Journal of Computational Mathematics
Vol. 00, Pages 000–000 (XXXX)

Shape approximation and deformation by piecewise Bezier curves and application to objects detection

Thierry Untel [1]
Jane Q. Doe [2]

[1] Inria Sophia Antipolis Méditerranée, France
E-mail address: thierry.untel@inria.fr
[2] Department of Mathematics, University of Minnesota, Minneapolis, MN, USA
E-mail address: doe082@math.umn.edu.

Abstract. This document is a short user's guide to the LaTeX class for articles submitted to *SMAI-JCM*. The abstract should generally be kept short. Important: Do not use "personal" LaTeX macros in the abstract, since the text will be extracted for further processing.

Keywords. hypersingular integral, multigrid.
Math. classification. 65N35; 15A15.

1. Introduction

2. Piecewise Bézier curves, deformations and algorithms

2.1. Bezier curves

2.1.1. *Basic definitions*

As Bezier curves are widely used in Computer Aided Geometric Design, it is common to defined them as parametric curves whose image lies in the euclidean space $\mathbb{R}^2$ or $\mathbb{R}^3$. However this definition can be extended to $\mathbb{R}^n$, for any $n \in \mathbb{N}^*$.

Given a set of $d+1$ points $P_0, ..., P_d$ of $\mathbb{R}^2$, the associated Bezier curve is defined as

$$\forall\ t \in [0;1],\ \ B([P_0, ..., P_d], t) = \sum_{i=0}^{d} P_i b_{i,d}(t)$$

where $b_{i,d}$ are the Bernstein polynomials

$$b_{i,d}(t) = \binom{d}{i} t^i (1-t)^{d-i}.$$

Notation : when we are talking about a curve, we denote it by $B([P_0, ..., P_d])$ to distinguish it from a point $B([P_0, ..., P_d], t)$. The set of Bezier curves of degree d is denoted $\mathcal{B}_d$.

The points $P_0, ..., P_d$ are called the control points or the control polygon of the curve and the integer d is the degree of the curve. One can remark that the order of the control points does matter.

Each point of the curve is a convex combination of the control points which implies that the curve lies in the convex hull of its control points. The curve does not goes through all of its control points, in general ; however it does starts at P_0 and finishes at P_d.

The first author was supported by the Inria project TEA.

2.1.2. *Interpolation*

Since a Bezier curve of degree d is defined using $d+1$ control points, one can hope to associate $d+1$ control points from a sampling of $d+1$ points on a curve. The following result shows that this is possible. But in fact, we do not have one Bézier curve of degree d but many ones. Each such curve is associated to a particular sampling of the parameter interval $[0,1]$.

Proposition 2.1. *Let $M_0,\dots,M_d \in E$, then there exists Bézier curves of degree d passing through these points.*

Lemma 2.2. *Let $t_0 = 0 < t_1 < \cdots < t_d = 1$, then there exists one and only one Bézier curve $B([P_0,...P_d])$ of degree d such that $B([P_0,...P_d],t_i) = M_i, \forall\ i \in \{0,\dots,d\}$.*

Proof. Denote M the $2\times(d+1)$ matrix built with the coordinate of M_i as i^{th} row, i.e. $M = (M_0,\dots,M_d)^t$, and denote P the $2\times(d+1)$ matrix built with the coordinate of P_i as i^{th} row, i.e. $P = (P_0,\dots,P_d)^t$. We consider the following matrix associated to $\boldsymbol{t} = \left(\ t_0,\dots,t_d\ \right)$:

$$B_{\boldsymbol{t},d} = \begin{pmatrix} b_{0,d}(0) & b_{1,d}(0) & \cdots & b_{d,d}(0) \\ b_{0,d}(t_1) & b_{1,d}(t_1) & \cdots & b_{d,d}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ b_{0,d}(1) & b_{1,d}(1) & \cdots & b_{d,d}(1) \end{pmatrix}. \tag{2.1}$$

The matrix of equation 2.1 is invertible (it is the Vandermonde matrix express in the Bernstein basis) and clearly if P is such that $B_{\boldsymbol{t},d}P = M$, then $B([P_0,\dots,P_d],t)$ give the wanted curve for the proof of the lemma. ∎

Remark that once $\boldsymbol{t}$ is know one can compute $B^{-1}_{\boldsymbol{t},d}$ once for all and that is possible to take advantage of its Vandermonde-like structure in order to improve the cost of the multiplication of a vector by $B_{\boldsymbol{t},d}$. Generally, we use a regular subdivision ($t_i = \frac{i}{d}$) but there are more suitable choices in regard of the stability of the computation.

In terms of mappings, we have the following result.

Definition 2.3. Given an integer d, we define the mapping $\Psi_d : (\mathbb{R}^2)^{d+1} \longrightarrow \mathcal{C}([0;1],\mathbb{R}^2)$ by $\Psi_d(P_0,...,P_d) = B([P_0,...,P_d])$ the Bezier curve associated to the control points $P_0,...,P_d$.

Definition 2.4. Consider a subdivision $0 = t_0 < t_1 < ... < t_d = 1$ of the interval $[0;1]$ that we will denote $\boldsymbol{t} = (t_0,...,t_d)$. We then define the sampling map $\mathcal{S}_{\boldsymbol{t}} : \mathcal{C}^0([0;1],\mathbb{R}^2) \longrightarrow (\mathbb{R}^2)^{d+1}$ by $\mathcal{S}_{\boldsymbol{t}}(\gamma) = (\gamma(t_0),...,\gamma(t_d))$.

Proposition 2.5. *The following diagram is commutative:*

$$\begin{array}{ccc} (\mathbb{R}^2)^{d+1} & \overset{\Psi_d}{\longrightarrow} & \mathcal{C}^0\left([0,1],\mathbb{R}^2\right) \\ & \underset{B_{\boldsymbol{t},d}}{\searrow} & \downarrow\ \mathcal{S}_{\boldsymbol{t}} \\ & & (\mathbb{R}^2)^{d+1} \end{array} \tag{2.2}$$

and Ψ_d is an invertible linear isomorphism between $\mathcal{B}_d = \mathrm{Im}(\Psi_d)$ and $(\mathbb{R}^2)^{d+1}$ and its inverse is $\Psi_d^{-1} = B^{-1}_{\boldsymbol{t},d} \circ \mathcal{S}_{\boldsymbol{t}}$.

Proof. Let $(P_0,...,P_d) \in (\mathbb{R}^2)^{d+1}$ and denote $(M_0,...,M_d) = \mathcal{S}_{\boldsymbol{t}}(\Psi_d(P_0,...,P_d))$. We have seen ealier that the matrix $B_{\boldsymbol{t},d}$ is invertible and its inverse is the interpolation matrix in order that

$B_{\boldsymbol{t},d}^{-1}(M_0,...,M_d) = (P_0,...,P_d)$. This shows that $(B_{\boldsymbol{t},d}^{-1} \circ \mathcal{S}_{\boldsymbol{t}}) \circ \Psi_d = id_{|(\mathbb{R}^2)^{d+1}}$.
Now let $\gamma \in \mathcal{B}_d$ a Bezier curve of degree d. The control points of γ are computed by first evaluating γ on the subdivsion $\boldsymbol{t}$ and then interpolate : $(P_0,...,P_d) = B_{\boldsymbol{t},d}^{-1}(\mathcal{S}_{\boldsymbol{t}}(\gamma))$. By construction, those control points define the Bezier curve of degree d going through the $d+1$ points $M_0,...,M_d$, meaning that $\Psi_d(P_0,...,P_d) = \gamma$ and $\Psi_d \circ (B_{\boldsymbol{t},d}^{-1} \circ \mathcal{S}_{\boldsymbol{t}}) = id_{|\mathcal{B}_d}$. ∎

2.2. **Piecewise Bezier curves**

2.2.1. *Basics on piecewise Bezier curves*

For a given Bezier curve, the more control points there are, the more complicated it is. But the degree can then be relatively high and one way to avoid this is to work with piecewise curves.

Let $N \in \mathbb{N}^*$ and a set of $N(d+1)$ control points $P_{1,0},...,P_{1,d},...,P_{N,d}$ of $\mathbb{R}^2$ such that

$$\forall\ 1 \leq i \leq N-1,\ P_{i,d} = P_{i+1,0} \tag{2.3}$$

the piecewise Bezier curve associated to these control points is defined by

$$B([P_{1,0},...,P_{N,d}],t) = B([P_{i,0},...,P_{i,d}],Nt-i+1),\ \text{if } t \in \left[\frac{i-1}{N};\frac{i}{N}\right]. \tag{2.4}$$

Notation : when there is no ambiguity, we will use the notation $\boldsymbol{P}$ for the list of control points $P_{1,0},...,P_{N,d}$ in order to lighten the formulas. Hence the piecewise Bezier curve $B([P_{1,0},...,P_{N,d}])$ can be written $B([\boldsymbol{P}])$ and a point on this curve will be denoted $B([\boldsymbol{P}],t)$.

Remark : when $t = \frac{i-1}{N}$, t belongs to both intervals $[\frac{i-2}{N};\frac{i-1}{N}]$ and $[\frac{i-1}{N};\frac{i}{N}]$ so there are two choices. In the first case, $B([\mathbf{P}],t) = B([P_{i-1,0},...,P_{i-1,d}],1) = P_{i-1,d}$ and in the second case, $B([\mathbf{P}],t) = B([P_{i,0},...,P_{i,d}],0) = P_{i,0}$ but thanks to the conditions (2.3) the two points are the same so the curve is well-defined in (2.4).

The global curve is composed by N Bezier curves of degree d : when $t \in [\frac{i-1}{N};\frac{i}{N}]$ the point lies on the i-th Bezier curve and the continuity comes from (2.3). Each Bezier curve is called a patch of the global curve. Here in this definition, since each Bezier patch has the same degree d, the curve is said to be uniform in the degree but it is possible to construct non-uniform curves with patches of different degrees. In the case where the first and the last control points are equal ($P_{1,0} = P_{N,d}$), the curve is said to be closed otherwise it is said to be open.

We denote $\mathcal{B}_{N,d}$ the set of piecewise continuous Bezier curves built with N Bezier curves of degree d. Now consider the linear map $\Psi_{N,d}$ which constructs a piecewise Bezier curve from a set of control points

$$\begin{array}{rccc} \Psi_{N,d} & : & (\mathbb{R}^2)^{N(d+1)} & \to & \mathcal{C}^0([0;1],\mathbb{R}^2) \\ & & \boldsymbol{P} = (P_{1,0},...,P_{N,d}) & \mapsto & B([\boldsymbol{P}]) \end{array}$$

We do not have that $\mathcal{B}_{N,d}$ is the image of the whole vector space $(\mathbb{R}^2)^{N(d+1)}$ because all the curves are not continuous (for example take $P_{1,d}$ different from $P_{2,0}$). However if we denote $\mathfrak{P} = \{\boldsymbol{P} \in (\mathbb{R}^2)^{N(d+1)} | P_{i,d} = P_{i+1,0}, \forall\ 1 \leq i \leq N-1\}$, we now have that $\mathcal{B}_{N,d} = \Psi_{N,d}(\mathfrak{P})$. We even have that $\Psi_{N,d}$ is an isomorphism between $\mathcal{B}_{N,d}$ and $\mathfrak{P}$: it is linear, surjective by construction and injective because if $B([\boldsymbol{P}]) = B([\boldsymbol{Q}])$, componentwise it is an equality between polynomials so they have the same coefficient, namely the control points if we write the polynomials in the Bernstein basis, hence

$\boldsymbol{P} = \boldsymbol{Q}$. It is clear that $\mathfrak{P}$ is isomorphic to $(\mathbb{R}^2)^{Nd+1}$ which is in its turn isomorphic to $B_{N,d}$.

Another way to see this is to extend the proposition 2.5 to the case of piecewise Bezier curves.

Definition 2.6. Consider a regular subdivision $\boldsymbol{t} = (t_{1,0}, ..., t_{N,d})$ of the interval $[0;1]$ such that $t_{i,0} = t_{i-1,d} = \frac{i-1}{N}$ and $t_{i,j} = t_{i,0} + \frac{j}{Nd}$. For each $1 \leq i \leq N$, we denote $\boldsymbol{t}_i = (t_{i,0}, ..., t_{i,d})$. We define the sampling map $\mathcal{S}_{\boldsymbol{t},N,d} : \mathcal{C}^0([0;1], \mathbb{R}^2) \longrightarrow (\mathbb{R}^2)^{N(d+1)}$ by $\mathcal{S}_{\boldsymbol{t},N,d}(\gamma) = (\mathcal{S}_{\boldsymbol{t}_0}(\gamma), ..., \mathcal{S}_{\boldsymbol{t}_d}(\gamma))$.

In the same way we define the evaluation matrix for the case of $(\mathbb{R}^2)^{N(d+1)}$.

Definition 2.7. Under the same hypothesis and notations of the previous definition, we define the linear evaluation map $B_{\boldsymbol{t},N,d} : (\mathbb{R}^2)^{N(d+1)} \longrightarrow (\mathbb{R}^2)^{N(d+1)}$ by $B_{\boldsymbol{t},N,d}(\mathbf{P}) = (B_{\boldsymbol{t}_1,d}(P_{1,0}, ..., P_{1,d}), ..., B_{\boldsymbol{t}_N,d}(P_{N,0}, ..., P_{N,d})$ In matrix notation, we have

$$B_{\boldsymbol{t},N,d} = \begin{pmatrix} B_{\boldsymbol{t}_1,d} & & \\ & \ddots & \\ & & B_{\boldsymbol{t}_N,d} \end{pmatrix}$$

Notation : we already introduced the notation $\mathfrak{P} = \{\boldsymbol{P} \in (\mathbb{R}^2)^{N(d+1)} | P_{i-1,d} = P_{i,0}, \forall\ 1 \leq i \leq N\}$. However the same set will be denoted as $\mathfrak{S}$ when it is used to represent the set of sample points, whereas the notation $\mathfrak{P}$ is related to the set of control points.

Proposition 2.8. *The following diagram is commutative:*

$$\begin{array}{ccc} \mathfrak{P} & \overset{\Psi_{N,d}}{\longrightarrow} & \mathcal{C}^0\left([0,1], \mathbb{R}^2\right) \\ & \underset{B_{\boldsymbol{t},N,d}}{\searrow} & \downarrow\ \mathcal{S}_{\boldsymbol{t},N,d} \\ & & \mathfrak{S} \end{array} \tag{2.5}$$

and $\Psi_{N,d}$ is an invertible linear isomorphism between $\mathcal{B}_{N,d} = \mathrm{Im}(\Psi_{N,d})$ and $\mathfrak{P}$ and its inverse is $\Psi^{-1}_{N,d} = B^{-1}_{\boldsymbol{t},N,d} \circ \mathcal{S}_{\boldsymbol{t},N,d}$.

Proof. The result is obtained by using proposition 2.5 and the fact that $\Psi_{N,d}$ is a cartesian product of isomorphisms ; the fact that the mappings are defined with $\mathfrak{P}$ and $\mathfrak{S}$ instead of the whole space $(\mathbb{R}^2)^{N(d+1)}$ insure the continuity of the curves.

■

2.2.2. *Interpolation for piecewise Bezier curves*

As in the case of a Bezier curve (not a piecewise Bezier curve), the question is to find the $N(d+1)$ control points defining the piecewise Bezier curve going through a set of given points. Let us denote $\mathbf{M} = (M_{1,0}, ..., M_{N,d})$ the points of $\mathbb{R}^2$ to interpolate which satisfy the following constraints

$$\forall\ 1 \leq i \leq N-1,\ M_{i,d} = M_{i+1,0}$$

and consider, as in definition 2.6, a regular subdivision $\boldsymbol{t} = (t_{1,0}, ..., t_{N,d})$ of the interval $[0;1]$ such that $t_{i,0} = t_{i-1,d} = \frac{i-1}{N}$ and $t_{i,j} = t_{i,0} + \frac{j}{Nd}$.

The interpolation problem is to find the control points $P_{1,0}, ..., P_{N,d}$ such that

$$\forall\ 1 \leq i \leq N,\ \forall\ 0 \leq j \leq d,\ B([\boldsymbol{P}], t_{i,j}) = M_{i,j}.$$

Using proposition 2.8, one can reformulate this interpolation problem by finding the control points $\mathbf{P} \in \mathfrak{P}$ such that $(\mathcal{S}_{\boldsymbol{t},N,d} \circ \Psi_{N,d})(\mathbf{P}) = \mathbf{M}$. And by this proposition, the solution $\mathbf{P}$ is given by $B_{\boldsymbol{t},N,d}(\mathbf{M})$. This means that it is enough to perform N interpolations for each patch.

Notation : $\mathcal{P}_i$ denotes the $d+1$ by n matrix of the control points $P_{i,0}, ..., P_{i,d}$ where the k-th row is composed by the coordinates of $P_{i,k}$. In the same way, we define the matrix $\mathcal{M}_i$ corresponding to the interpolation points $M_{i,0}, ..., M_{i,d}$.

Using these notations, the control points solution to the interpolation are computed as

$$\forall\ 1 \leq i \leq N,\ \mathcal{P}_i = \mathcal{B}^{-1}_{\boldsymbol{t}_{i,d}} \mathcal{M}_i.$$

2.2.3. *Tangent space $T\mathcal{B}_{N,D}$ and deformation of curve*

Recall that $\Psi_{N,D}$ define a linear isomorphism between the "space of control polygons" $\mathfrak{P}$ and the space of piecewize Bézier curves $\mathcal{B}_{N,d}$. We already saw that for any $\gamma \in \mathcal{B}_{N,d}$ then $\boldsymbol{P} \in \mathfrak{P}$ such that $\Psi_{N,d}(\boldsymbol{P}) = \gamma$ is given by $\chi_{\boldsymbol{t},N,d} = B^{-1}_{N,d} \circ \mathcal{S}_{\boldsymbol{t},N,d}(\gamma)$. This gives the following proposition:

Proposition 2.9. *We have that $T\Psi_{N,d} : T\mathfrak{P} \longrightarrow T\mathcal{B}_{N,d}$ is such that from any $\gamma \in \mathcal{B}_{N,d}$ we have $T\Psi^{-1}_{N,d}(\gamma) : T_\gamma \mathcal{B}_{N,d} \longrightarrow T_{\chi_{\boldsymbol{t},N,d}(\gamma)}\mathfrak{P}$ is given by $T\Psi_{N,d}(\chi_{\boldsymbol{t},N,d}(\gamma))^{-1}(\varepsilon) = B^{-1}_{N,d} \circ \mathcal{S}_N(\varepsilon) = \chi_{N,d}(\varepsilon)$ for any $\varepsilon \in T_\gamma \mathcal{B}_{N,d}$ and this is a linear isomorphism.*

An element of $\varepsilon \in T_\gamma \mathcal{B}_{N,d}$ is called a deformation curve. In fact, this proposition allows to express, given a piecewize Bézier curve and a deformation, how to deform its control polygon. This is an essential step proving that manipulating piecewize Bézier curve, it is enough to manipulate its control polygon. This is the object of the following lemma.

Lemma 2.10. *Let $\boldsymbol{P} \in \mathfrak{P}$, $\gamma = \Psi_{N,d}(\boldsymbol{P}) = B([\boldsymbol{P}]) \in \mathcal{B}_{N,d}$ and $\varepsilon \in T_\gamma \mathcal{B}_{N,d}$, then:*

i. $\varepsilon = \Psi_{N,d}(\chi_{\boldsymbol{t},N,d}(\varepsilon))$.

ii. $\gamma + \varepsilon = \Psi_{N,d}(\boldsymbol{P} + \chi_{\boldsymbol{t},N,d}(\varepsilon))$.

This lemma explain how to lift a deformation from the space of curves to the space of control polygons. The vector space structure of both the space of control polygons $\mathfrak{P}$ and of piecewize Bézier curves $\mathcal{B}_{N,d}$ allows to avoid the use of computationally difficult concept as exponential map between manifold and its tangent space and so on. This structure has also to define a simple notion of distance between two such curves.

2.3. Generalization of the Stone-Weierstrass theorem for Bezier curves in $\mathbb{R}^2$

Every curve in $\mathcal{C}^0([0;1], \mathbb{R}^2)$ cannot be represented as a Bezier curve but we can use this space $\mathcal{B}_{N,d}$ to approximate any continuous curve thanks to the Weierstrass approximation theorem.

Theorem 2.11. *Let $f \in \mathcal{C}([0;1], \mathbb{R}^2)$. For all $\epsilon > 0$, there exists $N, d \in \mathbb{N}$ and $\mathbf{P} \in \mathfrak{P}_{N,d}$ such that for all $t \in [0;1]$, $\|f(t) - B([\mathbf{P}], t)\|_\infty \leq \epsilon$.*

Proof. Let $\epsilon > 0$ and two integers N, i such that $1 \leq i \leq N$. Let us denote $f = (f_1, ..., f_n)$ the coordinates applications of f and consider an integer $1 \leq k \leq n$. We can apply the Weierstrass approximation theorem on the restriction of the function f_k to $[\frac{i-1}{N}; \frac{i}{N}]$: there exists a polynomial

application $p_{i,k}$ of degree $d_{i,k}$ such that for all $t \in [\frac{i-1}{N}; \frac{i}{N}]$, $|f_k(t) - p_{i,k}(t)| \leq \epsilon$. Written in the Bernstein basis, we have $p_{i,k}(t) = \sum_{j=0}^{d_{i,k}} p_{i,k,j} b_{j,d_{i,k}}(t)$, for all $t \in \mathbb{R}$.

Let us denote $d = \max_{\substack{1 \leq i \leq N \\ 1 \leq k \leq n}} d_{i,k}$. Now we define the control points $P_{i,j} = (p_{i,1,j}, ..., p_{i,n,j})$ with $p_{i,k,j} = 0$ if it has not been already defined, in order to have that for all $t \in [\frac{i-1}{N}; \frac{i}{N}]$, $B([P], t) = (p_{i,1}(t), ..., p_{i,n}(t))$. This Bezier curve is such that for all $t \in [0; 1]$, $\|f(t) - B([\mathbf{P}], t)\|_\infty \leq \epsilon$ by construction of the control points. ■

3. Topology change : the flip algorithm

Hypothesis :

- steps of deformation are smaller than the Bézier patches
- the patches are not self-intersecting patches

Notes :

- parler des structures de données (tableau des CP avec numéros de composantes et numéro de patch
- référence : *Computational geometry in C*, Joseph O'Rourke, pour l'intersection segment-segment
- référence : [*An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles*, Jon Louis Bentley, Derick Wood] + [*Fast software for box intersections*, Afra Zomorodian] pour trouver les intersections des bounding boxes
- les intersections des boites englobantes sont strictes

3.1. Overview of the flip algorithm

The flip algorithm acts on two cubic control polygons $\mathcal{P} = \{P_0, P_1, P_2, P_3\}$ and $\mathcal{Q} = \{Q_0, Q_1, Q_2, Q_3\}$ that intersect each other. The algorithm consists in rearranging the control points and produce two new control polygons of the form $\{P_0, ?, ?, Q_3\}$ and $\{Q_0, ?, ?, P_3\}$ that do not intersect. The question marks represents control points P_1,P_2, Q_1 and Q_2 which will be selected by the algorithm after phases of projection and sweeping along particular lines. But before going into details of how the algorithm works, one must say that there are two cases. The first case is when the two initial control polygons $\mathcal{P}$ and $\mathcal{Q}$ belong to the same closed piecewise Bézier curve and thus the flip algorithm will produce two independant components (see figure 1). The second case is the opposite situation : the control polygons $\mathcal{P}$ and $\mathcal{Q}$ are not on the same component and the flip algorithm will glue their respective components into one (see figure 2).

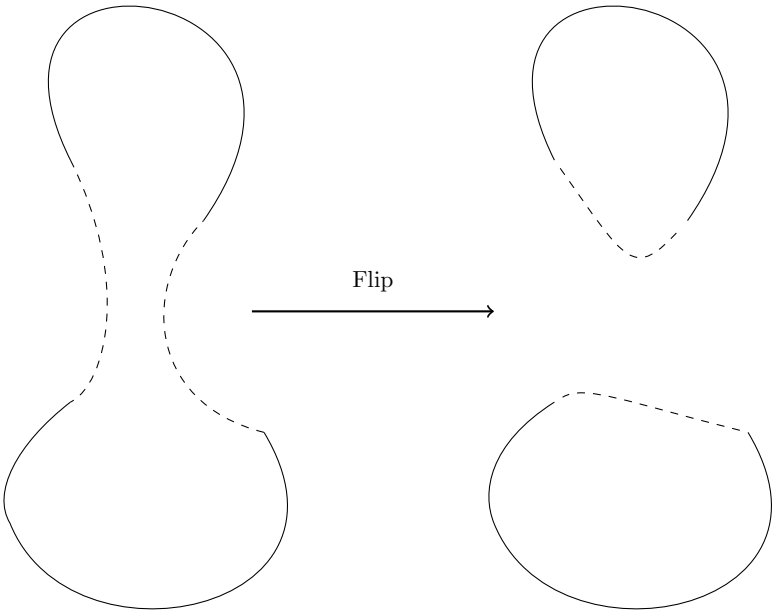


FIGURE 1. First case : one connected component is split in two components.

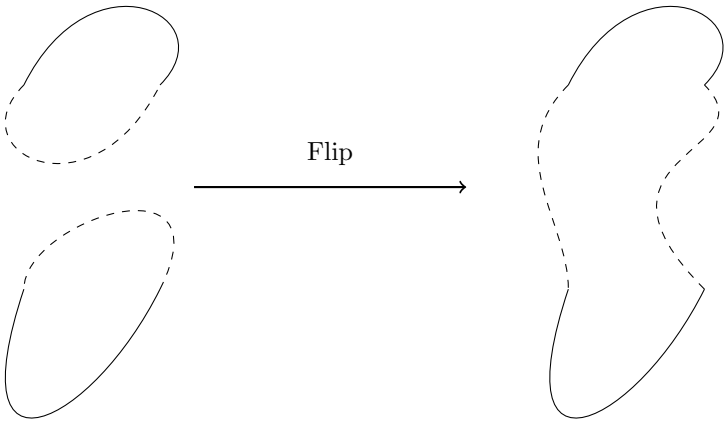


FIGURE 2. Second case : two connected components are merged together.

3.2. **A word about orientation**

One thing to be careful of is the orientation of the parametrizations or equivalently the order of the control points in the data structure storing them. This is not something to watch when being in the first case but it is crucial for the second case. Indeed, since the flip algorithm will produce two

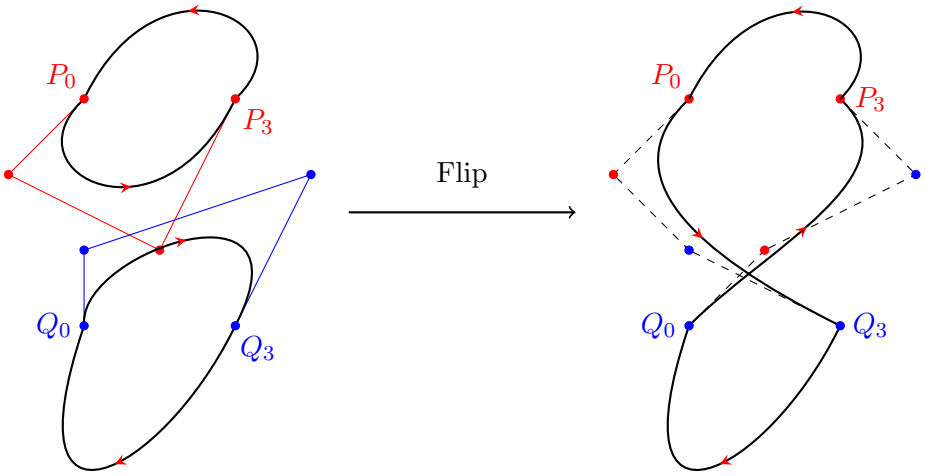


FIGURE 3. When the two components have opposite orientations, joining P_0 with Q_3 and Q_0 with P_3 will result in having two intersecting patches after the flip.

control polygons of the form $\{P_0, ?, ?, Q_3\}$ and $\{Q_0, ?, ?, P_3\}$, depending on the initial orientation of each component, the two new control polygons can still intersect. As the figure 3 shows, if the two components have opposite orientations, the resulting shape will have at least one self-intersection. However, if the two components have the same orientation, the problem will be avoided, as shown in figure 4. So before starting the flip, if it is the case of merging two components, one must check the orientations of each components and simply reverse one of them if necessary. Now concerning the orientation of the resulting curves, in both cases the orientation of the new control polygons are compatible with the untouched parts of the curve (see figures 4 and 5).

3.3. **The algorithm in details**

We can now present the flip algorithm precisely and assume that the two control polygons $\mathcal{P} = \{P_0, P_1, P_2, P_3\}$ and $\mathcal{Q} = \{Q_0, Q_1, Q_2, Q_3\}$ have the good orientations. Being in case 1 or 2 does not matter here, since the core of the algorithm does not make any difference between the two. We will be in the situation described by the figure 6 : only two control polygons intersects. This is not the only case of intersection, indeed, it can occur that three control polygons intersect like in figure 9. We will see later that there is not much work than the case with two control polygons.

As mentionned in the beginning of this section, the two new control polygons are of the form $\{P_0, ?, ?, Q_3\}$ and $\{Q_0, ?, ?, P_3\}$, so the flip algorithm must decide where to put the remaining control points P_1,P_2, Q_1 and Q_2 in the new polygons. The first step is to compute their projections on the line carried by the vector $\vec{P_0P_3}$. The two points having their projections closest to P_0 will belong to the new control polygon $\{P_0, ?, ?, Q_3\}$ and the other two to $\{Q_0, ?, ?, P_3\}$. In the situation depicted by the figure 6, Q_2 and P_1 will be selected to form the new control polygon with P_0 and Q_3 and P_2 and Q_1 will be part of the other control polygon with Q_0 and P_3. Note that doing so, one can find a line, perpendicular to the line where the points are projected, that separates each four control points, insuring the new control polygons will not intersect each other. Now to decide whether the

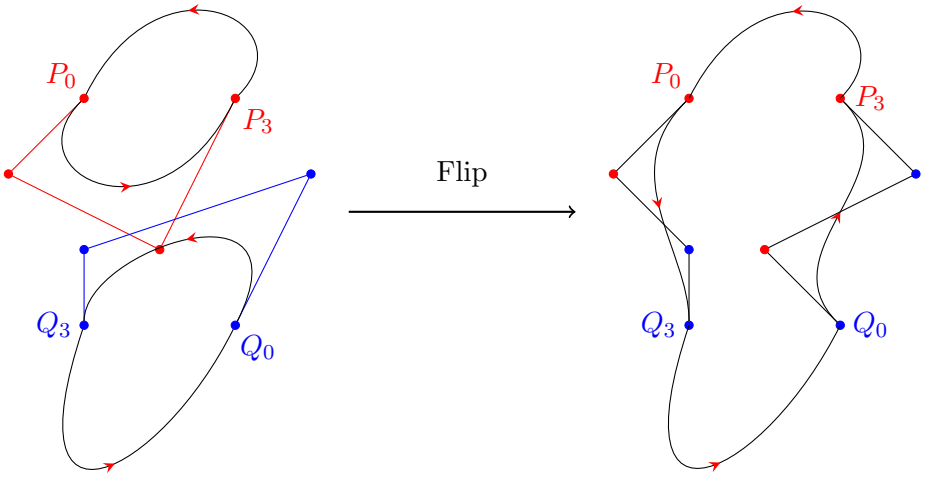


FIGURE 4. The two components have the same orientation, so that joining P_0 with Q_3 and Q_0 with P_3 will not produce two intersecting patches.

control polygon will be $\{P_0, Q_2, P_1, Q_3\}$ or $\{P_0, P_1, Q_2, Q_3\}$, the algorithm will perform another phase of projection on the line carried by $P_0\vec{Q}_3$ this time : the point whose projection is the closest to P_0 will follow P_0. Continuing with the example, figure 7 shows that the point Q_2 has its projection the closest to P_0 so that the new patch is $\{P_0, Q_2, P_1, Q_3\}$. The same will be done for the other polygon but using the line carried by $Q_0\vec{P}_3$. Figure 8 shows the patches before and after the flip.

We can now explain how to deal with the case of three patches, like it is in figure 9 : two consecutive patches intersect the same patch. The goal is to create two control polygons $\{P_0, ?, ?, R_3\}$ and $\{Q_0, ?, ?, P_3\}$ and the ? marks being points taken from the set $\{Q_1, Q_2, Q_3 = R_0, R_1, R_2\}$. So from three patches we will produce only two. The method is the same as in the situation with two patches : we project the five points Q_1, Q_2, Q_3, R_1 and R_2 on the line carried by $P_0\vec{P}_3$ and the two points whose projections are the closest to P_0 will belong to the control polygon $\{P_0, ?, ?, R_3\}$ and the two points whose projections are the closest to P_3 will belong to the control polygon $\{Q_0, ?, ?, P_3\}$. Then the second phase of projection is done along the line carried by $P_0\vec{R}_3$ (respectively $Q_0\vec{P}_3$) to decide the order of the two points in $\{P_0, ?, ?, R_3\}$ (resp. $\{Q_0, ?, ?, P_3\}$). In the example of figure 9, the new control polygon would be $\{P_0, R_1, R_2, R_3\}$ and $\{Q_0, Q_1, P_2, P_3\}$.

3.4. **The split function**

In order to describe more precisely the shapes, the algorithm dynamically increase the number of patches using the split function. At each iteration of the optimization loop, the length of each patch is checked and everytime one is above a certain threshold, the patch is split in two. If $\mathbf{P} = \{P_0, P_1, P_2, P_3\}$ has a size above the threshold, the split function will compute two patches $\mathbf{Q} = \{Q_0, Q_1, Q_2, Q_3\}$ and $\mathbf{R} = \{R_0, R_1, R_2, R_3\}$ by interpolation such that $\mathbf{Q}$ interpolates the first half of the curve and $\mathbf{R}$ the second half, as shown by picture 10. Note that splitting a patch in two does not change the curve,

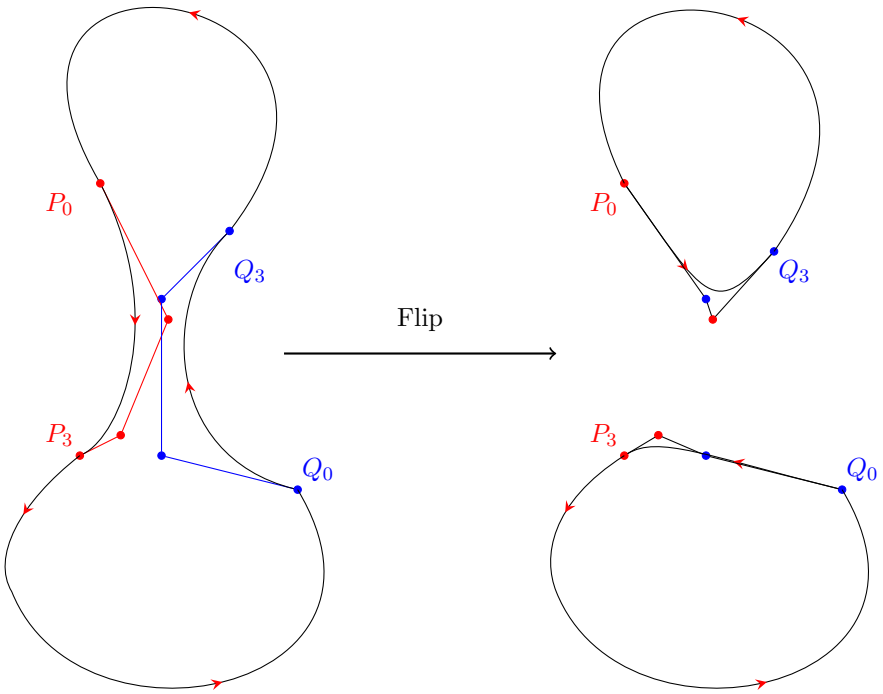


FIGURE 5. In the first case, we can see that joining P_0 with Q_3 and Q_0 with P_3 will produce two connected components whose orientations are both compatible with the untouched patches.

hence the shape will remain the same before and after the splits.

3.5. **The shape optimization algorithm using flip**

Now that the flip method has been explained, we will present the general form of the shape optimization algorithm. The shapes are represented by the control points of the boundary curves and we will denote P the data structure containing the control points of all the components. To each point in P we attach two numbers : the component number k and the patch number i to which it belongs. The number of components of the current shape will be denoted by M.

The first step in the algorithm is to compute the shape gradient G for the current shape defined by the control points P. We modify the control points by following the direction G, that is, we update $P \leftarrow P - \alpha G$ where α is the step. After this, we enter in the part where we must find which patches must be flipped, that is, those who intersect each other. We will first compute the bounding boxes of all the patches and compute the pairs of intersecting boxes and then investigate among these pairs if the patches intersect. The boxes are axis-aligned bounding boxes and consist in four real numbers denoted $left$, $right$, top and $bottom$. To each box, we attach the component number k and the patch number i. Each intersection is stored in a list as a pair $((k,i),(k^{'},i^{'}))$, meaning that the box of the

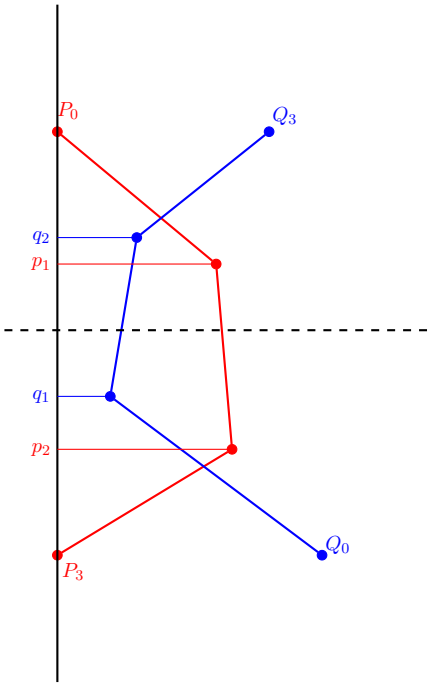


FIGURE 6. Projections of P_1, P_2, Q_1 and Q_2 along the line carried by $P_0\vec{P}_3$.

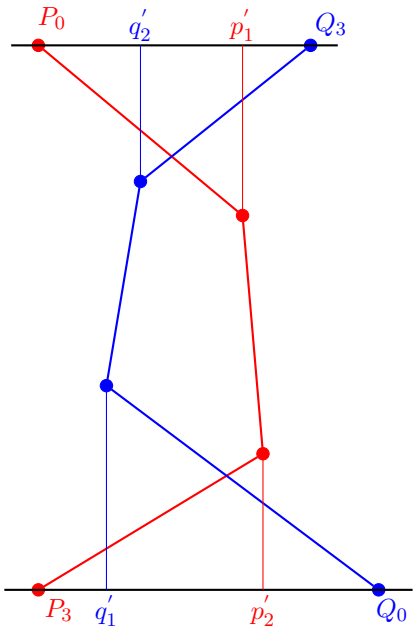


FIGURE 7. Projections of Q_2 and P_1 along the line carried by $P_0\vec{Q}_3$ and Q_1 and P_2 along the line carried by $Q_0\vec{P}_3$. The new control polygons are $\{P_0, Q_2, P_1, Q_3\}$ and $\{Q_0, P_2, Q_1, P_3\}$.

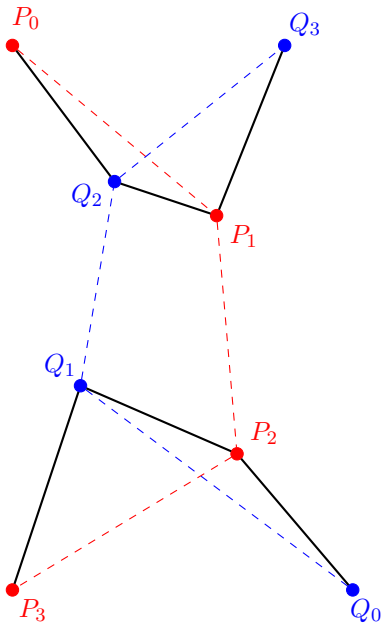


Figure 8. The new control polygons are $\{P_0, Q_2, P_1, Q_3\}$ and $\{Q_0, P_2, Q_1, P_3\}$.

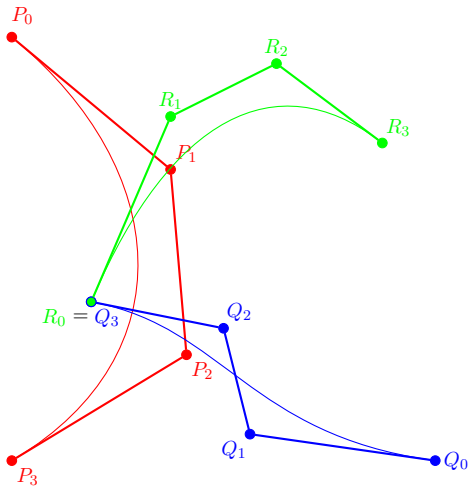


Figure 9. Case with three control polygons : two consectutive control polygons (on the right) intersect the same control polygon (on the left).

patch i of the connected component k intersect the box of the patch i' of the connected component k'.

Then we must check if these intersections occur between patches : we go through this list and create another list L of the same type. There will be as many flips as there are pairs in the new list L. To check if two patches intersect each other, since the patches are cubic patches, we will check 9 segment-segment intersections. The method to check these intersections is taken from the book *Computational geometry in C* : two segments intersect each other if for each segment, its endpoints are on different

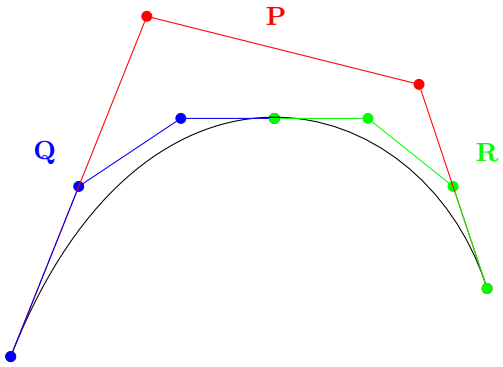


FIGURE 10. The split function increase the number of patches without changing the shape : $B([\mathbf{P}],[0,\frac{1}{2}]) = B([\mathbf{Q}],[0,1])$ and $B([\mathbf{P}],[\frac{1}{2},1]) = B([\mathbf{R}],[0,1])$.

side of the other segment.

Given the list L containing the pairs of intersecting patches, we will perform a flip. The pair $((k,i),(k^{'},i^{'}))$ is given to the flip function and the data structure P will be modified at the end of the flip. Now there is a problem : since the flip will either add a new component or glue two components, the component numbers and patch numbers associated to each control points will be modified. Let us consider an example where the current shape consists in $M = 5$ connected components and that in the third component, the patches 4 and 7 intersect each other. Then the pair $((3,4),(3,7))$ is an element of L and the flip function will be called. The figure 11 illustrates this example. The flip has created a sixth component and modified the third which has now five patches. The patch numbers have changed in the following way : $(3,8)$ became $(3,5)$, $(3,5)$ became $(6,1)$, $(3,6)$ became $(6,2)$ and the patches $(3,4)$ and $(3,7)$ have new control points and are labelled respectively as patches $(3,5)$ and $(6,3)$. Now these modification must be applied to the elements of the list L, so that the next pairs encountered in L make sense with the modified shape. The renumbering is also done when the flip merges two components.

Algorithm 1 Shape optimization algorithm with flip

```
while STOPPING CRITERIUM NOT SATISFIED do
    G ← Compute shape gradient for the shape defined by the control points P
    P ← P − αG
    Compute the bounding boxes for all the patches
    Compute the list of all the pairs ((k,i),(k',i')) of intersecting bounding boxes
    From the previous list, compute the list L of all the pairs ((k,i),(k',i')) of intersecting patches
    for each pair ((k,i),(k',i')) in L do
        P ← Flip(P,k,i,k',i')
        Update L
    end for
    Check the length of the patches and perform a split if needed
end while
```

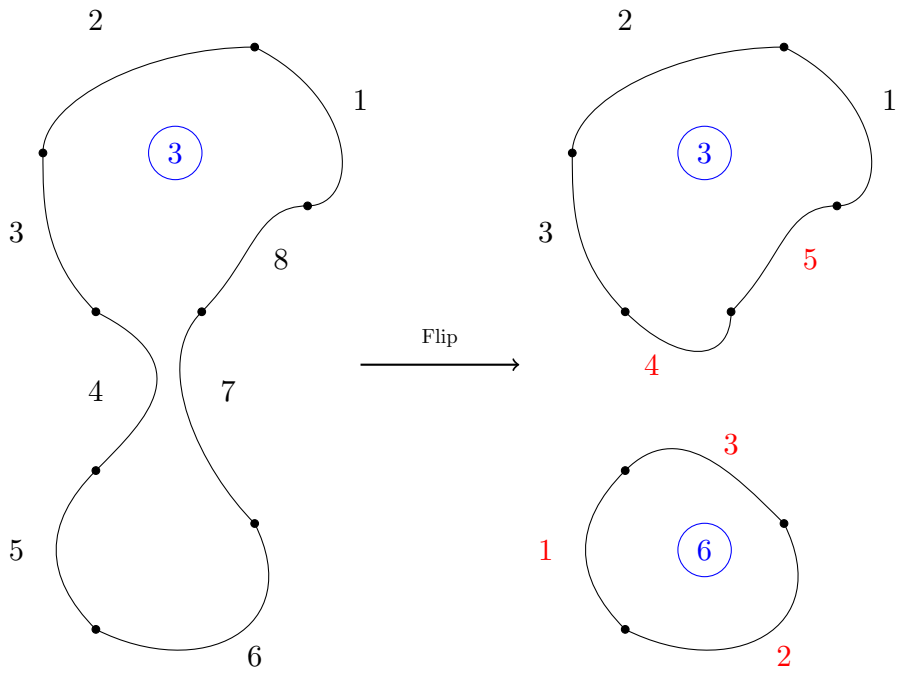


Figure 11. After a flip, the patch numbers are modified : the component number 3 gives another component whose number is $M + 1 = 6$.

3.6. **Complexity of the algorithm**

In this section, we want to investigate the complexity of the algorithm and show that, under some assumption, the operations needed for the flip run in $\mathcal{O}(n \log n)$ time, where n is number of patches. We will make the assumption that the steps of the optimization algorithm are negligible compared to the size of the patches, so that the number of intersecting patches does not depend on n. Looking at the algorithm 3.5, we will compute the cost of the operations from line 4 to 11.

Starting with line 4, computing the bounding boxes for each patch, the cost is clearly in $\mathcal{O}(n)$. The operation in line 5 is done in the following way. We sort all the $left$ and $right$ value of all bounding boxes, so the cost is in $\mathcal{O}(n \log n)$. Then we go through this sorted list and each time we encounter an intersection (the intersection is only in the x direction), we check if the intersection appears in the y direction too. The cost of the total operation comes from the sort, so that the step 5 of the algorithm is in $\mathcal{O}(n \log n)$. Step 6 goes through each pair in the previous list and check the 9 segment-segment intersections : the cost is linear in the size of the list. The **for** loop has a constant cost under the assumption of small steps as we already mentionned. The last operation of the algorithm consists in searching which patches need to be split and split them. Each split consists in sampling and interpolating cubic patches, so the cost of one split is independent of n. We must go through the list of patches, so the total cost is in $\mathcal{O}(n)$.

To conclude, the cost of all the operations related to the flip is in $\mathcal{O}(n \log n)$. The main cost of the algorithm is concentrated in the computation of the shape gradient by solving a PDE.

4. Objects detection via shape optimization

5. Free form approach and numerical result

5.1. Shape optimization using Bézier curves

Given the cost function $\mathcal{J}$ and its gradient $\nabla\mathcal{J}$ we can compute a descent direction for the control points as it follows. The set of admissible shapes will be the closed non-self intersecting Bézier curves denoted $\mathcal{B}^c_{N,d} = \{\gamma \in \mathcal{B}_{N,d}/\gamma(t) = \gamma(s),\ t \neq s \Leftrightarrow (t = 0 \text{ and } s = 1) \text{ or } (t = 1 \text{ and } s = 0)\}$. The associated vector space of control points is

$$H_{N,d} = \{(P_{i,j})_{i=1..N,j=0..d} \in ((\mathbb{R}^2)^{d+1})^N / P_{1,0} = P_{N,d}\}$$

Let's consider a curve $\gamma \in \mathcal{B}^c_{N,d}([0;1],\mathbb{R}^2)$ and a shape gradient

$$\begin{array}{rccc} \nabla\mathcal{J}(\gamma) & : & \gamma([0;1]) & \to & \mathbb{R}^2 \\ & & M & \mapsto & \nabla\mathcal{J}(\gamma)(M) \in \mathbb{R}^2 \end{array}$$

For each point M of the curve γ, the shape gradient associates a direction of deformation. We call the sampling deformation of the curve the map $\mathcal{T}_{N,d}(\mathcal{J})$ defined by

$$\mathcal{T}_{N,d}(\mathcal{J})(\gamma) = (\nabla\mathcal{J}(\gamma)(S_{t,N,d}(\gamma)))_{i=1..N,j=0..d} \in (\mathbb{R}^2)^{N(d+1)}$$

First we sample the curve thanks to $S_{t,N,d}$ in $N(d+1)$ points and we apply the shape gradient on them to obtain the $N(d+1)$ directions of deformation in the space $\mathbb{R}^2$. Then, by composing with $B^{-1}_{N,d}$, we obtain a vector field V on the space of control points $(\mathbb{R}^2)^{N(d+1)}$:

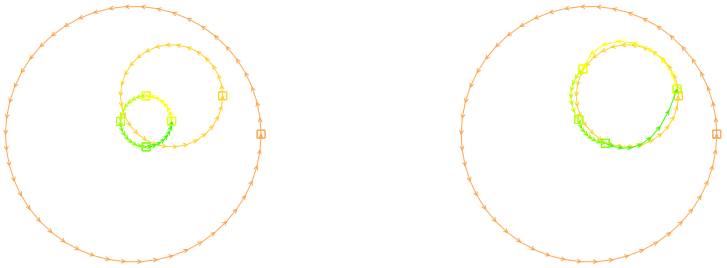
$$\begin{array}{rccc} V & : & \mathcal{B}^c_{N,d} & \to & (\mathbb{R}^2)^{N(d+1)} \\ & & \gamma & \mapsto & \mathcal{B}^{-1}_{N,d}(\mathcal{T}_N(\mathcal{J})(\gamma)) \end{array}$$

$V(\gamma)$ is the descent direction, the direction of the shape gradient, expressed for the control points : if we denote P the vector of control points defining the current shape and $\alpha \in \mathbb{R}$ a given step, at each iteration the operation $P \leftarrow P - \alpha V(\gamma)$ is performed to modify the shape.

5.2. Numerical results

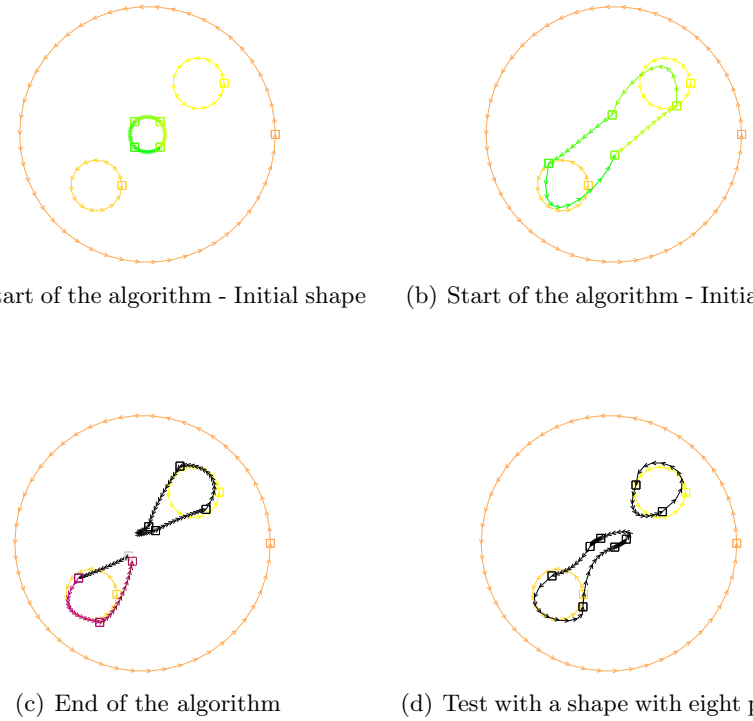
We wrote scripts in $FreeFem++$ for numerical simulations as it is very convenient to use for solving PDEs. (It is however not very well-suited if we want to change dynamically the number of patches of the curves.) We first tested our method on the problem of detecting only one object. That is, before testing the flip method to detect multiple objects, we had to be sure that the choice of Bézier parametrizations could give good results. Figure 12 shows that indeed Bézier shapes are well-suited for such a problem. The object to detect is the orange circle at the top right and the initial 4-patched Bézier shape is the smaller, green and yellow circle. The figure **??** shows that the Bézier shape has moved and grown to finally blend with the object.

We then tested our method with two objects to see how it would react with a flip. The figure 13 presents different states of the algorithm. The two objects are positionned in the bottom left corner and top right corner. The initial Bézier shape consists in a single component with four patches, located at the center (figure 13(a)). The shape grows and surrounds the two objects (figure 13(b)) until some patches come closer to each other. The flip is then performed and the shape is divided in two connected components. The figure 13(c) shows that the algorithm has found the two objects. We tried to increase the number of patches for the initial shape : after testing four patches, we used an initial Bézier shape composed by eight patches to see if we would get better results. Figure **??** shows the final shape in that case : the top right object was well detected by the two-patched Bézier shape whereas



(a) Initial shape (b) Final shape

Figure 12. Results for the detection of one obstacle.



(a) Start of the algorithm - Initial shape (b) Start of the algorithm - Initial shape

(c) End of the algorithm (d) Test with a shape with eight patches

Figure 13. Execution of the algorithm

the other component has too many patches (six) in order to blend with the second object. Note how one cubic patch is enough to describe well half of the object. We added a plot of the cost function $\mathcal{J}$ at each iteration in figure 14 to see the behaviour of $\mathcal{J}$ after a flip.

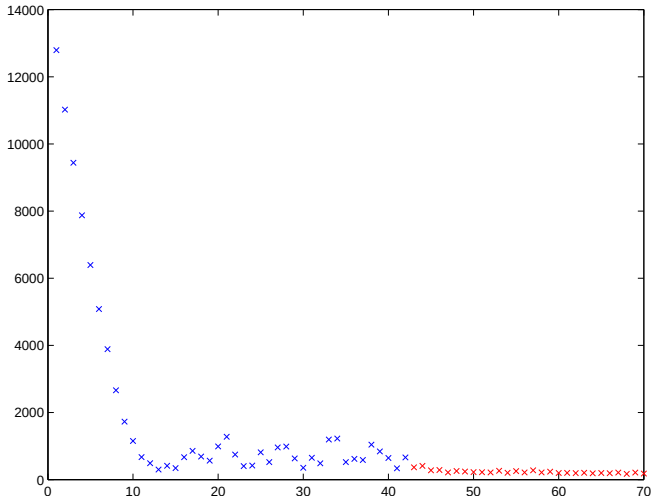


FIGURE 14. This graphic shows the objective functoin value at each iteration. After 42 iterations, the flip divides the shape in two connected components. The points in red corresponds to the part of the algorithm after the flip. We can note that the decrease of the objective function is smoother after the flip.Indiquer sur le graphique iteration en abscisse et J en ordonnee.

6. **Conclusion**

C'est formidable !