Analele Universităţii din Timişoara
Vol. XXXVIII, fasc. 2, 2000
Seria Matematică–Informatică

# A SYMBOLIC/NUMERIC TOOLBOX FOR COMPUTER AIDED GEOMETRIC DESIGN

Laureano González-Vega, Ioana Necula* and David Sevilla

Presented at $2^{nd}$ Int. Workshop "Symbolic and Numeric Algorithms on Scientific Computing" - SYNASC 2000, October 4-6, 2000, Timişoara, Romania

**Abstract.** This paper is devoted to show how the already widely used Scientific Computing Systems (in our case `Maple` and `Matlab`) integrating symbolic and numeric capabilities can be used to develop a Problem Solving Environment very useful to solve problems into a CAD/CAM framework. First section shows and motivates how algebraic techniques and Scientific Computing Systems can be very useful in CAGD. The remaining sections of this paper are `Maple` spreadsheets (with some calls to `Matlab` to solve some huge linear systems of equations) where some concrete problems in Computer Aided Geometric Design are solved. Our interest is focused on the generation and manipulation of B-spline entities (curves and surfaces) and on the computation (topologically exact) of a revolution surface sectioning.

## 1. Introduction

In this paper it is shown how widely used Scientific Computing Systems

Keywords and phrases: CAGD, B-spline entities (curves and surfaces), revolution surface sectioning, `Maple` and `Matlab` programming.
Partially supported by DGESIC PB 98-0713-C02-02 (Ministerio de Educación y Cultura)
Partially supported by the FEDER Project 1FD97-0409 (Ministerio de Educación y Cultura)

(in our case `Maple` and `Matlab`) integrating symbolic and numeric facilities can be used to develop a Problem Solving Environment which is very useful for solving problems into a CAD/CAM framework.

The utility of CAD/CAM systems as a way of increasing the efficiency of simulation and design processes in the productive area is nowadays unanswerable. Advantages as production time reduction, final product improvement and cost reduction are frecuentely called as the greatest benefits produced by introducing CAD/CAM systems in an industrial environment.

In the first section we show how algebraic techniques and Scientific Computing Systems (as `Maple` and `Matlab`) can be very useful in CAGD. The remaning sections of this paper are `Maple` spreadsheets, with internal calls to `Matlab` in order to solve some linear equation systems of big size, solving in this way some classical problems in Computer Aided Geometric Design. More precisely, the second section presents the computation, topologically exact, of a revolution surface sectioning. The following sections show how `Maple` can work with B-spline curves and surfaces.

This paper gathers three research lines:

- Studying and improving the graphic tools provided by the actual Scientific Computing Systems (`Mathematica`, `Matlab`, `Maple` and `Axiom`) together with simulation module development in these systems, for geometric modelling and visualization.
- Adapting, developing and integrating the algebraic equation system solution manipulation techniques developed in the framework of FRISCO project (ESPRIT/LTR 21024: European Union) to solve efficiently problems concerning parametric curve and surface manipulation.
- Solving a set of concrete problems (unsolved satisfactorily at this moment) for the CAD/CAM environment CSIS used by CANDEMAT (company dedicated to make dies for cars). This research line is approached by including the developed techniques inside the simulation methods mentioned above and/or including techniques or software before mentioned.

The CAGD systems traditionally have been developed using programming languages with good characteristics for scientific computing (`C`, `C++`, `Fortran`, `Basic`, `Pascal`,...). While the hardware technology is progressing rapidly, the software evolution goes on more slowly, although continuously too. This paper is focused specifically on the integrated symbolic and numeric computing packages having high graphical capacities. This kind of software has been reaching a large diffusion and importance in the last few

years. Together with their basic general statements, these systems offer additional modules which include more specific applications. That is the case of `Mathematica` packages and `Matlab` toolboxes.

In `Mathematica`, there is a package called 'Graphics Spline', having limited capacities for generating basic entities like Bezier curves and cubic splines. The strong graphical capacities on parametrics, operators, etc. allow these primitives to be applied for interesting but limited models. In `Matlab` there is a basic function for generating cubic splines and (since 1990) a toolbox called 'Spline' which contains B–spline functions for curves and surfaces, following the classic work of Carl de Boor ( [2], [3]).

Moreover in the last few years the international scientific community has looking for efficient (i.e. rapid) and as precise as possible (i.e. guaranteed validity for the obtained solutions) algorithms/methods for solving algebraic or polynomial equations and for manipulating the sets of its solutions. The exact meaning of the phrase above can be observed more clearly in the following example with a CAGD flavour.

**Example 1** *We consider the parametric equation of the bicubic surface* $\mathcal{S}$

$$\begin{aligned} x &= 3t(t-1)^2 + (s-1)^3 + 3s \\ y &= 3s(s-1)^2 + t^3 + 3t \\ z &= -3s(s^2 - 5s + 5)t^3 - 3(s^3 + 6s^2 - 9s + 1)t^2 + t(6s^3 + 9s^2 - 18s + 3) - \\ &\quad 3s(s-1) \end{aligned}$$

*for parameter values between 0 and 1, displayed in figure 1.*

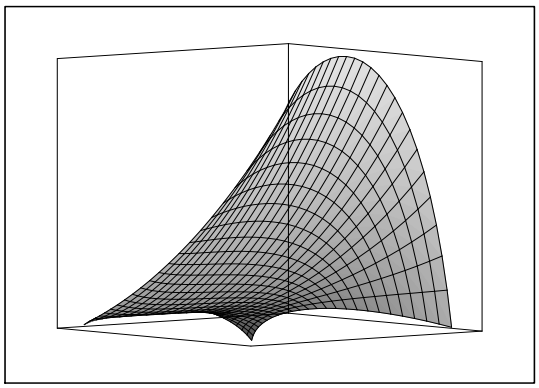

Figure 1: *Bicubic surface presented in Example 1*

A first problem which can arise in this situation is how to determine the intersection points of the surface $\mathcal{S}$ with the straight line $x = u, y = u, z = u$

(ifsuch a solution exists). That means solving the following equation system:

$$\begin{aligned} u &= 3t(t-1)^2 + (s-1)^3 + 3s \\ u &= 3s(s-1)^2 + t^3 + 3t \\ u &= -3s(s^2-5s+5)t^3 - 3(s^3+6s^2-9s+1)t^2 + t(6s^3+9s^2-18s+3) - \\ &\quad 3s(s-1) \end{aligned}$$

In this particular case we get only one intersection point, having coordinates

$$(0.5561, 0.5561, 0.5561)$$

reached at $s = 0.2748$ and $t = 0.0408$. Another way of solving this problem consists of determining the implicit equation of $\mathcal{S}$. If this equation has been computed, the previous problem reduces to solving the equation $H(u,u,u) = 0$. Analogously, every intersection of $\mathcal{S}$ with any curve can be dealt by solving a single variable equation. In this particular case, the implicit equation of $\mathcal{S}$ has the following structure:

$$H(x,y,z) = z^9 + \sum_{i=1}^{9} r_i(x,y) z^{9-i}$$

$$\begin{aligned} \mathbf{r}_1(x,y) &= -\tfrac{233469x}{2048} + \tfrac{188595y}{2048} - \tfrac{112832595}{262144} - \tfrac{81x^2}{64} + \tfrac{135xy}{32} - \tfrac{81y^2}{64} \\ \mathbf{r}_2(x,y) &= -\tfrac{20972672709381x}{536870912} + \tfrac{17975329363179y}{536870912} - \tfrac{729y^4}{8192} - \tfrac{729x^4}{8192} + \tfrac{1215x^3y}{2048} - \\ &\quad \tfrac{4779x^2y^2}{4096} + \tfrac{1215xy^3}{2048} - \tfrac{4105971x^3}{65536} + \tfrac{3129597y^3}{65536} + \tfrac{14456151x^2y}{65536} - \\ &\quad \tfrac{13181049xy^2}{65536} - \tfrac{54187594407x^2}{16777216} + \tfrac{48101467761xy}{8388608} - \tfrac{38812918311y^2}{16777216} - \\ &\quad \tfrac{22656991982391171}{137438953472} - \tfrac{1}{2}\left(\tfrac{233469x}{2048} - \tfrac{188595y}{2048} + \tfrac{112832595}{262144} + \tfrac{81x^2}{64} - \right. \\ &\quad \left. \tfrac{135xy}{32} + \tfrac{81y^2}{64}\right)\left(-\tfrac{233469x}{2048} + \tfrac{188595y}{2048} - \tfrac{112832595}{262144} - \tfrac{81x^2}{64} + \right. \\ &\quad \left. \tfrac{135xy}{32} - \tfrac{81y^2}{64}\right) \\ \mathbf{r}_3(x,y) &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned}$$

and by substituting $x = u, y = u, z = u$ we obtain one equation of degree 18 in $u$, very easy to solve:

$$5159780352u^{18} - 609499054080u^{17} + \ldots + 3707912273492242256259566313 = 0$$

The problem which occurs with this philosophy of work resides in the fact that, although there are methods for computing the implicit equation $H(x,y,z)$, these methods are very inefficient and can hardly be integrated in the CAD/CAM packages, where the solutions for any user request must be computed in real time. However, to overcome this drawback it is feasible the incorporation of a data base containing the implict equations (already computed and preprocessed) of the parametric surfaces of frequent use. For

instance the implicit equation of the parametric surface

$$\begin{aligned} x = &\quad x_{00}\frac{t_2-t}{t_2-t_1} + x_{01}\frac{t-t_1}{t_2-t_1} \\ y = &\quad y_{00}\frac{s_2-s}{s_2-s_1} + y_{11}\frac{s-s_1}{s_2-s_1} \\ z = &\quad \left(z_{00}\frac{s_2-s}{s_2-s_1} + z_{10}\frac{s-s_1}{s_2-s_1}\right)\frac{t_2-t}{t_2-t_1} + \left(z_{10}\frac{s_2-s}{s_2-s_1} + z_{11}\frac{s-s_1}{s_2-s_1}\right)\frac{t-t_1}{t_2-t_1} \end{aligned}$$

is

$$\begin{aligned} &(z_{00}-2z_{10}+z_{11})xy + (y_{00}z_{10}+y_{11}z_{10}-y_{00}z_{11}-y_{11}z_{00})x \\ &\quad +(x_{00}z_{10}+x_{01}z_{10}-x_{00}z_{11}-x_{01}z_{00})y + (x_{00}y_{11}-x_{00}y_{00}+x_{01}y_{00}-x_{01}y_{11})z \\ &\quad +x_{01}y_{11}z_{00}-x_{00}y_{11}z_{10}+x_{00}y_{00}z_{11}-x_{01}y_{00}z_{10} \end{aligned}$$

for most values of the parameters $x_{ij}$, $y_{ij}$ and $z_{ij}$. Likewise it is possible to determine the algebraic expressions describing $s$ and $t$ as functions of $x$, $y$ and $z$ (see [6]).

The problems considered in the following sections are examples of how algebraic techniques (such as subresultants, Sturm-Habicht sequences, symmetric functions etc.) can be used for the resolution of CAGD problems.

## 2. Revolution Surface Sectioning

The first problem considered is the computation, topologically exact, the section of a revolution surface by using the generic implicitation methodology before mentioned. For that it is used a serie of algebraic techniques in order to determine the exact shape of a revolution surface sectioning. The details of the algorithm can be found in [7] and [5].

**Example 2** *The curve in the plane $Y = 0$ defined by the parametrization $x = C_1(t)$, $z = C_3(t)$ will be rotated with respect to the $OZ$ axis. In figure 2 it is displayed the 3-dimensional curve $C$ defined by the parametrization $(C_1(t), 0, C_2(t))$.*

```
> plots[setoptions3d](scaling=CONSTRAINED,axes=FRAMED);
> plots[setoptions](scaling=CONSTRAINED,axes=FRAMED);

C1:=(2*t-1)/(1+t**2); C2:=0;
C3:=(1-t+t**2)/(2+t+t**2);
plot3d([C1,C2,C3],t=-15..15,s=-1..1,orientation=[60,75],
grid=[175,2]);
```

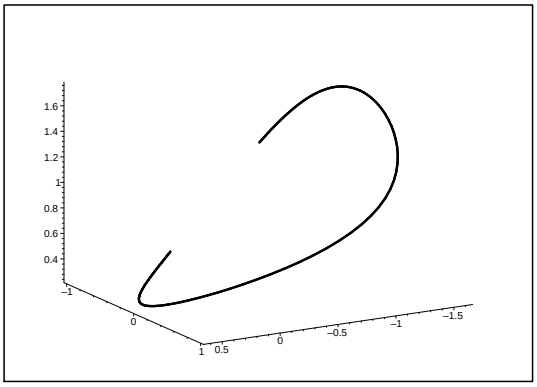


Figure 2: *Curve $C$ considered in Example 2*

The revolution surface generated by the curve $C$ is displayed as follows (see figure 3). The limits of the parameter intervals are $-3$ and 3 respectively $-6$ and 6.

```
> T1:=C1*(2*s)/(1+s**2)-C2*(1-s**2)/(1+s**2):
> T2:=C2*(2*s)/(1+s**2)+C1*(1-s**2)/(1+s**2):
> T3:=C3:
```

```
{P1:=(a,b)->plot3d([T1,T2,T3],s=a..b,t=-6..6,grid=[50,50],
axes=BOXED,scaling=UNCONSTRAINED,projection=1,
tickmarks=[0,0,0],orientation=[59,75]):
> P1(-3,3);
```

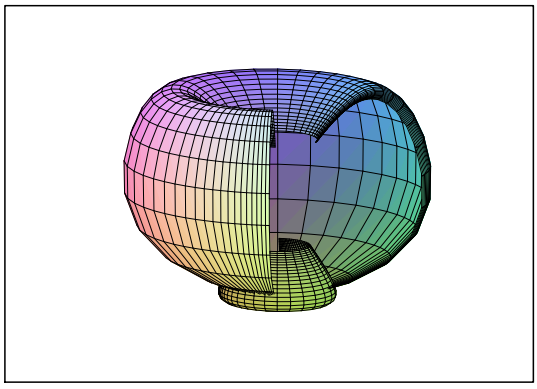

Figure 3: *Revolution surface generated by $C$*

For studying the sections of this revolution surface, we firstly compute its implicit equation by using the generic implicitation method, which is based on symmetric functions.

```
> read "ImplicitSuperfRevol.txt";
> toro:=ImplicitSuperfRevol(C1,C2,C3,x,y,z,t);
```

$$\begin{aligned} toro := {} & 9 + 8\,z\,x^2\,y^2 + 262\,z^2 - 308\,z^3 - 84\,z + 8\,z^4\,x^2\,y^2 + 121\,z^4 + \\ & 16\,z^2\,x^2\,y^2 + 20\,z\,x^2 + 20\,z\,y^2 - 3\,x^2 - 3\,y^2 + x^4 + y^4 + 16\,z^3\,x^2\,y^2 + \\ & 19\,z^4\,x^2 + 19\,z^4\,y^2 + 2\,x^2\,y^2 + 4\,z^4\,x^4 + 4\,z^4\,y^4 + 8\,z^3\,x^4 + \\ & 8\,z^3\,y^4 - 72\,z^3\,x^2 - 72\,z^3\,y^2 + 8\,z^2\,x^4 + 8\,z^2\,y^4 - \\ & 28\,z^2\,x^2 - 28\,z^2\,y^2 + 4\,z\,x^4 + 4\,z\,y^4 \end{aligned}$$

First we compute the section of this surface with the plane $y = 1/2$.

```
> Section1:=subs(y=1/2,toro);
```

$$\begin{aligned} Section1 := {} & -\frac{315}{4}z + 21z^4x^2 + \frac{511}{2}z^2 - \frac{651}{2}z^3 + 126z^4 + x^4 - \\ & \frac{5}{2}x^2 + 4z^4x^4 - 68z^3x^2 + 8z^3\,x^4 - 24z^2x^2 + \\ & 8z^2x^4 + 4zx^4 + 22zx^2 + \frac{133}{16} \end{aligned}$$

Initially, we determine the topology of this sectioning (it is important to mention that it has two isolated points), as displayed in figure 4a. In this step we are interested only in the critical and regular points of the curve, which are the graph points. The behaviour of the curve between these points is completely determined (taking into account the branch counting in each interval) so we can connect them by edges. The algorithm we have used to determine the topological structure of the curve is different from the classical algorithms because by a simple change of coordinates we can work with a curve which fulfils the following condition: for every root of the discriminant, we have at most a critical point of the curve (see [7] and [5]).

```
> read "prGrafos.txt";
> principal(Seccion1,x,z,20,'black');
```

After determining the topology of the curve, we are able to draw it exactly using for example the Newton method, without losing isolated or small components (figure 4b).

```
read "prDibujos.txt";
principal(Seccion1,x,z,20,500,10,'black');
```

```
> P2:=y->plot3d([s,y,t],s=-4..4,t=-2..2,style=PATCHNOGRID):
```

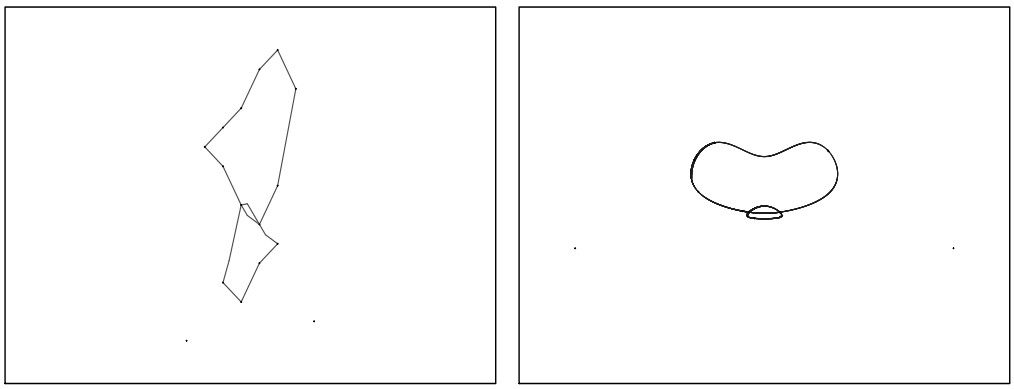

Figure 4: *(a) Sectioning topology and (b) sectioning curve*

In this way we have obtained, in an exact way, the section of our revolution surface we are interested in (figure 5).

```
plots[display](P1(-4,4),P2(1),grid=[50,50],axes=BOXED,
scaling=UNCONSTRAINED,projection=1,tickmarks=[0,0,0],
orientation=[13,130]);
```

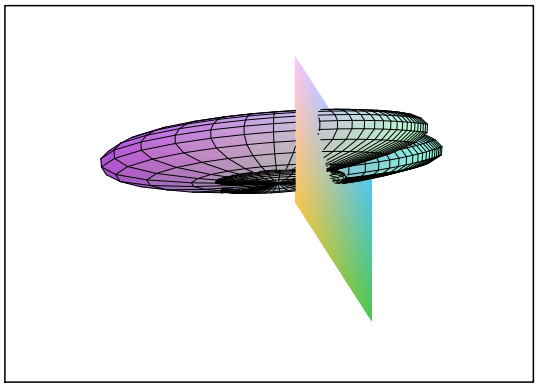

Figure 5: *Revolution surface sectioning*

## 3. Random Generation of Polynomial and Rational B-Spline Curves and Surfaces

In this section we will show how `Maple` can generate symbolically polynomial and rational B-spline entities. Before doing this, we will present some useful definitions [11] which will be used in this section and the in following ones.

**Definition 3** *The $N$th–order B-spline basis functions are defined recursively as:*

$$B_{i,1}(u) = \begin{cases} 1 & u_i \le u < u_{i+1} \\ 0 & otherwise \end{cases}$$

$$B_{i,N}(u) = \begin{array}{l} \frac{u-u_i}{u_{i+N-1}-u_i} B_{i,N-1}(u) + \\ \frac{u_{i+N}-u}{u_{i+N}-u_{i+1}} B_{i+1,N-1}(u) \end{array}.$$

**Definition 4** *A $N$th–order rational B-spline curve $R_N(u)$ is defined by:*

$$R_N(u) := \frac{\sum_{i=0}^{n-1} B_{i,N}(u) w_i P_i}{\sum_{i=0}^{n-1} B_{i,N}(u) w_i}, \qquad a \le u \le b,$$

*where $\{P_i\}$ are the control points, $\{w_i\}$ are the weights and $\{B_{i,N}\}$ are the $N$th–order B-spline basis functions defined on the nonperiodic knot vector*

$$U = \{u_0, u_1, \ldots, u_{n+N-2}, u_{n+N-1}\}$$

*with $u_0 = u_1 = \ldots = u_{N-1} = a$ and $u_n = u_{n+1} = \ldots = u_{n+N-1} = b$.*

*If for all $i$ $w_i = c$ $(c \neq 0)$ then the B-spline curve is called polynomial. In this case it has the following representation:*

$$R_N(u) := \sum_{i=0}^{n-1} B_{i,N}(u) P_i, \qquad a \le u \le b.$$

*If $a = 0$ and $b = 1$ we say that the knot vector is normalised.*

*The polygon formed by the $\{P_i\}$ is called control polygon.*

**Definition 5** *A $N_1, N_2$th–order rational B-spline surface $R_{N_1,N_2}(u,v)$ is defined by*

$$R_{N_1,N_2}(u,v) := \frac{\sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} B_{i,N_1}(u) B_{j,N_2}(v) w_{i,j} P_{i,j}}{\sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} B_{i,N_1}(u) B_{j,N_2}(v) w_{i,j}}, \quad a \le u \le b, c \le v \le d,$$

*where* $\{P_{i,j}\}$ *are the control points,* $\{w_{i,j}\}$ *are the weights ,* $\{B_{i,N_1}\}$ *and* $\{B_{j,N_2}\}$ *are the B-spline basis functions defined on the nonperiodic knot vectors* $U = \{u_0, u_1, \ldots, u_{n_1+N_1-1}\}$ *and* $V = \{v_0, v_1, \ldots, v_{n_2+N_2-1}\}$ *respectively, with* $u_0 = u_1 = \ldots = u_{N_1-1} = a$, $u_{n_1} = u_{n_1+1} = \ldots = u_{n_1+N_1-1} = b$, $v_0 = v_1 = \ldots = v_{N_2-1} = c$ *and* $v_{n_2} = v_{n_2+1} = \ldots = v_{n_2+N_2-1} = d$.

*If for all* $i, j \quad w_{i,j} = c \ \ (c \neq 0)$ *then the B-spline surface is called polynomial and its representation is the following one:*

$$R_{N_1,N_2}(u,v) := \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} B_{i,N_1}(u) B_{j,N_2}(v) P_{i,j}.$$

In the two following examples, the polynomial and rational expressions defining an entity will be generated in each interval $(u_i, u_{i+1})$ of the knot vectors. The `Maple` program will randomly generate the elements which define a B-spline entity.

**Example 6** *We consider the B-spline curves (polynomial and rational) defined by the following parameters:*

```
> read "simbC.txt";

Curve order: 6
Knot vector: [0,0,0,0,0,0,.1210225739,1.,1.,1.,1.,1.,1.]


Control points: [[1813.213542, 3155.072848, 7292.017654,
10852.09613, 11970.06777, 13138.01982, 5307.115079],
[15411.10969, 6888.316017, 7878.233333, 4268.225166,
9045.040000, 6034.080537, 9315.020747], [22834.01291,
13631.11796, 26325.11671, 23448.19923, 5654.282051,
18408.02733, 22853.31278]]


Weights: [.1883026555, .1081886648e-1, .5431078818e-1,
.5421364074, .1062056055, .6872239468, .2581224202e-1]
```

First we generate two polynomial expressions and by using them we determine the symbolic expression of the polynomial B-spline curve. We do the same in the rational case. Finally, we display (see figure 6) the control polygon (in red), the polynomial B-spline curve (in green) and the rational one (in blue).

**Example 7** *In this example we will consider the B-spline surfaces (polynomial and rational) defined by the parameters:*

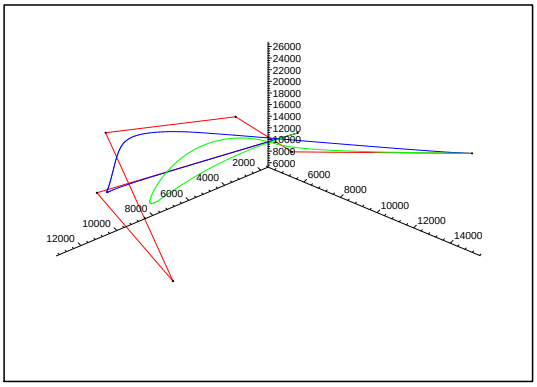


Figure 6: *Polynomial and rational B-spline curves presented in example 6*

```
> read "simbS.txt";

Surface orders: 3   4
Knot vectors: [0, 0, 0, 1., 1., 1.]  [0, 0, 0, 0, 1., 1., 1., 1.]


Control points: [[[3372.184932, 5567.160321, 7405.187500],
 [5472.689076, 4983.023544, 4507.052356], [7855.042857, 9212.056291,
 8272.083333], [11016.06109, 7460.165692, 3563.137184]],
 [[10457.15273, 17624.11443, 16674.14440], [4417.028646,
 15924.12000, 6384.092138], [12073.08661, 9125.182609, 10439.03580],
 [9301.109023, 17020.02273, 9594.065432]], [[20276.15139,
 19382.05319, 19076.02711], [17741.09483, 16728.07711, 16773.04392],
 [15851.06977, 5130.083645, 20085.10056], [3995.063830, 26465.03774,
 17510.08246]]]


Weights: [[.1467061818e-1, .2056977801, .8521961580],
[.6275754910, .2828278577, .3739758365e-1], [.1426845549,
.2486348262, .3187999143], [.5673102039, .3234548536,
.3034818421]]
```

We generate the expressions (polynomial and rational) which define symbolically the entities and afterwards we display the control polyhedra (in red) and the border curves of the polynomial and rational B-spline surface (in green and blue respectively) (see figure 7).

## 4. Polynomial and Rational B-Spline Curve Manipulation

In this section it is shown how `Maple` can perform, in a very efficient way, several operations with polynomial and rational B-spline curves (using

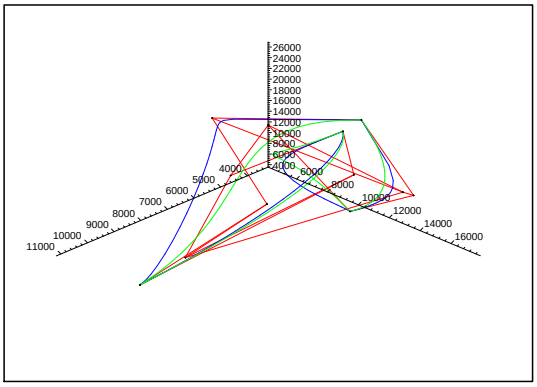


Figure 7: *Polynomial and rational B-spline surfaces presented in example 7*

the same knot vector and control points for both polynomial and rational entities). These operations are the following ones (see for instance [11]):

- Numerical generation of polynomial and rational B-spline curves.
- Computation of the B-spline curve derivatives, using:
  - basis function method
  - control point method.
- Knot vector refinement.
- Bezier decomposition.
- Approximation of rational B-spline curves with polynomial ones.

**Example 8** *Taking into acount definition 4, we consider the following parameters defining a polynomial B-spline curve and the following weight vector defining the corresponding rational B-spline curve:*

```
> read "manipBspline.txt";

Knots:=[0.,0.,0.,0.,0.,0.25,0.5,0.75,1.,1.,1.,1.,1.]:
order:=5:
nrPoints:=nops(Knots)-order:
Points:=[[0.,10.,10.,20.,30.,40.,25.,15.],
         [0.,20.,30.,35.,35.,25.,5.,10.],
         [0.,5.,5.,10.,10.,20.,5.,15.]]:
```

```
> Weights:=[]: for i from 1 to nrPoints do Weights:=[op(Weights),
evalf(rand()/10**12)] od: Weights;
```

[.7924959004, .7512095393, .6283634430, .3137460865, .005862664913, .07481365622, .6438424438, .1319057546]

For computing a point on a polynomial B-spline curve at a fixed value $u$, three steps are required:

1. Find the knot span in which $u$ lies.
2. Compute the non-zero basis functions.
3. Multiply the values of the non-zero basis functions with the corresponding control points.

In the rational case the algorithm is similar. Applying several times these algorithms and using afterwards the predefined `Maple` interpolation function `plottools[curve]`, we generate the curves (polynomial and rational one) which correspond to the anterior data and display them together with the control polygon using our `Maple` procedures `displayPol` and `displayRac`. In figure 8 the control polygon appears in red, the polynomial B-spline curve in green and the rational B-spline curve in blue.

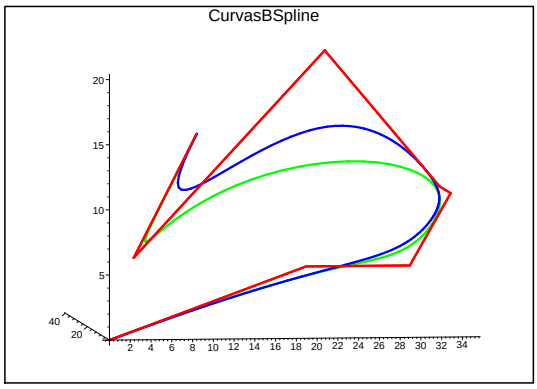


Figure 8: *Polynomial and rational B-spline curves presented in example 8*

It is important to mention that in the algorithms we have used, especially for the computation of the rational B-spline derivatives (which are complicated and involve denominators to high powers), we have represented the

rational B-spline entities in homogeneous coordinates, working with polynomial B-spline entities in four dimensions.

Let us start with a 3-dimensional point $P = (x, y, z)$. Then $P$ is written as $P^w = (wx, wy, wz, w) = (X, Y, Z, W)$ in 4-dimensional space, $w \neq 0$. $P$ is obtained from $P^w$ by dividing all coordinates by the fourth coordinate, $W$ (i.e. by mapping $P^w$ from the origin to the hyperplane $W = 1$). This mapping, denoted by $H$, is a perspective map with center at the origin.

$$P = H(P^w) = H((X, Y, Z, W)) = \begin{cases} (\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}) & , W \neq 0 \\ direction(X, Y, Z) & , W = 0 \end{cases}$$

Now for a given set of control points $\{P_i\} = \{(x_i, y_i, z_i)\}$ and weights $\{w_i\}$, we construct the weighted control points $P_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$ and then define the polynomial B-spline curve in 4-dimensional space

$$C^w(u) = \sum_{i=0}^{n} B_{i,N}(u) P_i^w.$$

Thus, rational B-spline entities can be processed in 4-dimensional space and the results are located in 3-dimensional space using the map $H$.

**Example 9** *Now we compute the first and the second derivatives of the curves, firstly with the basis function method and then with the control point method, at some points randomly generated.*

The basis funtion method consists basically in implementing an algorithm using the following formula:

$$C^{(k)}(u) = \sum_{i=0}^{n} B_{i,p}^{(k)}(u) P_i.$$

The control point method consists in representing the derivative of a polynomial B-spline curve as a polynomial B-spline curve of lower order:

$$C^{(k)}(u) = \sum_{i=0}^{n-k} B_{i,p-k}(u) P_i^{(k)}$$

$$\text{with} \qquad P_i^{(k)} = \begin{cases} P_i & , k = 0 \\ \frac{p-k+1}{u_{i+p+1} - u_{i+k}} (P_{i+1}^{(k-1)} - P_i^{(k-1)}) & , k > 0 \end{cases}$$

$$\text{and} \qquad U^{(k)} = \{\underbrace{0, \ldots, 0}_{p-k+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{1, \ldots, 1}_{p-k+1}\}.$$

The results are displayed as follows:

$$[[C_x(u), C_y(u), C_z(u)], [\frac{\partial C_x}{\partial u}, \frac{\partial C_y}{\partial u}, \frac{\partial C_z}{\partial u}], [\frac{\partial^2 C_x}{\partial u^2}, \frac{\partial^2 C_y}{\partial u^2}, \frac{\partial^2 C_z}{\partial u^2}]]$$

where $C_x(u)$, $C_y(u)$ and $C_z(u)$ represent the three components of the curve $C(u)$. After each operation the time is displayed.

```
> computeDerivatives(order,nrPoints,Knots,Weights,Points);


Argument:   .8458933315


Polynomial derivative BF: [[32.05687097, 17.50040607, 12.54996778],
[-54.43263906, -102.3150315, -37.93511739], [-749.8846345,
-41.0448997, -154.5126604]]


Time :   .10e-1


Polynomial derivative CP: [[32.05687097, 17.50040607, 12.54996778],
[-54.43263912, -102.3150317, -37.93511741], [-749.8846338,
-41.0448993, -154.5126601]]


Time :   .20e-1


Rational derivative BF: [[29.11414329, 13.78607835, 9.128659526],
[-37.30823064, -110.0511479, -37.68651421], [12.73743261,
732.2496156, 21.86782408]]


Time :   .260


Rational derivative CP: [[29.11414329, 13.78607835, 9.128659526],
[-37.30823073, -110.0511479, -37.68651425], [12.73743218,
732.2496156, 21.86782409]]


Time :   .20e-1
```

A very important operation with B-splines is the knot vector refinement. To state the problem, let $C^w(u) = \sum_{i=0}^{n} B_{i,N}(u) P_i^w$ be defined on the knot vector $U = \{u_0, \ldots, u_m\}$ and let $X = \{x_0, \ldots, x_r\}$ satisfy $x_i \leq x_{i+1}$ and $u_0 < x_i < u_m$ for all $i$. The elements of $X$ are to be inserted in $U$ and the corresponding new set of control points $\{Q_i^w\}, i = 0, \ldots, n + r + 1$ is to be computed. New knots $x_i$ should be repeated in $X$ with their multiplicities. The principal ideas of the refinement algorithm are the following ones:

- Find indices $a$ and $b$ such that $u_a \leq x_j < u_b$ for all $j$.
- Compute the new control points:
  - The control points $P_0^w, \ldots, P_{a-p}^w$ and $P_{b-1}^w, \ldots, P_n^w$ do not change.
  - The remaining control points are computed using the formula:

$$Q_{i,r}^w = \begin{cases} P_i^w & , r = 0 \\ \alpha_{i,r} Q_{i,r-1}^w + (1 - \alpha_{i,r}) Q_{i-1,r-1}^w & , r > 0 \end{cases}$$

$$\text{where} \qquad \alpha_{i,r} = \begin{cases} 1 & i \leq k - p + r - 1 \\ \frac{x_j - u_i}{u_{i+p-r+1} - u_i} & k - p + r \leq i \leq k - s \\ 0 & i > k - s + 1 \end{cases}$$

  and $s$ represents the multiplicity of $x_j$.

The applications of the knot refinement operation include:

- Decomposing the B-spline entities into its constituent Bezier components.
- Merging two or more knot vectors in order to obtain a set of curves defined on one common knot vector.
- Obtaining polygonal and polyhedral approximations for the B-spline curves and surfaces respectively. Refining knot vectors brings the control polygon (polyhedra) closer to the curve (surface), and in the limit the polygon (polyhedra) converges to the curve (surface).

**Example 10** *We will refine now our B-spline curves using the interior knots* $[0.4, 0.7, 0.7, 0.7]$. *After the refinement operation, we will redisplay the curves, together with the new control polygon.*

```
NewKnots:=[0.4,0.7,0.7,0.7]:
dib:=refine(order,nrPoints,Knots,Weights,Points,NewKnots):
plots[display]([op(dibPol),op(dib[1])],axes=NORMAL,
orientation=[7,96]);
plots[display]([op(dibRac),op(dib[2])],axes=NORMAL,
orientation=[7,96]);}
```

The refined curves are displayed in figure 9a,b. In figure 9a the polynomial B-spline curve is displayed in grey, the initial control polygon in orange and the final control polygon in red meanwhile in figure 9b the rational

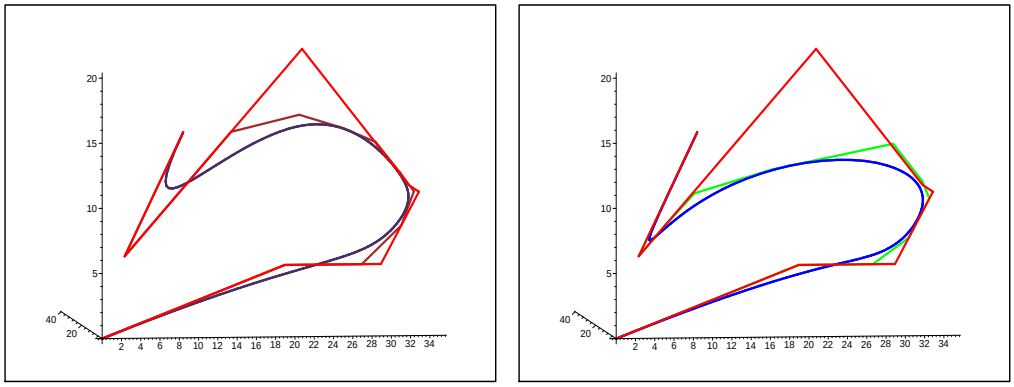

Figure 9: *Refined (a) polynomial and (b) rational B-spline curve*

B-spline curve is displayed in blue, the initial control polygon in orange and the final control polygon in green.

An important application of the knot vector refinement is the problem of decomposing a rational B-spline curve into its 4-dimensional polynomial segments. This operation is required for converting rational B-splines into other spline forms, for instance into an entity type in IGES format, the IGES spline parametric curve, Entity type 112 ( [9]). In order to perform this conversion, the first step consists in decomposing the curve into its Bezier components. The control points of the Bezier segments are obtained by inserting each interior knot until it has multiplicity equal to the order.

In our case, the inital curves (polynomial anr rational one) are decomposed into its Bezier components, which are displayed together with the corresponding control polygons.

```
> BezierPol:=BezCompPol(order,nrPoints,Knots,Points):
  BezierRac:=BezCompRac(order,nrPoints,Knots,Weights,Points):

> dib:=displayBezComp(order,BezierPol,BezierRac):
 plots[display]([op(dib[1])],axes=NORMAL);
 plots[display]([op(dib[2])],axes=NORMAL);
```

In figure 10a,b are displayed the Bezier components of the B-spline curves (polynomial anr rational respectively). The control polygons are displayed in orange and the four Bezier components (in each case) are displayed in a different color.

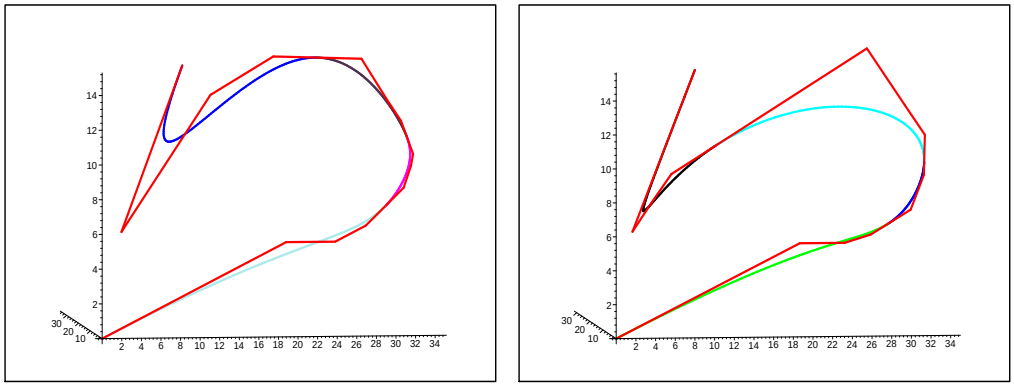

Figure 10: *(a) Polynomial and (b) rational Bezier curve components*

Another important operation for the rational B-spline curves is the approximation with polynomial B-spline curves. This operation is necessary for two reasons (which apply also for the surface case that will be studied in the next section):

- It is frequently necessary to interchange information between different working systems. These systems accept information in formats such as IGES or VDA, but IGES (see [9]) incorporates for curves and surfaces the rational B-spline representation, meanwhile VDA (see [12]) incorporates the polynomial representation in power basis form.
- During the process of implicitation, which could imply complicated computations, it is useful to firstly approximate the rational B-spline entities with polynomial ones, in order to obtain entities easier to implicitate.

The approximation algorithm consists of the following steps [10]:

1. Certain continuity conditions are imposed at the exterior points in order to determine some control points.
2. The remaining control points are determined by interpolation.
3. The differences between the curves (the initial one and the approximating one), the tangent vectors and the curvatures are evaluated at some adecuately choosen points and tested against the error parameters.
4. If the differences are not small enough, new internal knots are added and a new iteration begins, starting with the computation of the new control polygon.

```
> read "aprox.txt": dib:=aproxBezComp(order,BezierRac):
  plots[display]([op(dib[1]),op(dib[2])],axes=NORMAL,
  orientation=[7,96]);
```

The result of the approximation process is presented in figure 11: the rational B-spline curve is displayed in blue, the initial control polygon in red and the new control polygon (corresponding to the polynomial B-spline representation of our curve) in green.

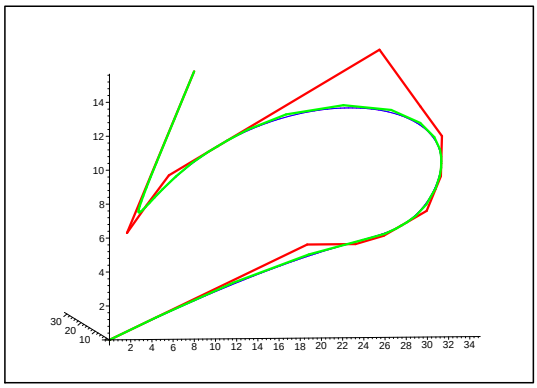


Figure 11: *B-spline curve approximation*

## 5. Polynomial and Rational B-Spline Surface Manipulation

In this section it is shown how `Maple` can perform, in a very efficient way, several operations with polynomial and rational B-spline surfaces. We will present the same operations as in the previous section:

- Numerical generation of polynomial and rational B-spline surfaces.
- Computation of the B-spline surface derivatives, using:
  - basis function method
  - control point method.
- Knot vector refinement.
- Bezier decomposition.

- Approximation of rational B-spline surface with polynomial ones.

**Example 11** *Taking into acount definition 5, we consider the following parameters defining a polynomial B-spline surface (the control points are read from the file "datos.txt") and the weights defining the corresponding rational B-spline surface:*

```
> read "manipBsplineSur.txt";
```

```
> KnotsU:=[0.,0.,0.,0.,0.,0.,0.25,0.25,0.25,0.25,
0.75,0.75,0.75,0.75,1.,1.,1.,1.,1.,1.]:
orderU:=6:nrPointsU:=nops(KnotsU)-orderU:
KnotsV:=[0.,0.,0.,0.,0.,0.,0.3,0.3,0.3,0.3,0.6,
0.6,0.6,0.6,1.,1.,1.,1.,1.,1.]: orderV:=6:
nrPointsV:=nops(KnotsV)-orderV;

> fis:=fopen(‘datos.txt‘, READ): PointsX:=[]: PointsY:=[]:
PointsZ:=[]:
 for i from 1 to nrPointsV do
  ListaX:=[]; ListaY:=[]; ListaZ:=[];
  for j from 1 to nrPointsU do
   ListaX:=[op(ListaX), fscanf(fis, \%f)[1]];
   ListaY:=[op(ListaY), fscanf(fis, \%f)[1]];
   ListaZ:=[op(ListaZ), fscanf(fis, \%f)[1]];
  od:
  PointsX:=[op(PointsX), ListaX];
  PointsY:=[op(PointsY), ListaY];
  PointsZ:=[op(PointsZ), ListaZ];
 od:
 fclose(fis): Points:=[PointsX, PointsY,PointsZ]:

> Weights:=[]: for i from 1 to nrPointsV do
  Lista:=[];
  for j from 1 to nrPointsU do
   Lista:=[op(Lista), evalf(rand()/10**12)] od:
  Weights:=[op(Weights), Lista] od:
```

Five steps are required to compute a point on a polynomial B-spline surface at fixed $(u, v)$ parameter values:

1. Find the knot span in which $u$ lies, say $u \in [u_i, u_{i+1})$.

2. Compute the non-zero basis functions $B_{i-p,p}(u), \ldots, B_{i,p}(u)$.

3. Find the knot span in which $v$ lies, say $v \in [v_j, v_{j+1})$.

4. Compute the non-zero basis functions $B_{j-q,q}(v), \ldots, B_{j,q}(v)$.

5. Multiply the values of the non-zero basis functions with the corresponding control points.

In the rational case the algorithm is similar. Applying several times these algorithms and using afterwards the predefined `Maple` interpolation function `plottools[curve]`, we generate the border curves of the B-spline surfaces (polynomial and rational one) which correspond to the anterior data and display them together with the control polyhedra using our `Maple` procedures `displaySurPol` and `displaySurRac`. In figure 12a the control polyhedra appears in red and the border curves of the polynomial B-spline surface in green, meanwhile in figure 12b the control polyhedra appears in magenta and the border curves of the rational B-spline surface in blue.

```
> dibPol:=displaySurPol(orderU,orderV,nrPointsU,nrPointsV,
KnotsU,KnotsV,Points,0.,1.,0.,1.,'red','green'):
dibRac:=displaySurRac(orderU,orderV,nrPointsU,nrPointsV,
KnotsU,KnotsV,Weights,Points,0.,1.,0.,1.,'blue','magenta'):
plots[display]([op(dibPol)], axes=NORMAL);
plots[display]([op(dibRac)], axes=NORMAL);
```

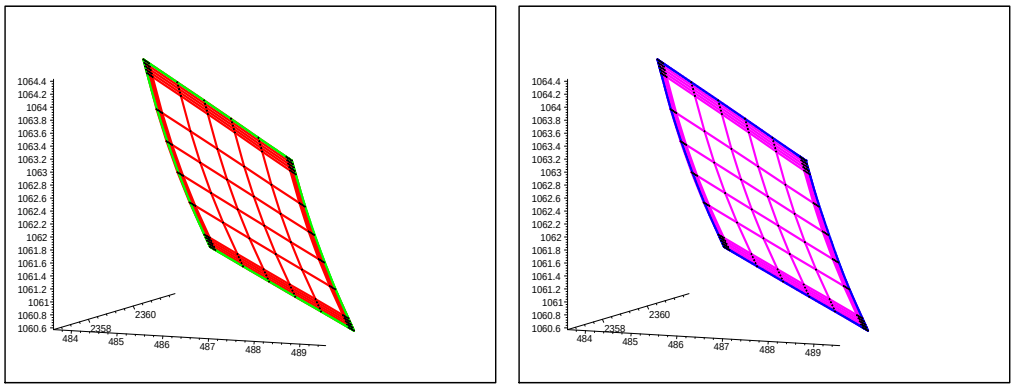


Figure 12: *(a) Polynomial and (b) rational B-spline surfaces presented in example 11*

Let denote by $S_{6,6}$ our polynomial B-spline surface and by $R_{6,6}$ our rational B-spline surface. Now we will compute the derivatives of total order at most 2:

$$S_{6,6}, \frac{\partial S_{6,6}(u,v)}{\partial u}, \frac{\partial S_{6,6}(u,v)}{\partial v}, \frac{\partial^2 S_{6,6}(u,v)}{\partial u^2}, \frac{\partial^2 S_{6,6}(u,v)}{\partial u \partial v}, \frac{\partial^2 S_{6,6}(u,v)}{\partial v^2},$$

where $S_{6,6}$ is one of our B-spline surfaces (polynomial or rational), in five points randomly generated, using two methods: the basis function method (used in the `Maple` procedures `DerivPolSurBF` and `DerivRacSurBF`) and the control point method (used in the `Maple` procedures `DerivPolSurCP` and `DerivRacSurCP`) (for details about these methods see section 3). In the case of the basis function method, the results are displayed as follows:

$$\left[S_{6,6}, \frac{\partial S_{6,6}(u,v)}{\partial u}, \frac{\partial S_{6,6}(u,v)}{\partial v}, \frac{\partial^2 S_{6,6}(u,v)}{\partial u^2}, \frac{\partial^2 S_{6,6}(u,v)}{\partial u \partial v}, \frac{\partial^2 S_{6,6}(u,v)}{\partial v^2}\right]$$

meanwhile in the case of the basis function method, the results are displayed in this order:

$$\left[S_{6,6}, \frac{\partial S_{6,6}(u,v)}{\partial v}, \frac{\partial S_{6,6}(u,v)}{\partial v^2}, \frac{\partial^2 S_{6,6}(u,v)}{\partial u}, \frac{\partial^2 S_{6,6}(u,v)}{\partial u \partial v}, \frac{\partial^2 S_{6,6}(u,v)}{\partial u^2}\right].$$

Part of the output has been deleted.

```
> Digits:=16: for i from 1 to 5 do argU:=evalf(rand()/10**12);
argV:=evalf(rand()/10**12); timp3:=time();
derPolFB:=DerivPolSurFB(argU,argV,orderU,nrPpointsU,
orderV,nrPointsV,KnotsU,KnotsV,Points);
timp3:=time()-timp3;timp4:=time();
derPolPC:=DerivPolSurPC(argU,argV,orderU,nrPpointsU,
orderV,nrPointsV,KnotsU,KnotsV,Points,2);
timp4:=time()-timp4;timp1:=time();

> derRacFB:=DerivRacSurFB(argU,argV,orderU,nrPpointsU,
orderV,nrPointsV,KnotsU,KnotsV,Weights,Points);
timp1:=time()-timp1; timp2:=time();
derRacPC:=DerivRacSurPC(argU,argV,orderU,nrPpointsU,
orderV,nrPointsV,KnotsU,KnotsV,Weights,Points,2);
timp2:=time()-timp2; lprint(‘Argumets:‘,argU,argV);

> lprint(‘Polynomial Derivatives BF:‘); for j from 1
tonops(derPolFB) do lprint(derPolFB[j]) od;
lprint(‘Time‘,timp3);lprint(‘Polynomial Derivatives CP:‘); for j
```

```
from 1 to nops(derPolPC) do lprint(derPolPC[j]) od;
lprint(‘Time‘,timp4);lprint(‘Rational Derivatives BF: ‘); for j
from 1 to nops(derRacFB) do lprint(derRacFB[j]) od;
lprint(‘Time‘,timp1);lprint(‘Rational Derivatives CP: ‘); for j
from 1 to nops(derRacPC) do lprint(derRacPC[j])od;
lprint(‘Time‘,timp2); od: Digits:=10:
```

```
Arguments:    .2343384383590000   .4638661136260000
Polynomial Derivatives BF:
[2359.518940493465, 487.0311374040819, 1061.277007256592]
[-1.6944287778418, -.87185075402776, 2.7732192030585]
[-10.7077915869695, 14.11671900866031, -1.9524893749358]
[-18.621023679972, -9.5753667293827, 30.463574828936]
[.601405722411, .201018274149, -.807455668135]
[6.569414433638, -9.286303630796, 1.756269732548]
Time:   .110
Polynomial Derivatives CP:
[[2359.518940493467, 487.0311374040819, 1061.277007256593],
[-10.70779158696980, 14.11671900866196, -1.952489374935699],
[6.569414433575617, -9.286303630814925, 1.756269732523466]]
[[-1.694428777840756, -.8718507540276519, 2.773219203059553],
[.6014057224017089, .2010182741482566, -.8074556681440215]]
[[-18.62102368007849, -9.575366729395748, 30.46357482892405]]
Time:    .520
```

Now we will refine the knot vectors, using the following interior knot: $[0.1, 0.5, 0.9]$ in the $u$ direction and $[0.2, 0.5, 0.8]$ in the $v$ direction. The refinement algorithm consisits basically in :

- Applying a $U$ knot refinement (by applying the refinement algorithm used in the curve case to all the columns of control points).
- Applying a $V$ knot refinement, by applying the refinement algorithm used in the curve case to all the lines of control points. The algorithm is organized so that redundant operations are eliminated.

Finally we display the obtained "new" surfaces, together with the new control polyhedras. In figure 13a the border curves of the polynomial B-spline surface are displayed in blue and the control polyhedra in red meanwhile in figure 13b the border curves of the rational B-spline surface are displayed in red and the control polyhedra in blue.

```
> NewKnotsU:=[0.1,0.5,0.9]: NewKnotsV:=[0.2,0.5,0.8]:
refinePol(orderU,orderV,nrPointsU,nrPointsV,KnotsU,
KnotsV,Points,NewKnotsU,NewKnotsV);
refineRac(orderU,orderV,nrPointsU,nrPointsV,KnotsU,
KnotsV,Weights,Points,NewKnotsU,NewKnotsV);
```

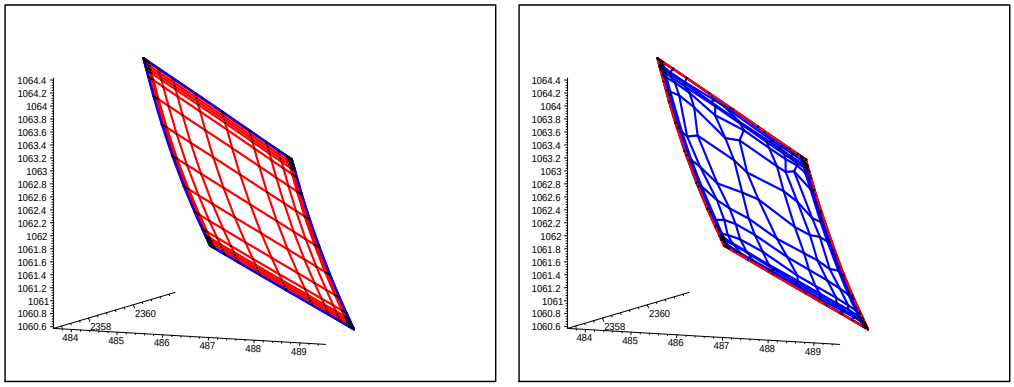


Figure 13: *Refined (a) polynomial and (b) rational B-spline surface*

We decompose now the B-spline surfaces into their Bezier components (segments). The algorithm [11] we have implemented computes a Bezier strip (i.e. a B-spline surface which is Bezier in one direction and B-spline in the other). The procedure has to be called twice, once in the $u$ direction to get the Bezier strips and then the strips must be fed into the procedure in the $v$ direction to get the Bezier patches. Finally we display the inital surfaces (polynomial and rational one) together with the corresponding control polyhedras.

```
> BezierPol:=PatchesBezPol(orderU,orderV,nrPointsU,
nrPointsV,KnotsU,KnotsV,Points):
dib:=CompBez(orderU,orderV,BezierPol):
plots[display](dib,axes=NORMAL,orientation=[-24,99]);
BezierRac:=PatchesBezPol(orderU,orderV,nrPointsU,
nrPointsV,KnotsU,KnotsV,Weights,Points):
dib:=CompBez(orderU,orderV,BezierRac):
plots[display](dib,axes=NORMAL,orientation=[-24,99]);
```

In figure 14a,b are displayed the Bezier components of the B-spline surfaces (polynomial and rational respectively). The control polygons are displayed in red and magenta respectively and the nine Bezier components (in each case) are displayed in a different color.

The rational B-spline surfaces are approximated with polynomial B-spline ones. The approximation algorithm [1] is similar to the curve approximation algorithm, described in the previous section:

1. The border curves are approximated (using the curve approximating algorithm) and the outermost control points are generated.

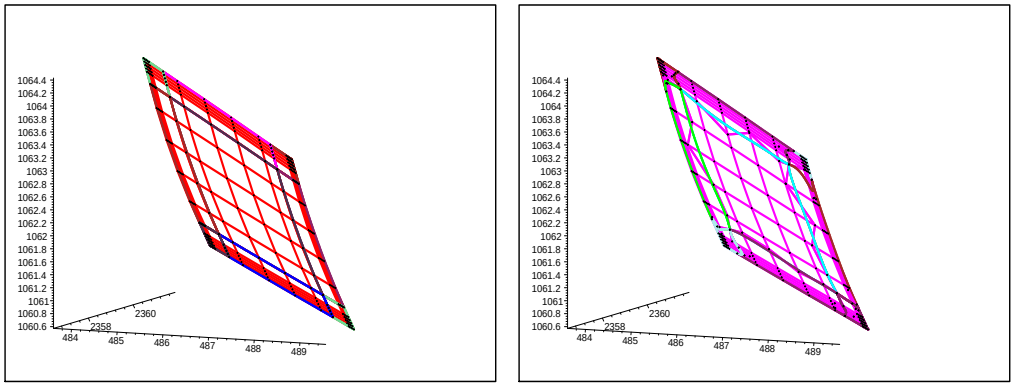


Figure 14: *(a) Polynomial and (b) rational Bezier surface components*

2. The remaining control points are determined by interpolation.

3. The differences between the surfaces (the initial one and the approximating one), the normal vectors and the curvatures are evaluated at some adecuately choosen points and tested against the error parameters.

4. If the differences are not small enough, new internal knots are added and a new iteration begins, starting with the computation of the new control polygon.

Three steps of the algorithm are implemented in `Maple` (steps 1,3 and 4) and step 2 is implemented in `Matlab`, because it requires the resolution of linear equation systems of big size ([Golu89]).

The C-shell presented immediately after realize the following operations:

- Reading IGES file (`Maple`)
- Approximating the rational B-spline surfaces with polynomial B-spline ones (`Maple` and `Matlab`)
- Generating output IGES file (`Maple`)

```
maple < varia20
cp nuevo.txt end.txt
while (-z end.txt)
   cp nuevo.txt finish.txt
   maple < varia2I
```

```
   while (-z finish.txt)
      cp nuevo.txt final.txt
      maple < varia211
      matlab < varia2Mat
      maple < varia221
      while (-z final.txt)
         maple < varia231
         matlab < varia2Mat
         maple < varia241
      end
   end
   maple < varia2F
end
cat Info.txt Out.txt > ficheroIGESOut.txt
maple < dibujarSup.txt
ghostview dibujos.ps&
```

After the execution of this C-shell, we obtain the results presented in figure 15, where there are displayed all the rational B-spline surfaces contained in the considered IGES file, together with the corresponding polynomial B-spline surfaces, which aproximate them.

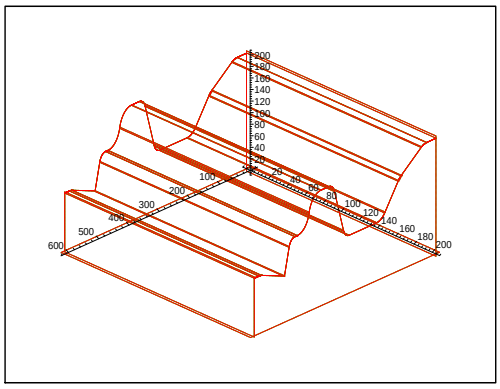


Figure 15: *Surface approximation*

# References


[1] **L. Bardis, N. Patrikalakis**; *Approximate conversion of rational B-spline patches*, Computer Aided Geometric Design, 6, 189–204 (1989).

[2] **C. de Boor**; *On calculating with B-splines*, J. Approx. Theory, 6, 50–62 (1972).

[3] **C. de Boor**; *A Practical Guide to Splines*, Springer, New York (1978).

[4] **G. Golub, C. van Loan**; *Matrix Computations*, Johns Hopkins University Press, London (1996).

[5] **Laureano González Vega, M'hammed El Kahoui**; *An Improved Upper Complexity Bound for the Topology Computation of a Real Algebraic Plane Curve* , Journal of Complexity, 12, 527–544 (1996).

[6] **Laureano González Vega, Ioana Necula**; *Approximate Implicitation of Rational Surfaces* , 1998 IMACS Conference on Applications of Computer Algebra, Praga (1998).

[7] **Laureano González Vega, Ioana Necula**; *Computing the Topology of Implicit Algebraic Plane Curves. Applications to CAGD* , Preprint.

[8] **J. Hoschek, F.J. Schneider**; *Aproximate Conversion for Integral and Rational Bezier and B-spline surfaces*, R. E. Barnhill (ed.), Geometry Processing for Design and Manufacturing, SIAM, p.45-86.

[9] **IGES/PDES Organization**; *The Initial Graphics Exchange Specification (IGES) Version 5.1.* , National Computer Graphics Association, Virginia-USA (1991).

[10] **N. Patrikalakis**; *Approximate conversion of rational splines*, Computer Aided Geometric Design, 6, 155–165 (1989).

[11] **L. Piegl, W. Tiller**; *The NURBS book*, Springer-Verlag, Berlin (1997).

[12] **VDA Working Group 'CAD/CAM'**; *VDA Surface, Data Interface (VDAFS) Version 2.0. Verband der Automibilindustrie e.v. (VDA)* , Frankfurt (1987).



Departamento de Matemáticas, Estadística y Computación
Universidad de Cantabria, Spain
{gvega,ioana,sevillad}@matesco.unican.es