A New Computational Framework For 2D Shape-Enclosing Contours

B. R. Schlei*

Los Alamos National Laboratory, Theoretical Division, T-1,
P.O. Box 1663, Los Alamos, NM 87545, USA

April 28, 2004

Abstract

In this paper, a new framework for one-dimensional contour extraction from discrete two-dimensional data sets is presented. Contour extraction is important in many scientific fields such as digital image processing, computer vision, pattern recognition, etc. This novel framework includes (but is not limited to) algorithms for dilated contour extraction, contour displacement, shape skeleton extraction, contour continuation, shape feature based contour refinement and contour simplification. Many of the new techniques depend strongly on the application of a Delaunay tessellation. In order to demonstrate the versatility of this novel toolbox approach, the contour extraction techniques presented here are applied to scientific problems in material science, biology and heavy ion physics.

Keywords: Contour; Isocontour; Edge; Unstructured grid; Delaunay tessellation; Skeleton; Shape morphology; Material surface; Bacterial colony; Freeze-out hypersurface

*E-mail address: pionpair@aol.com
1 Introduction

In two spatial dimensions, a lower-dimensional interface that partitions a two-dimensional (2D) space into separate subdomains with nonzero areas is called a contour. 2D spaces can be either continuous or discrete with respect to a field quantity that is defined across that space. For example, a 2D gray-level image represents a discrete 2D space with respect to the field quantity gray-level. Its area, which is covered by the image is broken into many regular 2D cells, i.e., pixels (= picture elements); each pixel has a constant shade of gray.

Many papers [1]–[6] have been written on the extraction of one-dimensional (1D) contours from 2D image data. It is not trivial to define a contour for a discrete space. For example, one has to specify how the final contour should be supported. Some contour extraction algorithms yield contours that connect only the centers of edge pixels (cf., e.g., Ref. [7]; an edge pixel is a pixel that is considered to represent a part of the boundary of a certain region of interest within a given image). Others may allow for the usage of points that lie on the boundary between two pixels (cf., e.g., Ref. [8]). More complications may arise because a resulting contour may not be closed or it may not enclose an area larger than zero (in the latter case, a contour is called degenerate). Furthermore, a contour may be self-intersecting and then it may enclose more than one of the 2D regions.

One also has to consider the level of information that is provided for building contours. In some applications, a 2D image is preprocessed through image segmentation [8]–[10], i.e., pixels are grouped together into so called blobs. A contour extraction method may then be applied to the image blobs. In other applications, a 2D image is preprocessed by an edge detector (cf., e.g., Ref. [11]), i.e., edge pixels are identified that are assumed to describe the transition of two different neighboring regions (note, that an edge pixel is a 2D object rather than a section of a 1D contour). A contour extraction method may then be applied to the resulting edge pixels. In particular, complications may arise when the edge pixels provided by an edge detector form only partially connected chains, or when the transition region (given by the edge pixels) of two zones in a 2D image exceeds the width of more than one pixel.

Refs. [1]–[10] demonstrate that many different image processing problems have resulted in many different approaches for building 1D contours from 2D image data. It is therefore the intent of this paper to provide a single computational framework for building 2D shape-enclosing contours from many different types of 2D discrete data sets, such as 2D gray-level images, or 1+1D (i.e., 1D space + 1D time) hydrodynamic simulation data, respectively. The framework presented here handles many different 2D image processing problems with one and the same set of tools. In particular, this novel framework will provide solutions for the problems described above and for others which one may be faced with when extracting contours from discrete 2D data sets.

This paper is structured as follows. In the next section, the contour extraction framework is explained. The topics that are covered in this section include (but are not limited to) dilated contour extraction [12], contour displacement, shape skeleton extraction, gap closure or contour continuation, shape feature based contour refinement and contour simplification. This section is followed by an application section, where the novel toolbox
approach for 1D contour extraction is applied to scientific problems in material science, biology and heavy ion physics, respectively. In particular, we address region-enclosing contour extraction for electron backscattered diffraction imagery, bacterial colony counting from image data taken from Petri dishes, and freeze-out hyper-surface extraction for 1+1D relativistic hydrodynamic simulation data. Finally, this paper concludes with a summary.

2 The Contour Extraction Framework

In this section, we describe how to extract 1D contours from discrete 2D spaces. As mentioned above, a discrete 2D space could be a 2D image. However, a proper discrete 2D space could also be given through the union of all triangles resulting from a 2D Delaunay tessellation [13] (including some additional field quantities that characterize each triangle further), etc. [2]. In the next subsection however, we shall restrict ourselves - without loss of generality - to the case of 2D images. Note, that the following contour extraction algorithm [12], which is also known under the name DICONEX, has been implemented successfully into software [14].

2.1 DICONEX - DIlated CONtour EXtraction

In the following subsection, a fast three-step ((A), (B), (C)) algorithm is described, which always yields perfect contours [15]–[17] for both binary and gray-level images. The contours are perfect in the sense that they are non-selfintersecting and non-degenerate, i.e., the contours always enclose an area larger than zero. Therefore, let us assume, that a 2D image has been segmented, i.e., all of its pixels have been assigned a new field quantity (e.g., as in the case of binary images, where pixels are either black or white, or any other given pair of colors or gray-levels, respectively). Fig. 1.a shows a binary image with 25 white and 11 gray pixels. Let us now construct contours for the gray pixels.

(A) As a first step, a set of disconnected vectors is constructed, which separates white pixels from gray ones. Each vector is attached to a pixel with its origin and its endpoint in such a way that the pixel always lies to the left of the vector (cf., Fig. 1.b). This ensures the counterclockwise circumscription of all pixels (or clusters of pixels) by the vectors. Conversely, all holes in a pixel cluster (blob) are circumscribed clockwise (cf., Fig. 1.c and Fig. 1.d). Note, that in order to accomplish this pixel enclosure by oriented vectors, it is only necessary to consider the four nearest neighbors of any given pixel, i.e., its upper, left, lower, and right pixel neighbor (cf., Fig. 1.b). Each vector is unique, double counting can never occur. Furthermore, there is no specific order required in which the neighborhood of any given pixel is evaluated. Therefore, this processing step is totally parallel.

(B) In the second step, which is linear, connected loops are constructed from the previously generated contour vector set. First the vectors are enumerated. Then, a new set is created, which is the set of corresponding vector origins (starting points) and endpoints. Every point is listed only once, but for each point we also list which vectors (their
identities are given through their enumeration) are connected to a given point. The reader can convince himself easily, that there are always either two or more (e.g., four in the case of pixel processing) vectors connected to a point in the point list. At this stage, all the knowledge for connecting the vectors is available. Initially, all edges of the contour vector set are labeled as “unused”. We take the first element of the contour vector set to construct the first contour loop. Its origin is added to a newly formed contour point chain list. We label this first edge as “used” and then look up the point list in order to examine to which other vectors’ origin our current vectors endpoint is connected to. At this point, we have to distinguish between two cases. In the first case, only two vectors are connected to the endpoint of the already as “used” labeled vector, wherein we just consider the as yet unused vector as our next contour vector. In the second case, more than two vectors are connected to the endpoint of the already as “used” labeled vector. At this point the user decides if the next pixel should be disconnected or connected to the current blob whose enclosing contour is being constructed; we either choose a left-turn (cf., Fig. 1.c) or a right-turn (cf., Fig. 1.d), respectively.

In doing so, we always ensure a consistent choice for building the contours. We thereby either weaken the connectivity between pixels, that touch each other only in one point, or we strengthen it. (In fact, as we shall see below, the dilated versions of the contours will lead either to a total separation (cf., Fig. 1.e) or to a merging (cf., Fig. 1.f) between two pixels that only share one common (corner) point.) After having found the next contour vector, we add its origin to the contour point chain list, and then we label that new edge as “used”. We iterate this procedure, until we encounter the first “used” element of the contour vector set. This concludes the construction of the first contour. While creating the contour, we may count the number of vectors used so far and compare it to the total number of vectors present in the contour vector set. If there are still unused edges present in the contour vector set we scroll through the edge list, until we find the first next “unused” edge and use it as the first edge of the next contour. We repeat the above algorithm to create the new contour. This process is repeated until eventually all edges have been used. The result is a set of point chains, of which each one represents a closed contour. At this point, we may note that the smallest contours from edge pixels consist of only four edge points. The latter result is valid for all isolated gray pixels (considering left-turns in the contour construction), i.e., those, which have no other gray pixel as one of the nearest four neighbors. Fig. 1.c and 1.d show the left-turn and right-turn contours, respectively, for the rendered bi-level (binary) image shown underneath.

(C) In particular, in the generated set of point chains each point chain is a list of the origins of the contour vectors. If one replaces these points with the midpoints between the origins and the endpoints of the contour vectors without changing the connectivity in the contour point chain list, one obtains modified contours, which are a dilation of the centers of the grain edge pixels. Figs. 1.e and 1.f depict the dilated contours according to the technique outlined here. We would like to emphasize, that the dilated contours are non-degenerate and they never cross or overlap each other. Furthermore, the dilated contours are oriented with respect to the shapes and their possible holes.

Note, that in some of the following figures the arrow heads of the vectors which represent either left- or right-turning dilated contours are not shown.
2.2 Boundary pixel tracing contours

Another type of contour, which is different from the previously discussed DICONEX (or dilated) contours can be obtained by tracing the pixel boundary of image blobs [7]. Such boundary pixel tracing contours (BPTCs) have the advantage that they can be obtained with very little memory requirement while making use of chain codes. A chain code is a sequence of directions (typically indicating the shortest path to one of the next eight neighbors of a given pixel), when starting from a particular boundary pixel of a given image blob. However, when holes are present in the shapes of a given pixel cluster, an algorithm may be unable to construct the contours successfully [6].

The DICONEX algorithm can also be used to construct BPTCs, i.e., contours which are supported by points that reside in the center of a boundary pixel. This can be accomplished by modifying step (C) in the previous subsection as follows. Instead of replacing the origin of a contour vector with the midpoint between its origin and its endpoint, the origin of a contour vector is moved to the center of the pixel to which the contour vector is attached to initially. Fig. 2 shows BPTCs according to this technique.

Note, that there are two contour solutions for the pixel configuration shown in Fig. 2 (cf., also Fig. 1.a), since the user has to decide, if a pixel that is in contact with another pixel in only one point should be disconnected from or connected to its apparent partner. In fact, it is this ambiguity that may cause an algorithm to crash in its effort to construct BPTCs successfully, because it may not have accounted for such cases consistently. Furthermore, we would like to stress that the contours shown in Fig. 2 are self-intersecting and also - partially or fully - degenerate (cf., e.g., the point in Fig. 2.a is a fully degenerate contour). In the following, we shall not make any further use of BPTCs.

2.3 Isocontours

Sometimes, 2D gray-level image data are processed with the intent to extract isocontours. An isocontour is a contour which has a constant value at all of its supporting points with respect to the field quantity that has been used for the contour extraction. The contour supporting points, which fulfill such a condition, are usually continuously distributed across the discrete 2D image data. Because the points which support a DICONEX contour always coincide with the midpoints of the boundary edge between two pixels, dilated contours are in general not isocontours. However, these contours can be transformed into isocontours very easily as the following example will demonstrate.

Fig. 3.a shows a 2D gray-level image with 36 pixels. Here, black pixels have a gray-level of value zero, whereas white pixels have a gray-level of value 255. We shall now construct an isocontour corresponding to a gray-level of value 100. First, all pixels with a gray-level of value larger than or equal to 100 are enclosed with contour vectors (cf., step (A) of subsection 2.1) as depicted in Fig. 3.b. From these contour vectors a dilated contour is constructed as shown in Fig. 3.c. In Fig. 3.c, additional vectors are drawn for each of the points that support the DICONEX contour. We will refer to these additional vectors as range vectors; each range vector connects the centers of the pair of pixels that share the boundary edge of the initial contour vectors. The origin of a range vector co-
incides with the pixel center of the larger gray-level value, whereas its endpoint coincides with the pixel center of the smaller gray-level value.

Fig. 3.d helps to illustrate how DICONEX contours can be transformed into isocontours. Two pixels - one with a gray-level value of 133, the other one with a zero valued gray-level - are initially separated by a DICONEX contour section that is located exactly in the middle between them (dotted line). A range vector connects the centers of the two pixels. Note, that the range vector defines the bounds within which the support point for a dilated contour may be displaced. Furthermore, the centers of each pixel are assumed to correspond exactly with their gray-level values. Since the isocontour is supposed to represent a gray-level of value 100, it should not be positioned at the middle of the range vector. This medium position actually represents a gray-level of value 66.5 assuming a linear interpolation between the gray-level bounds. In fact, the “true” location of the support point for the isocontour is located closer - and therefore has to be shifted - towards the center of the pixel with the gray-level of value 133. Hence, the isocontour (solid line) is supported by a point, which is located within the pixel with the gray-level of value 133. Note, that within this paper we use linear interpolation of gray-levels for moving the contour support points, although other interpolation techniques may be used instead.

In Fig. 3.e, the dilated contour of Fig. 3.c has been transformed into an isocontour representing a gray-level of value 100. Note, that the support points of the isocontour do not lie on a perfect circle due to a slight asymmetry in the radial distribution of gray-levels with respect to the center of the original image.

By using range vectors, one can transform a dilated contour also into a boundary pixel tracing contour (cf., the previous subsection). One simply has to move all points which support a dilated contour to the origins of the corresponding range vectors. In Fig. 3.f, a BPTC is shown based on this latter method. We would like to stress, that DICONEX contours can become self-intersecting or degenerate, when they are being transformed into isocontours by the above described procedure. However, the orientations of the DICONEX contours will be inherited by the isocontours, i.e., shapes with an area larger than zero will be circumscribed counter-clockwise, whereas holes with an area larger than zero will be circumscribed clockwise, respectively.

2.4 Delaunay tessellation and shape skeleton

Edge detection algorithms such as the Canny edge detector [11] return, when applied to a 2D gray-level image, a set of 2D pixels rather than 1D contours or contour segments. However, we would like to be able to generate 1D contours from a given set of edge pixels as well. In the following discussion, a set of edge pixels will represent one or more 2D shapes from which we extract one or more skeletons. Such skeletons will then represent 1D contours or (at least) contour segments.

Given a set of edge pixels, we shall first generate for each shape that is formed by the edge pixels one or more DICONEX contours. Note, that a shape with $N$ holes will yield exactly $N + 1$ contours. The dilated contours and their supporting point set then form the input to a constrained Delaunay tessellation (CDT). A constrained tessellation is applied, so that only the initial point set is used; no additional “Steiner” points [13]
are added. Furthermore, a subset of the triangular mesh edges, which is given by the initial set of dilated contour edges, is not altered. The CDT of a simple planar polygon (contour) is a decomposition of a polygon into triangles, such that the circum-circle of each triangle contains no vertex of the polygon inside it that is simultaneously visible to two vertices of the triangle [13] [18]. In Fig. 4.a, the interior of a 2D shape (which is enclosed by DICONEX contours) is decomposed into a set of triangles while applying a CDT. Note, that the CDT of the contours is the key step that allows for the skeleton extraction of the shape.

The triangles originating from the Delaunay tessellation can be classified into four types, namely those with three, two, or one external (i.e., polygonal boundary) edges, and those with no external edges, respectively. Triangles with three external edges are called isolated triangles, because none of their edges connects to another triangle. In the following, these are of lesser importance to us. Each kind of triangle carries morphological information [18]–[20] about the local structure of the shape’s enclosing polygon. Accordingly, they are given different names. A triangle with two external edges marks the termination of a “limb” or a protrusion of the polygon and is called a termination triangle or a T-triangle. A triangle with one external edge constitutes the “sleeve” of a “limb” or protrusion, signifying the prolongation of the polygon, and is called a sleeve triangle or S-triangle. Finally, a triangle that has no external edge determines a junction or a branching of the polygon, and is accordingly called a junction triangle or a J-triangle.

For each triangle, line segments or single points can be drawn ([cf., Fig. 4.b]), which in their union represent a skeleton of the processed shape.

Fig. 4.a shows in addition to the CDT of the 2D shape’s interior, also its shape skeleton. Because of small variations along the shape’s enclosing contour, structurally unimportant skeleton features may occur. However, these unimportant skeleton features can be removed (or pruned) according to the following method [19]. First, all junction triangles are evaluated based on their nesting level within a given shape. Junction triangles which are nested most deeply are processed first, those which are closest to the shapes contour(s) are processed last. Fig. 4.c provides an example of pruning. For a given junction triangle \(ABC\), we shall consider its edge \(AB\) and the shapes contour section \(AopqrB\). For each point of the set \(P = \{o, p, q, r\}\), we compute the distance \(d\) to the junction triangle’s edge \(AB\). Let \(\rho \equiv d/|AB|\) be the ratio of morphological significance. If for any of the points in \(P\), the ratio \(\rho\) exceeds or equals a fixed threshold, e.g., \(\rho_0 = 0.6\), the contour section and the corresponding skeleton branch will be preserved. However, if all of the points in \(P\) have a ratio \(\rho\) with \(\rho < \rho_0\), all skeleton parts which belong to the area covered by the closed polygon \(AopqrBA\) will be removed. In addition, the junction triangle will turn into a sleeve triangle, where the triangle edge \(AB\) becomes a new (virtual) shape boundary edge. Note, that before the junction triangle is turned into a sleeve triangle, the above algorithm is also applied to the edges \(BC\) and \(CA\), respectively. Thus, a junction triangle can change even into a terminal or into an isolated triangle, if more than one of the skeleton branches are removed.

Junction triangles, which have been removed by the pruning algorithm before they have been processed for pruning themselves will not be considered any further by the pruning algorithm ([cf., e.g., junction triangle \(ApB\) in Fig. 4.c]). This will ensure a fast
processing for pruning. In Fig. 4.d, the pruned skeleton for the initial 2D shape of Fig. 4.a is shown. How this shape skeleton can be best converted into 1D contours and how gaps can be closed in the presence of contour segments, will be explained in detail in the next subsection.

2.5 Frame addition and gap closure

Edge pixels are very often generated in order to decompose 2D image data into several disjunct regions or shapes. In the previous subsection, we explained how one can obtain skeletons from shapes that represent clusters or chains of such edge pixels through their enclosing DICONEX contours. However, in general these skeletons may not provide a complete partitioning of the underlying 2D space (or image data), because some of the initial edge pixel chains may have been fragmented such that the generated skeletons are not closed contours. Furthermore, edge pixels are usually not generated for the image boundary itself, such that skeleton extraction will not yield any contour segments that mark the outer boundary of the image data.

Let us consider the discrete 2D space (image) with 4x4 pixels as shown in Fig. 5.a. Furthermore, we are given a skeleton, that suggests that our space should be partitioned into three areas. In order to convert the skeleton into shape-enclosing contour sections, we proceed as follows [17]. First, two vectors are assigned to each line segment of length larger than zero of the shape skeleton. The length and orientation of each vector pair coincide with the length and the orientation of each of the skeleton’s line segments, but their directions are chosen to be opposite (cf., the shape skeleton in Fig. 5.b). Because we intend to partition the whole image area, a frame is added around the original image given in Fig. 5.a (cf., Fig. 5.b). In fact, the added outer frame consists of a set of single vectors that are arranged counterclockwise around the original image as shown in Fig. 5.b. The point set that supports the frame’s vector set is sampled at the rate of pixels available along a side of the image, and it always includes the four corners of the image. In Fig. 5.b, it can be seen that the shape skeleton is disconnected from the images (vector set) frame, i.e., there are gaps present between limb-like arcs of the skeleton and the image frame.

In order to close the gaps we apply the following technique. A CDT is applied to the point set, which supports the skeleton and the image frame. Fig. 5.c shows the Delaunay triangular mesh for the initial state given in Fig. 5.b. Certain edges (shown dotted in Fig. 5.d) of this unstructured grid connect the terminal points of the skeletons’ limb-like arcs with the outer image frame. For instance, one could either select the shortest edges (cf., Fig. 5.e) or the edges, which preserve mostly the orientation of the terminating vector pair in a skeleton’s limb (cf., Fig. 5.f), among the edges bridging a gap between a terminal point and a frame point. In fact, the gaps are closed with vector pairs of opposing direction. Considering the vector sets of (i) the shape skeleton, (ii) the image frame, and (iii) the gap vectors, one can now apply the above described contour element connection algorithm (B) (cf., subsection 2.1) in order to connect all vectors into discrete region-enclosing contours (only left-turns will be performed this time, whenever junctions are encountered). Hence, we end up partitioning our image area into three disjunct shapes, each of which is enclosed by a non-degenerate contour. Note, that this gap closure technique can
be extended easily to general gap closure or edge continuation between disjunct skeletons.

### 2.6 Contour refinement using shape features

Very often shapes, that have been extracted from 2D image data and that are represented by their shape-enclosing contours, do not resemble the shapes that a human observer would have extracted from the image data. For example, Fig. 6.a shows a gray-level image with six circular shaped dark pixel clusters, of which four of them slightly overlap. Isocontour extraction as described in subsection 2.3, gives us three shape-enclosing contours for an isovalue of 130 as shown in Fig. 6.b, but not six. Note, that the initial DI-CONEX contours are chosen to be left-turning, i.e., pixel disconnecting. In the following, we describe a technique [21], that allows us to refine the initially found three contours (i.e., we break-up one of the contours into four such that the final count of contours will be six) by making use of higher-level shape features.

To be more specific, it is the intent of this contour processing tool to take a 2D shape and to decompose it into convex shaped constituents. In Fig. 6.c, the interiors of all of the three contours have been decomposed with a constrained Delaunay tessellation. Furthermore, the unpruned shape skeletons are depicted. Let us now introduce high-level shape features, which we are going to call limbs and torsos. Ref.s [18]–[20] have made clear the value of Delaunay triangulations in obtaining structurally meaningful decompositions of shapes into simpler components, similar to the human visual system parsing of complex shapes. 2D shapes can be decomposed into limbs and torsos [18, 19] which can be viewed as generic shape components. A “limb” is a chain complex of pairwise adjacent triangles, which begins with a junction triangle and ends with a termination triangle. A “torso” is a chain complex of pairwise adjacent triangles, which both begins and ends with a junction triangle (cf., Fig. 6.d). Thus each limb and torso, is represented by a line segment in the shape skeleton, which have their endpoints accordingly in points of bifurcations and/or terminations (note, that a string-like shape, i.e., \( TS...ST \), is a degenerate limb, whereas a torus-like shape, i.e., \( S_oS_1...S_NS_o \), is a degenerate torso).

For our particular task of contour refinement the torsos play an important role. In Fig. 6.e, some of the junction triangles are enumerated. Let us now consider the torsos which are encapsulated by the junction triangle pairs 1&2, 3&4, and 5&6, respectively. For these torsos only none of the longest edges of both encapsulating junction triangles face their other junction triangle partner. A vector pair (with opposing directions) is placed at the narrowest local width of each torso, eluding to its break-up (cf., Fig. 6.e). Note, that the initial shape-enclosing contours consist of vector sequences, which enclose the shapes counterclockwise. After insertion of the vector pair, we apply the above described contour element connection algorithm (B) (cf., subsection 2.1) in order to connect all vectors into discrete region-enclosing contours (only left-turns will be performed, whenever junctions in the contours are encountered). Hence, we end up partitioning the contour representing the four overlapping circular shaped dark pixel clusters into four separate contours, whereas the other two contours remain unchanged (cf., Fig. 6.f). Note, that the eventual vector pair insertion has resulted in the areal splitting of shapes, similar to the decomposition of the discrete 2D space as described in the previous subsection.
2.7 Contour simplification

When boundary pixel tracing contours and/or isocontours are constructed by the techniques outlined in this paper, some of the final contour vectors may have a zero length and some points which support the final contours are accounted for multiple times. We leave it up to the reader to implement a proper filter, which removes the zero-length vectors and the redundant points.

Very often it is possible to reduce the number of points in the region-enclosing contours further. For example, contour sections which form straight lines may be represented by two points only. The same may apply for contour segments which turn only very little. For the down-sampling of contours various techniques [22]–[24] have been proposed. Here, a method [17] is presented, which requires only local contour information when points are evaluated for removal. This technique is based on the repeated application of constrained Delaunay tessellations.

As a first step, a network of connecting contours is broken up into contour segments, which reach from one bifurcation point to the next. In particular, the corners of an image frame are considered as bifurcation points. Contours, which loop back onto themselves without being in contact with another contour are broken up at an arbitrary point (note, that other choices are possible). All contour segments are evaluated piecewise for the removal of supporting points. The two filters described in the following paragraphs are applied to the contour segments, and the segments are finally reattached to their initially given network. Without loss of generality we consider here the example shown in Fig. 7.

(i) Points between the two endpoints of a contour segment that lie on a straight line are removed. Figs. 7.a-7.c depict this procedure. The initially given contour segment consists of the five line segments $12, 23, 34, 45,$ and $56$, respectively (cf. Fig 7.a). Starting with the pair $12$ & $23$, there is no directional change between these two line segments. Therefore, point 2 is marked for removal (cf. Fig 7.b). However, there are directional changes between the subsequent pairs $23$ & $34$, $34$ & $45$, and $45$ & $56$. No further points are marked for removal during this processing step. Only point 2 is removed from the contour supporting point set (cf., Fig. 7.c).

(ii) Points between the two endpoints of a contour segment that fit into a minimum-enclosing rectangle (MER) of a certain width are removed, provided the last considered line segment has an accumulated turn-angle $\omega$ with $|\omega| \leq \pi/2$. Figs. 7.c-7.k depict this procedure. The initially given contour segment consists of the four line segments $13, 34, 45, \text{ and } 56$, respectively (cf., Fig 7.c). Starting with the pair $13$ & $34$, there is a directional change between these two line segments, which is larger than $\pi/2$, i.e., $|\alpha| > \pi/2$ (cf., Fig. 7.d). No point is marked for removal and the next pair, $34$ & $45$, is considered. Here the directional change between the two line segments is smaller than $\pi/2$, i.e., $|\beta| < \pi/2$ (cf., Fig. 7.e). Now a CDT of the three points 3, 4 and 5 is performed and a MER is placed around the convex hull (which is given here by a triangle) of the three points. Let us assume that the width $w_1$ of the MER is less or equal to an initially given threshold $w_0$. In this case point 4 is marked for removal (cf., Fig. 7.f) and the next line segment $34$ is added to the set $\{34, 45\}$. Now the directional change between the two line segments $34$ & $45$ is evaluated. The modulus of the accumulated turn angle is again smaller than
$\pi/2$, i.e., $|\gamma| < \pi/2$ (cf., Fig. 7.g) and this time a CDT of the four points 3, 4, 5 and 6 is performed. A MER is placed around the convex hull of the four points. This time, let us assume that the width $w_2$ of the MER is larger than the initially given threshold $w_0$ (cf., Fig. 7.h). Point 5 cannot be considered for removal, because the latter MER exceeds the initially given tolerance. The algorithm recognizes the end of a segment series and attempts to start again with line segment 56 and its successor (which does not exist) in order to evaluate the removal of point 6 (cf., Fig. 7.i). However, there is no further line segment; point 6 is the last point of the contour segment under consideration. No further points will be marked for removal for this contour segment. Finally, the marked point 4 is removed from the contour segment (cf., Fig. 7.k). The final contour segment consists of the four points 1,3,5 and 6, respectively.

This subsection concludes the theoretical section of the new framework for 1D contour extraction from discrete 2D data sets.

3 Applications

In this section, several applications for 1D contour extraction from discrete 2D data sets are discussed. It is the particular intent of this section to demonstrate the versatility of the above described toolset for contour extraction by addressing a rather diverse group of applications. In the first application for electron backscattered diffraction imagery, we use dilated contour extraction, Delaunay tessellations, shape skeletons, frame addition, gap closure, and contour simplification, respectively (cf., subsections 2.1, 2.4, 2.5 and 2.7). In a second application, we use dilated contour extraction, isocontour extraction, a Delaunay tessellation, shape skeletons and contour refinement, respectively (cf., subsections 2.1, 2.3, 2.4 and 2.6), in order to provide improved image processing techniques for bacterial colony counting. In a third and last application, we process 1+1D relativistic hydrodynamic simulation data with dilated contour and isocontour extraction, respectively (cf., subsections 2.1 and 2.3), in order to obtain a freeze-out hyper-surface for subatomic multi-particle production.

3.1 Region-enclosing contours for EBSD imagery

The accurate characterization of the structures and properties of grain boundary networks is one of the fundamental problems in interface science. The Electron BackScattered Diffraction (EBSD) technique provides experimental results on grain boundary properties and grain growth in metal surfaces. In EBSD experiments, images of various material surfaces are recorded by secondary electron or backscattered contrast and corrected for instrumental distortions. To extract the information contained in the images, it is important to locate the grain boundaries and triple junctions between grains. This localization task is accomplished by shape processing techniques, which have been presented in the previous sections. The resulting region-enclosing contour information is essential for mesh generation and the characterization of the morphology and topology of grain distributions.
In this section, we follow the processing of an experimental EBSD image with the above described shape processing algorithms. Fig. 8 shows an example of a backscattered contrast image [27] of a thin Aluminum film with a columnar grain structure. Using an image which is recorded simultaneously from secondary electron emission, researchers are able to determine the grain edge pixels with rather standard [8]–[10] pixel processing techniques (cf., Refs. [25, 26]; e.g., in Ref. [28], grain boundary information is obtained from imagery taken by transmission electron microscopy). Fig. 9 shows the resulting bi-level image with grain edge pixels rendered in black.

In the following, we shall restrict our discussion to a sub-region (cf., Fig. 10.a) of the bi-level image shown in Fig. 9 without loss of generality. The goal is to process the given grain edge pixels to obtain as a final result a set of contours, where (i) each contour encloses a grain (region) counter-clockwise with a minimum number of supporting points according to the initially given edge pixels, and where (ii) the whole area of the image has been taken under consideration (cf., Fig. 10.d). This can be accomplished as follows.

First, right-turning, i.e., pixel connecting, DICONEX contours are generated for the gray pixels shown in Fig. 10.a. Then, a CDT is applied to the dilated contours and their supporting point set, in order to decompose the interior of the shape into triangles. Using the individual morphological roles of the triangles, a pruned skeleton (with \( \rho_0 = 0.6 \)) is generated. The shape skeleton is shown in Fig. 10.b. It encloses only three of the nine visible grains fully. Because it is our intent to construct region-enclosing contours for all nine visible grains, a frame (which is also shown in Fig. 10.b) is added around the original image area. A subsequent CDT is applied to the point set of the skeleton and of the image frame (cf., Fig. 10.c). For this current application, we choose to preserve the orientation of the last line segment for a limb-like skeleton arc as much as possible when connecting it to the frame (cf., Fig. 10.c). Considering the vector sets of (i) the shape skeleton, (ii) the image frame, and (iii) the gap vectors, we apply the above described contour element connection algorithm (B) (cf., subsection 2.1) in order to connect all vectors into discrete region-enclosing contours (note, that only left-turns will be performed, whenever contour junctions are encountered). However, the region-enclosing contours are rather densely sampled. In Fig. 10.c, the contours are sampled with 599 points, and the Delaunay mesh shown here consists of 986 triangles. Therefore, we simplify the contours with the technique outlined in subsection 2.7 (in particular, we have chosen \( w_0 = 0.7 \) of the pixel width). In Fig. 10.d, the final contours are sampled with only 38 points, and the Delaunay mesh shown here consists of only 64 triangles! Note, that the region-enclosing contours stay within the limits defined by the gray pixels.

Full processing of the binary image shown in Fig. 9 with 25,518 grain edge pixels leads to only 1,368 region-enclosing contour support points for 177 grain regions, and the accurate coverage of all grains requires only 2,678 Delaunay triangles (cf., Fig. 11). The maximum processing time is below 5 seconds on a 800 MHz Pentium III CPU, which has LINUX as an operating system [17].
3.2 Bacterial colony counting

In studies of the population dynamics of the intestinal population of mice, the most basic measure of how the population of a certain species is behaving is the abundance of the organisms \[29\]. As such, fecal sampling and plating at several dilutions on selective media allows the biologist to get a measure of the abundance of organisms in the lower colon of mice. Counting colony forming units (CFUs) is a traditional way of measuring the population density of any bacterial culture or natural bacterial source. Fig. 12 shows the interior of a Petri dish with E. coli colonies, which have been recovered from the fecal pellets of mice \[30\]. E. coli is the abbreviated name of the bacterium in the family Enterobacteriaceae named Escherichia coli.

In clinical studies, researchers usually take repeated samples for statistical reasons. Giving the mice under consideration various treatments which alter their intestinal flora results in different effects in the population densities. Even a small experiment with – let’s say – 36 mice, will produce hundreds of plates per sample point and can be sampled several times daily. The abundance of visual data makes the automation of visual bacterial colony counting highly desirable. However, the separation of overlapping colonies is a challenging task for standard image processing techniques, i.e., it is not trivial to provide the correct number of CFUs present in a given image. In Fig. 13, we show all 415 isocontours (for an arbitrarily chosen isovalue of 130) that have been obtained for Fig. 12 by the contour extraction techniques outlined in subsections 2.1 and 2.3, respectively. In particular, some of the contours enclose more than a single CFU. Therefore, this current number of contours does not reflect the correct number of CFUs present in Fig. 12.

In subsection 2.6, it was explained how one can refine the contours of Fig. 13, in order to decompose the enclosed shapes into a maximum of convex shaped constituents (cf., also Ref. \[21\]). In fact, Fig. 6.a is a subregion of the image in Fig. 12 at an increased resolution. In subsection 2.6, we were able to count all of the shown CFUs correctly. Fig. 14 shows the refined contours of Fig. 13. Note, that the number of contours (and therefore the number of CFUs) increases from 415 in Fig. 13 to 451 in Fig. 14. However, there are still quite a number of contours, which the reader possibly would have refined as well. The contour refinement technique as described in subsection 2.6 is very sensitive to the particular result of the constrained Delaunay tesselation. The CDT in return, is very dependent on the quality of the contours. Small variations in the contours can be caused, e.g., by noise in the image data. However, it is beyond the scope of this paper to address a proper treatment of noise (as well as other topics such as proper illumination of the sample, etc.) for the image data shown in Fig. 12. Hence, we conclude this subsection with the notion that we have obtained a significant improvement in our attempt to count bacterial colonies through our new framework for contour extraction.

3.3 Freeze-out hyper-surface extraction

Relativistic fluid dynamical models are widely used to describe heavy ion collisions \[31\]. Their advantage is that one can choose explicitly the equation of state of the nuclear matter and test its consequences on the reaction dynamics and the outcome. This makes
fluid dynamical models a very powerful tool to study possible phase transitions in heavy ion collisions such as the liquid-gas or the quark-gluon plasma phase transition \[32\]. The initial and final, freeze-out stages of the reaction are outside the domain of applicability of the fluid dynamical model. For example, fluid dynamics is not valid when the fluid becomes diffuse. When it is believed that the transition from a fluid to subatomic particles occurs (freeze-out), a popular approach for the calculation of multi-particle production probability distribution functions of hadrons (i.e., subatomic particles, which are composed of quarks and gluons) is represented by the integration of source or emission functions. The source functions are expressed in the case of relativistic hydrodynamic models in terms of hydrodynamic fields \[33\] across a freeze-out hyper-surface (FOHS).

A FOHS is considered to be a lower-dimensional interface representing the union of all subatomic particle production events. In 3+1D (i.e., 3D space plus 1D time) hydrodynamic simulations, a FOHS is a 3D volume which is embedded in the 4D space-time. Events of subatomic particle production take place at a space-time 4-vector, \(x_\mu (\mu = 0, 1, 2, 3)\), on the FOHS. The index \(\mu = 0\) typically refers to the temporal dimension and the indices \(\mu = 1, 2, 3\) refer to the three spatial dimensions, respectively. If one intends to calculate probability distribution functions for the production of hadrons, one has to know further quantities \[34, 35\]. These are, e.g., the 4-normal vector of the FOHS, \(d\sigma_\mu(x_\mu)\), the 4-velocity vector of the fluid at freeze-out, \(u_\mu(x_\mu)\), the temperature at freeze-out, \(T_f(x_\mu)\), etc. Very often, the FOHS is assumed to be an hyper-isosurface with respect to the temperature field \[33, 35\] of the relativistic fluid, i.e., \(T_f(x_\mu) = T_f = \text{const}\). The exploitation of spacial symmetries may allow the physicist to investigate certain aspects of relativistic fluid dynamics simulations in reduced dimensions, such as in a 1D radial space plus 1D time. Then the problem of FOHS extraction becomes identical to a 1D thermal isocontour extraction on a discretized 2D hydrodynamic simulation history. The 2D hydrodynamic simulation history is comprized of a discretized 1+1D space-time lattice (similar to 2D image data), on which field quantities such as temperature or fluid velocity components, etc., have been stored.

Fig. 15.a shows a 2D hydrodynamic simulation history of the discrete fluids temperature field, i.e., the temporal temperature evolution of a 1D relativistic fluid. In other words, Fig. 15.a is an image, \(T = T(t, r)\), where \(T\), \(t\), and \(r\) are the (continuous) fluid temperature, the (discretized) time and a (discretized) spacial dimension (e.g., radius, because a radial symmetry may apply), respectively. Darker pixels refer to lower fluid temperatures, whereas brighter ones refer to higher fluid temperatures. The origin of the space-time lattice (i.e., \(r = t = 0\)) is located in the center of the lower left image pixel. The FOHS is defined here as a thermal isocontour of value \(T_f\) and is constructed as follows. First, all lattice points (i.e., pixels) which have a temperature higher or equal to \(T_f\) are enclosed with a left-turning (i.e., pixel disconnecting) DICONEX contour. This is depicted in Fig. 15.b, which also shows the range vectors necessary for the following isocontour extraction. In the next step, the isocontour is constructed (\(\text{cf.}, \) Fig. 15.c). When the contour support points are relocated through linear interpolation, the corresponding field quantities such as fluid velocity field components, etc., are evaluated for the isocontour supporting points as well. Note, that points which sit directly on the boundary of the image data are not moved. Contour edges which have both supporting points with coor-
coordinates $r \leq 0$ and/or $t \leq 0$, are physically irrelevant (for reasons, which are not explained here). They have been removed in the final result shown in Fig. 15.d. In Fig. 15.d, we also show the corresponding 4-normal vectors of the FOHS at freeze-out, $d\sigma_\mu(x_\mu)$. Note, that in 1+1D these 4-vectors are just the normal vectors of the isocontour vectors \cite{36}. Furthermore, the 1+1D freeze-out events, $x_\mu = (t_f, r_f)$, are associated with the middle of each isocontour vector (cf., Fig. 15.d), and so are all corresponding field quantities (i.e., $d\sigma_\mu(x_\mu)$, $u_\mu(x_\mu)$, etc.). $t_f$ and $r_f$ denote the freeze-out times and freeze-out radii, respectively.

This subsection concludes the application section of the new framework for 1D contour extraction from discrete 2D data sets.

4 Summary

In summary, we have introduced a new framework for 1D contour extraction from discrete 2D data sets. Within this toolbox approach, we can generate up to five different types of contours. These are (i) the contours made up by the connected sets of contour vectors which initially separate pairs of pixels, (ii) DICONEX or dilated contours, (iii) boundary pixel tracing contours, (iv) isocontours, and (v) contours from shape skeletons, respectively. All of the contours can be computed rather fast and 100% robustly. In particular, the DICONEX contours resemble a class of perfect contours in the sense that they are always non-selfintersecting and non-degenerate, i.e., they always enclose an area larger than zero.

An important integral part of the contour extraction toolbox is a constrained Delaunay tessellation tool, which aids the gap closure and/or continuation of contour fragments such that closed contours can be obtained at all times. The Delaunay tessellations also support contour simplification, as well as extraction and potential pruning of shape skeletons (through the identification of the morphological roles which the individual triangles within the tessellations may play). The introduction of high-level shape constituents (e.g., torsos) allow for contour refinement through 2D shape manipulations.

Finally, we have demonstrated that a wide range of rather diverse applications can be addressed with this novel contour extraction framework.

5 Acknowledgements

This work has been supported by the Department of Energy under contract W-7405-ENG-36.
References

[1] S. Di Zenzo, L. Cinque, and S. Levialdi, “Run-Based Algorithms for Binary Image Analysis and Processing,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 18 (1996) 83 – 89.

[2] E. Bribiesca, “A new chain code,” Pattern Recognition, 32 (1999) 235 – 251.

[3] N. L. Jones, M. J. Kennard, A. K. Zundel, “Fast algorithm for generating sorted contour strings,” Computers and Geosciences, 26 (2000) 831 – 837.

[4] S. Kaygin, M. M. Bulut, “A new one-pass algorithm to detect region boundaries,” Pattern Recognition Letters, 22 (2001) 1169 – 1178.

[5] Y. B. Bai, X. W. Xu, “Object Boundary Encoding – a new vectorisation algorithm for engineering drawings,” Computers in Industry, 46 (2001) 65 – 74.

[6] M. Ren, J. Yang, H. Sun, “Tracing boundary contours in a binary image,” Image and Vision Computing, 20 (2002) 125 – 131.

[7] T. Pavlidis, Algorithms for Graphics and Image Processing, Computer Science Press, 1982, 142 – 148.

[8] W. K. Pratt, Digital Image Processing, John Wiley & Sons, 2001.

[9] J. R. Parker, Algorithms for Image Processing and Computer Vision, John Wiley & Sons, 1997.

[10] B. Jähne, Digital Image Processing, Springer, 1997.

[11] J. Canny, “A Computational Approach to Edge Detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8 (1986) 679 – 698.

[12] B. R. Schlei, L. Prasad, “A Parallel Algorithm for Dilated Contour Extraction from Bilevel Images,” Los Alamos Preprint LA-UR-00-309, Los Alamos National Laboratory, cs.CV/0001024, 2000.

[13] P. L. George and H. Borouchaki, Delaunay Triangulation and Meshing, Hermes, 1998.

[14] B. R. Schlei, “DICONEX - Dilated Contour Extraction Code, Version 1.0,” Los Alamos Computer Code LA-CC-00-30, Los Alamos National Laboratory.

[15] B. R. Schlei, L. Prasad and A. N. Skourikhine, “Geometric Morphology of Granular Materials,” Proceedings of SPIE 2000, 4117 (2000) 196 – 201.

[16] B. R. Schlei, L. Prasad and A. N. Skourikhine, “Geometric morphology of cellular solids,” Proceedings of SPIE 2001, 4476 (2001) 73 – 79.

[17] B. R. Schlei, “Region-Enclosing Contours from Edge Pixels,” Vision Geometry XI, Proceedings of SPIE’s 47th Annual Meeting, Seattle, WA, 4794 (2002) 63 – 70.
[18] L. Prasad, “Morphological Analysis of Shapes,” CNLS Newsletter, No. 139, July ‘97, LALP-97-010-139, Center for Nonlinear Studies, Los Alamos National Laboratory.

[19] L. Prasad, R. Rao, “A Geometric Transform for Shape Feature Extraction,” Proceedings of SPIE 2000, 4117 (2000) 222 – 233.

[20] J. J. Zou, H.-H. Chang, H. Yan, “Shape skeletonization by identifying discrete local symmetries,” Pattern Recognition, 34 (2001) 1895 – 1905.

[21] B. R. Schlei, “Counting Bacterial Colonies,” Theoretical Division - Self Assessment, Special Feature, a portion of LA-UR-02-1409, Los Alamos (2002) 113 – 114.

[22] L. Prasad, R. Rao, “Multi-scale Discretization of Shape Contours,” Proceedings of SPIE 2000, 4117 (2000) 202 – 209.

[23] L. J. Latecki, R. Lakämper, “Convexity rule for shape decomposition based on discrete contour evolution,” Comput. Vision Image Understanding, 73 (1999) 441 – 454.

[24] L. J. Latecki, R. Lakämper, “Application of planar shape comparison to object retrieval in image databases,” Pattern Recognition, 35 (2002) 15 – 29.

[25] M. C. Demirel, B. S. El-Dasher, B. L. Adams, A. D. Rollet, “Studies on the Accuracy of Electron Backscatter Diffraction Measurements,” Electron Backscatter Diffraction in Materials Science, Kluwer Academic-Plenum Publishers, New York, 2000, 65 - 74.

[26] A. Kuprat, D. George, G. Straub, M. C. Demirel, “Modeling microstructure in three dimensions with Grain3D and LaGrit,” Comp. Mat. Sci. 28 (2003) 199 – 208.

[27] courtesy, M. C. Demirel, Department of Engineering Science and Mechanics, Pennsylvania State University, PA.

[28] D. T. Carpenter, J. M. Rickman, and K. Barmak, “A methodology for automated quantitative microstructural analysis of transmission electron micrographs,” J. Appl. Phys., 84 (1998) 5843 – 5854.

[29] J. Wilson, T. Hunt, Molecular Biology of the Cell: A Problems Approach, Garland Science, 2002.

[30] courtesy, B. Kirkup, Osborne Memorial Laboratory, Yale University, New Haven, CT.

[31] R. B. Clare, D. Strottman, “Relativstic Hydrodynamics and Heavy Ion Collisions,” Phys. Rep. 141 (1986) 177 – 280.

[32] L. P. Csernai, Introduction to Relativistic Heavy Ion Collisions, John Wiley & Sons, 1994.
[33] F. Cooper, G. Frye, E. Schonberg, “Landau’s hydrodynamic model of particle production and electron-positron annihilation into hadrons,” Phys. Rev. D11 (1975) 192–213.

[34] J. Bolz, U. Ornik, R. M. Weiner, “Relativistic hydrodynamics of partially stopped baryonic matter,” Phys. Rev. C46 (1992) 2047–2056.

[35] J. Bolz, U. Ornik, M. Plümer, B. R. Schlei, R. M. Weiner, “Resonance decays and partial coherence in Bose-Einstein correlations,” Phys. Rev. D47 (1993) 3860–3870.

[36] E. M. Lifschitz, L. D. Landau, The Classical Theory of Fields: Volume 2, Butterworth-Heinemann, 1980.
6 Figure Captions

Figure 1: (a) Initial binary image; (b) contour vectors for a single pixel; (c) left-turning and (d) right-turning contour vectors for initial binary image; (e) dilated contours for left-turning contour vectors (“disconnect” mode); (f) dilated contours for right-turning contour vectors (“connect” mode).

Figure 2: (a) boundary pixel tracing contours (“disconnect” mode); (b) boundary pixel tracing contours (“connect” mode).

Figure 3: (a) Initial gray-level image; (b) as in (a), but with left-turning contour vectors; (c) as in (a), but with range vectors and dilated contour; (d) contour displacement for a pixel pair (see text); (e) gray-level image superimposed with range vectors and an isocontour of value 100; (f) as in (e), but with boundary pixel tracing contour.

Figure 4: (a) A shape with internal triangle decomposition and skeleton; (b) triangles with shape skeleton segments (the dashed lines indicate segments of a shape contour); (c) geometric pruning for skeletons (see text); (d) as in (a), but with a pruned skeleton.

Figure 5: (a) A pixel frame superimposed with a shape skeleton; (b) shape skeleton vectors and frame vectors; (c) as in (b), but with additional Delaunay triangular mesh; (d) as in (c), but with suggested gap closure lines (dotted); (e) as in (c), but with gap closing vector pairs (gray) of shortest length; (f) as in (e), but with vector pairs (gray), which attempt to conserve directions.

Figure 6: (a) Initial gray-level image; (b) three isocontours for the dark spots in image (a); (c) shape skeletons, which are enclosed by the contours in (b); (d) a limb and a torso (see text); (e) torso-splitting vector pairs (black); (f) six final shape-enclosing contours.

Figure 7: (a) – (k): Processing steps for point removal in contour segments (see text).

Figure 8: Backscattered contrast image [27] of a thin Aluminum film.

Figure 9: Bi-level image [27] with 25,518 grain edge pixels (black). Image dimensions: 442 x 441 pixels = 194,922 pixels.

Figure 10: Processing a sub-region of the binary image Fig. 9: (a) dilated contours enclose the gray pixels; (b) shape skeleton and added outer frame; (c) CDT for the final closed contours (the closed gaps are drawn with larger line width); (d) gray pixels superimposed with down-sampled shape-enclosing contours and a CDT grid.

Figure 11: Shape-enclosing contours and a CDT grid for the binary image Fig. 9.

Figure 12: Gray-level image [30] of the interior of a Petri dish with E. coli colonies.
Figure 13: Shape-enclosing contours for the E. coli colonies in Fig. 12. before contour refinement (see text).

Figure 14: Shape-enclosing contours for the E. coli colonies in Fig. 12. after contour refinement (see text).

Figure 15: (a) A gray-level image representing the temporal temperature evolution of a 1D relativistic fluid (see text); (b) a dilated contour encloses pixels with temperatures \( T \geq T_f \) and corresponding range vectors; (c) as in (b), but after contour displacement with respect to the fluid temperatures; (d) final FOHS with normal vectors \( d\sigma_\mu(x_\mu) \).
This figure "figure01.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure02.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure03.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure04.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure05.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure06.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure07.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure08.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure09.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure10.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure11.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure12.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure13.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure14.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1
This figure "figure15.gif" is available in "gif" format from:

http://arXiv.org/ps/cs/0405029v1