

Selfintersections of a Bézier Bicubic Surface

André Galligo, Jean Pascal Pavone
Nice (France)

* Supported by the EU contract GAIA II(IST-2002-35512)

Summary

The computation of selfintersections is a major problem in Computer Aided Geometric Design (CAD). We present two contributions :

- First, a specific sparse bivariate resultant adapted to the corresponding elimination problem,
- Second a semi-numeric polynomial solver able to deal with large system of equations with floating point coefficients.

Definition and notations

A parametrized bicubic patch is the image of :

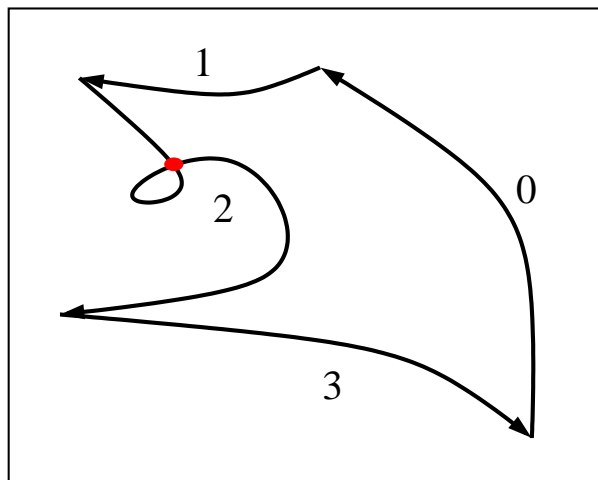
$$\Phi : [0, 1] \times [0, 1] \rightarrow \mathbb{R}^3$$

$$(t, u) \mapsto \Phi(u, v) = \left(\frac{\Phi_1(u, v)}{\Phi_0(u, v)}, \frac{\Phi_2(u, v)}{\Phi_0(u, v)}, \frac{\Phi_3(u, v)}{\Phi_0(u, v)} \right)$$

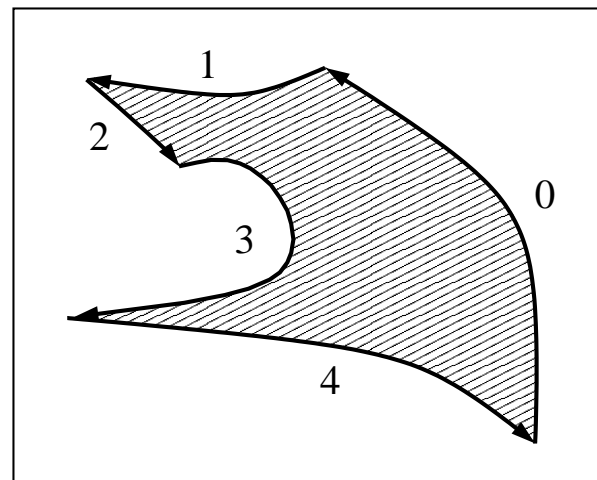
$\Phi_0, \Phi_1, \Phi_2, \Phi_3$ are polynomials of bidegree $(3, 3)$.

The patch is called Bézier when $\Phi_0(u, v) = 1$. (Our hypothesis).

A domain in the plane is represented in C.A.D. by the list of the curve segments delineating its border. This list represents a well-defined domain if each segment is free of loop.



(a)



(b)

C.A.D. systems use a similar representation for volumes. This problem is one of the main topics of the European project GAIA II.

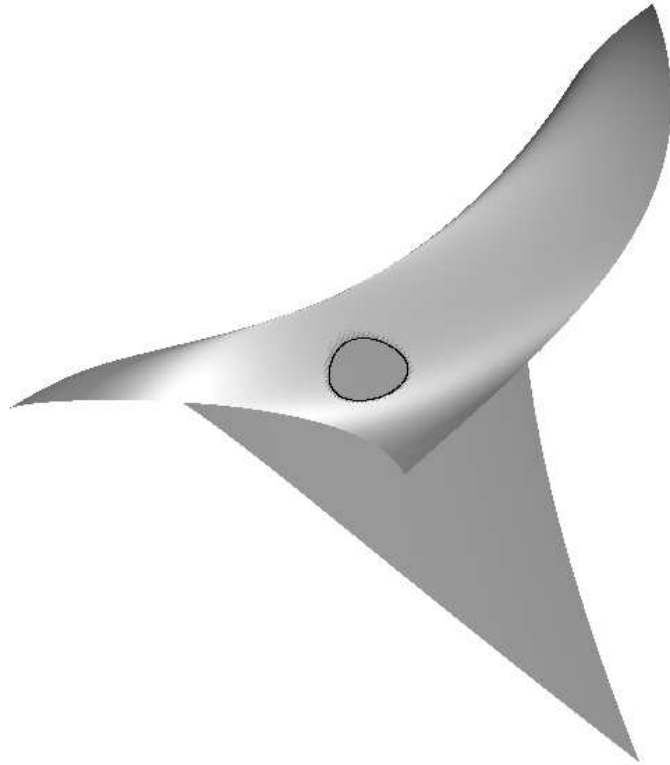
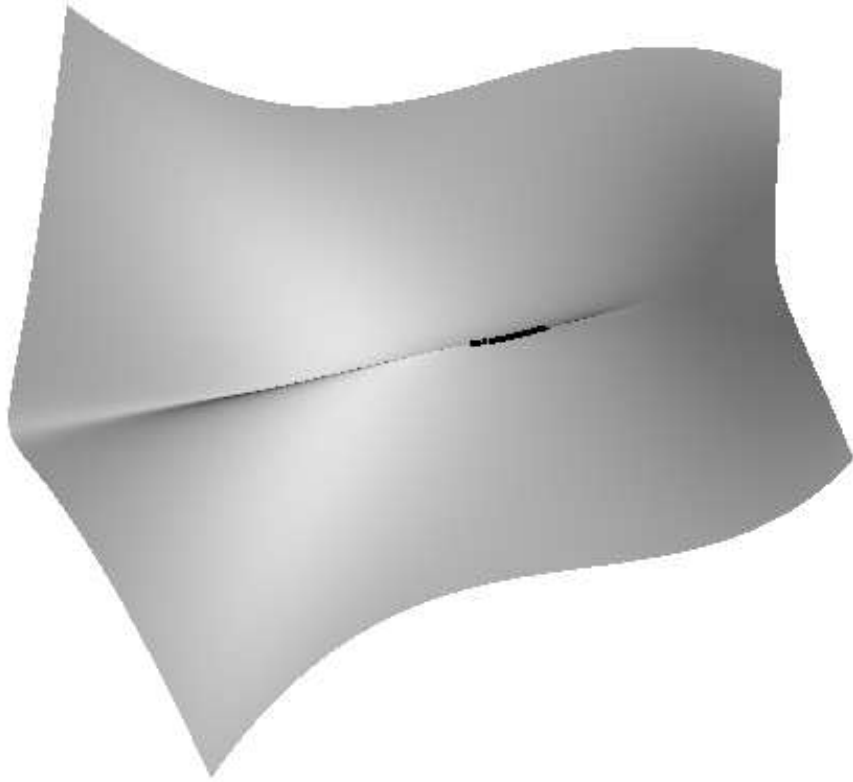


FIG. 1 – A selfintersection loop



Setting the equations

The set of double points of Φ is characterized by :

$$\mathcal{C} = \{(u, v, u', v') \text{ s.t. } (u, v) \neq (u', v'), \Phi_i(u, v) = \Phi_i(u', v')\}.$$

We assume $u > u'$ and let :

$$u = u_1 + l; v = v_1; u' = u_1; v' = v_1 + lk.$$

- We get 3 equations :

$$T_i := \frac{\Phi_i(u_1 + l, v_1) - \Phi_i(u_1, v_1 + lk)}{l}.$$

An elimination problem

The algebraic question amounts to eliminate l and k in T .

All T_i have multidegree $(3, 3, 2, 3)$ in (u_1, v_1, l, k) ; but only $k^3l^2, k^2l, l^2, k, l, 1$ do appear. We set :

$$T := m^2 * \text{subs}(l = \frac{1}{m}, T)$$

and get 3 polynomials in the 6 monomials :

$$k^3, k^2m, 1, km^2, m, m^2.$$

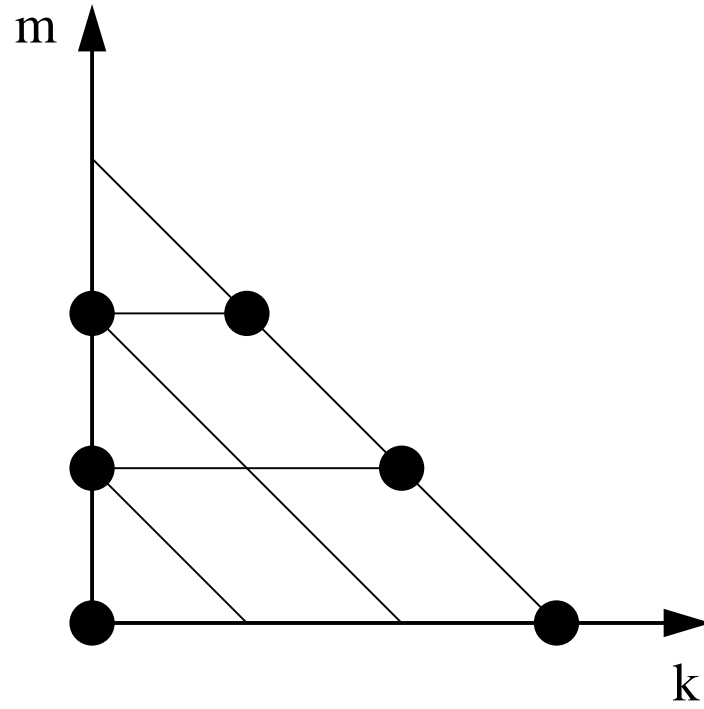


FIG. 2 – A very sparse support

A specific sparse resultant

- Several families of multivariate resultants exist.
- Some implementations are available as Maple packages
- multires, see the web page of galaad at INRIA.
- bires by A. Khetan, see his web page.
- Next, is our recipe for computing a “home-made resultant” for our specific data :

$$g_1(x, y); g_2(x, y); g_3(x, y)$$

with support $x^3, x^2y, xy^2, 1, y, y^2$.

- We construct g_4 and g_5 via

$$f_2 := \text{reduce}(g_2, [g_1]), \quad f_3 := \text{reduce}(g_3, [g_1, g_2]),$$

$$g_4 := \text{reduce}(x f_3, [f_1, f_2]),$$

$$g_5 := \text{sply}(g_4, \text{reduce}(x g_4, [f_2, f_3])).$$

We will also need the following notations :

$$B_2 := \text{coeff}(f_2, x^2 * y) = a_1 * b_2 - a_2 * b_1;$$

$$B_5 := \text{coeff}(f_2, y) = a_1 * b_5 - a_5 * b_1;$$

$$C_3 := \text{coeff}(f_3, x * y^2)$$

- We denote by R_1 the determinant of the square matrix of the coefficients of 21 product by choosen monomials of these g_i .

- R is the exact quotient $\frac{R_1}{B_2^2 * C_3^2 * B_5}$.

Implicit equations

$$P(u_1, v_1) := R(T_1(u_1, v_1), T_2(u_1, v_1), T_3(u_1, v_1)).$$

- The degree of $P(u_1, v_1)$ in u_1 is a priori bounded by $3 \times 24 = 72$.

But due to the special form of the coefficient, generically the degree of $P(u_1, v_1)$ in u_1 is 44.

- This is exactly the maximal number predicted by our geometric analysis.
- To compute the critical points, solve :

$$P(u_1, v_1) = 0 ; \frac{\partial P}{\partial u_1}(u_1, v_1) = 0.$$

- An alternative approach returns to T_i .

A numerical approach

- To compute the critical points, we express

$$T_4 := \det\left(\left[\frac{\partial T}{\partial v_1}, \frac{\partial T}{\partial l}, \frac{\partial T}{\partial m}\right]\right) = 0.$$

- T_4 is a polynomial of multidegree $(9, 8, 5, 8)$ in (u_1, v_1, l, m) .
- We have to solve (numerically) a system of 4 polynomials in 4 variables : $(T_1, T_2, T_3, T_4) = 0$.
- We developed a solver, based on multivariate Bernstein representation of polynomials, it is adapted to our setting.

A polynomial solver

- The basic principle of our solver, is similar to that of the IPP solver of Sherbrooke and Patrikalakis. They use control polyhedrons.
- The solvers perform a sequence of carefully chosen subdivisions and reductions to determine a finite number of very small boxes which may contain roots of the system.
- These solvers rely on projections of the control polyhedrons of the graphs of the equations to reduce the domain for each variable involved in the process.

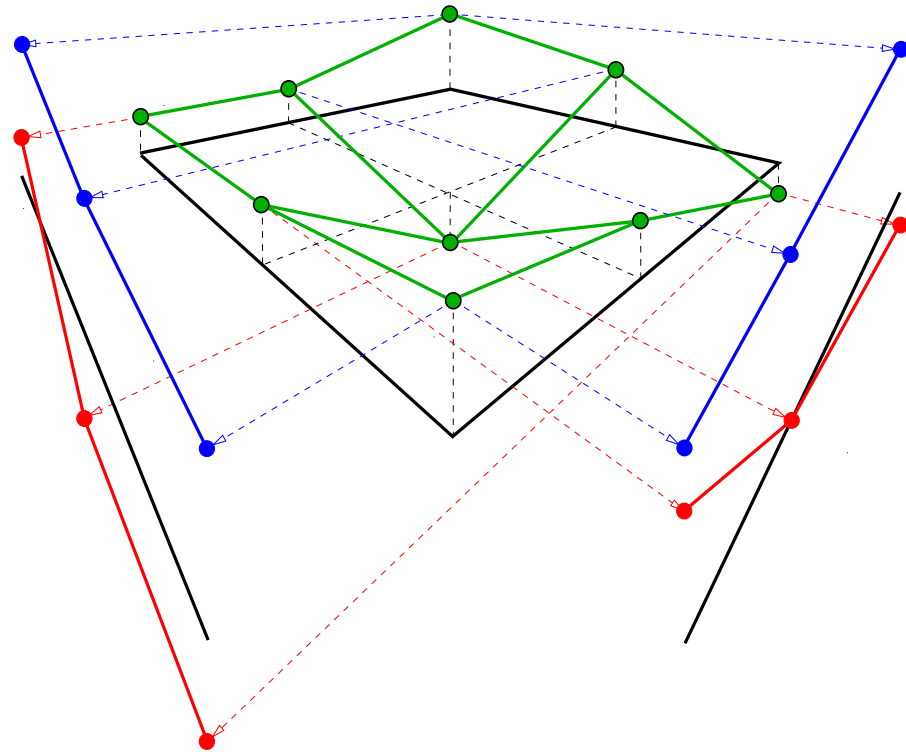


FIG. 3 – Principle of the *IPP* projection

Our solver uses

- a preconditioner inspired by interval analysis,
- proceeds to much less subdivisions, compared to IPP.

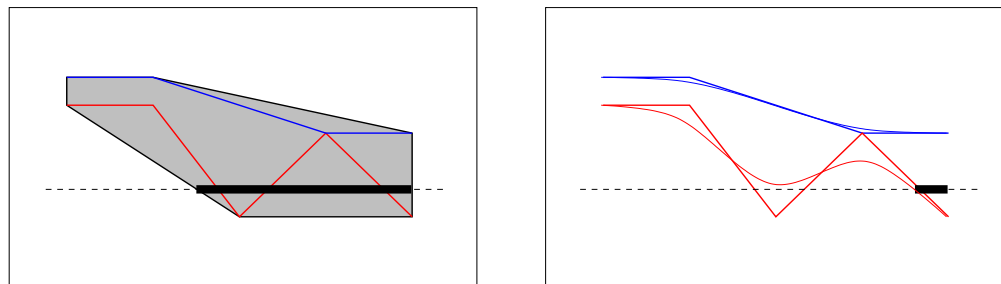


FIG. 4 – *IPP* reduction (a), our solver (b)

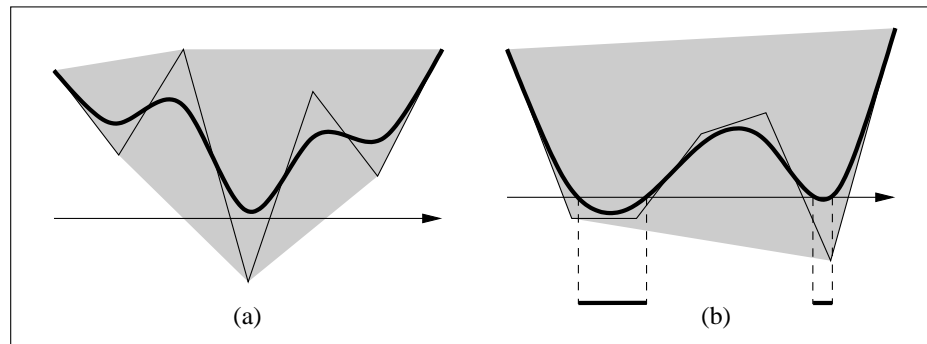


FIG. 5 – exclusion (a) and multiple reductions (b)

Examples and timings

We treat the examples corresponding to the first 2 presented pictures.

$$\begin{array}{l} x= \\ y= \\ z= \end{array} \begin{bmatrix} 1.718019 & 0.045451 & 1.220627 & -8.437654 \\ 2.245235 & 0.572671 & -0.528670 & -1.472482 \\ -2.706853 & -3.948356 & -0.572671 & -2.245235 \\ -2.565772 & -4.284708 & -3.824920 & -1.718019 \\ -5.030251 & -3.145330 & -2.801745 & -4.636559 \\ -3.561677 & -1.676751 & 0.410812 & 1.075778 \\ -2.077333 & -0.843524 & 1.676751 & 3.561677 \\ -1.994069 & 0.916993 & 3.530333 & 5.030251 \\ 1.935424 & 2.807696 & 0.101320 & -2.755474 \\ -0.227130 & 0.645141 & -0.162758 & 0.015489 \\ 0.790476 & -1.114797 & -0.645141 & 0.227130 \\ -1.779410 & -0.718161 & -2.442728 & -1.935424 \end{bmatrix}$$

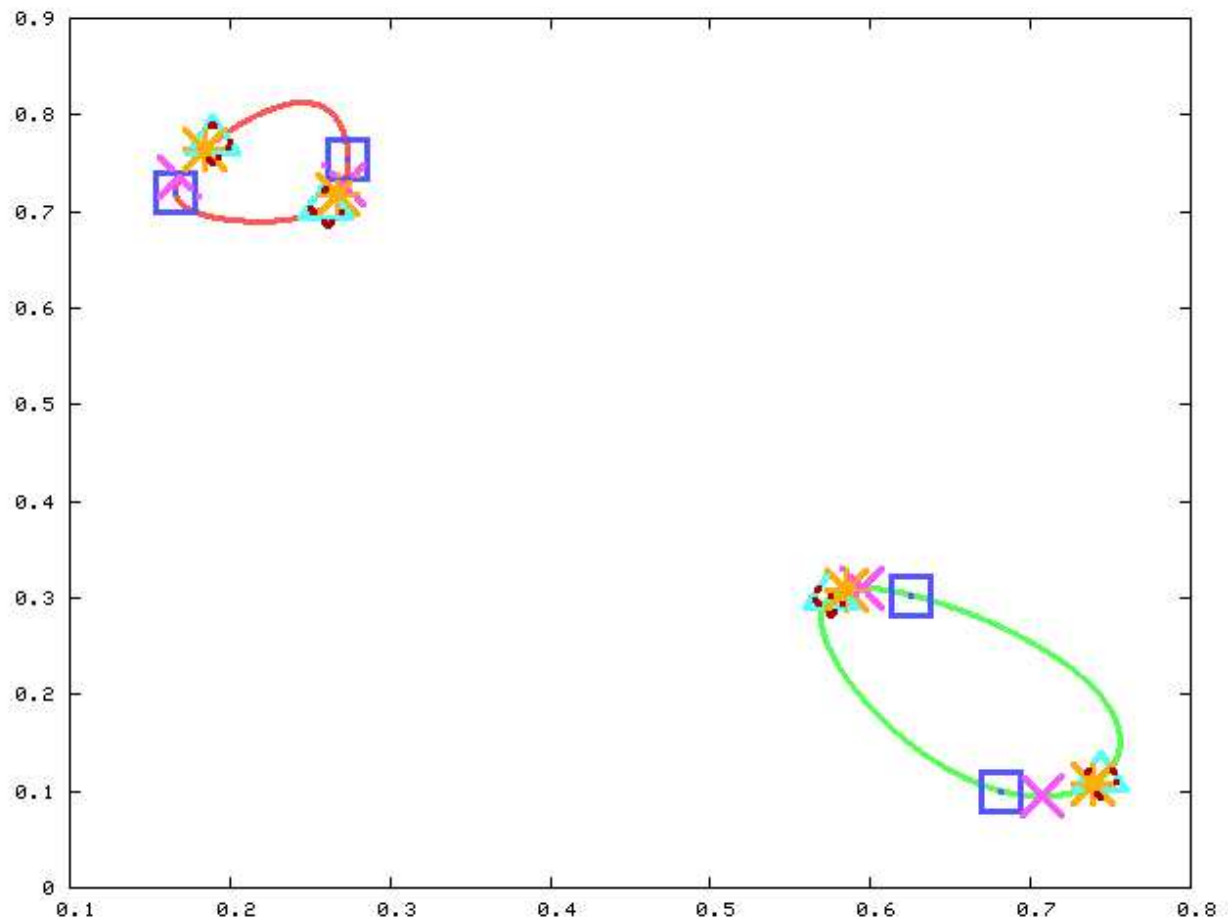


FIG. 6 – Extrema points in the domain 1

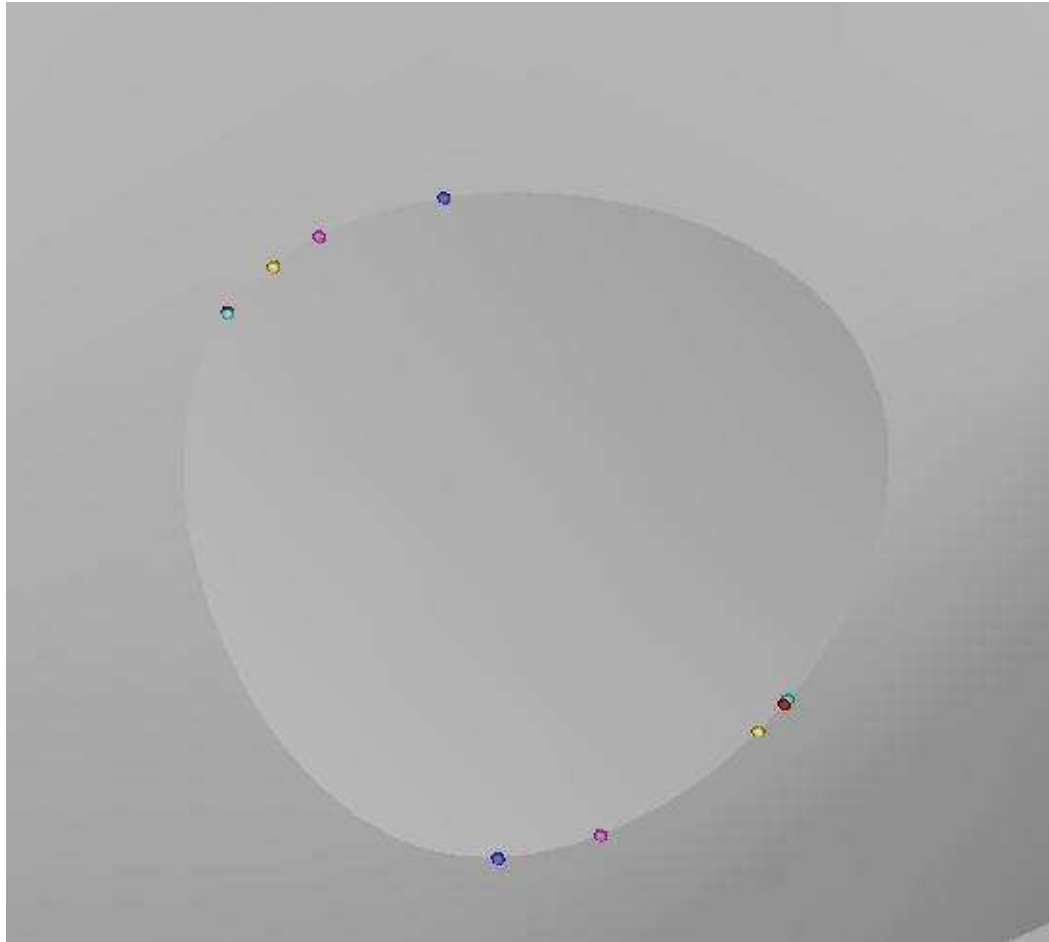


FIG. 7 – Extrema points on the surface 1

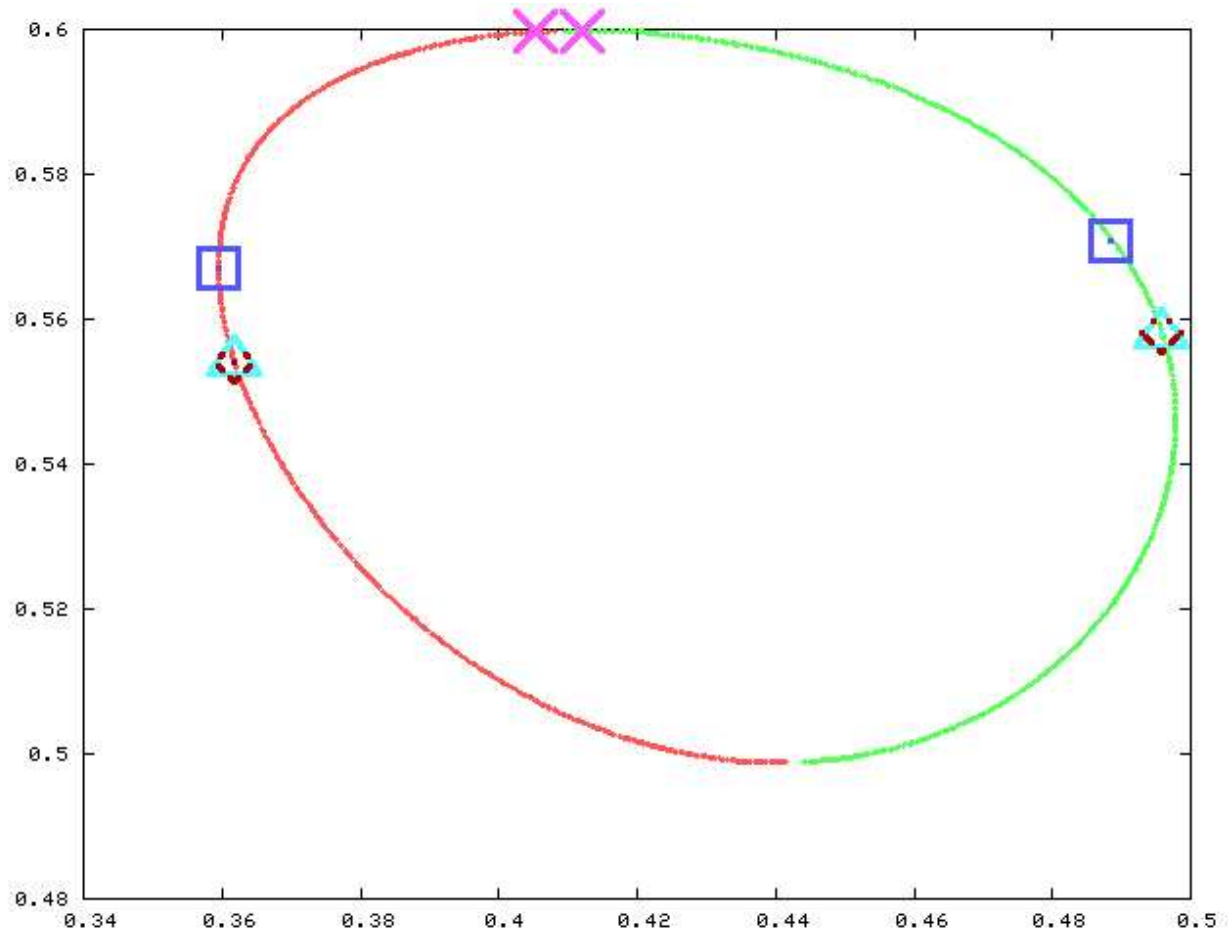


FIG. 8 – Extrema points in the domain 2

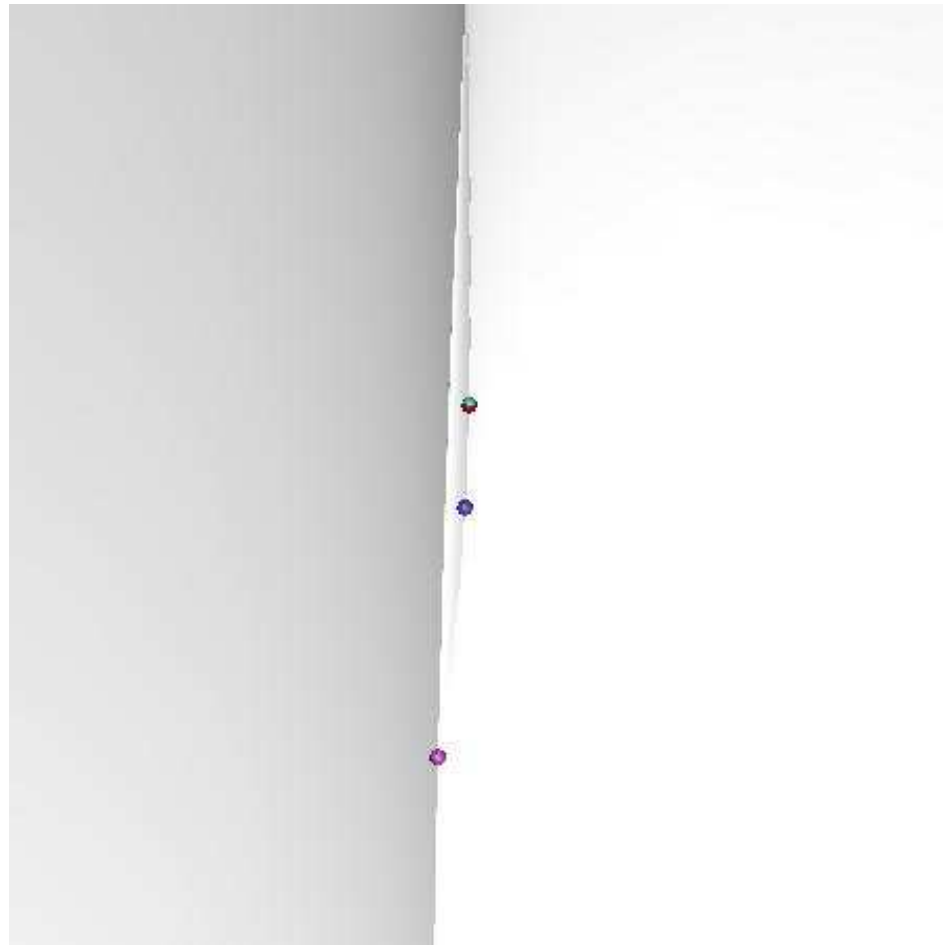


FIG. 9 – Extrema points on the surface 2

time(s)	u_1	v_1	l	k
ex 1	5.1	5.7	5.3	5.5
ex 2	2.5	1.9	2.4	2.2
extrema 1	u	v	u'	v'
u_1	0.625524	0.302895	0.273844	0.754153
	0.681597	0.0998794	0.166012	0.72042
v_1	0.595082	0.310081	0.27086	0.727953
	0.706713	0.0946811	0.168988	0.736224
l	0.574816	0.301324	0.260516	0.706247
	0.744083	0.113855	0.18959	0.771738
k	0.584561	0.308121	0.267303	0.71729
	0.73906	0.107815	0.184838	0.765172

Conclusion

- In theory, we should be able to check and improve the quality of the output of the semi-numeric solver, by using the specific resultant built in section 5 but :
 - The norm of the matrix whose determinant gives the resultant (ours or Khetan's one) is too large and induces ill-conditioned behavior.
 - This numeric problem is tough.
- Our second tool is more satisfactory :
 - The timings, we presented in section 8, are good and show that the program can be used in a C.A.D. system.
 - It is not only efficient but also very robust as it delivers a solution together with a (small) bound on the possible error.