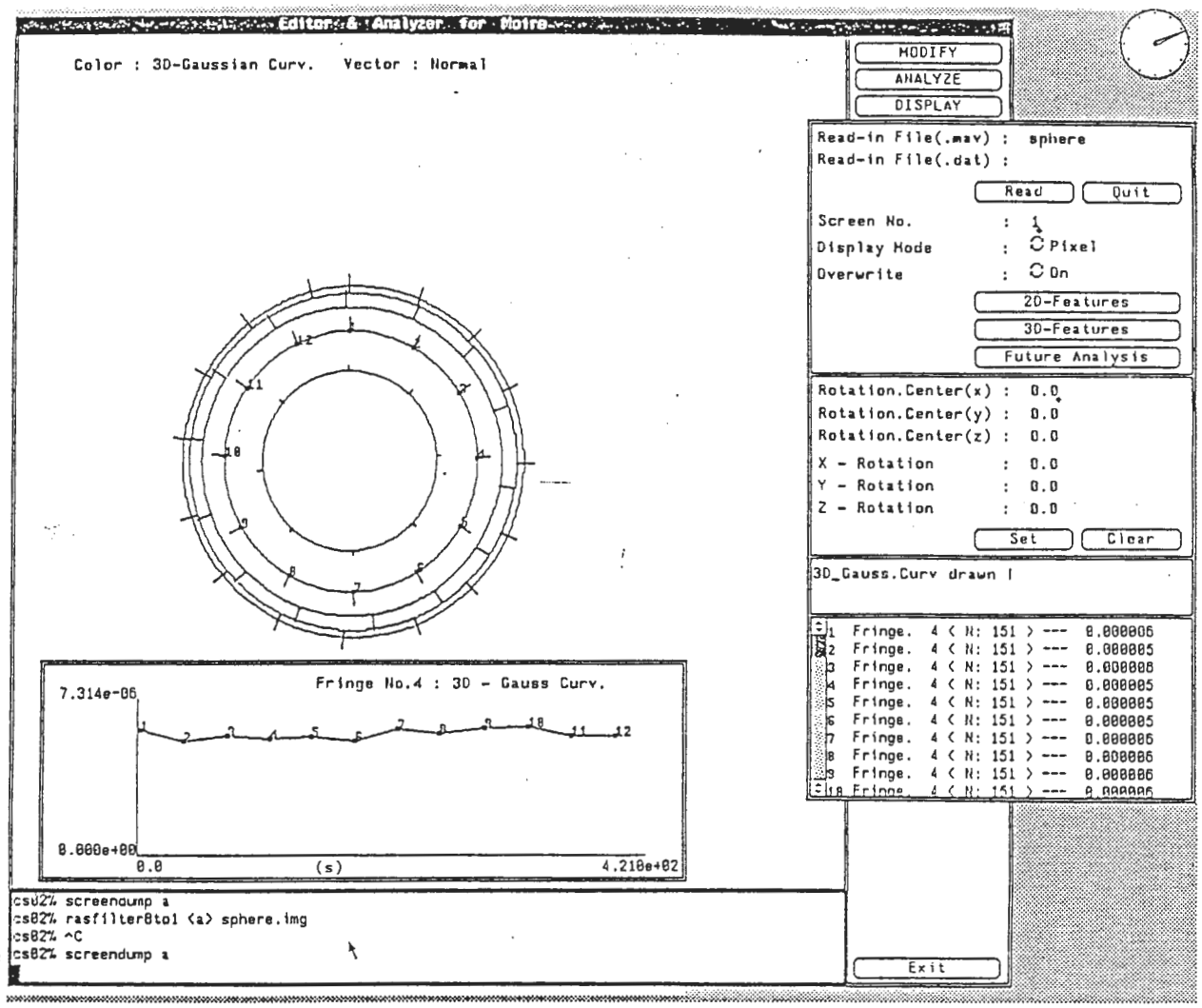


Fig. IV-16 Computation results of a hemisphere.

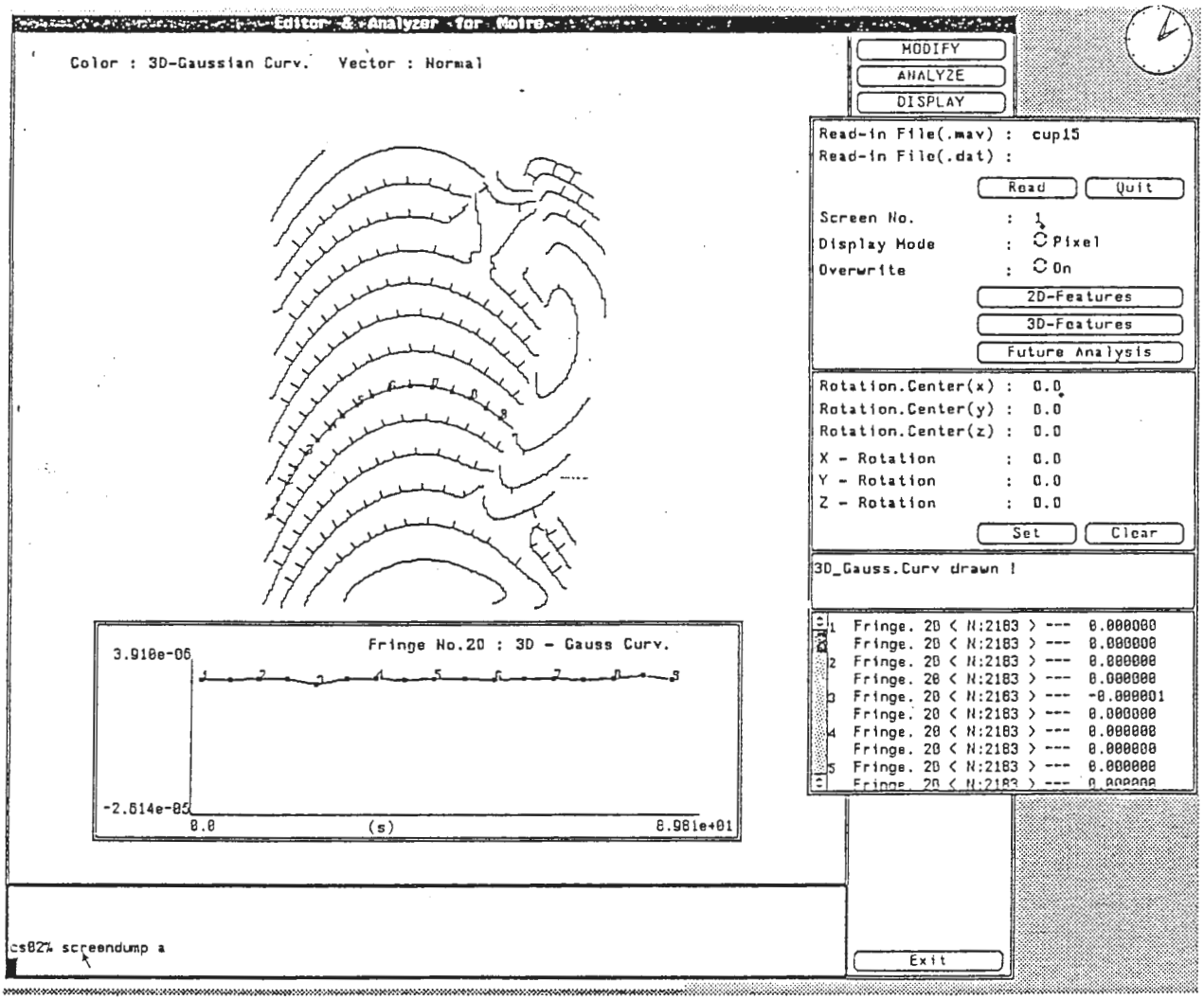


Fig. IV-17 Computation results of a coffee cup.

IV CONCLUSIONS

We have shown how surface curvatures can be computed from the equidistance range contours directly without any explicit estimates of partial derivatives. Our procedure is based on using simple geometric construction to obtain the normal sections and using osculating circles for curvatures computations. Starting with the surface

curves on the 2D contour plane, the algorithm reconstructs the normal sections in a step by step fashion so that the geometric quantities are computed as accurately as possible, within the resolution of the acquisition system. We have also shown by error analysis of the construction steps that the errors are bounded and the computation steps are stable. Our method is general and can be applied to any dense range image data. It should offer advantages in terms of accuracy, speed and robustness against existing methods that rely on the explicit computation or implicit estimation of partial derivatives.

Appendix I

The Derivation of Curvature Formula (1) of section 4.3

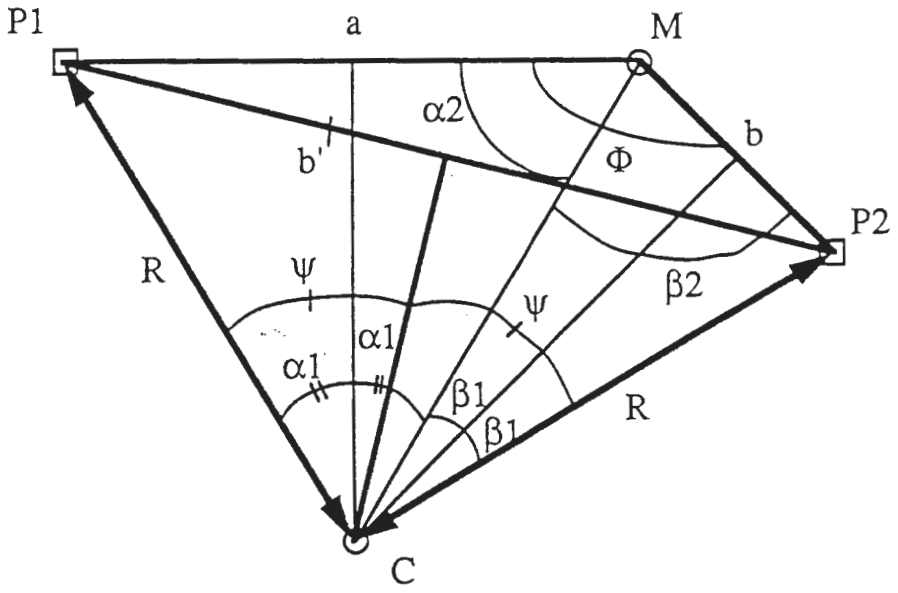


fig. 1 : curvature computation from points M, P1 and P2.

Let P1, P2 and M lie on the circle of center C and of radius R.

The secant P1P2 has length : (1) $b' = 2 R \sin \psi$

But that segment also belongs to the triangle (M P1 P2). Its length is then :

(2) $b'^2 = a^2 + b^2 - 2 a b \cos \Phi$

Now, let us write the angle relations in the figure :

(3) $2 \psi = 2 \alpha 1 + 2 \beta 1$

(4) $\Phi = \alpha 2 + \beta 2$

$$(5)\ \alpha 1 = \frac{\pi}{2} - \alpha 2$$

$$(6)\ \beta 1 = \frac{\pi}{2} - \beta 2$$

If we replace (3) we get :

$$(7)\ \psi = \pi - \alpha 2 - \beta 2 = \pi - \Phi$$

Thus combining (1), (2) and (4) :

$$(8)\ 2\ R\ \sin(\pi - \Phi) = \sqrt{a^2 + b^2 - 2\ a\ b\ \cos\ \Phi}$$

Thus :

$$(9)\ \frac{1}{K} = R = \frac{\sqrt{a^2 + b^2 - 2\ a\ b\ \cos\ \Phi}}{2\ \sin\ \Phi}$$

c.q.f.d.

Appendix II

Normal Coordinates Computation

As seen in Chapter IV II.3.2 we need to compute the sine and cosine of the angle between the normal and the orthogonal vector to the tangent. For that purpose we need to study four cases according to the signs of the curvature k and of the inner product $\overrightarrow{MP2} \cdot \overrightarrow{t}^{\perp}$.

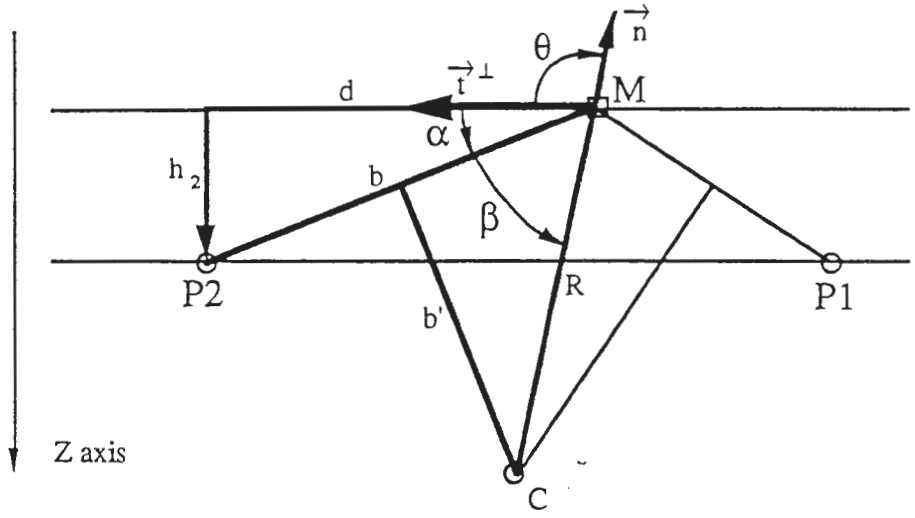
We computed cos θ and sin θ for each of the cases and we summed the results in those formulas :

$$\cos\theta = \frac{\text{sign}(\overrightarrow{MP2} \cdot \overrightarrow{t}^{\perp})}{R}\left(- \text{sign}(k) \frac{d}{2} + h_2 \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)$$

$$\sin\theta = - \frac{1}{R}\left(- \text{sign}(k) \frac{h_2}{2} - d \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)$$

The calculus for each case is detailed in the four following pages.

case 1 : $\boxed{\overrightarrow{MP2} \cdot \vec{t}^{\perp} > 0 \quad \text{and} \quad k > 0.}$



Here we have : $\boxed{\alpha + \beta - \theta = \pi}$ and thus :

(1) $\cos \theta = \cos \alpha + \beta - \pi = - \cos \alpha + \beta$

(2) $\sin \theta = \sin \alpha + \beta - \pi = - \sin \alpha + \beta$

using :

$$\cos \alpha = \frac{d}{b} \qquad \cos \beta = \frac{b/2}{R} x$$

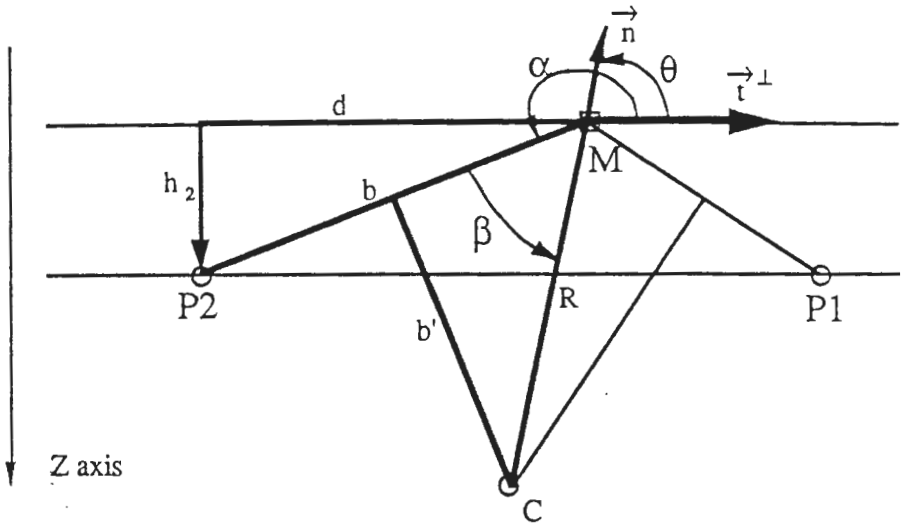
$$\sin \alpha = \frac{h_2}{b} \qquad \sin \beta = \frac{b'}{R} \qquad \text{and} \qquad b' = \sqrt{R^2 - \frac{b^2}{4}}$$

We get :

$$\boxed{(3) \quad \cos \theta = \frac{1}{R}\left(- \frac{d}{2} + h_2 \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

$$\boxed{(4) \quad \sin \theta = \frac{1}{R}\left(- \frac{h_2}{2} - d \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

case 2 : $\boxed{\overrightarrow{MP2} \cdot \vec{t}^{\perp} < 0 \quad \text{and} \quad k > 0.}$



Here we have : $\boxed{\theta - (\alpha + \beta) = \pi}$ and thus :

(1) $\cos \theta = \cos \alpha + \beta + \pi = - \cos \alpha + \beta$

(2) $\sin \theta = \sin \alpha + \beta + \pi = - \sin \alpha + \beta$

using :

$$\cos \alpha = -\frac{d}{b} \qquad \cos \beta = \frac{b/2}{R}x$$

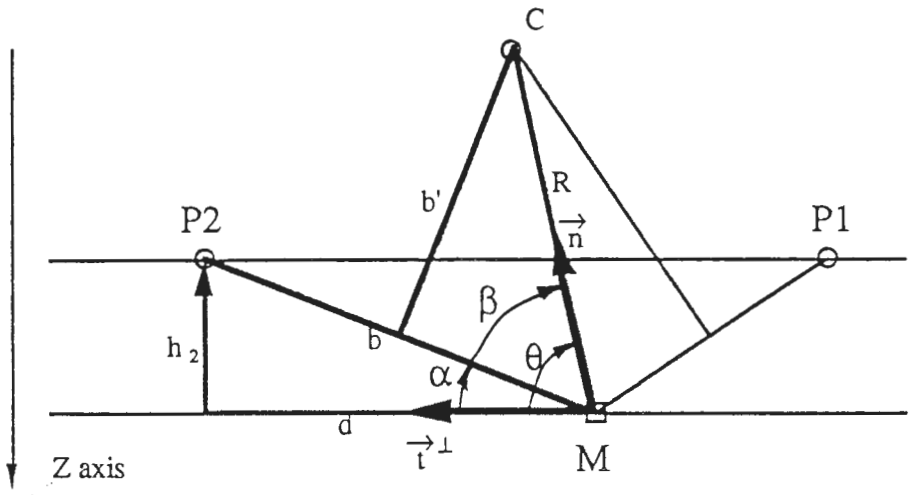
$$\sin \alpha = \frac{h_2}{b} \qquad \sin \beta = -\frac{b'}{R} \quad \text{and} \quad b' = \sqrt{R^2 - \frac{b^2}{4}}$$

We get :

$$\boxed{(3) \quad \cos \theta = \frac{1}{R}\left(\frac{d}{2} - h_2 \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

$$\boxed{(4) \quad \sin \theta = \frac{1}{R}\left(-\frac{h_2}{2} - d \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

case 3 : $\boxed{\overrightarrow{MP2} \cdot \vec{t}^{\perp} > 0 \quad \text{and} \quad k < 0.}$



Here we have : $\boxed{\theta = \alpha + \beta}$ and thus :

(1) $\cos \theta = \cos \alpha + \beta$

(2) $\sin \theta = \sin \alpha + \beta$

using :

$$\cos \alpha = \frac{d}{b} \qquad \cos \beta = \frac{b/2}{R}x$$

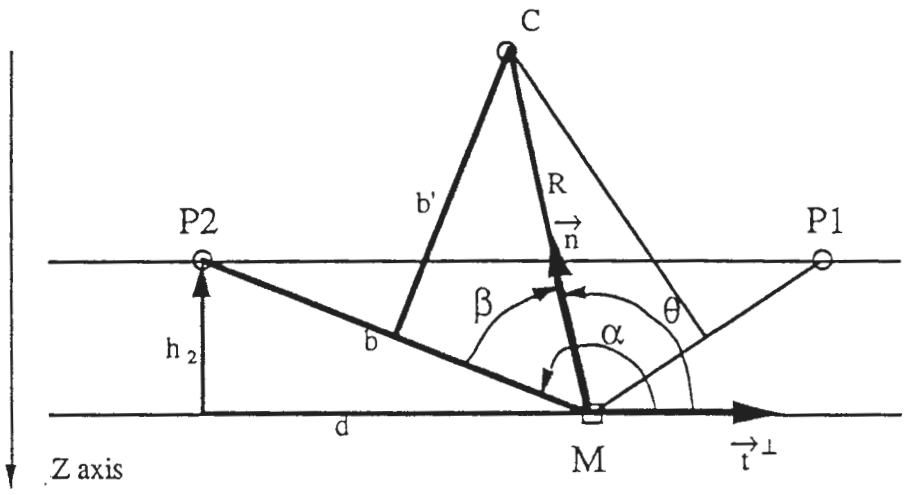
$$\sin \alpha = \frac{h_2}{b} \qquad \sin \beta = -\frac{b'}{R} \quad \text{and} \quad b' = \sqrt{R^2 - \frac{b^2}{4}}$$

We get :

$$\boxed{(3) \quad \cos \theta = \frac{1}{R}\left(\frac{d}{2} + h_2 \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

$$\boxed{(4) \quad \sin \theta = \frac{1}{R}\left(\frac{h_2}{2} - d \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

case 4 : $\boxed{\overrightarrow{MP2} \cdot \vec{t}^{\perp} < 0 \quad \text{and} \quad k < 0.}$



Here we have : $\boxed{\theta = \alpha + \beta}$ and thus :

(1) $\cos \theta = \cos \alpha + \beta$

(2) $\sin \theta = \sin \alpha + \beta$

using :

$$\cos \alpha = -\frac{d}{b} \qquad \cos \beta = \frac{b/2}{R}x$$

$$\sin \alpha = \frac{h_2}{b} \qquad \sin \beta = \frac{b'}{R} \qquad \text{and} \qquad b' = \sqrt{R^2 - \frac{b^2}{4}}$$

We get :

$$\boxed{(3) \quad \cos \theta = \frac{1}{R}\left(-\frac{d}{2} - h_2 \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

$$\boxed{(4) \quad \sin \theta = \frac{1}{R}\left(\frac{h_2}{2} - d \sqrt{\left(\frac{R}{b}\right)^2 - \frac{1}{4}} \right)}$$

References

[Aho et al. 83] "DATA STRUCTURES AND ALGORITHMS" by **A.V. Aho, J.E. Hopcroft and J. D. Ullman**. Addison-Wesley publishing company .

[Carmo 76] "DIFFERENTIAL GEOMETRY OF CURVES AND SURFACES" by **Manfredo P. do Carmo**. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[Cline et al. 84] "AUTOMATIC MOIRE CONTOURING" by **H. E. Cline, W. E. Lorensen and A. S. Holik** in Applied Optics / Vol. 23, No. 10 / 15 may 1984.

[Fan et al. 87] "SURFACE SEGMENTATION AND DESCRIPTION FROM CURVATURES FEATURES" by **T.J Fan, G. Medioni and R. Nevatia**, in Proceedings of Image Understanding Workshop 1987.

[Kling et al. 89] "SURFACE CURVATURES FROM EQUIDISTANCE CONTOURS" by **O. Kling, H. Tanaka and D. Lee,** Submitted to 1989 IEEE International Conference on Computer Vision and Pattern Recognition, San Diego CA, June 4-8 1989.

[Koezuka et al. 88] "A CONSIDERATION ON AUTOMATIC ACQUISITION OF THE SHAPE OF A THREE DIMENSIONAL OBJECT USING MOIRE TOPOGRAPHY" by **T. Koezuka, J.B. Thomine, K. Akiyama and Y. Kobayashi** in Proceedings of the 1988 IEEE International Conference on Robotics and Automation, Philadelphia, Pennsylvania, April 24-29, 1988.