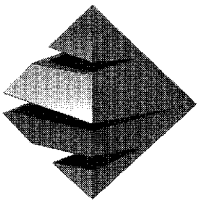
Pattern Analysis & Applic. (1998)1:2–17



Digital Boundary Tracking

G. T. Herman[1] and D. F. Robinson[2]

[1]*Department of Radiology, University of Pennsylvania, Philadelphia, PA, USA;* [2]*Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand*

Abstract: In many three-dimensional imaging applications, the three-dimensional space is represented by an array of cubical volume elements (voxels) and a subset of the voxels is specified by some property. Objects in the scene are then recognised by being 'components' of the specified set and individual boundaries are recognised as sets of voxel faces separating objects from 'components' in the complement of the specified set. This paper deals with the problem of algorithmic tracking of such a boundary specified by one of the voxel faces lying in it. The paper is expository in that all ideas are carefully motivated and introduced. Its original contribution is the investigation of the question of whether the use of a queue (of loose ends in the tracking process which are to be picked up again to complete the tracking) is necessary for an algorithmic tracker of boundaries in three-dimensional space. Such a queue is not needed for two-dimensional boundary tracking, but published three-dimensional boundary trackers all make use of such a thing. We prove that this is not accidental: under some mild assumptions, a boundary tracker without a queue will fail its task on some three-dimensional boundaries.

Keywords: Boundary tracking; Digital geometry; Digital spaces, Digital Topology; Voxels

1. MOTIVATION

Many imaging devices will produce estimated values of a physical quantity at certain points (referred to as *grid points*) in three-dimensional space. For example, a CT (Computerized Tomography) scanner estimates the X-ray attenuation coefficient inside a human body at points of a three-dimensional rectangular grid. When displaying the results of such an estimation, we usually use a sequence of two-dimensional images. An example is given in Fig. 1, in which bone appears light, softer tissues appear grey, and the air appears dark.

In Fig. 1 a cursor is placed at the right edge of a large pieces of bone in the upper part of the image. Since this CT scan was taken of a trauma victim, an important question to ask is: are all other bones which are normally 'connected' to this main piece of bone still connected for this patient? For example, the biggish piece of bone just to the right and below the cursor (it is the mandible) is not connected to any other bone in this slice, but it should be connected to the rest of the skull (at our resolution) somewhere in the whole three-dimensional array. Some of what we are doing will be concerned with the analysis of connectivity properties in three-dimensional images.

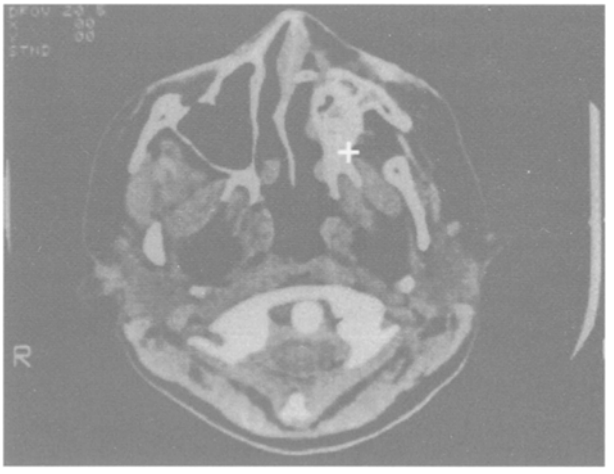


Fig. 1. A slice of a CT scan of the head of a trauma victim.

We turn aside for a moment to introduce some notation. We use R to denote the set of real numbers and R^N to denote the set of N-dimensional (row) vectors, all of whose components are real numbers. If

Received: 29 July 1997
Received in revised form: 5 January 1998
Accepted: 28 January 1998

$v \in R^N$, v_n denotes the nth component of v, for $1 \leq n \leq N$. The norm of a vector v is denoted by $\|v\|$. We use the algebraic notion of a vector in R^N and the geometric notion of a point in an N-dimensional Euclidean space quite interchangeably, and actually refer to R^N as the N-dimensional (Euclidean) space. We use Z to denote the set of all integers, and Z^N to denote the set of N-dimensional (row) vectors all of whose components are integers.

Let G be any set of points (a *grid*) in R^N. The *Voronoi neighbourhood* in G of any element g of G is defined as:

Definition 1

$$N_G(g) = \{v \in R^N \mid \text{for all } h \in G,\ \|v-g\| \leq \|v-h\|\}.$$

In other words, the Voronoi neighbourhood of g consists of all those vectors which are not nearer to any other point of G than they are to g. For example, the Voronoi neighbourhood of $(c_1, c_2) \in Z^2$ in Z^2 is $\{(v_1, v_2) \in R^2 | \max(|v_1-c_1|, |v_2-c_2|) \leq \frac{1}{2}\}$. This is a closed square with side-length 1 centred at the point (c_1, c_2). When perceived as a set of points in R^2, Z^2 is referred to as the *square grid*. The Voronoi neighbourhoods in a grid in R^2 are referred to as *pixels*.

We now return to the way in which the two-dimensional images of Fig. 1 are produced. Let us assume that the X-ray attenuation coefficients have been estimated at grid points with integer coordinates (c_1,c_2,c_3), $1 \leq c_1 \leq I$, $1 \leq c_2 \leq J$, $1 \leq c_3 \leq K$, and that Fig. 1 is the kth of these K slices. The region over which the images are defined is the union of the pixels associated with those $(c_1,c_2) \in Z^2$ for which $1 \leq c_1 \leq I$ and $1 \leq c_2 \leq J$. The grey value assigned to the pixel associated with one of these (c_1,c_2)s is proportional to the estimated X-ray attenuation coefficient at (c_1,c_2,k).

The overall aim of the type of work on which we are reporting is the display and analysis of selected objects based on the estimated values of the physical quantity at grid points in three-dimensional space. Examples for bones in the head are shown in Fig. 2. On the top we show a computer graphic display of the boundary of the connected piece of bone which is indicated by the cursor in Fig. 1. (The cursor location in the top image of Fig. 2 corresponds to the cursor location in Fig. 1.) We now see that the mandible is disconnected from this bone in the whole three-dimensional data set. The boundary of the mandible can be tracked separately, and then the two boundaries can be displayed together in their correct relative locations, as is shown in the bottom image of Fig. 2.

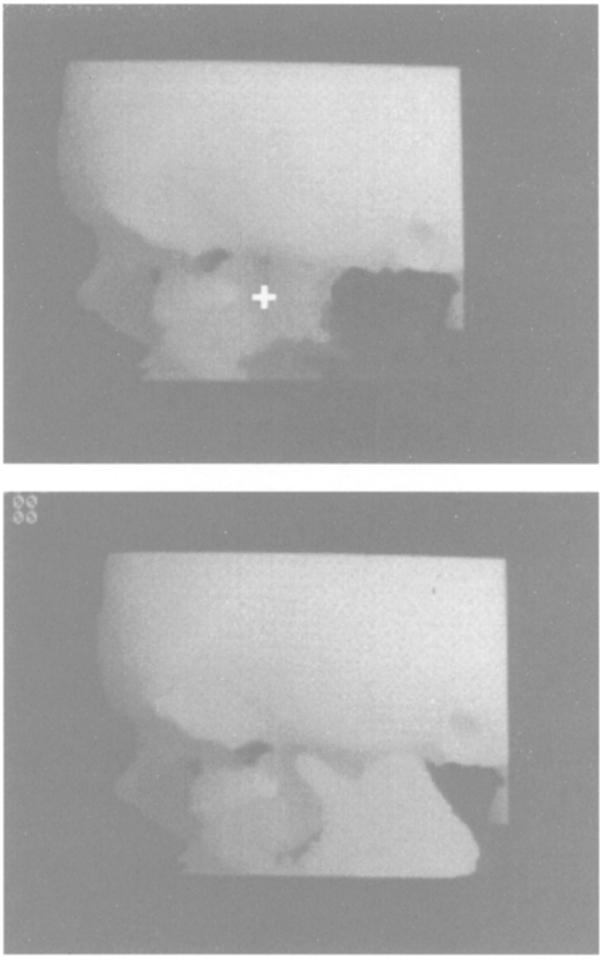


Fig. 2. Computer graphic displays of detected boundaries of bones in the head, based on the data set which contains the slice shown in Fig. 1.

2. A METHODOLOGY FOR EXTRACTING OBJECT BOUNDARIES

We now discuss a particular methodology for extracting boundaries of objects based on values assigned to points of the grid Z^3 in R^3, the so-called *cubic grid*. The Voroni neighbourhoods associated with a grid in R^3 are referred to as *voxels*. The voxel associated with the element (c_1,c_2,c_3) of Z^3 is the closed unit cube: $\{v \in R^3 | \max(|v_1 - c_1|, |v_1 - c_2|, |v_3 - c_3|) \leq \frac{1}{2}\}$. The tessellation of R^3 into voxels of a cubic grid is sometimes referred to as a *cuberille*. In Fig. 3, we illustrate a grid point g of Z^3 together with all other grid points which lie on the $2 \times 2 \times 2$ cube whose centre is g, as well as the Voronoi neighbourhood of g.

Now suppose that we have some methodology to determine which grid points belong to what kind of material. For example, in CT bone attenuates X-rays more than any other type of tissue in the human body, and so we may say that if the value assigned to

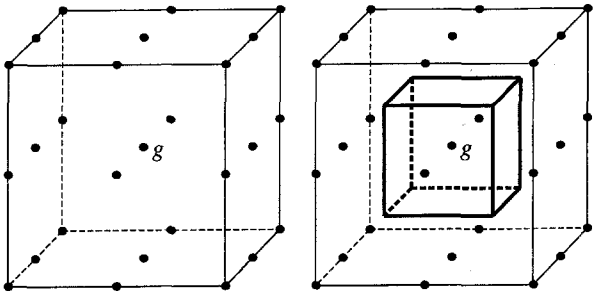


Fig. 3. On the left we show a point g of the cubic grid Z^3 together with the 26 grid points which lie on the $2 \times 2 \times 2$ cube whose centre is g. On the right we show (using heavy lines) the Voronoi neighbourhood of g (a voxel).

a grid point is above a certain value, then that grid point is in bone. So, as an approximation, the part of space that is occupied by bone may be considered to be the union of the voxels associated with those pixels for which the estimated X-ray attenuation coefficient is above the threshold for bone. This collection of voxels may not necessarily form a connected subset of three-dimensional space: some pieces of bone may be disconnected from the rest and some voxels may be included because noise in the estimation process caused us to identify a grid point as being in bone, when in fact it is not. Let us say that the 'object' whose boundary we wish to display is a 'component' of the set of points occupied by the 'bone' voxels.

Even this specification is not precise enough for describing the intuitive notion of a 'boundary' of an object. This is because an object of the kind we described in the previous paragraph may have cavities inside it (just look at Fig. 1; the inside of bones contains a lot less X-ray attenuating tissue and may well be identified as a result of thresholding as 'not bone') and so it may have multiple boundaries (one exterior one and possibly many interior ones). Let us say that our task is to identify exactly one of these boundaries.

How do we specify which one? One way is to point at a boundary face, that is at a face which separates a bone voxel from a not-bone voxel, and say that we wish to display that boundary of the object which contains that boundary face. (At this point, it is not even clear that this specification is legitimate. Is the 'boundary containing a boundary face' a well defined concept? We will show that it is; we will set up an environment in which for any boundary face there will be one and only one boundary containing it.)

An intuitively useful picture is the following. We consider the cuberille, the tessellation of three-dimensional space into cubic voxels. A finite number of these voxels are occupied by sugar cubes of just the right size. We point at an uncovered face of one of these sugar cubes (i.e. a face such that the voxel on the other side of it is not occupied by a sugar cube), and ask that there be delivered to us the boundary which contains that face. The problem is to design an algorithm which is guaranteed to do this for us for all possible arrangements of the sugar cubes.

3. FLIES IN FLATLAND

Before attacking the three-dimensional problem we consider its simpler two-dimensional analog. Following Abbott [1], we refer to the plane as Flatland and to the computing device which we hope will deliver us the required boundary surfaces as a Flat Fly. As an example, consider a configuration of two-dimensional sugar cubes in Flatland shown in Fig. 4 with a Flat Fly on one of the flat faces (i.e. on an edge) of one of these flat sugar cubes.

3.1. Algorithm for Flat Flies

1. Dirty the face on which you are standing.
2. Crawl onto the face which meets the one on which you are standing at the vertex in front of you.
3. See if it is dirty.
   (a) If it is, fly away.

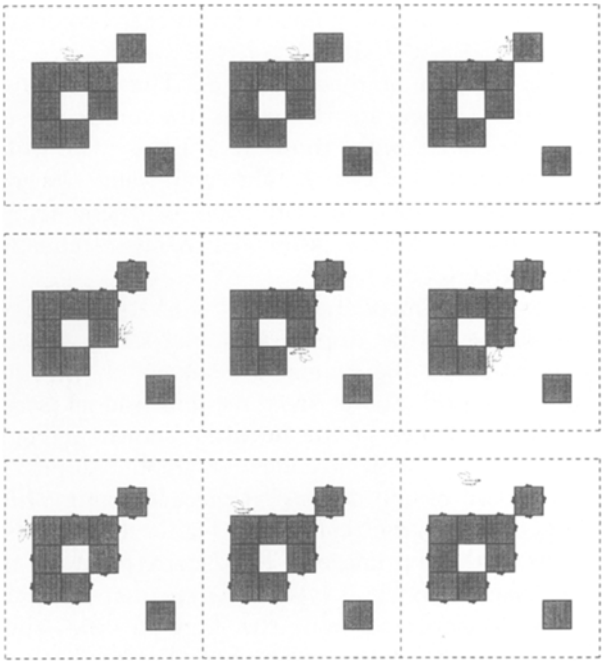


Fig. 4. Illustration of boundary tracking in two dimensions by the Algorithm for Flat Flies when a Flat Fly is initially placed on a flat face (i.e. on an uncovered edge of an occupied pixel): the first row is the beginning, the second row is the middle, and the third row is the end of the execution of the algorithm.

(b) If it is not, start again at Instruction (1).

This algorithm actually does something that can be made precise. To achieve that precision, we need to introduce some mathematical terminology.

4. COMPONENTS DETERMINED BY BINARY RELATIONS

Although our paper is self-contained, the interested reader may wish to consult elsewhere [2–4] for background and related material to the mathematics that we introduce in this and in the following two sections.

Let M be a set, and ρ be a binary relation on M. If $(c,d) \in \rho$, then we say that c is *ρ-adjacent to d* and that d is *ρ-adjacent from c* and, in case ρ is *symmetric* (meaning that $(c,d) \in \rho$ if, and only if, $(d,c) \in \rho$), that c and d are *ρ-adjacent*. We use 'adjacency' to refer to a symmetric binary relation.

In the case when $M = \mathbb{Z}^N$, we will be repeatedly dealing with two adjacencies, α_N and ω_N, defined as follows:

Definition 2

$(c,d) \in \alpha_N \Leftrightarrow (c \neq d$ and, for $1 \leq n \leq N$, $|c_n - d_n| \leq 1)$.

Definition 3

$$(c,d) \in \omega_N \Leftrightarrow \sum_{n=1}^{N} |c_n - d_n| = 1$$

For $N = 2$, $(c,d) \in \alpha_2$ means that the associated pixels have a non-empty intersection, but are not identical, and $(c,d) \in \omega_2$ means that the associated pixels have exactly one edge in common. These are often used notions and go under a number of commonly used names: ω_2 is referred to both as the edge-adjacency and as the 4-adjacency (since by it, any pixel is adjacent to four others), while α_2 is referred to both as the edge-or-vertex-adjacency and as the 8-adjacency (since by it, any pixel is adjacent to eight others).

For any c and d in a subset A of M, we call a sequence $\langle c^{(0)},\cdots,c^{(K)}\rangle$ of elements of A and *ρ-path in A connecting c to d*, if $c^{(0)} = c$, $c^{(K)} = d$ and, for $1 \leq k \leq K$, $c^{(k-1)}$ is ρ-adjacent to $c^{(k)}$. (In graph theory [5] this would be called a 'walk', since $c^{(i)} = c^{(j)}$ for $i \neq j$ is not forbidden; however, the term 'path' is standard in digital geometry [2].) A nonempty subset A of M is said to be *ρ-connected* if, for any c and d in A, d is ρ-connected in A to c.

If ρ is an adjacency, then ρ-connectedness in A is an equivalence relation. Hence, it partitions A into *ρ-components* (i.e. into nonempty ρ-connected subsets which are not proper subsets of any other ρ-connected subset of A). Referring to Fig. 4, we see that there are three ω_2-components of the set of grid points whose pixels are painted grey: one has seven elements and the other two have one element each. However, there are only two α_2-components.

We denote the *complement* $M - A$ of A in M by $\overline{A}$. In Fig. 4, $\overline{A}$ (the set of grid points whose pixels are not painted grey) is α_2-connected subset (and hence it has only one α_2-component). However, it is not ω_2-connected.

In the special case when $A = M$, we use the phrases *ρ-path* and *ρ-connected* instead of ρ-path in A and ρ-connected in A.

5. WHAT DOES A FLAT FLY DO?

We first specify the outcome of the algorithm for Flat Flies in the specific case identified in Fig. 4. The Flat Fly is initially put on the face which separates the grid point g from the grid point h. After a while it flies away, leaving behind a bunch of faces that have been dirtied. This set of faces can be described as consisting of those faces which are between a grid point painted grey which is α_2-connected in the set of grey grid points to g and a grid point not painted grey which is ω_2-connected in the set of not grey grid points to h.

This is, in fact, representative of the general behaviour of the Flat Fly. Suppose that in the square grid $\mathbb{Z}^2$ the pixels of a finite number of grid points are filled with flat sugar cubes, and that a Flat Fly is put on top of these sugar cubes into a pixel which is not occupied by a sugar cube (see Fig. 4). Let g be the name of the grid point of the sugar cube on top of which the Flat Fly is placed into the pixel of a grid point named h (see Fig. 4).

Claim 5.1 After a finite number of loops through instructions (1), (2), and (3) of the Algorithm for Flat Flies, the Flat Fly will fly away. At that time, the set of dirty faces is exactly the set of those faces which are between a pixel associated with an element of the α_2-component of the set of grid points with sugar cubes which contains g and a pixel associated with an element of the ω_2-component of the set of grid points without sugar cubes which contains h.

This claim is Corollary 1 of Artzy et al [6]. One reason why it is important is that the set of faces dirtied by the Flat Fly has some desirable properties.

We first observe that the face onto which the fly

is initially placed can be identified by the pair (g,h) of grid points on either side of it. Note that this is an ordered pair, interpreted as "the fly is put into h, with its feet touching g". If we were to draw the Flat Fly on (h,g), then its feet would be touching the same edge, but it would be hanging upside down. Similarly, every ordered pair of ω_2-adjacent grid points can be thought of as such an edge with an orientation across it from the first grid point in the pair towards the second and, conversely, all such oriented edges can be described by a unique element of ω_2, as defined by definition (3). Thus, we make the fundamental observation that ω_2 has a dual purpose: it is an adjacency on Z^2 and its elements can be used as unique identifiers of elements of the set of all oriented edges between pixels of the square grid. We note immediately that, for arbitrary positive integer N, ω_N is an adjacency on the grid Z^N, and its elements can be used as unique identifiers of elements of the set of all oriented (hyper-) faces between points of the grid. We now introduce two important concepts for the grid Z^N.

For any subsets O and Q of Z^N, we define the *boundary* between them by:

Definition 4

$$\partial(O,Q) = \{(c,d) \mid c \in O,\ d \in Q,\ (c,d) \in \omega_N\}.$$

Let A consist of the nonempty finite set of grid points which are filled with sugar cubes, according to the supposition preceding Claim 5.1, and let the Flat Fly be placed on (g,h), where $g \in A$ and $h \in \overline{A}$. Let O be the α_2-component of A containing g and Q be the ω_2-component of $\overline{A}$ containing h. Then Claim 5.1 says that the set of faces dirtied by the Flat Fly is exactly $\partial(O,Q)$.

Let $\langle c^{(0)},\cdots,c^{(K)}\rangle$ be an ω_N-path and S be any subset of ω_N. We say that $\langle c^{(0)},\cdots,c^{(K)}\rangle$ *crosses* S, if there is a k, $1 \le k \le K$, such that either $(c^{(k-1)},c^{(k)}) \in S$ or $(c^{(k)},c^{(k-1)}) \in S$.

Claim 5.2 Let A be any nonempty proper subset of Z^2. Let O be an α_2-component of A and Q be an ω_2-component of $\overline{A}$, such that $\partial(O,Q)$ is not empty. Then there exist two uniquely defined subsets I and E of Z^2, which have the following properties:

(i) $O \subseteq I$ and $Q \subseteq E$.

(ii) $\partial(O,Q) = \partial(I,E)$.

(iii) $I \cup E = Z^2$ and $I \cap E = \varnothing$. ($\varnothing$ denotes the empty set).

(iv) I is α_2-connected and E is ω_2-connected.

(v) Every ω_2-path from an element of I to an element of E crosses $\partial(O,Q)$.

This claim is special case for Flatland of the much more general results of Herman [2]. Since, according to Claim 5.1, the output of the Algorithm for Fat Flies is a boundary $\partial(O,Q)$ satisfying the premises of Claim 5.2, the latter claim says that there are two sets of grid points I and E, of which we will think intuitively as the interior and the exterior, which have certain properties, reminiscent of the famous Jordan Curve Theorem [7]. According to that theorem, the set of all points in the plane which are not on a simple closed curve can be partioned into two subsets, one may be called the interior and the other the exterior. Both of these are connected sets in the sense that we can get from any point of the interior to any other point by drawing continuously a curve between them which never leaves the interior (and similarly for the exterior), but one cannot draw continuously a curve from a point in the interior to a point in the exterior which does not contain at least one point of the simple closed curve, which is in fact the boundary between the interior and the exterior.

The boundary that is the output of the algorithm for Flat Flies shares important properties with simple closed curves in R^2: its interior and exterior partition the whole space (iii), both are conntected in some sense (iv), but one cannot get from the interior to the exterior without crossing the boundary (v) which is, in fact, the boundary between the interior and the exterior (ii). (Mathematically speaking there is also a difference: a simple closed curve is a subset of the plane and the Jordan Curve Theorem says that the complement of the curve, rather than the whole plane, has precisely two components. Since our boundary is a subset of ω_2, rather than of Z^2, it is possible to demand that the interior and exterior partition the whole of Z^2; if anything a more attractive-sounding aim than that of the classical theorem.)

It is important to recognise that in spite of this formal similarity to the Jordan Curve theorem, Claim 5.2 is not a result about the continuous plane R^2, but a thoroughly digital theorem. Notions of 'connectedness', 'path' and 'crosses' have all been defined for the square grid Z^2 and while they are analogous to corresponding notions in R^2, the analogy breaks down sometimes. It is the difficulty of applying the continuous notions directly to the digital environment to prove claims regarding algorithms operating in discrete domains which motivates us to develop a geometry directly for digital spaces.

For example, it is tempting to think of the boundary output by the Algorithm for Flat Flies as the point set in R^2 which is the union of all the dirtied faces. However, in spite of the discrete Jordan curve properties of this boundary (Claim 5.2), the Jordan Curve Theorem is not applicable to the corresponding point

set in R^2, since it does not form a simple closed curve. (It 'touches itself' at one vertex and so it is not a simple curve.)

Having said this, we must admit that the argument is only half convincing. It is all very well that we can give a self-contained theory in digital spaces with nice theorems which resemble those of famous results of continuous mathematics; nevertheless, an interpretation in the underlying continuous space is unavoidably needed for some applications. For example, no physician would think of the boundaries shown in Fig. 2 as collections of ordered pairs of voxels; rather, they would be perceived as continuous biological surfaces. Hence, it is not really desirable to have boundaries which touch themselves in the fashion of the boundary consisting of the dirtied faces at the end of Fig. 4.

While we are pondering this, we may also ask the question: Is it reasonable that two pixels which only share a vertex are considered connected to each other? This came about quite naturally in the way the Flat Fly crawled about on the sugar cubes in Fig. 4, but – looking at the result – the connection is rather suspect. Would it not be better to use only the edge-adjacency ω_2 to define connectivity and to avoid such flimsy looking connections?

Once we raise this objection on physical grounds, we note that the use of α_2 is also not elegant from a mathematical point of view. Why do we use different types of components for grid points with sugar cubes and for those without in Claim 5.1? Why do we use different types of components for A and for $\overline{A}$ in Claim 5.2? Since in point (v) of Claim 5.2 only ω_2 is used, it would be much nicer to replace α_2 by ω_2 everywhere in Claim 5.2. Unfortunately, the resulting claim would not be true; as is illustrated, for instance, in Herman [2]. Thus, we need ω_2 (it is used to describe the edges which make up the boundary), and we need a broader adjacency such as α_2 for the validity of the desirable Claim 5.2.

We have thus seen that in two dimensions, although boundary tracking is computationally easy, the conceptual underpinning is somewhat complex. We cannot expect the three-dimensional situation to be easier in either aspect.

6. ON THE CUBERILLE

In three-dimensional space we would like to have an Algorithm for Fat Flies (in Fig. 5 we show one placed on a sugar cube), which would have behaviour resembling that of Flat Flies as expressed in Claims 5.1 and 5.2.

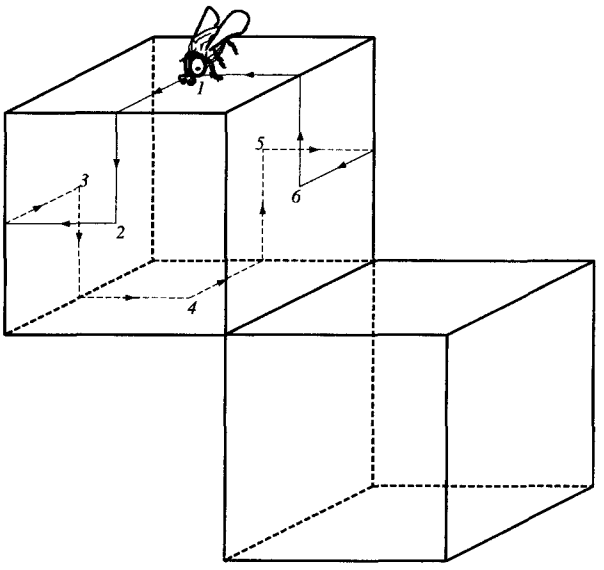


Fig. 5. The route of a Model 1 Fat Fly placed on one of a pair of sugar cubes.

We first need to decide on the appropriate analog of α_2 for the case of a cuberille. One candidate is α_3, as defined by Definition 2. The geometrical interpretation of α_3 is: two points of the cubic grid Z^3 are α_3-adjacent if, and only if, the corresponding voxels have either exactly one face, or one edge, or one vertex in common (see Fig. 3). The question is: for the purpose of the three-dimensional analogue of Claim 5.1, do we want to consider two sugar cubes to be adjacent if they meet only at a vertex? Even in two dimensions such a connectivity was undesirable, but was forced upon us in order to have Claim 5.2. In three dimensions we need ω_3 to define the boundary surface, but also something wider to obtain an analogue of Claim 5.2. The difference is that now we have an alternative, another analogue of α_2 in which two grid points are adjacent when the corresponding voxels have either exactly one face or exactly one edge in common. Formally, for any positive integer N, we define the adjacency δ_N on Z^N as follows. For any c and d in Z^N,

Definition 5

$$(c,d) \in \delta_N \Leftrightarrow \left((c,d) \in \alpha_N \text{ and } \sum_{n=1}^{N} |c_n - d_n| \leq 2 \right)$$

It is clear from Definition 2 that $\delta_2 = \alpha_2$ and so δ_3 and α_3 are equally legitimate three-dimensional analogues of α_2. However, δ_3 is intuitively preferable over α_3, in as much as it does not allow adjacency by vertices only; see Fig. 6. (For obvious reasons, ω_3 has been referred to both as face-adjacency and as 6-adjacency, δ_3 has been referred to both as face-or-edge-adjacency and as 18-adjacency – since by them

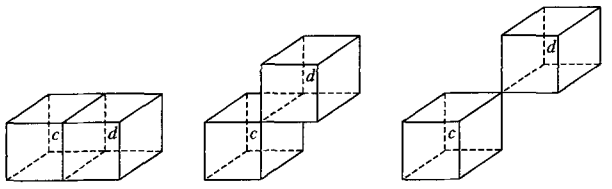


Fig. 6. The grid points c and d on the left are α_3-, δ_3- and ω_3-adjacent; those in the middle are α_3- and δ_3-adjacent, but not ω_3-adjacent; those on the right are α_3-adjacent, but are not δ_3- or ω_3-adjacent.

each voxel is adjacent to six and eighteen other voxels, respectively – and α_3 has been referred to both as face-or-edge-or-vertex-adjacency and as 26-adjacency.) In view of this, our aim is as follows.

Aim 6.1 Design an algorithm for Fat Flies which will have the following property. Suppose that in the cubic grid Z^3 the voxels of a finite number of grid points are filled with sugar cubes, and that a Fat Fly is put on top of one of these sugar cubes into a voxel which is not ocupied by a sugar cube. Let g be the name of the grid point of the sugar cube on top of which the Fat Fly is placed into the voxel of a grid point named h. Let O be the δ_3-component of the set of grid points with sugar cubes which contains g, and Q be the ω_3-component of the set of grid points without sugar cubes which contains h. After a finite number of steps, the Fat Fly should fly away and, at that time, the set of faces that have been dirtied should be exactly $\partial(O,Q)$.

The output $\partial(O,Q)$ of such an algorithm is an intuitively desirable boundary, due to the following consequence of the general results of Herman [2].

Claim 6.2 Let A be any nonempty proper subset of Z^3. Let O be a δ_3-component of A and Q be an ω_3-component of $\overline{A}$, such that $\partial(O,Q)$ is not empty. Then there exist two uniquely defined subsets I and E of Z^3, which have the following properties:

(i) $O \subseteq I$ and $Q \subseteq E$.
(ii) $\partial(O,Q) = \partial(I,E)$.
(iii) $I \cup E = Z^3$ and $I \cap E = \varnothing$.
(iv) I is δ_3-connected and E is ω_3-connected.
(v) Every ω_3-path from an element of I to an element of E crosses $\partial(O,Q)$.

We see that this claim is a digital three-dimensional version of the Jordan Curve Theorem. Apart from being a mathematically pleasing fact, this observation is of the utmost importance. In practical applications, we are interested in finding boundaries of objects for two reasons. One is to create computer graphic displays of them, such as those shown in Fig. 2, and the other is to analyse certain properties of the objects, such as their volumes. From the graphical display point of view, property (v) can be used to prove that when we display the boundary $\partial(O,Q)$ as it appears from any exterior point, then the surface that is being displayed will hide all the interior points.

For the purpose of analysis, Claim 6.2 works as follows. Suppose that a device has given us estimates of a physical quantity at points of the cubic grid Z^3. Suppose further that we can identify from such data that set, A, of grid points which are in cardiac muscle. Assuming that A itself is δ_3-connected, and that the set of grid points in any one of the chambers of the heart is an ω_3-component of $\overline{A}$, we can estimate the volume of the left ventricle, say, by selecting ω_3-adjacent voxels g and h, so that g is the cardiac muscle and h is in the left ventricle, defining O and Q as in Aim 6.1, and estimating the volume of the left ventrical of the heart as the combined volume of the voxels associated with the grid points of the exterior E, whose existence is guaranteed by Claim 6.2. Alternatively, if we wish to find the volume of the whole heart, then assuming that the set of grid points outside the heart form an ω_3-component of $\overline{A}$, we can select ω_3-adjacent voxels g and h, so that g is in the cardiac muscle and h is outside the heart, defining O and Q as in Aim 6.1, and estimating the volume of the whole heart as the combined volume of the voxels associated with the grid points of the interior I, whose existence is guaranteed by Claim 6.2. This also indicates how the achievement of Aim 6.1 would allow us to detect individually the boundaries of the cardiac muscle.

7. ALGORITHMS FAT FLIES

First, we observe that just by simply changing the word 'vertex' in the Algorithm for Flat Flies into the word 'edge' we get a set of instructions which are meaningful in the three-dimensional case. However, the resulting algorithm will not fulfil Aim 6.1. Even if there is only one grid point with a sugar cube (see Fig. 5), this simply modified version will fail. The Fat Fly will keep moving forward and, having dirtied only four faces of the cube, it will get back to the original face and will fly away. Clearly, $\partial(O,Q)$ in this case consists of all six faces of the sugar cube, and the proposed algorithm resulted in only four of them being dirtied. So consider instead the following:

7.1 Algorithm for Fat Flies Model 1

1. Dirty the face on which you are standing.

2. Crawl onto the face which meets the one on which you are standing at the edge in front of you.
3. Dirty the face on which you are standing and turn right.
4. Crawl onto the face which meets the one on which you are standing at the edge in front of you.
5. See if it is dirty
   (a) If it is, fly away.
   (b) If it is not, turn left and start again at Instruction (1).

It turns out that this algorithm behaves as required for the case of a single sugar cube. It can be seen in Fig. 5 that the Fat Fly under the control of the Model 1 algorithm will visit (and dirty) every face of a single sugar cube.

However, the route of the Fat Fly shown in Fig. 5 indicates that the Model 1 algorithm cannot behave as is needed for Aim 6.1: there are edges which are not traversed by this route. If we attach a second sugar cube to one of these edges, the grid points of the two sugar cubes are δ_3-adjacent, and so $\partial(O,Q)$ consists of the twelve faces of the pair of sugar cubes. However, the route of the Fat Fly does not change, it still only dirties six faces; see Fig. 5.

In fact, we see that this indicates a general principle. If the proposed algorithm is such that the edge towards which the Fat Fly starts crawling is not influenced by the absence or presence of nearby sugar cubes, then in order for the algorithm to work, it must traverse each edge when started on the face of a single sugar cube. (Otherwise, we can attach a second sugar cube to the untraversed edge, just as indicated in Fig. 5.) However, if each edge is traversed once during a route on a single sugar cube, then (since there are six faces and twelve edges) each face must be visited twice; so the simple mechanism of dirtying the faces the first time they are visited and flying away the second time one is visited will not do. This by itself is not a problem; we can have the fly "mark" a face the first time it visits it, "dirty" it when it finds that it has already been marked, and fly away when it finds that the face has already been dirtied. The interesting question is: what should be the rule for deciding the edge towards which the Fat Fly starts crawling when it is on a particular face?

We now establish a further principle. Having accepted that the route of the Fat Fly must traverse each edge, let us aim for the shortest type of route on a single sugar cube which satisfies this condition. If we draw the route of such a fly on the sugar cube in the manner as was done for Fig. 5, then we see that on each face there have to be four arrows (one for each edge of the face), two of which point from

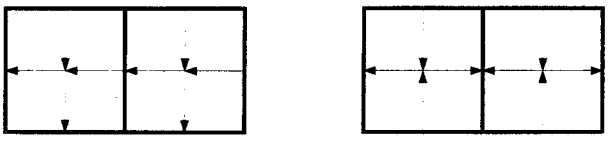


Fig. 7. Pairs of faces with two types of arrow orientation; only the one on the left is acceptable.

the centre of the face towards an edge and two which point from the other two edges towards the centre of the face. Futher, there has to be some consistency between these arrows: if an edge has an arrow pointing to it on one face, on the face which meets this edge the arrow has to be pointing away from the edge. Finally, we consider the desirable simplicity of an Algorithm for Fat Flies. We impose the condition that the way in which the arrows point on the face on which the fly is standing depends only upon the orientation of the face (conventionally, North, South, East, West, Up, Down). It is as though all the sugar cubes had identical sets of arrows printed on them.

The arrangement of four arrows on a single sugar cube face (two to the centre and two out) can take only two forms. Either the two inward arrows are next to each other (making an angle of 90°), in which case it is possible for the fly to proceed straight ahead, or the two inward arrows are opposite (consequently, so are the outward arrows). But, the second arrangement will not do for us. Suppose the object being identified consists of two cubes side by side and the uppermost faces, which must have an identical arrangement of arrows, are of the second kind. Then at the edge where the upper faces meet the arrows will be pointing in opposite directions, as in Fig. 7.

This implies that the route of the Fat Fly for a single sugar cube must be of the general form indicated by Fig. 8. We say 'of the general form', since the considerations above still leave us with some freedom;

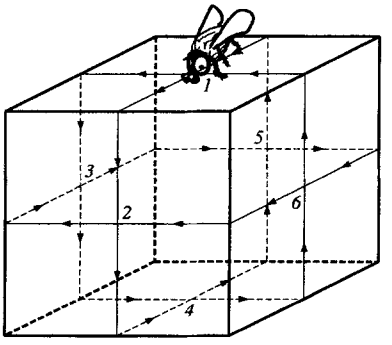


Fig. 8. Legitimate routes of the Fat Fly on a single sugar cube follow the arrows and cross every edge once. The numbers refer to the orientations of the faces and not to the order in which they are visited.

the horizontal cycle of arrows around the vertical faces could be reversed without violating the conditions described above and the same is true for the two vertical cycles of arrows, leaving us with eight possible legitimate arrangements. However, these are symmetric versions of each other and, for what we are going to do, there is no harm in restricting our attention to the arrangement shown in Fig. 8. (We make this more precise in Section 8.) A legitimate route for the Fat Fly is defined as one which follows the arrows and traverses each edge once.

Let us suppose in designing the Algorithm for Fat Flies that the fly is aware of the orientation of the face (one of six possible orientations) on which it finds itself. If it is now to behave so that it will follow one of the legitimate routes of Fig. 8, then it has only two choices: go straight on or turn 90° in the legitimate direction. Note that the knowledge that the Fat Fly is visiting the face for the first or for the second time it is not relevant to this decision: to achieve the required behaviour, if it goes straight the first time, it must also go straight the second time and if it turns the first time, it must also turn the second time. (Otherwise, some edge would be traversed twice.) Thus the behaviour of the Fat Fly can be controlled by a function g of the following kind. The domain of the function is $\{1,2,3,4,5,6\}$, indicating one of the six possible orientations of a face of a sugar cube. (For the sake of being specific, consider Fig. 5. Let 1 denote the orientation of the face with the Fat Fly on it and let the orientations of the other faces be numbered by the order in which they are visited by the Fat Fly as it executes the Model 1 algorithm.) The value of the function g is either s (for go straight) or t (for turn). We call such a function a *routing function*. There are 64 possible routing functions; we are going to show that none of them do any good for us. This statement is the 'research' – as opposed to 'expository' – contribution of our paper. We validate it by first introducing, for each routing function g, an algorithm for Fat Flies and then showing that none of them achieve what we wish to do.

7.2. Algorithm for Fat Flies Model g

1. Mark the face on which you are standing.
2. Crawl onto the face which meets the one on which you are standing at the edge in front of you. Check its orientation $i \in \{1,2,3,4,5,6\}$.
3. See if it is dirty.
   (a) If it is, fly away.
   (b) If it is not, see if it is marked.
   - If it is, dirty it.
   - If it is not, mark it.
4. If $g(i) = t$, turn 90° in the legitimate direction indicated in Fig. 8.
5. Start again at Instruction (2).

Consider the case when $g(i) = t$, for $i \in \{2,3,4,5,6\}$. (We will fix the value of $g(1)$ later.) If the Fat Fly is placed on top of a single sugar cube as indicated in Fig. 8, then after the sixth execution of Instruction (2) in the algorithm for Fat Flies Model g it will have followed the route indicated in Fig. 5 and each of the faces have been marked, but none have been dirtied. At this moment $i = 1$, but the Fat Fly is not facing in the same direction as it was facing in its initial position indicated in Fig. 5; rather, it is facing in the direction indicated by the incoming arrow on the top face. According to Instruction (3) the top face gets dirtied. Now we have to apply Instruction (4).

If $g(1)$ happens to be t, then the Fat Fly turns to the left, putting itself back into its initial position, and repeats the same route as before (dirtying as it goes), until it gets back to face 1, which is dirty and so it flies away. At this time, the same edges are covered as by the Model 1 algorithm, and so while all faces are dirty, it has not fulfilled the other part of its task, to cross all edges.

Otherwise $g(1) = s$ and the Fat Fly will not turn and will, according to Instruction (2), climb onto the face with orientation $i = 3$. This is also marked, but not dirty. According to Instruction (3) it gets dirtied and, according to Instruction (4), the Fat Fly turns to the right. Then, according to Instruction (2) it climbs onto the face with orientation $i = 5$. This is also marked, but not dirty. According to Instruction (3) it gets dirtied and, according to Instruction (4), the Fat Fly turns again to the right. Then, according to Instruction (2), it climbs again onto the face with orientation $i = 1$. This face is dirty, and so the Fat Fly flies away, having dirtied only three of the six faces of the sugar cube.

This shows that if $g(i) = t$, for $i \in \{2,3,4,5,6\}$, then the algorithm for Fat Flies Model g will not do what we wish it to do (irrespective of the choice of $g(1)$). This behaviour is not accidental, as we now show.

We call a routing function g *legitimate* if whenever a Fat Fly is placed on a single sugar cube as indicated in Fig. 8, and made to behave according to the algorithm for Fat Flies Model g, by the time it flies away it will have traversed each edge at least once. (The Fat Fly will eventually fly away, since in the execution of Instruction (3), if the Fat Fly does not fly away, it will either mark a face or it will dirty a face. As the algorithm keeps looping through Instructions, more faces get first marked and then dirtied and, since there

are only six faces, eventually the face onto which the fly crawls has to be dirty.)

We have shown prior to stating the definition that g is not legitimate if $g(i) = t$, for $i \in \{2,3,4,5,6\}$, irrespective of the choice of $g(1)$. We could carry out a similar study for each of the remaining 62 routing functions to check whether or not it is legitimate. This could be done by following the behaviour of a Fat Fly placed on a single sugar cube as indicated in Fig. 5. We take here a more elegant alternative approach.

Let us ignore for now the routing function itself and just look at the possible routes taken by a Fat Fly when placed on a single sugar cube, as indicated in Fig. 8 under the control of an algorithm for Fat Flies Model g, for any g. The first stage of such a route is the same for all gs: the Fat Fly moves from face *1* to face *2*. The simplification in the approach that is presented here is due to this fact: the initial segments of the routes taken by the Fat Fly are common for many different gs. So we list all possible initial routes of length 1, 2, 3, and so on. To ensure that a route corresponds to a routing function, we must make sure that if a face is visited the second time along a route, then the route is continued in a consistent way: if we went straight the first time, then we must do so the second time, and if we turned the first time, then we must turn the second time. A particular route need not be expanded any further if the face that is being visited is visited for the third time (this corresponds to the Fat Fly flying away). After we have traversed all twelve edges, this condition must be satisfied, and so the longest route that we consider will be of length 12. Also, if an initial route (of length no more than 12) is such that the same edge is visited twice, then we need not expand that route any further, since from this time on the Fat Fly will keep retracing its previous route, and so will never traverse the so far untraversed edges. (This situation corresponds to the case $g(1) = t$ in the example given above.)

The initial routes of length up to 5 are indicated in Fig. 9. At the bottom of the figure, a * indicates that the particular route need not be expanded any further because the edge from the face with orientation *1* to the face with orientation *2* has been traversed twice. (Letters *a* to *l* label routes which need to be investigated further.) Since g is a function of the orientation of the face, the second time the face with orientation *2* is visited on a route, the same decision has to be taken regarding going straight as was taken the first time; this results in the arrow which ends at *e* not being one of a pair.

In Fig. 10 we can expand the routes which end their first five steps at the points indicated by the

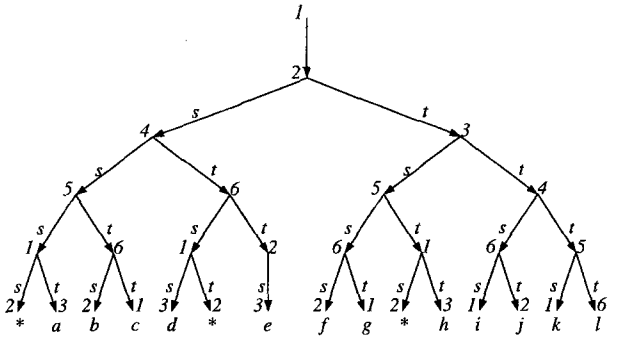


Fig. 9. All routes of length 5 that a Fat Fly may take according to an Algorithm for Fat Flies Model g.

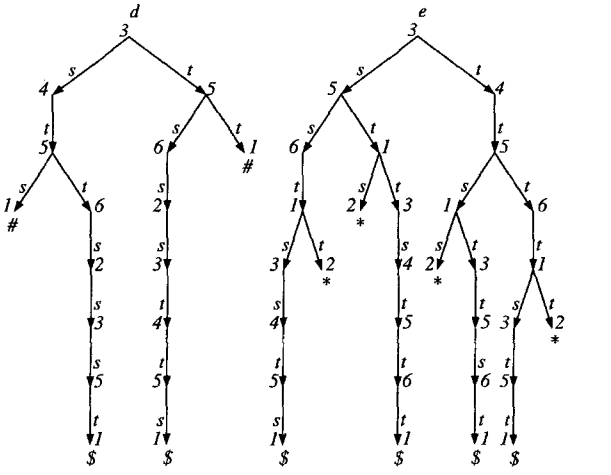


Fig. 10. The rest of the possible routes that a Fat Fly may take under the control of an algorithm for Fat Flies Model g expanded from points *d* and *e* in Fig. 9.

letters *d* and *e* in Fig. 9. Again, * indicates that the particular route need not be expanded any further because the edge from the face with orientation *1* to the face with orientation *2* has been traversed twice. In Fig. 10 we also see some # and $ signs; these mean that the route cannot be expanded further, since the face with orientation *1* has just been entered the third time and so the Fat Fly flies away at this point. The difference is that # indicates that at this time not all 12 edges have yet been traversed, while $ indicates the opposite. It is the latter which signals success, the routing function which gives rise to such a route is a legitimate one.

For example, the route which leads to the leftmost $ sign in Fig. 10 shows that the routing function g for which $g(i) = s$, except for $i = 4$ or 5 is legitimate and the route which leads to the right-most $ sign in Fig. 10 shows that the routing function g for which $g(i) = t$, except for $i = 1$ or 2 is legitimate. By expanding all the nodes, we can in this fashion discover all the

legitimate routing functions. They are listed in Table 1. (The first and second row in this table, labelled by *d1* and *d2*, respectively, correspond to the first and second $ signs in the expansion of the point *d* in Fig. 10. Similarly, the rows labelled by *e1*, *e2*, *e3* and *e4*, correspond to the four $ signs, left to right respectively, in the expansion of the point *e* in Fig. 10.) However, we did not actually produce this table by an exhaustive expansion of all nodes, rather we relied on a graph-theoretical approach in order to simplify our work.

8. DIGRAPHS

A *digraph* (sometimes called a directed graph) is an ordered pair (M,ρ), where M is any non-empty finite set (in the graph-theoretical context we refer to its elements as *nodes*) and ρ is an anti-reflexive binary relation on M, i.e. for any $c \in M$, $(c,c) \notin M$. (In the graph-theoretical context we refer to the elements of ρ as *arcs*.) In view of this, all the terminology introduced in Section 4 is immediately applicable to digraphs. Since in a digraph the ρ is fixed, we use the phrases *adjacent to*, *adjacent from*, *path*, etc. instead of ρ-adjacent to, ρ-adjacent from, ρ-path, etc., respectively.

An example of a digraph is provided by Fig. 8. In this case, $M = \{1,2,3,4,5,6\}$ and $(i,j) \in \rho$ if, and only if, we can get in Fig. 8 from the centre of a face with orientation *i* to the centre of a face with orientation *j* following two arrowed line segments across an edge. There is an alternative simple representation of this digraph, which is given in Fig. 11. In this representation, the nodes correspond to the vertices of the hexagon and the arcs are represented by the arrowed line segments connecting the vertices.

Table 1. Definitions of all the legitimate routing functions

	1	*2*	*3*	*4*	*5*	*6*
d1	s	s	s	t	t	s
d2	s	s	t	t	s	s
i	s	t	t	s	s	s
f1	t	t	s	s	s	s
a1	t	s	s	s	s	t
c	s	s	s	s	t	t
e1	s	s	s	t	s	t
b2	s	s	t	s	t	s
f2	s	t	s	t	s	s
a2	t	s	t	s	s	s
g	s	t	s	s	s	t
b1	t	s	s	s	t	s
e2	t	s	s	t	t	t
e4	s	s	t	t	t	t
l	s	t	t	t	t	s
k	t	t	t	t	s	s
j1	t	t	t	s	s	t
h1	t	t	s	s	t	t
e3	t	s	t	t	s	t
j2	s	t	t	s	t	t
h2	t	t	s	t	t	s

The first thing we do with this representation is to show that the eight possible ways of drawing the three cycles (see Fig. 8) on a sugar cube lead to exactly the same consequences. This is because we can label the orientations of the faces on the sugar cubes (after the cycles are drawn) in such a way that the resulting digraph has the alternative representation shown in Fig. 11. (For example, we can call the orientation of the top face *1* and call the orientations of the two faces adjacent to it *2* and *3* in such a way that 3 is adjacent to 2, etc.) So by the simple trick of choosing the names of the orientations of the faces of a sugar cube in a way which is dependent on how the directions of the cycles have been selected, we end up with the same digraph in all cases. Hence, all the things that we will say regarding the specific case that we happened to select (the one in Fig. 8) will also be valid for the seven alternative ways.

A path $\langle c^{(0)},\cdots,c^{(K)}\rangle$ in a digraph (M,ρ) is an *Eulerian trail* if:

(i) $c^{(K)} = c^{(0)}$.

(ii) $\{c^{(0)},\cdots,c^{(K)}\} = M$.

(iii) $\{(c^{(k-1)},c^{(k)}) | 1 \le k \le K\} = \rho$.

(iv) For $1 \le k < k' \le K$, if $c^{(k)} = c^{(k')}$, then $c^{(k-1)} \ne c^{(k'-1)}$.

The first condition means that an Eulerian trail has to be a *closed path* or, as we will also call such a thing, a *loop*. The second condition says that the

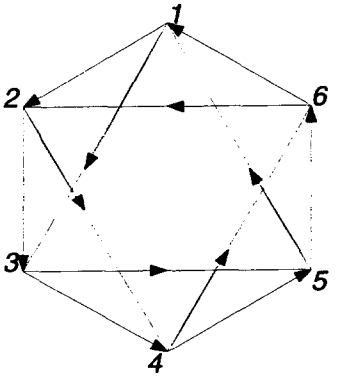


Fig. 11. Alternative representation of the digraph of legitimate routes in Fig. 8.

Eulerian trail has to visit all nodes. The third condition says that it makes use of all the arcs. The fourth condition says that no arc occurs in the trail more than once. For the digraph of Fig. 8, the path $\langle 1,2,3,4,5,6,1\rangle$ provided by the route of the Model 1 Fat Fly of Fig. 5 satisfies conditions (i), (ii) and (iv), but it fails to be an Eulerian trail since it does not satisfy (iii), e.g. $(1,3) \in \rho$, but 1 and 3 are never consecutive in the given path. The path $\langle 1,3,5,1,2,4,5,1,2,3,4,6,1\rangle$ is also not an Eulerian trail; it satisfies conditions (i), (ii) and (iii), but fails to satisfy (iv) since, for example, *2* occurs at two different places in the path and is preceded by *1* in both cases. On the other hand, the path $\langle 1,2,4,6,1,3,4,5,6,2,3,5,1\rangle$ provided by the route of the Model *d1* Fat Fly, is an Eulerian trail.

Ligitimate routes for the Fat Fly correspond exactly to those Eulerian trails in the digraph of Fig. 11 in which the first two elements of the path are *1* and *2*. The choice of these first two elements is not important: if $\langle c^{(0)},\cdots,c^{(K)}\rangle$ is an Eulerian trail and $1 \le k \le K$, then $\langle c^{(k)},\cdots,c^{(K)},c^{(1)},\cdots,c^{(k)}\rangle$ is also an Eulerian trail and, in fact, these two Eulerian trails are considered to be one and the same. From the point of view of our specific example, consider any of the legitimate routing functions *g* of Table 1. Suppose that the Fat Fly under the control of the Algorithm for Fat Flies Model *g* (started on the top of the face with orientation *1* and looking towards the edge shared with the face with orientation 2, see Fig. 8) crawls during its execution from a face in orientation *i* to a face in orientation *j*. If instead the Fat Fly is placed on top of the face with orientation *i* looking towards the edge of the face with orientation *j*, then it will still trace the same Eulerian trail (in the sense of sameness just discussed), following the rest of the trail as if it has started from face *1* looking towards *2* until it reaches that position, and then following the beginning of that trail until it reaches face *i* looking towards *j* again.

This, combined with the third defining condition of an Eulerian trail, means that if *g* is a legitimate routing function and the Fat Fly is placed on top of *any* face of a single sugar cube towards *either* of the edges indicated by an arrow in Fig. 8, then under the control of the Algorithm for Fat Flies Model *g* it will visit each face twice and traverse each edge once before flying away.

We have already noted that the number of legitimate routing functions is exactly the number of different Eulerian trails of the digraph of Fig. 11. There is a formula, applicable to all digraphs which have Eulerian trails (sometimes referred to as the BEST theorem [5, p. 204], using which we can calculate the number of Eulerian trails in the digraph. An application of this formula to Fig. 11 consists of having to work out the determinant of a simple 5×5 matrix and yields immediately (without having to explicitly find them) that the number of legitimate routing functions is indeed 21, as is indicated in Table 1.

The next thing we are going to do is show that even though there are 21 different Eulerian trails of the digraph of Fig. 11, there are only four such Eulerian trails which are 'essentially' different from each other, in the sense that all the others can be derived from these four by an easy manipulation: if we rotate the figure clockwise 60° around its geometrical centre, then all the nodes and arcs will map onto themselves (only the labels will appear to have been shifted). Defining the notation n_6 to mean exactly n, if $1 \le n \le 6$, with $7_6 = 1$ and $8_6 = 2$, it follows that if a routing function *g* gives rise to an Eulerian trail, then the routing function *g'*, defined by $g'(n) = g((n+1)_6)$ must also give rise to an Eulerian trail (and so is a legitimate routing function). To illustrate this, consider the Eulerian trail $\langle 1,2,4,6,1,3,4,5,6,2,3,5,1\rangle$, which is the route for the Algorithm for Fat Flies Model *d1* (started in the situation shown in Fig. 8). Rotation clockwise by 60° turns this into the Eulerian trail $\langle 6,1,3,5,6,2,3,4,5,1,2,4,6\rangle$, which is the same Eulerian trail as $\langle 1,2,4,6,1,3,5,6,2,3,4,5,1\rangle$. This is in fact the route of the fly under the control of the Algorithm for Fat Flies Model *d2* (started in the situation shown in Fig. 8), as indicated in Figs 9 and 10. This is expected, since simple observation of the first two lines of Table 1 shows that $d2 = d1'$.

We have thus shown that the fact that *d2* is a legitimate routing function follows from the fact that *d1* is a legitimate routing function, and does not require an independent confirmation of tracing the route of the fly under the control of the Algorithm for Fat Flies Model *d2*. By repeating this, we get consecutively that *i*, *f1*, *a1*, *c* are legitimate routing functions. In fact, by observing the entries of Table 1, we see that the facts that *d1*, *e1*, *e2* and *e3* are legitimate routing functions (demonstrated in Figs 9 and 10) imply that the other 17 entries are legitimate routing functions as well. Since we know from the BEST theorem that there are 21 such functions, there is no need to look any further. We have now shown that Table 1 is complete, without having had to produce expansions of the routes in addition to those shown in Fig. 10.

Of equal importance is that the same approach can be used to simplify proofs of negative results. For example, if the Algorithm for Fat Flies Model *d1* does not satisfy Aim 6.1, then it follows that Algorithm for Fat Flies Model *g* will not satisfy Aim 6.1 if *g* is

d2, *i*, *f1*, *a1*, or *c*. Similar statements can be made, in place of *d1*, for *e1*, *e2*, and *e3*. We now demonstrate by a counter-example that the algorithm for Fat Flies Model *d7* does not satisfy Aim 6.1.

In Fig. 12, we show a pair of sugar cubes, labelled *I* and *II*, respectively. The Fat Fly is placed on cube *I* on the face with orientation *1* (we use the notation *I1* to denote this face), looking towards the edge of face *I2*. Application of the algorithm for Fat Flies Model *d1* provides us with the route $\langle I1,I2,I4,II4,II5,II6,II2,I2,I3,I5,I1,I2,\cdots\rangle$, which now repeats from the beginning, and so the Fat Fly will fly away without ever dirtying *II1*. It follows that the Algorithm for Fat Flies Model *d1* does not satisfy Aim 6.1. From what we have said about the relationship between legitimate routing functions, counter-examples (also consisting of pairs of cubes touching along a face) can be constructed for functions *d2*, *i*, *f1*, *a1* and *c*.

9. WHAT CAN A FAT FLY DO?

We have shown by the use of two sugar cubes meeting in a face that the Algorithm for Fat Flies Model *d1* fails and, therefore, so do the other five legitimate models associated with it. The same pair of sugar cubes produces a counter-example for *e3*, so it and its associated functions *j2* and *h2* also fail.

A cube pair does not provide a counter-example for *e1* or *e2*, so we consider instead the arrangement of four cubes meeting along edges only (with a hole through the middle), as shown in Fig. 13. Using this figure it is easy to check that both *e1* or *e2* (and therefore all their associated models) fail.

We have now shown that none of the Algorithms for Fat Flies Model *g* will do. We can formalise these results as follows.

Theorem 9.1 For every routing function g, there exists a finite δ_3-connected set A of points in Z^3 such that $\overline{A}$ is ω_3-connected and, when the voxels associated with the points in A are filled with sugar cubes, one can place the Fat Fly on top of an uncovered face of a sugar cube in such a way that if the Fat Fly is made to follow the instructions of the Algorithm for Fat Flies Model g, then it will fly away without having dirtied all the faces in $\delta(A,\overline{A})$.

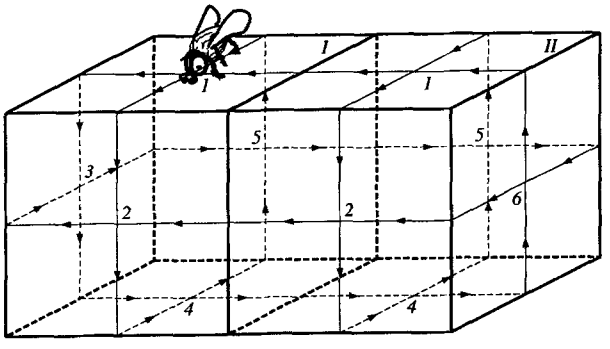


Fig. 12. A pair of sugar cubes, providing a counter-example which shows that the Algorithm for Fat Flies Model *d1* does not satisfy Aim 6.1.

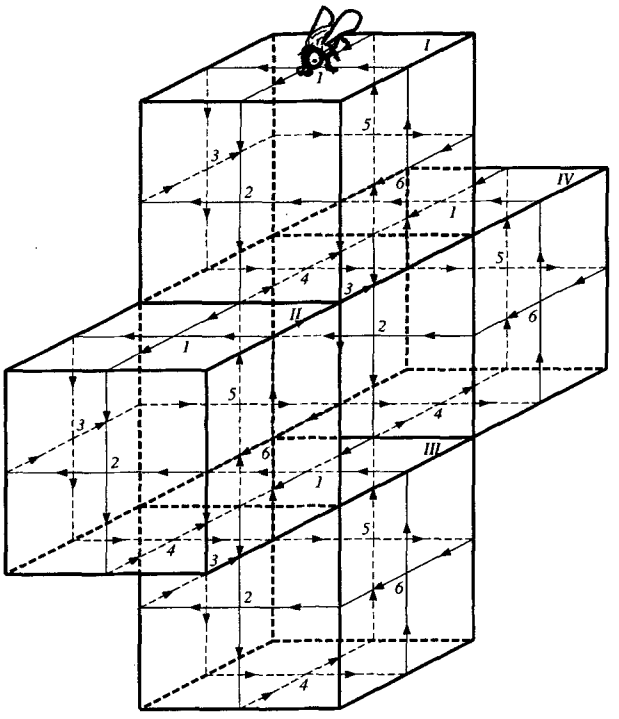


Fig. 13. An arrangement of four sugar cubes (with a tunnel through the middle of them), providing a counter-example which shows that the Algorithms for Fat Flies Models *e1* and *e2* do not satisfy Aim 6.1.

In view of this theorem, we need to consider more complex alternatives.

10. ALGORITHMS FOR CLONING FLIES

One approach to achieving Aim 6.1 is due to Artzy [6]. His idea was to allow the flies to clone themselves and so be able to follow both directions on the face of a cube (see Fig. 8) simultaneously. This results in the following:

10.1 Algorithm for Cloning Flies

1. Crawl onto the face which meets the one on which you are standing at the edge in front of you.
2. See if it is dirty.
   (a) If it is, fly away.

(b) If it is not,

- dirty it,
- clone yourself into two flies facing in the two legitimate directions indicated in Fig. 8 and both starting at Instruction (1).

If a Cloning Fly is placed on a single sugar cube as shown in Fig. 8, then after executing one iteration of Instructions (1) and (2) of the Algorithm for Cloning Flies the situation will be as depicted in Fig. 14.

The Algorithm for Cloning Flies satisfies the conditions of Aim 6.1. We will not bother to prove this here because although the Cloning Flies are a wonderful concept, it is hard to manufacture them. We now discuss an algorithm that can be run on a computer. The problem is that of simulating the behaviour of a device which can follow many courses of action simultaneously (such as the Cloning Fly, if we consider it and all its descendants as one device) by a device which can execute only one action at a time. The usual trick is to create a queue of the things whose handling is postponed until the future. For us, the elements in the queue are faces. (All published three-dimensional boundary trackers use a queue to store information about faces that have been found, but for which it has not yet been pursued whether or not they give rise to additional faces in the boundary. However, the material in the last section is the first indication in the published literature that some sort of queue is actually necessary.) The instructions of the algorithm are not given to a fly, but to a computer which is simulating in a sequential mode the behaviour of the Cloning Fly, with the aim of eventually producing the same set of dirty faces as would be produced by the Cloning Fly.

10.2. Algorithm Simulating Cloning Flies

1. Dirty the face which meets the one on which you are standing at the edge in front of you and put it into the queue.
2. Remove a face f from the queue and find the faces $f1$ and $f2$ in the boundary to which the Cloning Fly would get from the face f.
   (a) If $f1$ is not dirty, then dirty it and put it into the queue.
   (b) If $f2$ is not dirty, then dirty it and put it into the queue.
3. Check if the queue is empty.
   (a) If it is, STOP.
   (b) If it is not, start again at Instruction (2).

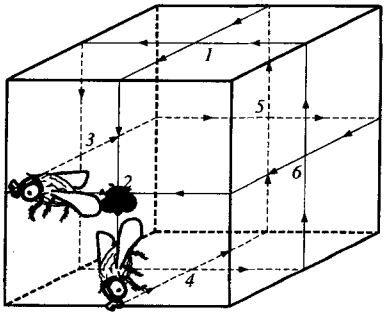


Fig. 14. The situation after execution of one iteration of Instructions (1) and (2) of the Algorithm for Cloning Flies when the Cloning Fly is initially placed as depicted in Fig. 8.

The Algorithm Simulating Cloning Flies satisfies the conditions of Aim 6.1. We do not prove this, since we wish to press on with a development which is one of the most pleasing aspects of Artzy's idea [6].

The difficulty that arises in the Algorithm Simulating Cloning Flies is subtle; it comes from the phrases "If $f1$ is not dirty, ..." and "If $f2$ is not dirty, ..." at the beginning of Instructions 2a,b for the Algorithm Simulating Cloning Flies. How do we find out, if a face is dirty? In a medical application, surfaces such as the ones depicted in Fig. 2 may consist of millions of faces, and checking whether or not a face has already been dirtied requires lots of computational resources (storage space and/or computer time).

The observation that allows us to considerably reduce the size of this problem is the following: there are exactly two ways to get to any face in the boundary. We now rephrase this observation in standard mathematical terminology. Consider the digraph whose nodes are the faces in the boundary and in which a face f is adjacent to a face f' if, and only if, f' is one of the faces in the boundary to which the Cloning Fly would get from the face f. Then we see that every face in the boundary is adjacent from exactly two faces in the boundary; in other words, every node in the digraph has *indegree* two.

We make use of this mathematical property by introducing the additional concept of a *list* of faces. The idea is that we can keep the number of faces in the list to be generally much smaller than the number of dirty faces, but the algorithm can do everything that it needs to do by checking for particular faces whether or not they are in the list, rather than whether or not they are dirty.

10.3. Artzy's Algorithm

1. Dirty the face which meets the one on which you are standing at the edge in front of you, put it into the queue and put two copies of it in the list.
2. Remove a face f from the queue and find the faces

f1 and *f2* in the boundary to which the Cloning Fly would get from the face *f*.

(a) Try to find one copy of *f1* in the list.

- If successful, remove it from the list.
- If not, then dirty *f1*, and put it into the queue and the list.

(b) Try to find one copy of *f2* in the list.

- If successful, remove it from the list.
- If not, then dirty *f2*, and put it into the queue and the list.

3. Check if the queue is empty.
   (a) If it is, STOP.
   (b) If it is not, start again at Instruction (2).

The claim regarding Artzy's Algorithm is the following (first made in Artzy [6]), proved using combinatorial topology in Herman and Webster [8], with a more elegant proof in Rosenfeld et al [9].

Claim 10.1 Suppose that in the cubic grid Z^3 the voxels of a finite number of grid points are filled with sugar cubes and a Cloning Fly is put on top of one of these sugar cubes into a voxel which is not occupied by a sugar cube. Let g be the name of the grid point of the sugar cube on top of which the Cloning Fly is placed into the voxel of a grid point named h. Let O be the δ_3-component of the set of grid points with sugar cubes which contains g, and Q be the ω_3-component of the set of grid points without sugar cubes which contains h. Artzy's Algorithm will stop after a finite number of steps and, at that time, the set of faces that have been dirtied is exactly $\partial(O,Q)$.

The boundaries shown in our Fig. 2 have been detected using Artzy's Algorithm. Other such boundaries are illustrated in earlier work [6,8,10]; the same papers, especially Frieder et al [10], also discuss the computational performance of the algorithm. The algorithm has been widely used in medical applications, such as the study of craniofacial deformities [11], of disk disease [12], of acetabular fractures [13], and of maxillofacial trauma [14].

11. DISCUSSION

Previously known algorithms for boundary tracking in two-dimensional space (e.g. the Algorithm for Flat Flies) and in three-dimensional space (e.g. Artzy's Algorithm) differ in an essential aspect: the two-dimensional algorithms do not need to make use of a queue (of loose ends in the tracking process which are to be picked up again to complete the tracking), while the three-dimensional ones do. Our Theorem 9.1 shows that this is not accidental: under some mild assumptions, a boundary tracker without a queue will fail its task on some surfaces. This is our main result.

It follows that we cannot hope to reproduce the simplicity of two-dimensional boundary tracking in three-dimensional space. In applications (such as medicine) in which the number of faces in a boundary may be of the order of millions, this implies that the management of the order in which faces are visited often places significant demands on available computational resources. An approach towards three-dimensional boundary tracking is that of the Cloning Flies; the desire to reduce its computational demands leads to Artzy's Algorithm and the 'list' that is introduced in there seriously reduces, but does not totally eliminate, the problem. Based on the timings reported in Frieder et al [10], it appears that about 80% of computer time in medical applications of Artzy's Algorithm is taken up with the management of the order of tracking. Since those times were of the order of tens of minutes (in 1985) and are even today far from real time, the potential five-fold speed-up (that could be obtained by eliminating the part of the computer time is taken up with the management of the order of tracking) is important.

There may be alternate approaches not requiring queues: for example, we may allow the Fat Fly to have a memory (Model *g* Fat Flies do not) and so possibly achieve Aim 6.1 without using a queue. Our attempts at designing such an algorithm have not proved successful, but we do not have a proof that it cannot be done. It is by no means clear that, even if it existed, such an algorithm would outperform Artzy's Algorithm in practical applications.

An alternative way to try to get around the problem is to replace Aim 6.1 by something different, but still useful from the point of view of applications. One such approach was advocated by Gordon and Udupa [15]: by replacing δ_3 with an adjacency with less symmetric properties, the authors reduced the time required by boundary tracking to nearly a half. It may be possible to design algorithms without queues for subsets of the set of boundaries defined in Aim 6.1; in applications such as medicine this does not seem to be a reasonable approach, since we could never be sure if our algorithm is being applied to track one of the boundaries for which it is known to work. A more promising approach is to change the underlying grid. For example, by adopting the so-called face-centred cubic grid (as advocated for pattern analysis in [16]) we get voxels (in the shape of rhombic dodecahedra) which cannot share an edge without sharing a face. Thus the situation depicted in Fig. 5 cannot arise, and the possibility of boundary tracking without a queue

is not ruled out. The investigation of efficient boundary tracking for such alternative grids is an open area of research.

References

1. Abbott EA. Flatland, A Romance of Many Dimensions. Little, Brown and Co, 1899
2. Herman GT. Oriented surfaces in digital spaces. CVGIP: Graph Models Image Proceedings 1993; 55: 381–396
3. Kong TY, Rosenfeld A. Digital topology: Introduction and survey. Comput Vision Graph Image Proc 1989; 48: 357–393
4. Udupa JK. Multidimensional digital boundaries. GVGIP: Graph Models Image Proceedings 1994; 56: 311–323
5. Harary F. Graph Theory. Addison-Wesley, 1969
6. Artzy E, Frieder G, Herman GT. The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. Comput Graph Image Proc 1981; 15: 1–24
7. Munkres JR. Topology: A First Course. Prentice Hall, 1975
8. Herman GT, Webster D. A topological proof of a surface tracking algorithm. Comput Vision Graph Image Proc 1983; 23: 162–177
9. Rosenfeld A, Kong TY, Wu AY. Digital surfaces. GVGIP: Graph Models Image Proc 1991; 53: 305–312
10. Frieder G, Herman GT, Meyer C, Udupa J. Large software problems for small computers: an example from medical imaging. IEEE Software 1985; 2(5): 37–47
11. David DJ, Hemmy DC, Cooter RD. Craniofacial Deformities: Atlas of Three-Dimensional Reconstruction from Computed Tomography. Springer-Verlag, 1990
12. Herman GT, Coin CG. The use of three-dimensional computer display in the study of disk disease. J Comput Assist Tomogr 1980; 4: 564–567
13. Burk Jr DL, Mears DC, Cooperstein LA, Herman GT, Udupa JK. Acetabular fractures: Three-dimensional computed tomographic imaging and interactive surgical planning. CT: J Comput Tomogr 1986; 10: 1–10
14. DeMarino DP, Steiner E, Poster R, Katzberg RW, Hengerer AS, Herman GT, Wayne WS, Prosser DC. Three-dimensional computed tomography in maxillofacial trauma. Arch Otolaryng – Head Neck Surg 1986; 112: 146–150
15. Gordon D, Udupa JK. Fast surface tracking in three-dimensional binary images. Comput Vision Graph Image Proc 1989; 45: 196–241
16. Preston Jr K. Multidimensional logical transforms. IEEE Transactions PAMI 1983; 5: 539–554

Gabor T. Herman received the M.Sc. degree in mathematics from the University of London, England, in 1964, the M.S. degree in engineering science from the University of California, Berkeley, in 1996, and the Ph.D. degree in mathematics from the University of London, England, in 1968. From 1969 to 1981, he was with the Department of Computer Science, State University of New York at Buffalo, where in 1976 he became the Director of the Medical Image Processing Group. Since 1981, he is a Professor in the Medical Imaging Section of the Department of Radiology at the University of Pennsylvania. He is involved editorially with a number of journals – in particular he was for a while Editor-in-Chief of the IEEE Transactions on Medical Imaging – and is the author of numerous articles and books on medical imaging and related topics, including "Geometry of Digital Spaces", to be published by Birkhauser in 1998.

David Robinson graduated B.Sc.(Hons) in Mathematics at the University of London in 1959. After two years in the electronics industry he was appointed Lecturer in Mathematics at the University of Canterbury, New Zealand. He took his Ph.D. degree in 1967 from that university with a thesis in graph theory, and was promoted to Senior Lecturer the following year. His research publications, mostly based on applications of discrete mathematics, range from pure mathematics to operations research, and most recently, botany.

Correspondence and offprint requests to: G.T. Herman, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Blockley Hall, Fourth Floor, 423 Guardian Drive, Philadelphia, PA 19104-6021, USA. Email: gabor@mipg.upenn.edu