



Published in Image Processing On Line on 2014-07-30.
 Submitted on 2013-02-10, accepted on 2013-05-06.
 ISSN 2105-1232 © 2014 IPOL & the authors CC-BY-NC-SA
 This article is available online with supplementary materials,
 software, datasets and online demo at
<https://doi.org/10.5201/ipol.2014.67>

Digital Level Layers for Digital Curve Decomposition and Vectorization

Laurent Provot¹, Yan Gerard², Fabien Feschet²

¹ LIMOS, Clermont-Ferrand, France (laurent.provot@udamail.fr, <http://lprovot.fr>)

² ISIT, Clermont-Ferrand, France (yan.gerard@udamail.fr, research@feschet.fr, <http://isit.u-clermont1.fr>)

Communicated by Bertrand Kerautret

Demo edited by Bertrand Kerautret

Abstract

The purpose of this paper is to present Digital Level Layers and show the motivations for working with such analytical primitives in the framework of Digital Geometry. We first compare their properties to morphological and topological counterparts, and then we explain how to recognize them and use them to decompose or vectorize digital curves and contours.

Source Code

The source code and the online demonstration are accessible at the IPOL web part of this article¹.

Keywords: digital level layers; vectorization; curve decomposition; digital geometry; digital curve recognition

1 Overview

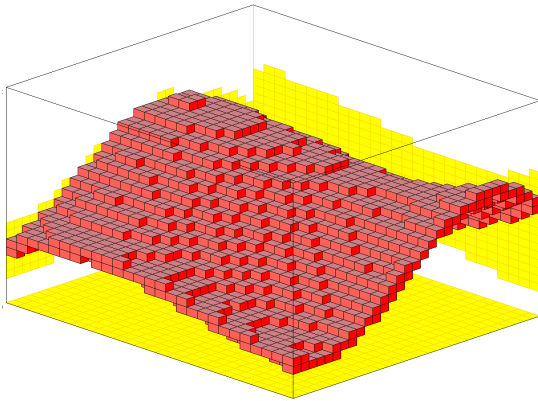
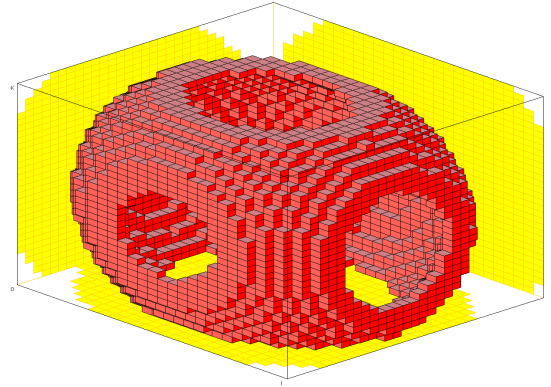
The purpose of the paper is to present the class of Digital Primitives that we call Digital Level Layers [4] and associated techniques in the framework of Digital Geometry [3]. The first part of this work is devoted to the investigation of the properties of these analytical primitives with respect to the topological and morphological approaches. Then, we focus our attention on a classical task of Digital Geometry which is the decomposition of a digital curve into pieces of digital primitives. This problem is solved for digital lines or digital circles [2, 1]. We present how we can extend the segmentation in these very small classes of digital primitives to the very large class of Digital Level Layers by using a single algorithm which is a variant of the well-known GJK collision detection algorithm [5].

¹<https://doi.org/10.5201/ipol.2014.67>

Some Examples of Digital Level Layers

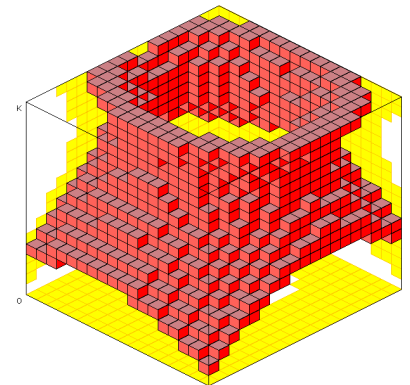
An ellipsoidal Digital Level Layer of double inequality:

$$\begin{aligned}
 -2361656972824902 \leq & 785990096201xy + 7668661544529y^2 + \\
 778342918331yz - & 225060348805176x - 226992843736355y - \\
 118704433125882z + & 7962496587660x^2 - 209524996389xz + \\
 3858710236743z^2 \leq & -2361656972824902 + 321418613523258
 \end{aligned}$$



A Digital Level Layer of double inequality:

$$\begin{aligned}
 3739455720000 \leq & -748631400x^3 - 628794398xy + 1642966215y^3 - \\
 44450164110y^2 - & 408432083400x - 124389916200y + \\
 33670461598xy + & 37939212300x^2 - 870370875xy^2 + \\
 623242620000z \leq & 3739455720000 + 910277121600
 \end{aligned}$$



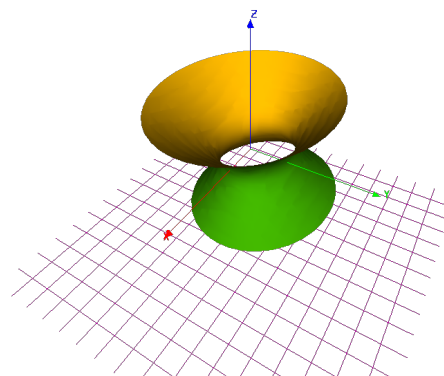
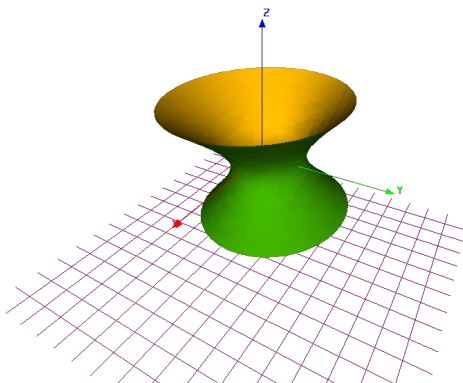
A hyperboloidal Digital Level Layer of double inequality:

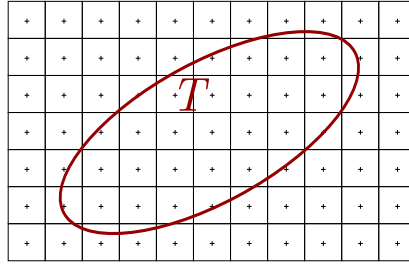
$$\begin{aligned}
 3739455720000 \leq & -748631400x^3 - 628794398x^2y + 1642966215y^3 - \\
 44450164110y^2 - & 408432083400x - 124389916200y + 33670461598xy + \\
 37939212300x^2 + & -870370875xy^2 + 623242620000z \leq \\
 3739455720000 + & 910277121600
 \end{aligned}$$

An Example of a Level Set

Two views of one of the continuous hyperboloids bounding the previous Digital Level Layer:

$$\begin{aligned}
 -233050932002 = & 9706766352x + 34545685472y - 50669609424z - 1006860048x^2 \\
 - 1419723424y^2 + & 1813925040z^2 + 346913568xz - 191023296yz
 \end{aligned}$$




 Figure 1: Our Euclidean template T : an ellipse.

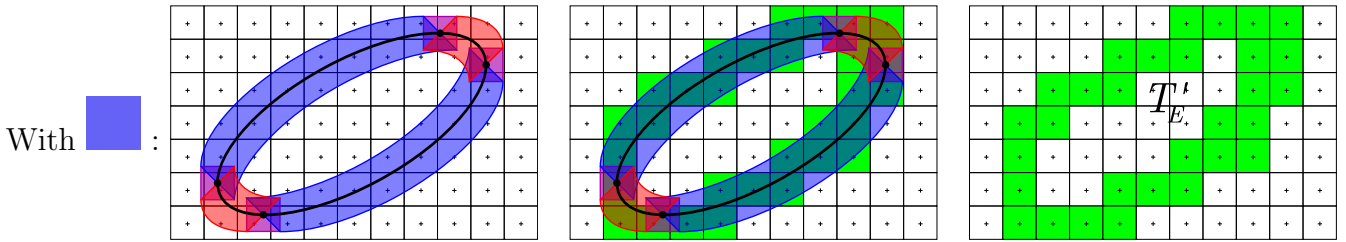
 Some possible structuring elements E :


Figure 2: **Morphological approach.** Several structuring elements E can be chosen. The most usual ones are the balls according to classical norms or a cross in the Bresenham approach. The structuring element is added to the initial shape T and we keep the integer points in the sum. The figure presents the case where the structuring element is a pixel. In this framework, T'_E is the cover of T namely the set of pixels whose intersection with T is non empty.

1.1 Morphological, Topological and Analytical Primitives

Generally speaking, Digital Level Layers (DLL) are the extension of Level Sets where the equality $f(x) = 0$ is relaxed in a double inequality $-\delta \leq f(x) \leq \delta$ (the kind of inequality can be independently chosen large or strict on each side) and where the domain of the variable x is restricted from \mathbb{R}^d to the digital grid \mathbb{Z}^d .

Why this? The main idea is to introduce digital primitives that derive from the Euclidean ones. Let us take a Euclidean template $T \subset \mathbb{R}^d$ — for instance an ellipse as depicted in Figure 1. There are at least three general ways to digitize it. Each one is lead by the idea of preserving a property of the Euclidean template T .

1. **Morphological Primitives:** The main idea is to preserve the morphology of the template T . We take a structuring element E and define the discretization T'_E of T as the intersection $T'_E = (T + E) \cap \mathbb{Z}^d$ of the Minkowski's sum $T + E$ with the grid \mathbb{Z}^d (Figure 2). The parameter of this approach is the set E . It can be a pixel, a vertical segment of length one, a ball of radius one, a square of length one in a diagonal position according to the grid, a cross — it corresponds to the Bresenham approach of digital primitives — or even more abstract shapes. According to this choice, the digital primitives T'_E defined from the Euclidean primitives T can have several topological properties.
2. **Topological Primitives:** We consider now a template T with an interior and an exterior which is a restrictive condition. In this case, we can just take the lattice points inside and outside the template and define the digital template T' as their boundary (Figure 3). There is again the choice to take the points of the exterior that are neighbors from the interior or on

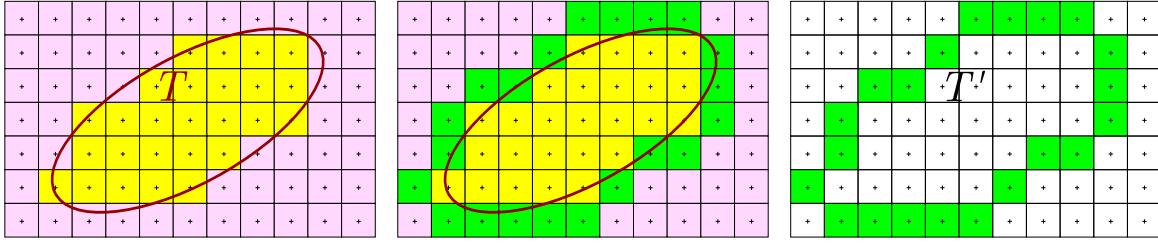


Figure 3: **Topological approach.** We just take the pixels whose center is inside and outside the ellipse. In the figure, we chose to keep the pixels outside the template which have a 4-neighbor inside T . Hence, the notion of neighborhood considered in this framework is the 4-connectivity.

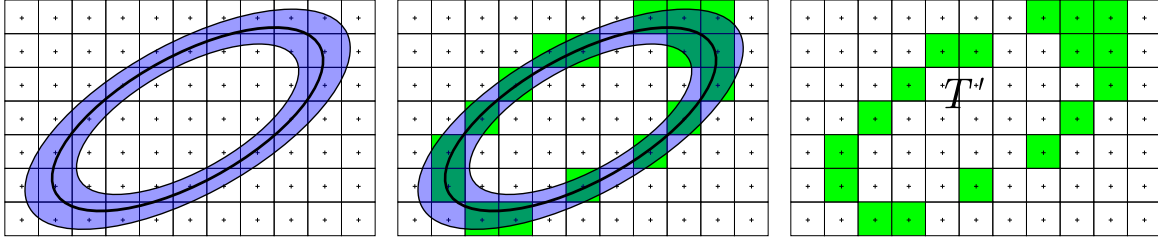


Figure 4: **Analytical approach.** We relax the equation of the ellipse $f(x) = 0$ in a double inequality $-\delta \leq f(x) \leq \delta$ for some real value of δ . It yields an elliptic annulus and we keep the pixels whose center belongs to it.

the contrary the points of the interior that are neighbors of the exterior points. The parameter of this approach is the notion of neighborhood which is used.

3. **Analytical Primitives:** We consider a template T defined as a level set $f(x) = 0$ in \mathbb{R}^d . We introduce the corresponding Digital Level Layers by the double inequality $-\delta \leq f(x) \leq \delta$ (Figure 4). The parameter of this approach is the value that should be taken for δ .

As a conclusion we have for a single template T at least three different ways to introduce a corresponding digital primitive T' .

In the easiest case of the digital straight lines, these three approaches coincide: a choice of the structuring element E corresponds to a notion of neighborhood and to a value δ depending on the normal of the line so that the primitives obtained according to the three different approaches are actually the same. But what happens for digital straight lines is no longer true for general shapes. Hence, the question is to determine which one is the good one, but this question has no clear answer. Each approach has its own advantages and drawbacks.

1.2 Advantages and Drawbacks of Each Approach

We consider the properties of each kind of primitives according to four criteria: topology, morphology, analytical characterization and the knowledge of a recognition algorithm.

1. **Topology:** We can first be interested in the preservation of the topology by going from the template T to its digitization T' . The first thing to consider is the number of connected components of T' . The morphological approach and topological approaches with good parameters preserve it while it is not necessarily the case with the analytical approach especially in the case where the gradient of f becomes large. A curve in 2D can for instance lose its connectivity as for the ellipse drawn in Figure 4. About the number of connected components of the complementary, pathological cases can be built in both cases (see Figure 5), so that this number

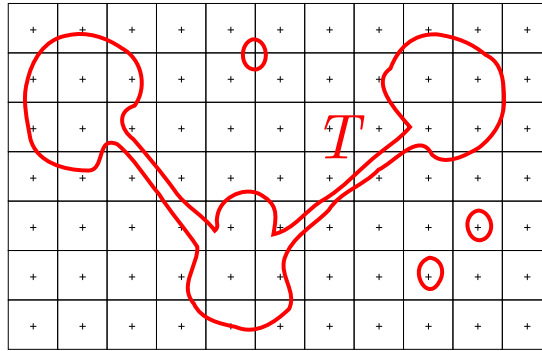


Figure 5: A pathological case: whatever the method, the number of connected components of the complementary of T is not preserved for any shapes. It may decrease or increase, depending on the position of the shape in the grid.

may increase or decrease dramatically. Whatever the approach, holes of the shape T can be forgotten or new holes can be created while they do not exist in the template.

2. **Morphology:** We can also be interested in the local thickness of the shape. With the morphological and topological approach, this morphological property is controlled by the structuring element or by the chosen notion of neighborhood. Except in pathological cases, these approaches have a good behavior. With the analytical approach, the gradient of f can again lead to problems. If it becomes very small in some points of a curve in 2D and very large in others, it is difficult to find a good compromise for the value of δ . It means that we cannot preserve the connectivity of the curve without introducing clusters of pixels in regions with small gradient. It happens for instance with the ellipse drawn in Figure 4. Hence, the analytical approach has bad morphological behavior.
3. **Analytical Characterization:** Let us consider now the property of having an analytical characterization by some equations or inequations. By definition, DLL satisfy it. This property can be discussed in detail for the morphological approach but it would require much more than some lines. We can even say briefly that, as far as we know, there exists no general analytical characterization of morphological primitives. Several inequations can be introduced by using the vertices of the convex hull of the structuring element but as soon as the template becomes complicated, it is not easy to determine which constraints are active or inactive: it depends on the normal of the shape. The result is not a single analytical characterization of the primitive. Nevertheless, we do not claim that analytical characterizations do not exist. They exist but they are most often local and it is not completely trivial to build from them a complete description of the digital primitive. With the topological approach, as far as we know, no analytical description of the digital shape has been given, even for fundamental shapes as spheres.
4. **Recognition Algorithm:** At last, we consider the existence of algorithms that allow to recognize some classes of digital primitives like, for instance, spheres or conics. We are going to discuss extensively this class of problem for the analytical approach namely in the framework of DLL in Section 2 about the DLL Recognitions Problems. In the topological case, this question is related to a problem of separation of the pixels inside the template from the pixels outside. This kind of problem has been extensively investigated in the framework of Support Vector Machines and we can notice that the main algorithm that we extend in the following (GJK) yields an efficient solution to solve it efficiently. In the case of morphological surfaces, this is still a research topic. We can nevertheless notice that, as far as we know, these algorithms deal only with rather basic shapes — lines, circles, spheres — in dimension 2 or at most 3.

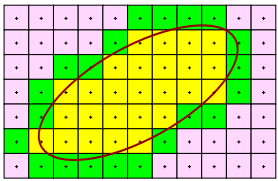
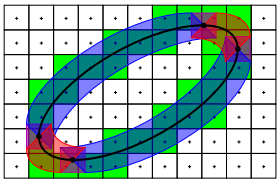
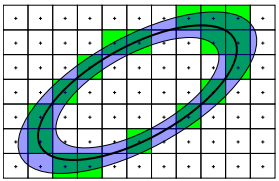
	Topology	Morphology	Analytical
Properties			
Topology	✓	✓	✗
Morphology	✓	✓	✗
Analytical characterization	✗	✗**	✓
Recognition algorithm	✓*	✗	✓

Figure 6: Advantages and drawbacks of each approach.

* The recognition problem of topological primitives is a problem of separation of two sets of points by a surface. This kind of question has been deeply investigated in the field of Support Vector Machines.

** As far as we know, most often, there is no general characterization of morphological primitives but local ones that can be used to build a complete description but it is much more complicated than a general analytical formula.

Figure 6 summarizes the strengths and weaknesses of each approach. As a conclusion, the main advantage of DLL is on the side of the analytical properties. We can notice that the DLL recognition problem is in fact a problem of vectorization of a digital shape or most often of a part of a digital shape in a DLL. The purpose of this paper is to show that this problem can be tackled very efficiently with a *single* and *efficient* algorithm that allows to solve many cases.

1.3 Curve Decomposition and Vectorization with DLL

The decomposition or segmentation of a digital curve in a list of naive digital straight segments or digital arcs, namely pieces of very basic digital primitives — lines and circles — is a very classical task in the framework of Digital Geometry [2, 1]. The purpose of this paper is to extend the set of possible classes of primitives to more complex shapes than only lines or circles. We extend it to the classes of DLL obtained with functions f in a finitely generated space of functions F .

If we consider an 8-connected curve without extra-points (if we remove an interior point, the curve is no more 8-connected) and the linear space of functions $F = \{ax + by + c \mid a, b, c \in \mathbb{R}\}$, this approach generalizes the case of the segmentation of a curve into digital straight segments [2].

If we consider the set of functions that gives circular level sets $F = \{a(x^2 + y^2) + by + cy + d \mid a, b, c, d \in \mathbb{R}\}$, we have the case of the segmentation into digital arcs [1].

One can use many other linear spaces of functions F . One can for instance deal with general conical DLL if F is the set of polynomial of degree at most 2, $F = \{ax^2 + by^2 + cxy + dx + ey + f \mid a, b, c, d, e, f \in \mathbb{R}\}$, or algebraic primitives of higher degree. This means that we have a general approach with a generic space of functions F which allows us to generalize and extend previous methods with a unique and efficient algorithm — a basic generalization of a classical algorithm called GJK [5] — for all computations.

Let us define this framework more precisely. We consider an 8-connected curve $p_j = (x_j, y_j)$ with

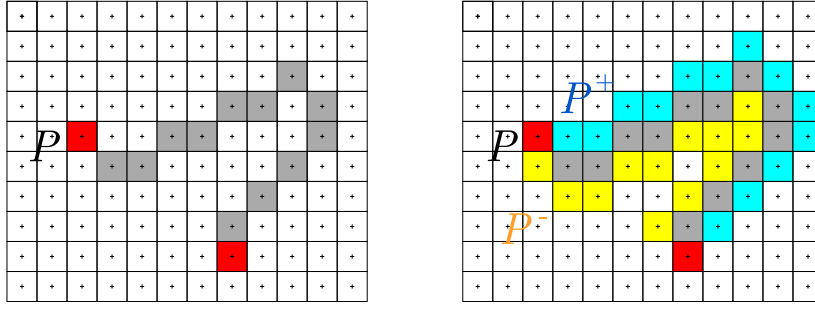


Figure 7: Path decomposition of the 4-border of an 8-connected path. In grey, the interior points of the path. We consider their 4-neighbors, more precisely, all points of \mathbb{Z}^2 whose Euclidean distance to one point p_j with $2 \leq j \leq k - 1$ is 1. This yields an 8-connected curve around the path P and we decompose it into two curves (yellow and blue) according to the position of the first and last point of the path P in their list. Notice that the first point of P is not necessarily a 4-neighbor of the next point, as it occurs on the left of the path P . This nevertheless allows us to decompose the border into two curves.

j going from 0 to m and a basis of functions $f_i : \mathbb{Z}^2 \rightarrow \mathbb{R}$ with i from 1 to n . Let us denote C the class of DLL $-\delta \leq f(x) \leq \delta$ with f in the linear space F generated by the functions f_i and δ a constant.

The task is to decompose the curve into pieces of DLL of C . It is a way to vectorize the curve because it provides an analytical representation of all of its points. The principle is to *describe* the first pixels by a DLL of our class C until such a *description* becomes no more possible. Then, we start again with the remaining pixels and a new DLL.

Let us define what we mean by the word *description*: we say that an 8-connected curve $P = \{p_j \mid 1 \leq j \leq k\}$ is described by the DLL $-\delta \leq f(x) \leq \delta$ if all points p_j verify the double inequality and if their 4-neighbors remain on both sides of it. We do not consider exactly all of the exterior neighbors from P but only the 4-neighbors of P without considering its first and last points. We decompose the 4-border of $P \setminus \{p_1, p_k\}$ into two exterior curves P^+ and P^- . If the path P is not simple, a pixel can belong to P^+ and P^- but it is not a problem in the following.

Hence we have three sets of pixels: P , P^+ and P^- , as illustrated in Figure 7. The initial path P provides the inliers of the DLL, namely the points that should be in the primitive and thus verify $-\delta \leq f(x) \leq \delta$. The sets P^+ and P^- should remain on both sides of it, namely satisfy $f(x) < -\delta$ or $f(x) > \delta$.

For explanations about this problem of recognition, see Section 2. As a conclusion, the input is a digital curve and a class of DLL given through some functions from \mathbb{Z}^2 to \mathbb{R} and the output is a decomposition of the curve into pieces of DLL as depicted in Figure 8.

The final decomposition with the end points of each piece and the equation of the corresponding DLL provides a complete vectorization of the path P . It can be used, for instance, for curve analysis, local curvature extraction and many other tasks.

2 The DLL Recognition Problems

2.1 Expression of the Classical Recognition Problem

The problem of recognition of a digital primitive is usually stated as follows:

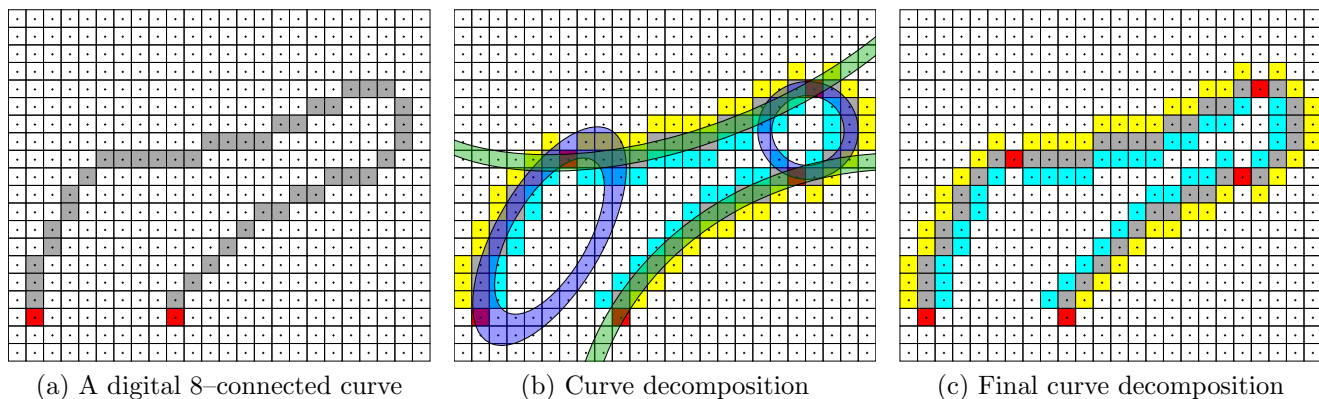


Figure 8: The curve is decomposed into pieces of conical DLL. Notice that the borders of each piece of the curve (in yellow and blue) remain outside the DLL. We have here two elliptic DLL and two hyperbolic ones.

Classical Recognition Problem of Primitives

Input: A finite subset S of points of \mathbb{Z}^d and a class C of digital primitives.

Output: Does there exist a primitive P in C such that $S \subset P$? If there exists one, provide P .

We should notice that the condition is the inclusion of S in a primitive P and not the equality $S = P$ namely the fact that S belongs directly to C . The problem is stated in these terms because the problem that occurs in practice is not to recognize complete straight lines, planes or spheres but finite pieces of these primitives.

This problem is the most classical one for the recognition of primitives in digital geometry. Let us call it the *classical* problem of recognition of digital primitives and we are going to show in the following that under some assumptions on the chosen class C of DLL, it can be solved efficiently. It remains however to precisely specify the class of digital primitives that should be chosen.

If we take as class C the whole set of DLL with any function f and any value of δ , any finite subset of \mathbb{Z}^d is by itself a DLL and the question becomes trivial. The interest of DLL is to work with functions f chosen in a given space F , for instance the polynomials of given maximal degree as it occurs for the algebraic curves, lines, conics or cubics. Even with a restriction that f belongs to a finitely generated space of functions F , the parameter δ cannot take any value. If we allow it to be as large as we want, for any finite subset S of \mathbb{Z}^d , we can increase δ so that the DLL $-\delta \leq f(x) \leq \delta$ contains S . Hence any subset S is a piece of DLL in C which is of no interest. The class C of DLL becomes interesting if a maximal value of δ is given for any function f in F . Let us denote it $\phi(f)$ (with $\phi : F \rightarrow \mathbb{R}^+$). As F is linear, the function ϕ makes sense if it is homogeneous, namely if $\phi(\lambda f) = |\lambda|\phi(f)$. Norms or absolute values of linear forms satisfy this condition and they are the most classical choices for ϕ .

As an example, we can for instance consider the framework of digital straight lines in \mathbb{Z}^2 . It corresponds to the DLL $-\delta \leq f(x) < \delta$ with functions $f(x) = a \cdot x + b$ where \cdot denotes the dot product, the normal vector a belongs to \mathbb{R}^2 and b is a real value. The class of the DLL with linear space F of functions $a \cdot x + b$ and $\delta = \frac{1}{2}\|a\|_\infty$ is the set of naive straight lines. If we consider $\delta = \frac{1}{2}\|a\|_1$, we have the set of the standard digital straight lines. If we take $\delta = \frac{1}{2}|a_1|$, we have the digital straight lines of vertical thickness equal to 1 which are functional in the first coordinate while if $\delta = \frac{1}{2}|a_2|$, we have the digital straight lines of horizontal thickness equal to 1 which are functional in the second coordinate.

This means that interesting classes of DLL with functions f in a linear space F of finite dimension are in fact defined by the function ϕ that we choose for the maximal value of δ . Let us denote by

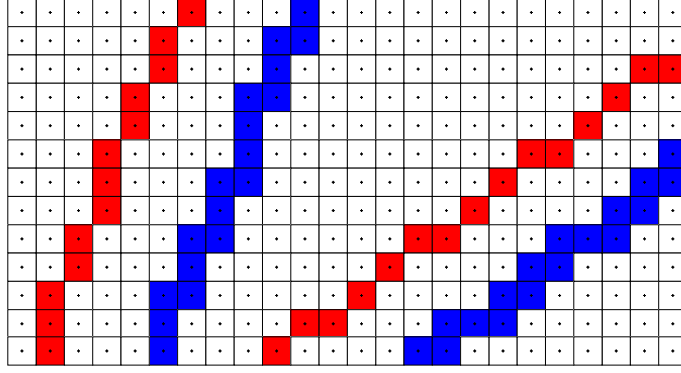


Figure 9: Digital straight lines. The double-inequalities $-\delta \leq a \cdot x + b < \delta$ with $\delta = \frac{1}{2}\|a\|_\infty$ define naive digital lines (in red) while the digital lines are said standard (in blue) if $\delta = \frac{1}{2}\|a\|_1$.

$C_{F,\phi}$ the class of DLL characterized by the double-inequalities $-\delta \leq f(x) \leq \delta$. We can notice that the class of $C_{F,\phi} \cup C_{F,\phi'}$ can also be defined by $C_{F,\max(\phi,\phi')}$ while $C_{F,\phi} \cap C_{F,\phi'}$ can also be defined by $C_{F,\min(\phi,\phi')}$. The recognition of the pieces of DLL in $C_{F,\phi}$ can be expressed as follows:

Classical Recognition Problem of DLL

Input: A finite subset S of points of \mathbb{Z}^d , a linear space of functions F of finite dimension and a homogeneous function $\phi : F \rightarrow \mathbb{R}$.

Output: Does there exist $f \in F$ such that for any x in S , we have $-\phi(f) \leq f(x) \leq \phi(f)$? If there exists one, provide f .

In practice, we choose as set F the polynomials of bounded degree, but it could be as well the space of functions generated by any finite family of functions. We choose as function ϕ the absolute value of a linear form of F . With the combination of these classes according to $C_{F,\phi} \cup C_{F,\phi'} = C_{F,\max(\phi,\phi')}$, linear forms allow in fact to work with classes where ϕ is a norm with a polyhedral ball.

2.2 Solutions for the Classical DLL Recognition

Let us take the linear space of functions F generated by some functions f_i with an index i going from 1 to n and a constant 1, and the absolute value of the linear form $\phi(\sum_{i=1}^n a_i f_i + b) = |\sum_{i=1}^n \mu_i a_i|$ as function ϕ , then the problem of recognition can be expressed as follows:

Classical Recognition Problem of DLL

Input: A finite subset S of points of \mathbb{Z}^d , n functions f_i and a vector $(\mu_i)_{1 \leq i \leq n} \in \mathbb{R}^n$.

Output: Does there exist $(a_i)_{1 \leq i \leq n}$ in \mathbb{R}^n and b in \mathbb{R} such that for any x in S , we have $-\sum_{i=1}^n \mu_i a_i \leq \sum_{i=1}^n a_i f_i(x) + b \leq \sum_{i=1}^n \mu_i a_i$? If there exists one, provide $(a_i)_{1 \leq i \leq n}$ and the constant b .

There are at least three different approaches to solve the problem: *Linear Programming*, *Collision Detection* or *Thickness Computation*.

1. **Linear Constraints (Linear Programming):** We can notice that if we have a solution $(a_i)_{1 \leq i \leq n}$ and b where the sum $\sum_{i=1}^n \mu_i a_i$ is negative, we just have to take the opposite of $(a_i)_{1 \leq i \leq n}$ and b to make it positive. This change of signs preserves all the equalities. This trick shows that the nonlinear constraints $-\sum_{i=1}^n \mu_i a_i \leq \sum_{i=1}^n a_i f_i(x) + b \leq \sum_{i=1}^n \mu_i a_i$ can be reduced to the linear constraints $-\sum_{i=1}^n \mu_i a_i \leq \sum_{i=1}^n a_i f_i(x) + b \leq \sum_{i=1}^n \mu_i a_i$ without the absolute value. In the case where the inequalities are not of the same type — weak on one side and strict on the other — we can use the same trick with the assumption that S is finite — the value b is moved from ϵ).

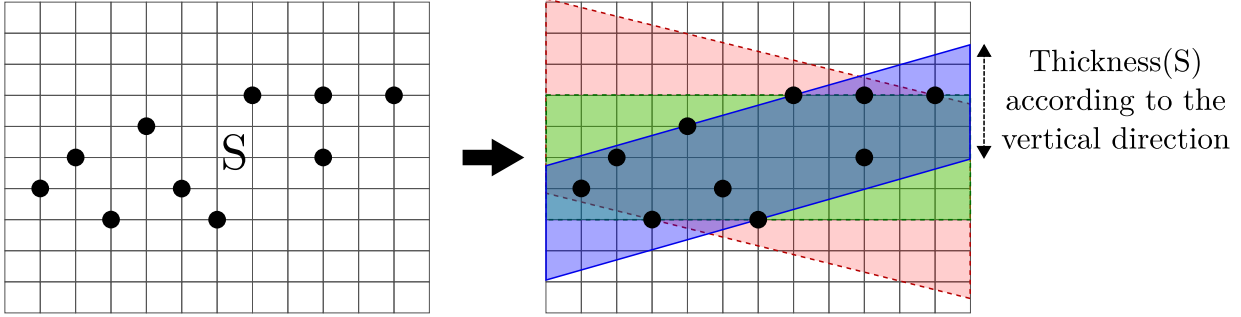


Figure 10: The notion of thickness. Given a set of points S and a direction D —vertical on the picture— we define the thickness of S in the direction D as the minimum of the thickness of the strips which contain S where the thickness of a strip is the length of its intersection with a line in direction D .

It follows that we have in fact two linear inequalities $-\sum_{i=1}^n \mu_i a_i \leq \sum_{i=1}^n a_i f_i(x) + b$ and $\sum_{i=1}^n a_i f_i(x) + b \leq \sum_{i=1}^n \mu_i a_i$ for each point of S . The unknowns are the values a_i and b . This provides a linear program with $2|S|$ linear constraints and $n + 1$ variables that can be solved by any method of the field (Simplex, Interior points, ...).

2. **Thickness Computation (Chords algorithm):** Linear Programming is an option to solve the problem but we can also tackle it with tools of computational geometry. Let us understand geometrically the meaning of the double-inequalities $-\left|\sum_{i=1}^n \mu_i a_i\right| \leq \sum_{i=1}^n a_i f_i(x) + b \leq \left|\sum_{i=1}^n \mu_i a_i\right|$. It just means that the points $(f_i(x))_{1 \leq i \leq n} \in \mathbb{R}^n$ with x in S are all in a strip of \mathbb{R}^n between the two parallel affine hyperplanes of equations $\sum_{i=1}^n a_i x_i + b = -\left|\sum_{i=1}^n \mu_i a_i\right|$ and $\sum_{i=1}^n a_i x_i + b = \left|\sum_{i=1}^n \mu_i a_i\right|$. These two parallel hyperplanes are translated one from each other by a vector $2(\mu_i)_{1 \leq i \leq n}$.

This leads us to introduce briefly the notion of thickness in a given direction. For all strips between parallel hyperplanes containing a given set S , we can compute the length of the intersection segment between the strip and a line in the prescribed direction. This length can go to infinity if the direction of the hyperplanes become parallel to the direction and it has a minimum. This minimum is the thickness of the set. If we denote $F(\cdot)$ the function $F : \mathbb{Z}^d \rightarrow \mathbb{R}^n$ defined by $F(x) = (f_i(x))_{1 \leq i \leq n}$, the question is just to determine whether or not the thickness of $F(S)$ in the direction of $(\mu_i)_{1 \leq i \leq n}$ is less or equal to $2\|(\mu_i)_{1 \leq i \leq n}\|$. If it is, the normal direction of the hyperplanes of the strip provides a possible value for a and b can be computed afterwards. This question can be solved by several algorithms of computational geometry. The first one is the chords algorithm described in [3].

3. **Collision Detection (GJK):** The principle is easy to understand. We just use an equivalence: a set S is between H and its translation $H + v$ by a vector v if and only if the hyperplane H separates S from its image by translation $S - v$. We can briefly explain the idea: S is between H and $H + v$ means (i) S is above H and (ii) below $H + v$. (ii) is equivalent with (iii) $S - v$ below $H + v - v = H$. It follows that (i) and (ii) can be replaced by (i) and (iii) which means exactly that H separates S from $S - v$. Now we apply the separation theorem of convex sets: there exists a hyperplane H that separates S from $S - v$ if and only if the intersection of their convex hull is empty. This means that the condition that the thickness in one direction in the direction of $(\mu_i)_{1 \leq i \leq n}$ is less or equal to $2\|(\mu_i)_{1 \leq i \leq n}\|$ can be checked by computing the distance between the convex hull of S and its image by translation $S + 2(\mu_i)_{1 \leq i \leq n}$. If this minimal distance is null, it means that S and $S + 2(\mu_i)_{1 \leq i \leq n}$ cannot be separated strictly one from each other by an hyperplane. If it is positive, they can be separated and the pair of the closest

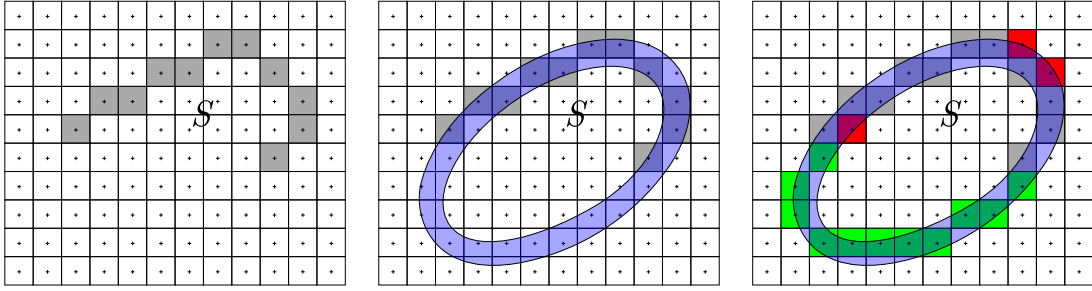


Figure 11: Why the classical recognition problem is not well-suited for DLL. On the left, we have the input set S . In the middle image, we have an elliptic DLL in a convenient class C that contains S . We notice on the right that new points, in red, have been added to the path S . This DLL shall be excluded because we don't want to add new pixels to the curve in a region where it is already well defined.

points from the two convex hulls provides a normal direction of H . Some other details should be discussed in the case where the inequalities of the DLL are large, because the information that the minimal distance is null is not sufficient to conclude. The answer depends on the fact that the intersection of S and its image by translation is reduced to points on the boundary or the intersection has a non empty interior. All these questions about collision detection can be solved by a very efficient algorithm called GJK.

2.3 Unconventional Recognition Problem

We have shown how we can solve the classical Recognition problem for DLL but there is a difficulty: the problem itself is not completely satisfactory in practice, if we work with DLL.

Why? Simply because DLL have bad morphological properties. A consequence is that we can find a DLL of C which contains P but it can introduce undesired neighbors around the points of S as shown in Figure 11.

Hence we suggest a new version of the recognition problem where the input is made of a class C of primitives, a subset S of points that should be recognized as a piece of primitive, as in the classical version but we add two new subsets of integer points in the input. We call them *up* and *down* for convenience. These two finite subsets are subsets of *outliers* while the points of S are *inliers*. They should not belong to the primitive: they should be on both sides of it.

This leads to work with primitives P which separate the space \mathbb{Z}^d into two sides. In other words, we have two sides P^- and P^+ such that the disjoint union $P \cup P^+ \cup P^-$ is \mathbb{Z}^d . Let us say that these primitives are separating and it is of course the case of the DLL with $P^- = \{x \in \mathbb{Z}^d \mid f(x) < -\delta\}$ and $P^+ = \{x \in \mathbb{Z}^d \mid f(x) > \delta\}$.

Unconventional Recognition Problem of Primitives

Input: Three finite subsets S , S^- and S^+ of \mathbb{Z}^d and a class C of separating digital primitives.

Output: Does there exist a primitive P in C such that $S \subset P$, $S^+ \subset P^+$, $S^- \subset P^-$, or $S \subset P$, $S^+ \subset P^-$, $S^- \subset P^+$?

In this framework, we do not have to extensively discuss the class of DLL that should be chosen to make the problem suitable in practice: we just choose a finitely generated space of functions described by a basis f_i with an index i going from 1 to n . We do not have to choose a maximum value of δ , because in our new version of the problem, the restricted thickness of the primitive is controlled by the outliers. It follows that we do not have to choose arbitrarily a maximal algebraic

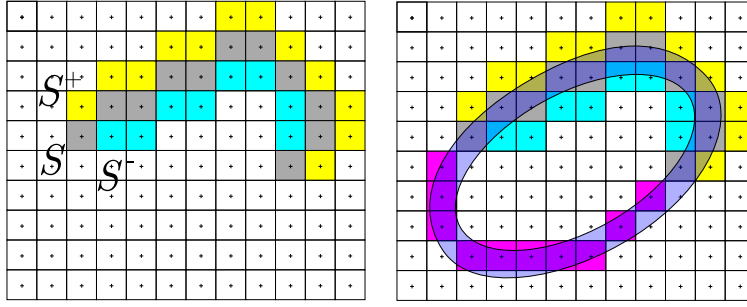


Figure 12: Unconventional recognition problem. On the left, the three points sets S , S^+ and S^- given in input. On the right, an elliptic DLL in a convenient class C that contains the inliers of S but which lets the outliers of S^+ and S^- on both of its sides. By construction, this DLL cannot introduce undesired neighbors around the points of S in the primitive.

thickness $\phi(f)$ with respect to f . It is simply replaced by the outliers. This leads us to the final expression of the problem:

Unconventional Recognition Problem of Primitives

Input: Three finite subsets S , S^- and S^+ of \mathbb{Z}^d and a linear space of functions F generated by functions $f_i : \mathbb{Z}^d \rightarrow \mathbb{R}$ with an index i going from 1 to n and a constant δ .

Output: Does there exist a function f in F such that $-\delta \leq f(x) \leq \delta$ for all points of S , with one of the complementary condition $f(x) < -\delta$ for all x in S^- , $f(x) > \delta$ for all x in S^+ or $f(x) < -\delta$ for all x in S^+ , $f(x) > \delta$ for all x in S^- ? If there exists one, provide f .

The complementary condition means that S^- and S^+ are each one on its own side of the DLL (see Figure 12).

2.4 Solutions for Unconventional DLL Recognition

Let us state the problem in a geometrical way. We denote by $a = (a_i)_{1 \leq i \leq n}$ the coefficients of f in F and b the constant. We can also introduce for any x in \mathbb{Z}^d , the point $F(x) = (f_i(x))_{1 \leq i \leq n} \in \mathbb{R}^n$. With these notations, the problem is to find a , b and δ such that:

$$\begin{cases} \forall x \in S, & -\delta \leq a \cdot F(x) + b \leq \delta \\ \forall x \in S^-, & a \cdot F(x) + b < -\delta \\ \forall x \in S^+, & a \cdot F(x) + b > +\delta \end{cases} \quad \text{or} \quad \begin{cases} \forall x \in S, & -\delta \leq a \cdot F(x) + b \leq \delta \\ \forall x \in S^+, & a \cdot F(x) + b < -\delta \\ \forall x \in S^-, & a \cdot F(x) + b > +\delta \end{cases} .$$

It means exactly that the set of points $F(S)$ is the strip between the parallel hyperplanes $a \cdot X = -b - \delta$ and $a \cdot X = -b + \delta$, while $F(S^+)$ and $F(S^-)$ are on both sides. In other words, the problem is just to search a strip of \mathbb{R}^n that contains $F(S)$ with $F(S^+)$ and $F(S^-)$ on both sides.

So we face a problem of separation of three finite sets of planes by two parallel hyperplanes. The problem of separation of two sets by an hyperplane can be solved by the collision detection algorithm GJK. With three sets, we have provided in IWCI'A'11 [6] a generalization of this well-known algorithm which allows us to separate as many sets as given by parallel hyperplanes. Let us examine its principle.

3 Algorithm

The GJK distance algorithm [5] is an iterative method for computing the minimal distance between the convex hulls of two point clouds in arbitrary dimension. In 3D, this technique is used to detect

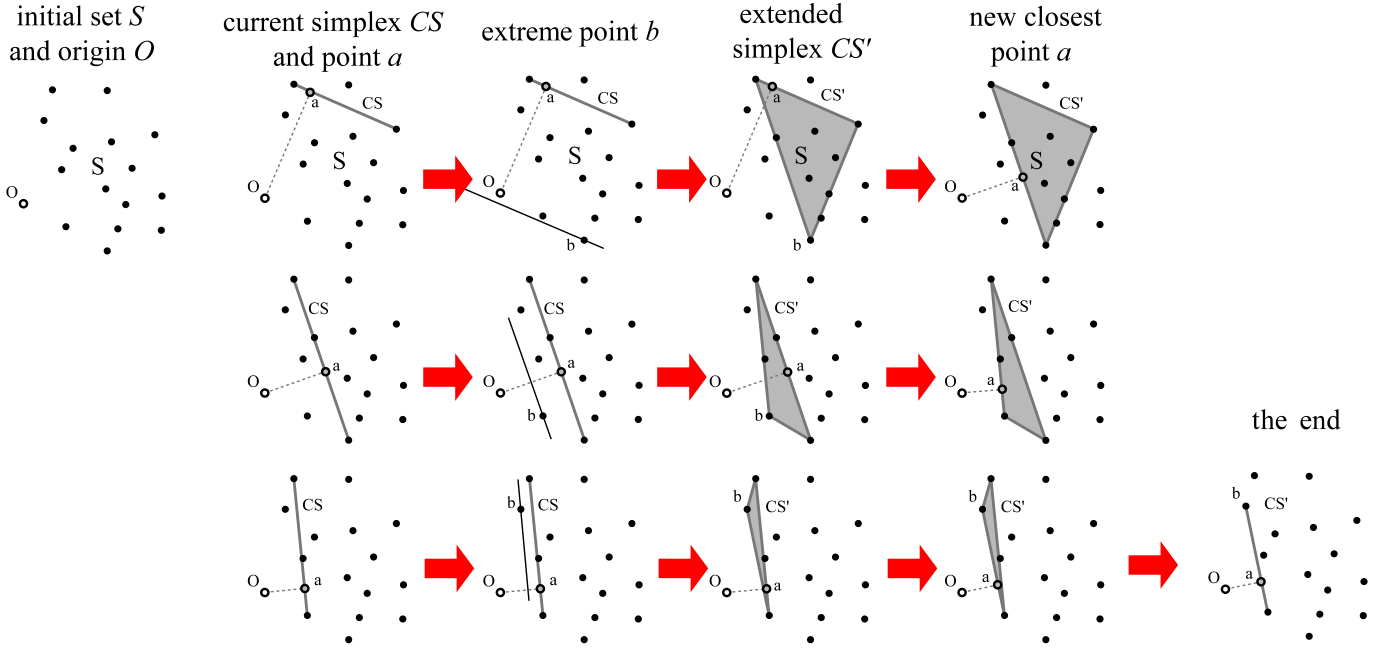


Figure 13: The main GJK steps. On the left, the initial set S and the origin O . We want to compute the distance of $\text{ConvexHull}(S)$ to the origin. In the first column, we have a current simplex CS whose vertices belong to S , and its closest point to the origin a . In the second column, we compute the extreme point b of S according to the direction a . In the third column, we add the vertex b to the simplex CS which yields the simplex CS' . In the fourth column, we compute the new point a of the simplex CS' which is the closest to the origin. We go one row down (and to the left) by taking the face of CS' which contains a as the new current simplex CS . On the bottom right, we reach the **Case 1** which ends the computation and provides the closest point a to the origin.

collisions between 3D objects in real-time applications such as video games. This algorithm can be used in order to separate two point clouds S_1 and S_2 because the closest pair of points in their convex hulls — the first point is in $\text{ConvexHull}(S_1)$, the other in $\text{ConvexHull}(S_2)$ — corresponds exactly, by duality, to the largest affine strip which separates S_1 from S_2 . If the distance is null, then the two point clouds cannot be separated.

The aim of this section is to explain our variant of GJK which allows us to separate, if it is possible, three sets S , S^+ and S^- by two parallel hyperplanes. It requires however to understand how the classical version of the algorithm works. We refer of course to the initial paper [5] for complete explanations, but we provide in the next subsection the main ideas.

3.1 The Classical GJK Algorithm

Let us consider two point clouds S_1 and S_2 of \mathbb{R}^d . We can denote by $x_1 \in \text{ConvexHull}(S_1)$ and $x_2 \in \text{ConvexHull}(S_2)$ the pair of points — or in degenerated cases, one of the pairs — that provide the minimal distance between the convex hulls of S_1 and S_2 . The first idea of GJK is to consider the difference body obtained by taking the convex hull of $S = S_2 - S_1 = \{y_2 - y_1 \mid y_1 \in S_1, y_2 \in S_2\}$. By construction, the difference $x_2 - x_1$ is the closest point to the origin in the convex hull $\text{ConvexHull}(S_2 - S_1)$.

This means that we can reduce in some way the initial problem to the particular case where one set is S and the other is reduced to the origin. In this framework, the idea is to work at each step with a current simplex CS whose vertices belong to S and whose dimension can increase and decrease all along the algorithm. In fact we start the algorithm with an initial simplex of dimension

0 (a point) and the algorithm guarantees that the dimension is always less than or equal to d . Let us denote a the closest point of CS to the origin. Two cases can arise:

- **Case 1:** All the points of S , and thus all the points of the convex hull of S , verify $a \cdot x \geq a \cdot a$. It follows that a is the closest point to the origin. This ends the computation.
- **Case 2:** The minimum of $a \cdot x$ among all the points x in S is strictly less than $a \cdot a$. We introduce a point b of S providing a minimum. We extend the current simplex CS into CS' by adding the vertex b . By construction, the distance between CS' and the origin is strictly less than the distance with CS . We compute the new closest point to the origin in CS' and update a . There are two sub-cases:
 - **Case 2.1:** The point a is in the interior of CS' . This is only possible if CS' contains the origin, which means that we are in a non-separable case (convex hulls overlap).
 - **Case 2.2:** Otherwise a is on a face of CS' . It follows that there are unnecessary vertices in CS' . We remove them by taking as the new current simplex CS the face of smallest degree of CS' containing a . And we start again until the case 1 or 2.1 occurs (see Figure 13).

We should notice that the computation of b is done directly on S_1 and S_2 — in linear time with respect to the cardinality of these two sets. Last, we have the guarantee that there is no loop because the norm of a decreases at each step.

3.2 Our GJK Variant

We have now three input sets S , S^+ and S^- . It is possible to separate them by two parallel hyperplanes if and only if it is possible to separate the origin from the union $(S^+ - S) \cup (S - S^-)$.

This means that instead of working with $S' = S_2 - S_1$ as in the classical version of the algorithm, we just have to work with the set $S' = (S^+ - S) \cup (S - S^-)$.

3.3 Complexity Analysis

The worst case complexity of the algorithm can be easily bounded. At each step, we use a current simplex whose vertices belong to $S = S_2 - S_1$ and we are in dimension d . The first computation is the search for the extreme point b which can be done in linear time in the sum of the cardinalities of S_1 and S_2 . The second computation is the one of the closest point from the origin in the extended simplex. Its computation time does not depend on the cardinalities of the input sets but only on the dimension d — we can bound it by $O(d^5)$ with a Gaussian elimination algorithm in order to compute the projections of the origin on the hyperplane of each face and check if it lies inside the face —. If we consider that d is a constant, it follows that the worst-case complexity can be bounded by the cardinality of the chords set $S = S_2 - S_1$ multiplied by the number of visited simplices. The problem is that this number of visited simplices is hard to bound with a tight value. We can give a brute-force bound which does not correspond to the experimental behavior of the algorithm. The question is how many simplices can the algorithm visit? We should first notice that the dimension may change during the process but even if it happens, we can consider that its dimension is maximal by keeping in the current structure some previous vertices which are not used for the next steps. This leads us to the idea that we can count the number of simplices which have their vertices in $S = S_2 - S_1$. Since we cannot visit twice the same simplex, because the distance to the origin decreases at each step, the number of visited vertices can be bounded by the total number of simplices of maximal dimension namely $(card(S_2)card(S_1))^{d+1}$. This provides a bound for the worst-case complexity in $(card(S_2)card(S_1))^{d+2}$ which is very high. As far as we know, there has been no result published in the literature which provides a tighter bound on the number of simplices visited by the algorithm.

4 Implementation

The following algorithm (Algorithm 1) summarizes the main part of our GJK variant. The steps are actually very similar to the classical GJK algorithm except that some functions have been adapted to deal with the third point set.

Let us detail these different steps. First we initialize the closest simplex to the origin (CS) with an arbitrary point in $S^+ - S$ — we could also choose a point in $S - S^-$, it does not really matter. In our implementation this is done through the `pickPoint(S)` function that picks the first point of the set S . We then start the main loop to incrementally converge toward the real closest simplex to the origin.

Algorithm 1: GJK variant algorithm.

```

Input  : Three finite point sets  $S$ ,  $S^+$  and  $S^-$  in  $\mathbb{Z}^d$ .
Output: The existence of an affine strip  $H(a, h, h')$  that separates  $S$  from  $S^+$  and  $S^-$ 
1 begin
  // Initialization
2   $p \leftarrow \text{pickPoint}(S)$ 
3   $p^+ \leftarrow \text{pickPoint}(S^+)$ 
4   $CS \leftarrow \{p^+ - p\}$ 
5   $a \leftarrow p^+ - p$ 
  // Main loop
6  do
7     $b \leftarrow \text{supportPoint}(a, S, S^+, S^-)$ 
8    if  $a * b = a * \text{pickPoint}(CS)$  then
9      break
10    $CS' \leftarrow \text{addVertex}(b, CS)$ 
11    $a \leftarrow \text{closestPointToOrigin}(CS')$ 
12    $CS \leftarrow \text{removeVertices}(CS', a)$ 
13 if  $\text{dimension}(CS) = d$  then
14   return false
15 else
16    $h \leftarrow -a * \text{supportPoint}(-a, S)$ 
17    $h' \leftarrow a * \text{supportPoint}(a, S)$ 
18   return true

```

The function `addVertex(b, CS)` adds the vertex b to the current simplex CS . The function `closestPointToOrigin(CS')` returns the closest point a to the origin in the simplex CS' while function `removeVertices(CS', a)` returns the minimal face of the simplex CS' containing the point a . These two functions are of course deeply related — that is why they are merged into the same function in the provided source code.

For that purpose, we use a recursive decomposition of the simplex CS' into its faces. Let us suppose that the simplex K of dimension k is a face of CS' . We compute the barycentric coordinates of the orthogonal projection of the origin with respect to this k -simplex. If all the barycentric coordinates $\lambda_{i_{1 \leq i \leq k+1}}$ are strictly positive, it means the k -simplex is the closest to the origin. If it is not the case, we go one step further in the recursion and check all its $k - 1$ -simplices that are associated to a negative λ_i coordinate². We can note that this technique is however not optimal

²If we denote K_i the vertices of K , a $k - 1$ -simplex that is associated to a λ_i coordinate is a $k - 1$ -simplex of K

because a k -simplex in \mathbb{R}^d belongs to $d - k$ simplices of higher degree so it might be checked several times. It would be worth making this an iterative procedure — but the recursive procedure has been coded and gives good results.

The most time consuming function for large instances is `supportPoint`. It computes at each step the “furthest” point b of S according to the direction a . This is the main difference with the classical GJK algorithm, as we have to deal with three sets. To do so, we do not need to explicitly compute the set S . We can indeed notice that for the set $S^+ - S$ the point b that maximizes the value $a \cdot b$ is given by $P = M^+ - m$ where M^+ and m are the points that respectively maximize the value $a \cdot x$ for all $x \in S^+$ and minimize the value $a \cdot x$ for all $x \in S$. The same holds for $S - S^-$ with $Q = M - m^-$. To have the desired b we just have to keep between P and Q the point associated to the $\min(a \cdot P, a \cdot Q)$. The computations are thus linear with respect to the cardinality of S, S^+ and S^- .

The next step is to check the validity of b . If $a * b = a * \text{pickPoint}(CS)$ it means CS was the closest simplex to the origin so we can stop the algorithm.

5 Source Code

A C++ implementation is provided and distributed under the [GPL³](#) license and accessible at the [IPOL web part of this article⁴](#).

Basic compilation using *CMake*, and usage instructions are included in the `README.txt` file. This code requires the `libpng5` library.

- Linux. You can install `libpng` with your package manager.
- Mac OSX. You can get `libpng` from the [Fink project⁶](#).
- Windows. Precompiled DLLs are available online for `libpng7`.

6 Examples

We show here some results of the decomposition of different images with three (the `StraightLine`, the `Circle` and the `Conic`) DLL models.

First a binary shape (black) and the different steps of its 8-connected digital contour decomposition (Figure 14). Inliers are depicted in grey and outliers in yellow and cyan. The red pixels show the beginning of each DLL segments.

For the next images (figures 15, 16 and 17), each DLL segment is represented with a color, alternating between red, green and blue, with the first segment of each contour drawn in yellow. One can thus clearly see the decomposition into DLL segments. In addition, the number of DLL segments is shown in parentheses.

As expected, by using digital primitives with a degree higher than one, the number of segments has been drastically reduced in the curved parts of the contours. The difference between `Circle` DLL and `Conic` DLL decompositions is not impressive although noticeable: `Conic` DLL segments are

that does not contain the vertex K_i

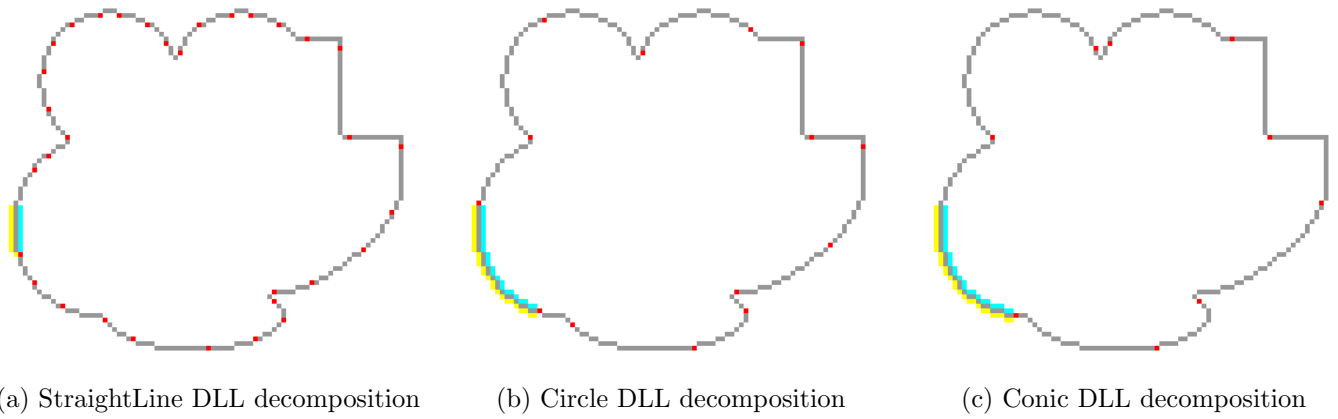
³<http://www.gnu.org/licenses/gpl.html>

⁴<https://doi.org/10.5201/ipol.2014.67>

⁵<http://www.libpng.org/pub/png/libpng.html>

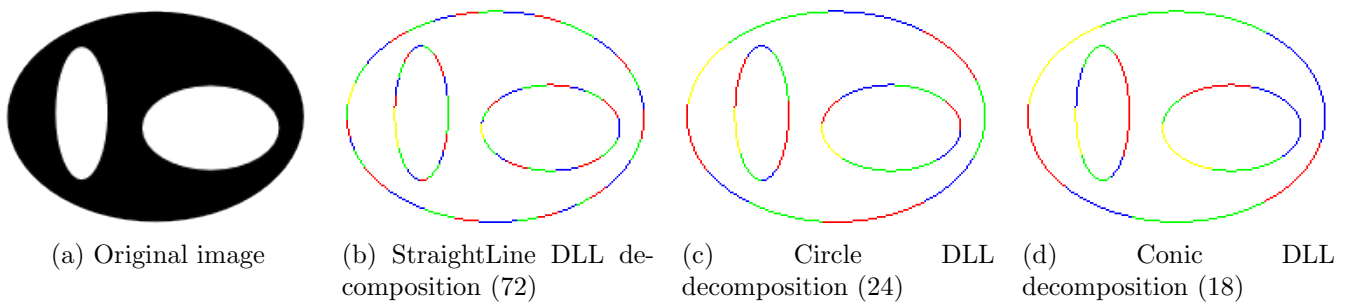
⁶<http://www.finkproject.org/>

⁷<http://gnuwin32.sourceforge.net/packages/libpng.htm>



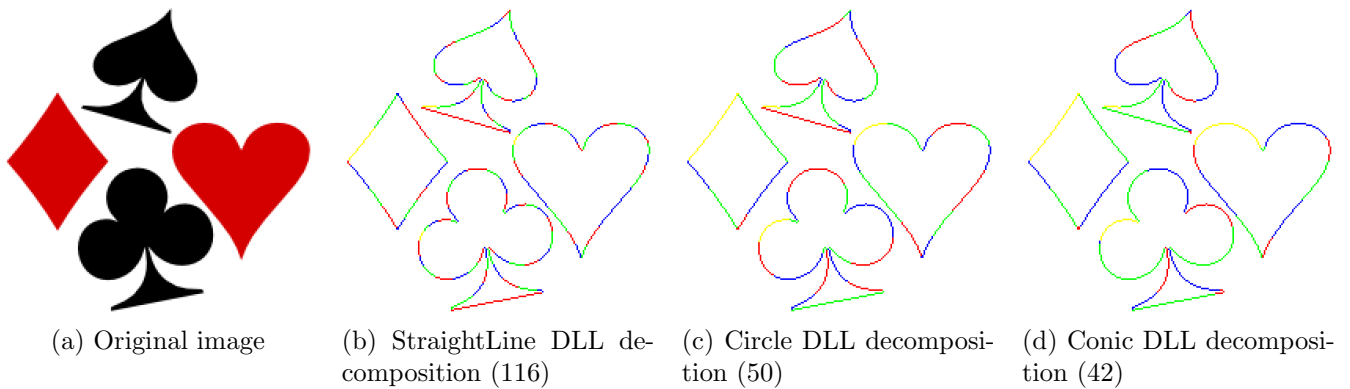
(a) StraightLine DLL decomposition (b) Circle DLL decomposition (c) Conic DLL decomposition

Figure 14: Illustration of the decompositions with the last DLL segment (represented in yellow and cyan).



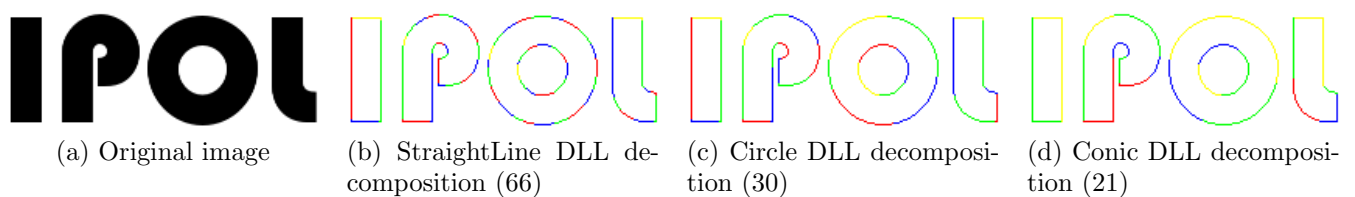
(a) Original image (b) StraightLine DLL decomposition (72) (c) Circle DLL decomposition (24) (d) Conic DLL decomposition (18)

Figure 15: Decomposition of an ellipse with two elliptic holes.



(a) Original image (b) StraightLine DLL decomposition (116) (c) Circle DLL decomposition (50) (d) Conic DLL decomposition (42)

Figure 16: Decomposition of the set of card suits.



(a) Original image (b) StraightLine DLL decomposition (66) (c) Circle DLL decomposition (30) (d) Conic DLL decomposition (21)

Figure 17: Decomposition of the IPOL logo.

a little bit longer and can even fit right angles — a typical occurrence of a sharp hyperbolic DLL segment.

Furthermore, the fact that we use 4-neighbors as outliers in our method makes it quite sensitive to imperfections — or noise — according to our DLL definition. But this is not a surprise. For instance, we can see that the digitization of the ‘O’ letter in Figure 17, although very close to a circle, is segmented into several circular DLL segments. This is because the inliers (contour points) do not totally fit inside the annulus defined by the two sets of outliers. But this is directly related to our *digital circle*’s definition. One way to prevent this type of over-segmentation would be to choose outliers farther from the inliers, but this would probably introduce an additional parameter, and the method would not be automatic anymore (for a given DLL model).

Image Credits

All images and contours are given by the authors.

References

- [1] D. COEURJOLLY, Y. GÉRARD, J.-P. REVEILLÈS, AND L. TOUGNE, *An elementary algorithm for digital arc segmentation*, Discrete Applied Mathematics, 139 (2004), pp. 31–50. <http://dx.doi.org/10.1016/j.dam.2003.08.003>.
- [2] I. DEBLED-RENNESON AND J.P. REVEILLÈS, *A linear algorithm for segmentation of digital curves*, International Journal of Pattern Recognition and Artificial Intelligence, 09 (1995), pp. 635–662. <http://dx.doi.org/10.1142/S0218001495000249>.
- [3] Y. GÉRARD, I. DEBLED-RENNESON, AND P. ZIMMERMANN, *An elementary digital plane recognition algorithm*, Discrete Applied Mathematics, 151 (2005), pp. 169–183. <http://dx.doi.org/10.1016/j.dam.2005.02.026>.
- [4] Y. GÉRARD, L. PROVOT, AND F. FESCHET, *Introduction to digital level layers*, in Discrete Geometry for Computer Imagery, vol. 6607 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 83–94. http://dx.doi.org/10.1007/978-3-642-19867-0_7.
- [5] E.G. GILBERT, D.W. JOHNSON, AND S.S. KEERTHI, *A fast procedure for computing the distance between complex objects in three-dimensional space*, IEEE Journal of Robotics and Automation, 4 (1988), pp. 193–203. <http://dx.doi.org/10.1109/56.2083>.
- [6] L. PROVOT AND Y. GÉRARD, *Recognition of digital hyperplanes and level layers with forbidden points*, in Combinatorial Image Analysis, vol. 6636 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 144–156. http://dx.doi.org/10.1007/978-3-642-21073-0_15.