Towards an Invertible Euclidean Reconstruction of a Discrete Object

Rodolphe Breton[1], Isabelle Sivignon[3], Florent Dupont[2], and Eric Andres[1]

[1] Laboratoire IRCOM-SIC, Université de Poitiers, BP 30179,
86962 Futuroscope Chasseneuil Cedex, France
{andres,breton}@sic.sp2mi.univ-poitiers.fr
http://www.sic.sp2mi.univ-poitiers.fr
[2] Laboratoire LIRIS – FRE 2672 CNRS, Université Claude Bernard Lyon I,
Bât. NAUTIBUS, 8, bd Niels Bohr, 69622 Villeurbanne cedex, France
fdupont@liris.cnrs.fr
http://liris.cnrs.fr
[3] Laboratoire LIS, Domaine universitaire Grenoble, BP 46,
38402 St Martin d'Hères Cedex, France
sivignon@lis.inpg.fr
http://www.lis.inpg.fr

Abstract. An invertible Euclidean reconstruction method for a 2D curve is proposed. Hints on an extension to 3D are provided. The framework of this method is the discrete analytical geometry. The reconstruction result is more compact than classical methods such as the Marching Cubes. The notions of discrete cusps and patches are introduced.

Keywords: Discrete object, invertible Euclidean reconstruction.

1 Introduction

The reconstruction of discrete objects is mainly performed in practice with the "Marching Cubes" method [1] (and all its follow ups). For a couple of years another approach, based on discrete analytical geometry, is investigated in the discrete geometry community. The aim is to decompose the boundary of a discrete object into discrete analytical polygons and then these polygons into Euclidean polygons. The method has to be *invertible*, i.e. the discretization of the reconstructed boundary has to be equal to the original discrete object. We don't want any information to be added or lost. The aim of this new approach is to provide a more compact reconstruction. Several other attempts have already been made in this direction that are not satisfying and usually not invertible (see [2] for details). Our method is based on Vittone's recognition algorithm for the decomposition of the discrete boundary into discrete line pieces in 2D and discrete plane pieces in 3D. The analytical framework is provided by the standard discrete analytical model that defines 2D and 3D discrete polygons [3]. A working solution in 2D and indications on how to tackle the 3D case are proposed. The method works basically as follows: a discrete boundary is decomposed with Vittone's algorithm [4] into discrete line pieces in 2D (resp. discrete

I. Nyström et al. (Eds.): DGCI 2003, LNCS 2886, pp. 246–256, 2003.

plane pieces in 3D). The result of Vittone's algorithm is adapted to the standard analytical model as it is, for the moment, the only suitable discrete analytical model [3]. The reconstruction process is guided by so called discrete cusps in order to propose a reconstruction that fits better a "common sense" recontruction. A Euclidean line (resp. 3D plane) candidate is chosen among all the possible solutions. This is done for each discrete line piece (resp. 3D plane piece). All these 2D lines (resp. 3D planes) form a Euclidean 2D polygon (resp. 3D polyhedron).

The discretization of this Euclidean object (2D polygon or 3D polyhedron) is not necessarily equal to the boundary of the discrete object but is usually larger. In 2D, in order to avoid this problem, and provide the revertibility property, patches are introduced. In 3D, the problem is more difficult and not completely solved so far. Not only the vertices but also the 3D edges of the polyhedron can be located outside the discrete object. Several hints are given on how to solve these problems, especially with convex and non-convex discrete objects.

In section 2, a new discrete curve reconstruction method is provided. Notions such as discrete cusps and patches are introduced. In section 3, the 3D case is examined. The convex and non-convex case are studied and hints on solutions are given. We conclude in section 4 with some perspectives.

Brief recall on the standard model. The standard digitization of a Euclidean object consists in all the pixels (resp. voxels) that are cut by the object. The standard lines (resp. planes) can be defined arithmetically: a discrete standard line (resp. plane) of parameters (a, b, μ) (resp. (a, b, c, μ)) is the set of integer points (x, y) (resp. (x, y, z)) verifying $-\omega \leq ax + by(resp. + cy) + \mu < \omega$ where $\omega = \frac{|a|+|b|\,(resp.+|c|\,)}{2}$. A standard line (resp. plane) is a 4-connected line (resp. 6-connected plane). If we denote $St(O)$ the standard digitization of the object O, the following useful properties can be derived from the geometrical definition of this model: $St(O_1 \cap O_2) \subseteq St(O_1) \cap St(O_2)$ and $St(O_1 \cup O_2) = St(O_1) \cup St(O_2)$

2 Reconstruction of a 2D Discrete Curve

2.1 Principle

We consider here 4-connected curves. To reconstruct a discrete curve, we first choose a point on that curve, recognize a discrete straight-line segment and then, repeat this process along the curve.

The recognition algorithm used was developed by Vittone [5]. For a given discrete edge, it provides the set of all corresponding Euclidean straight lines as a polygon in a parameter space (well studied by Veelaert in [6]). The standard discretization [7] [3] of any of these Euclidean lines contains the original discrete edge. It has been proven that the set of solutions is a (3 or 4)-vertex convex polygon (see [8]) in the (α, β) parameter space $\mathcal{P}$, and can only have one of the five shapes illustrated in fig. 1.

A Euclidean straight line $y = \alpha x + \beta$, in the cartesian space $\mathcal{C}$, corresponds to a point (α, β) in $\mathcal{P}$. Thus, the three (resp. four) vertices of the solution set correspond to three (resp. four) Euclidean straight lines in $\mathcal{C}$. We chose one

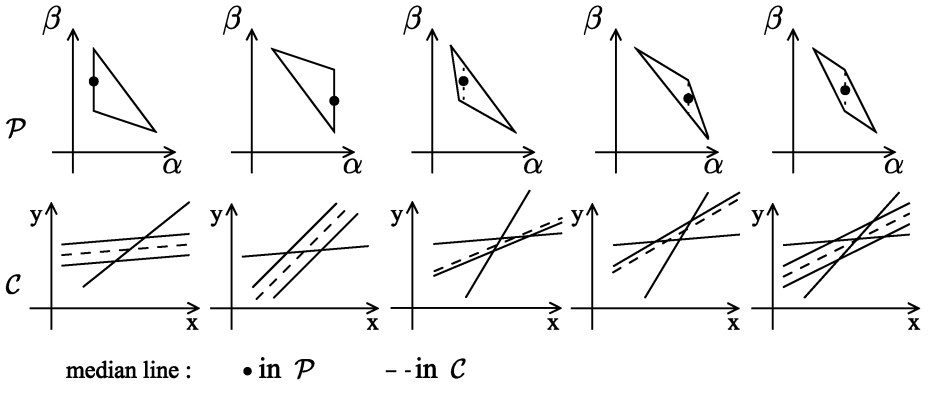

Fig. 1. The 5 possible shapes of the solution set and in each case, the chosen solution.

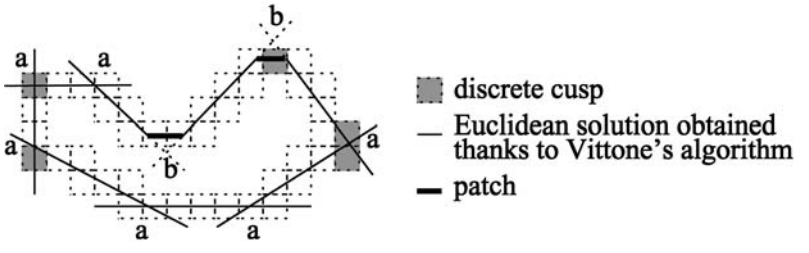

Fig. 2. Example of discrete cusps and Euclidean solutions. In (a), a regular case. In (b), addition of a patch.

particular line as a solution and called it the *median solution*. This seems to be a reasonable choice, as illustrated on fig. 1. This figure shows the median solution in $\mathcal{P}$ and $\mathcal{C}$ for each possible shape of the set.

Prior to the recognition process, we look for remarkable points on the discrete curve. We call those points *discrete cusps* and define them as follows: a point of a discrete curve is a discrete cusp iff the segment composed of this point, the next two points and the previous two points, is not a discrete segment. We use the Freeman code to determine whether or not such a 5-pixel set is a discret segment. Fig. 2 shows an example of discrete cusps. These cusps act like "anchors" and help us to adjust segments' extremities: during the recognition of a discrete segment, we preferably begin (and end) a segment on a discrete cusp.

Starting Point. If there are cusps, we choose as a starting point of the algorithm, the cusp with the smallest x-coordinate and then with the smallest y-coordinate. If there are no cusps on the curve, we choose a regular point that fits the same conditions. This choice ensures the unicity of the process. We proceed then with the recognition of the curve counter clockwise.

2.2 Details on the Reconstruction

Before going on, we have to introduce some useful notations: p_i denotes the i-th pixel of a curve and s_k is the k-th segment of the polygonalized curve.

After the Vittone's algorithm, for each discrete segment we found, we obtain an equivalence class of all the lines that match this discrete segment and we choose the median line as a solution (see fig. 2). Then, we have to handle the intersections between those Euclidean lines. The most simple case occurs when two lines intersect in a pixel which belongs to the two corresponding discrete segments s_k and s_{k+1} (see fig. 2 (a)).

In this section, we explain the different cases we face during the reconstruction.

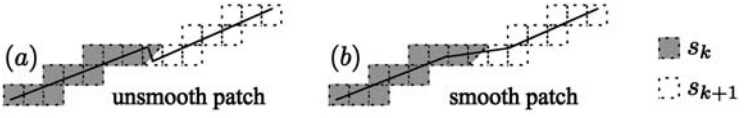

Fig. 3. (a) addition of a patch. (b) smoothing of this patch.

Patch. Our first problem is the intersection of two lines outside a pixel of the curve, or even, the non-intersection of two lines. As we must "constrain" the Euclidean curve inside the discrete curve, we decide to add a little *patch* to join the lines together (see fig. 2 (b)).

But for some cases, adding a patch causes undesired visual results as shown on fig. 3 (a). We soften this patch by extending it to the neighbouring pixels, as illustrated on fig. 3 (b).

In order to reduce the number of patches, we allow two solution lines to intersect in a 3-pixel long area, that is, the pixel common to the two discrete segments and its two neighbours. This little trick still allows reversibility.

Post-process Patch Removal. Sometimes, we can get rid of a patch thanks to a second pass of the recognition algorithm in the opposite direction. In fig. 4 (a), we see the result of a first reconstruction. As the two solution lines do not intersect in the permitted intersection area, we normaly should add a patch. But a second recognition, in the opposite direction, leads to (b) and a valid intersection. So, we eventually end up with the result (c).

2.3 The Algorithm

Initialization:

- we consider a discrete curve, i.e. a sorted sequence of n pixels: $p_1 \dots p_n$
- the cusps of the curve are determined

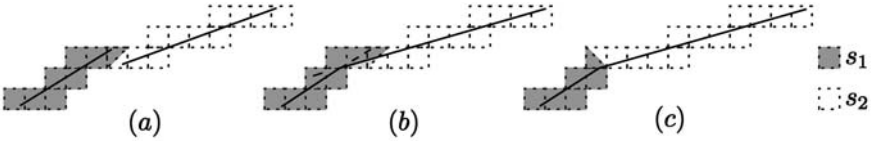

Fig. 4. Patch removal thanks to a reverse recognition.

Step 1: Recognition

- s_k denotes the current segment (at first $k = 1$)
- p_i denotes the current pixel (at first $i = 2$)
- we use Vittone's algorithm to recognize a discrete segment:
  - we insert pixel p_i in s_k
  - if this extended s_k is still a discrete segment, we go on: $i = i + 1$
  - else s_k ends up on p_{i-1} and either p_{i-1} or p_{i-2} becomes the starting point of the new segment: $i = i - 1$ (or $i = i - 2$) and $k = k + 1$
- until we reach the last pixel ($i = n$)
- in the case of a closed curve we carry on the recognition until we meet a cusp, and then we possibly merge the last and the first segment
- at this point, the curve is entirely recognized and splitted into k discrete segments and each one is linked to a coset of Euclidean solutions in the parameter space

Step 2: Reconstruction

- for each coset of solutions, we choose the median line d_k
- we must now create the Euclidean segments that are contained in d_k
- so, we set the first extremity of the first Euclidean segment r_1 (a point on d_1 that belongs to p_1, the first pixel of the curve)
- then, we enter a loop through the lines d_k:
  - if d_k (segment $s_k = [p_a, p_b]$) and d_{k+1} (segment $s_{k+1} = [p_b, p_c]$) intersect in p_b, p_{b-1} or p_{b+1}
  - then*, this intersection point becomes the second extremity of r_k and the first one of r_{k+1}
  - else (intersection outside or no intersection), we launch another recognition between p_c and p_a, which can lead to two cases:
    - ▷ we still have the same two segments s_k and s_{k+1}, therefore, the patch is unavoidable, and then, the second extremity of r_k is the first vertex of the patch, and the first extremity of r_{k+1} is the second vertex of the patch
    - ▷ s_{k+1} has been extended and the intersection between d_k and the new solution line allows us to avoid the patch; thus we go back to the regular case (see $*$)
- we eventually have a sequence of Euclidean segments r_k (each one defined by two Euclidean points) and this sequence forms a polygonal line of which discretization perfectly matches the starting discrete curve

3 Discrete Object Surface Polygonalization

In this section, we present the problem for 3D discrete volumes. We point out the type of difficulties we encounter and give some indications on the possible solutions to solve them.

3.1 Discrete Surface Segmentation

We consider an 18-connected discrete volume and its surface defined as the set of voxels sharing one face with the background object. Since discrete naive planes are the thinnest 18-connected discrete planes without 6-connected holes [9], they are well adapted for a segmentation of an object surface. In an arithmetical way, a discrete naive plane of parameters (a, b, c, μ) is the set of integer points (x, y, z) fulfilling the conditions $0 \leq ax + by + cz + \mu < max(|a|, |b|, |c|)$. We use, as in 2D, a discrete naive plane recognition algorithm proposed by Vittone [10] in 3D. For a given discrete plane, it provides the set of all corresponding Euclidean planes as a polyhedron in a parameter space. The standard discretization of any of these Euclidean planes contains the original discrete plane.

Consider a discrete point (x_0, y_0, z_0) and the parameter space (α, β, γ) where a point $(\alpha_0, \beta_0, \gamma_0)$ stands for the plane $\alpha_0 x + \beta_0 y + z + \gamma_0 = 0$. The discrete point corresponds to a double constraint defined by the double inequality $0 \leq \alpha x_0 + \beta y_0 + z_0 + \gamma < 1$ in the parameter space. Hence, the recognition algorithm adds the voxels one by one, reducing the solution set in the parameter space according to the corresponding double inequality. Figure 5 gives an illustration of a piece of plane and the corresponding set of solutions in the parameter space.

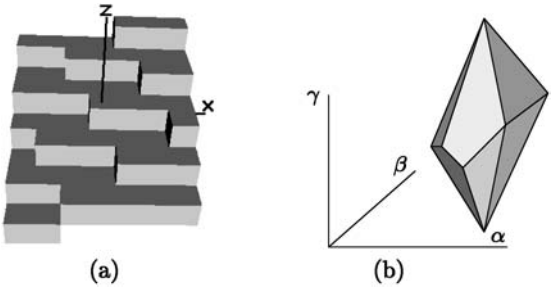

Fig. 5. A piece of the discrete naive plane $(1, 3, -5, 0)$ and the corresponding set of solutions in the parameter space.

We proposed in [11] a discrete surface segmentation based on this algorithm. We will not describe precisely this algorithm but just give some hints. The general idea is to propose a coplanarity test ensuring a "regular shape" for the recognized plane pieces. To do so, we use a local configuration of discrete planes called *tricube*. Let P be a discrete plane in the first quadrant. Then, a tricube is a set of 9 voxels of P such that the projection fo those voxels onto the plane (x, y)

is a 3×3 square: $T(i,j) = \{(x,y,z) \in P \mid i \leq x < i+3,\ j \leq y < j+3\}$. There exist 40 different tricubes [12–14] and it has been shown that any discrete plane can be built using tricubes. In our algorithm, we impose that any voxel of a plane piece belongs to a tricube of this plane, which means that at least 3 out of 8 neighbours of any voxel of a plane piece P belong to P. Moreover, we allow planes overlapping to avoid as much as possible tiny plane pieces. The pieces of planes recognized may contain holes that can be removed splitting them around the holes. Hence, the result of the algorithm is a labelling of the voxels faces with discrete plane pieces numbers.

3.2 Use of the Standard Model

After the discrete surface segmentation, we need to define discrete polygons onto this surface in order to get a polygonal reversible surface. This implies the definition of vertices and edges and thus the study of the discrete planes intersections. Unfortunately, naive planes, that were well adapted for the segmentation step, do not have the geometrical consistency properties needed to define discrete edges and vertices.

To solve this problem, we choose to swap to another model, called *standard model* that was already presented briefly for lines in the introduction.

We use the connectivity characteristics of naive and standard planes to add to the naive plane pieces, the voxels needed to get standard planes. As we do not want to add information to the initial object, we must add those voxels inside the object. If we look at the arithmetical definition of naive and standard plane, this means that we only add voxels (x,y,z) which satisfies $-(|a|+|b|+|c|-max(|a|,|b|,|c|)) \leq ax+by+cy+\mu < 0$ and which lies "under" a surface voxel of the considered plane piece. Once we have done this transformation, we need to move the set of solutions in the parameter space, in order to fit to the definition of standard plane we gave. Consider a point (a,b,c,μ) of the parameter space, solution for the piece of naive plane P. Then, the point $(a,b,c,\mu + \frac{|a|+|b|+|c|-2max(|a|,|b|,|c|)}{2})$ is a solution for the standard plane defined by the previously given transformation.

3.3 From a Discrete Surface to a Polygonal Surface

We have shown how to get a segmentation of a discrete surface into pieces of standard planes. In the following, we show how to get a polygonal surface for convex objects, and give some hints on the problems encountered for non convex objects.

First Approach for Convex Objects. For each piece of discrete plane of the segmentation, we know the whole set of solutions in the parameter space. Thus, one can choose a solution for each piece of plane, and the intersection of all those half-spaces is a polygonal approximation of the object surface. Figure 6(a) gives the result we get with such a solution for a discrete sphere of radius 20.

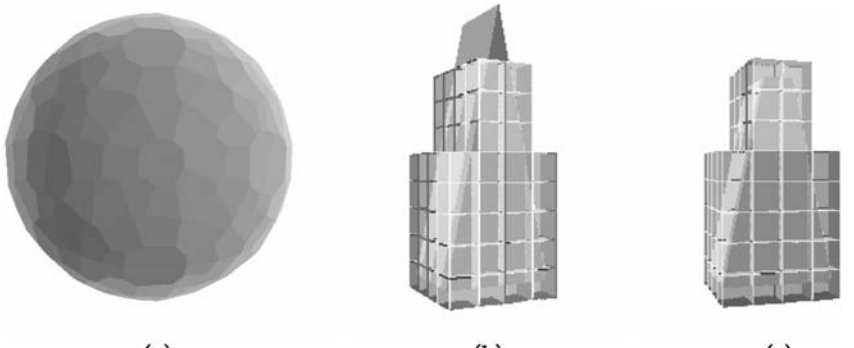

Fig. 6. Some examples on convex discrete volumes.

This solution is, however, usually not a reversible one. Figure 6(b) shows an example where some of the reconstructed edges and vertices are outside the discrete volume. Thus, the standard digitization of this polygonal surface contains more voxels than the original volume. This is exactly the same type of problems we discussed and solved by adding patches for discrete curves. In 3D, such patches are more difficult to define but a solution would be to run the discrete plane recognition algorithm on the surface places where the polygonal surface goes through the discrete object. This new plane would give the needed patch as shown on figure 6(c).

General Case and Specific Problems. Solving the reversibility problems is a second step after the construction of a polygonal surface. The half-spaces intersection method presented above can not work on non convex volumes. In order to reconstruct a polygonal surface from the segmentation for any object, we propose a contruction face by face. Moreover, this allows us to control the position of edges and vertices as we calculate them one by one. The general algorithm we propose is shown in Algorithm 1.

Algorithm 1 Construction of a polygonal surface

POLYGONAL_SURFACE(S)

1: For each piece of discrete plane of S, choose an Euclidian solution.
2: Let p be a piece of discrete plane, and $\mathcal{E}(p)$ the Euclidian solution chosen.

- track the 6-connected border of p, numbering its neighbour planes p_i, $0 \leq i < n$, $n \geq 3$;
- for all i, compute $L_i = \mathcal{E}(p) \cap \mathcal{E}(p_i)$; `[edges]`
- for all i, compute $L_i \cap L_{i+1}$. `[vertices]`

3: Repeat for each p_i, $0 \leq i < n$ until each discrete plane has been treated.

From the face by face construction, we derive that this very simple algorithm is valid for convex and non convex objects. Nevertheless, the discrete structure of the volume induces many problems. Let us look at this algorithm step after step.

The first important step is to track the border of each piece of plane in order to get an order on the plane neighbours. This step highly depends on the segmentation we get. Indeed, the segmentation algorithm we proposed allows planes overlapping and this leads to many neighbourhood relationships between discrete planes whatever neighbourhood definition we use. It is sometimes impossible to get an order on the neighbours which is consistent with the contruction of a polygonal face. We tried other strategies to get rid of this problem, the underlying idea always being the suppression of useless neighbourhood relationships. Algorithm 2 describes the solution we propose to compute the neighbourhoods.

Algorithm 2 Neighbourhood calculation

NEIGHBOURS()

1: Apply the segmentation algorithm allowing only one piece of discrete plane for each voxel: the voxels already labelled by another plane piece are added to the current plane but not labelled.
2: Compute the 4-connected border $B(p)$ of the projection of each piece of plane p;
3: Order the neighbour planes of each p tracking $B(p)$: two planes are neighbours when there exist $v_1 \in p_1$ and $v_2 \in p_2$ such that v_1 and v_2 are 18-neighbours.
4: For each plane piece, label the voxels that were added but not labelled during step 1.

With Algorithm 2, we use the minimal plane number to compute the neighbourhood relationships, but finally get the same pieces of planes as before. This method gives most of the time good neighbourhood relationships but needs to be improved because the order in the plane segmentation has an influence on the result we get.

The next and last problem of algorithm 1 occurs during the vertices calculation when one vertex should be the intersection of more than three planes. For instance, let us consider a vertex that should be the intersection of four planes p_0, p_1, p_2 and p_3. This vertex is computed four times, one for each polygon, and we denote them $\alpha_0 = p_0 \cap p_1 \cap p_2$, $\alpha_1 = p_0 \cap p_1 \cap p_3$, $\alpha_2 = p_0 \cap p_2 \cap p_3$ and $\alpha_3 = p_1 \cap p_2 \cap p_3$. Figure 7 illustrates this situation. Those four vertices are either confounded or all different. Thus, either we get one point or four. Moreover, in the case of four points, they cannot be coplanar.

In the case of four different points, we need to make some changes in order to get a surface. For instance, if at least one of the α_i is outside the discrete object, then we need to add a patch. An other case is when the four vertices belong to the same voxel: then we can delete some of those vertices or add some little triangle faces. Otherwise, the four vertices are inside the object but do not lie in the same voxel. This case may be very tricky and the most simple way to

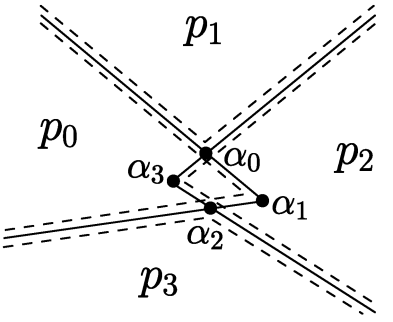

Fig. 7. The multiple vertices problem: four planes and four different vertices. The polygonal faces computed are drawn with dashed lines.

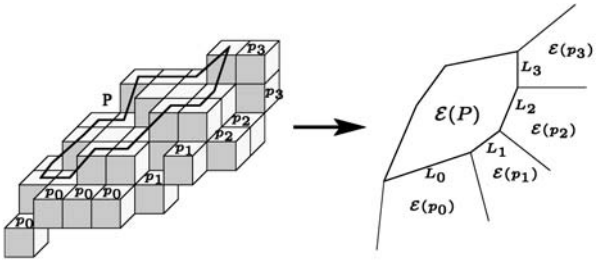

Fig. 8. Illustration of the different steps for the reconstruction of a polygon.

solve the problem is probably to try to recognize a new piece of digital plane with the voxels containing the vertices α_i.

Figure 8 illustrates the whole process described in this section: on the left, a digital piece of plane P: the 4-connected border of its projection is represented by a polygonal line, and the labels of the neighbour voxels are depicted; on the right, an illustration of the reconstructed polygon from the neighbour planes.

4 Conclusions and Future Work

In this paper we described a framework to find a polygonal curve (resp. surface in 3D) from a discrete curve (resp. surface in 3D) with an invertible method. In 2D a new algorithm has been developed to vectorize a discrete curve. We first introduce some remarkable points called discrete cusps and use the Vittone's algorithm for line recognition. The addition of patches allows to keep the Euclidean curves inside the discrete curve. Then a post-processing stage removes patches in order to give a visually correct result. In 3D, a solution has been presented for convex objects which is for the moment not reversible. We have also proposed a general algorithm to construct a polygonal surface based on the Vittone's algorithm and a face by face neighbourhood calculation. We have pointed out the main problems encountered to find neighboorhood relationships

and have proposed some solutions. In a future work improvements have to be done in order to keep the Euclidean surface inside the object even on identified particular cases.

References

1. Lorensen, W., Cline, H.: Marching cubes: a high resolution 3d surface construction algorithm. In: SIGGRAPH '87, Computer Graphics J. Volume 21., Anaheim, USA (1987) 163–169
2. Cœurjolly, D.: Algorithmique et géométrie discrète pour la caractérisation des courbes et des surfaces. PhD thesis, Université Lumière, Lyon 2, France (2002)
3. Andres, E.: Discrete linear objects in dimension n: the standard model. Graphical Models (2003) (To appear).
4. Vittone, J., Chassery, J.M.: $(n-m)$-cubes and farey nets for naive plane understanding. In: 8th Int. Workshop on Discrete Gometry for Computer Imagery. Volume 1568., Marne-la-Vallée, France (1999) 76–87
5. Vittone, J.: Caractérisation et reconnaissance de droites et de plans en géométrie discrète. PhD thesis, Université Joseph Fourier - Grenoble 1, France (1999)
6. Veelaert, P.: Geometric constructions in the digital plane. Journal of Mathematical Imaging and Vision **11** (1999) 99–118
7. Andres, E.: Defining discrete objects for polygonalization: the standard model. In A. Braquelaire, J.O.L., Vialard, A., eds.: Discrete Geometry for Computer Imagery 2002. Volume 2301 of Lecture Notes in Computer Science., Bordeaux, France, Springer (2002) 313–325
8. Lindenbaum, M., Bruckstein, A.: On recursive, $o(n)$ partitioning of a digitized curve into digital straight segments. IEEE Transactions on Pattern Analysis and Machine Intelligence **15** (1993) 949–953
9. Andres, E., Acharya, R., Sibata, C.: Discrete analytical hyperplanes. Graphical Models and Image Processing **59** (1997) 302–309
10. Vittone, J., Chassery, J.M.: Recognition of digital naive planes and polyhedrization. In: Discrete Geometry for Computer Imagery. Volume 1953 of LNCS., Springer-Verlag (2000) 296–307
11. Sivignon, I., Dupont, F., Chassery, J.M.: Decomposition of a 3d discrete object surface into discrete plane pieces. Algorithmica, Special Issue on Shapes Algorithmics (To appear)
12. Debled-Rennesson, I.: Étude et reconnaissance des droites et plans discrets. PhD thesis, Université Louis Pasteur, Strasbourg, France (1995)
13. Schramm, J.: Coplanar tricubes. In Ahronovitz, Fioro, eds.: Discrete geometry for computer imagery. Volume 1347 of LNCS., Springer-Verlag (1997) 87–98
14. Vittone, J., Chassery, J.M.: Coexistence of tricubes in digital naive plane. In: Discrete Geometry for Computer Imagery. Volume 1347 of LNCS., Springer-Verlag (1997) 99–110