A Mathematica Package for CAGD and Computer Graphics

Andrés Iglesias[1], **Flabio Gutiérrez**[1,2] and **Akemi Gálvez**[1]
[1]Departament of Applied Mathematics and Computational Sciences
University of Cantabria, Santander, Spain
[2]Departament of Mathematics, Sciences Faculty
National University of Piura, Peru
iglesias@ccaix3.unican.es, flabio@tallan.unp.edu.pe, uc8031@cclx1.unican.es

Abstract

This paper introduces a Mathematica package for learning and teaching Computer-Aided Geometric Design (CAGD). The package incorporates commands for the treatment of the most usual curves and surfaces in CAGD (Bézier, B-spline, rationals, etc.). The powerful symbolic and graphical Mathematica capabilities, the functional and pattern recognition programming and the Mathematica visualization environment have been extensively applied to get a user-friendly, didactic and powerful tool for education, with applications in areas as CAGD and Computer Graphics. Some illustrative examples showing the performance of the package are also given.

Keywords: *Symbolic programming, visualization, computer graphics, CAGD.*

1. Introduction

Computer graphics and geometric modeling constitute fundamental parts of the engineering design process. Geometric modeling methods are applied to construct a precise mathematical description of the shape of a real object and its efficient computer representation. On the other hand, computer graphics capture the visual and quantitative aspects of this object. They are also essential topics in the engineering education process (see [1]).

1. In the last few years, engineering teachers have witnessed an extraordinary change in the way of teaching and developing students' skills for their professional future. The extraordinary advances in hardware and software have made possible the appearance of a new generation of scientific Symbolic Computation Programs (SCPs), as Mathematica or Maple. In comparison with the traditional program-ming languages, (Fortran, Pascal, C, etc.) the SCPs incorporate a great number of symbolic, graphical and programming facilities. Today, teachers can take advantage of these graphical capabilities when introducing students to computer graphics and/or visualizing the geometry of the objects under analysis. In addition of this, as we show below, SCP symbolic tools allow to get not only a graphical output but also a mathematical expression for curves and surfaces.

Mathematica [2] has emerged as the most popular and widely used symbolic computation program. In this paper, we present a Mathematica package, `CAGD.m`, for learning and teaching CAGD and Computer Graphics. The package incorporates many commands for the treatment of the most usual curves and surfaces in CAGD (Bézier, B-spline, rationals, etc...). Some classical geometric operations with interesting applications to real-world problems have also been performed. In the following paragraphs, we describe the main features of the package and show some interesting and illustrative examples of its application.

2. Description of the Package

Tha package `CAGD.m` to be introduced in this paper has been implemented in the Mathematica programming language. We think that Mathematica is an ideal environment for developing programs for computer graphics, because of its powerful programming language (supporting procedural, functional, based-on rules, based-on pattern recognition and object-oriented programming), and because it is easy to understand, it reproduces many C language structures, it has several remarkable graphical capabilities and, mainly, it works *in a symbolic way*. This important feature will be used later to obtain the symbolic expression of some curves and surfaces.

2.1. How Does the Package Works?

Perhaps the best way to understand how the package has been built is to take a look to some of the routines it incorporates. For example, let $\mathbf{P}=\{\mathbf{P}_0,\mathbf{P}_1,...,\mathbf{P}_n\}$ be a set of points in $\mathbf{R}^d$, d=2,3 ($\mathbf{R}$ meaning the set of real numbers). As already know (see [3]) the Bézier curve associated with the set $\mathbf{P}$ is defined by:

$$\sum_{i=0}^{n} \mathbf{P}_i B_{i,n}(t) \qquad (1)$$

where $B_{i,n}(t)$ represents the Bernstein polynomials, which are given by:

$$B_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}t^i \quad i=0,\dots,n \qquad (2)$$

n being the polynomial degree. The points $\mathbf{P}_i$ are called *control points* or *Bézier points* of the curve. These expressions can be easily translated to Mathematica. For example, Equation 2 becomes:

```
Bernstein[i_,n_,v_]:=
 Binomial[n,i]*(1-v)^(n-i)*v^i
```

Table 1. Code for the Bernstein polynomials.

while Equation 1 takes the simple form:

```
BezierCurve[pt:{{_,_}..}|{{_,_,_}..},v_]:=
 Module[{n=Length[pts]-1},
 Simplify[
 Table[Bernstein[i,n,v],{i,0,n}].pt
 ]
 ];
```

Table 2. Code for the BézierCurve command.

Let us analyze the previous code: in line 1 we introduce the variables, namely `pt` indicating the (two- or three-dimensional) points, and `v` the variable of the curve. After obtaining the curve degree (line 2), we translate Eq. 1 to Mathematica. Thus, line 4 corresponds to the scalar product of two vectors: the table with the Bernstein functions and the vector of points. Finally, the result is simplified.

From the previous codes, it becomes clear that:

1. Mathematical expressions (as Eqs. 1 and 2) can be easily implemented in Mathematica; in most cases, this process consists of a simple translation of these expressions to its programming language.
2. Note also that the previous code allows us to work in a symbolic way. For example, after loading the package:

   ```
   In[1]: <<CAGD.m
   ```

   we can obtain the following mathematical expressions for the Bernstein polynomial and the Bézier curve:

   ```
   In[2]: Bernstein[1,3,t]
   ```
   Out[2]: $3(1-t)^2\, t$

   Fig. 1 shows the Bernstein functions of degree 4:

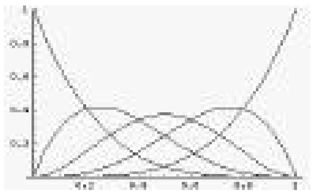


Figure 1. Bernstein polynomials for n=4.

```
In[3]: pts={{1,1},{2,4},{3,0},
 {4,3},{5,-1},{6,0}};
In[4]: BezierCurve[pts,t]
```
Out[4]: $\{1+5t,\ 1+15t-70t^2+140t^3-140t^4+54t^5\}$

We remark here that we have obtained the symbolic formula for the Bézier curve. This fact represents a clear advantage with respect to the traditional programming languages. These mathematical formulas are exact, because they can be obtained by, more or less complex, symbolic operations. Since many computer graphics methods are very susceptible to round-off errors, this capability is specially valuable.

Plotting the corresponding curve reduces to this simple input:

```
In[5]: PlotBezierCurve[pts,t]
```
Out[5]: Figure 2(left).

Figure 2(left) shows a Bézier curve, its Bézier points and the control polygon (a set of segments joining the control points of the curve).

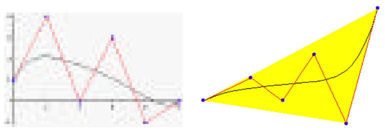


Figure 2. (left) A Bézier curve; (right) Convex hull property of a Bézier curve

An interesting property of the Bézier curves is the *convex hull property*: the curve lies entirely within the convex figure set by the extreme points of the polygon determined by the control points (see [1, 4, 6]). This fact is illustrated in Figure 2(right), where the grey area indicates such a convex figure. To get this figure, we introduce the control points and obtain the plot of the corresponding Bézier curve:

```
In[6]: p1={{1/2,1},{2,3/2},
 {3,1},{4,2},{5,1/2},{6,3}};
In[7]: cur=PlotBezierCurve[p1,t];
```

Then, we load the Mathematica package for calculating the convex hull of the given points. Finally, we display the resulting area (in grey) and the curve:

```
In[8]: <<DiscreteMath`
 ComputationalGeometry`
```

```
In[9]: ConvexHull[puntos];
```

```
In[10]: Show[
 {Graphics[
 {RGBColor[1,1,0],
 Polygon[%]}],cur}
 ]
```

Out[10]: Figure 2(right)

In computer graphics, curves and surfaces are usually dealt with by using numerical routines, which are required to be as short, fast, elegant, numerically stable and general as possible. For example, the problem to evaluate a Bézier curve at a given parameter value $t=t_0$ [0,1] (say t=0.3) is solved by applying the de Casteljau algorithm (see [3], Chapter 4). However, in our case this problem just reduces to compute:

```
In[11]: BezierCurve[pts,0.3]
```

Out[11]: {2.5,1.97722}

3. We have used pattern recognition to identify the dimension of the points through the pattern `{_,_}` or `{_,_,_}`. Moreover, the pattern `{{_,_}..}` (`{{_,_,_}..}` resp.) indicates that one or more pairs (triplets resp.) are expected. Otherwise, no evaluation is performed:

```
In[12]:r={{1,1},{2,4,3,2},{4,3}};
```

```
In[13]: BezierCurve[r,t]
```

Out[13]: BezierCurve[{{1,1},{2,4,3,2},{4,3}},t]

Finally, the operator "`|`" (line 1 in Table 2) is used to indicate "this pattern" or "this another one". This means that the same commands are able to work with two- and three-dimensional curves and surfaces:

```
In[14]: PlotBezierCurve[{{1,4,4},
 {2,2,3},{3,0,3},{3,4,0},
 {4,5,2}},t]
```

Out[14]:

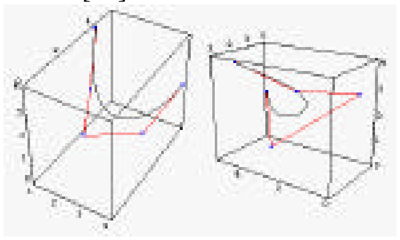


Figure 3. Two different viewpoints of a three-dimensional Bézier curve.

4. The graphical Mathematica capabilities can be successfully applied to visualization. They include the rendering of parametric, implicit and explicit curves and surfaces, colors, grids, different points of view (see Figure 3), shading, shadows, intersection algorithms, etc. In addition of this, the package incorporates many other options, as two- and three-dimensional transformations, projections, perspectives, visible lines and surfaces techniques, etc.

We remark that the previous examples have been chosen primarily for simplicity and illustrative purposes, rather than to correspond to valuable codes or very complicated algorithms. However, very similar discussions can be applied to many other commands. As an illustration, Table 3 shows the corresponding code for a Bézier surface. In this code, we have used the functional programming, a powerful alternative to the standard procedural programming. Note that the pattern recognition and functional Mathematica programming avoid conditionals and loops, respectively. Motivated by the different kinds of programming that Mathematica can support, the final syntax of our commands is very short, elegant and easy to understand.

```
BezierSurf[pts:{{{_,_,_}..}..},v1_,v2_]:=
 Module[{m=Length[pts]-1,U,V,
 n=Length[First[pts]]-1},
 {U,V}=MapThread[Table[
 Bernstein[i,#1,#2],{i,0,#1}]&,
  {{m,n},{var1,var2}}
  ];
       Plus @@ (U.pts*V) //Simplify
 ]
```

Table 3. Code for the Bézier surface.

x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
0	0	1	0	1	2	0	2	3	0	3	3	0	4	2	0	5	1
1	0	2	1	1	3	1	2	4	1	3	4	1	4	3	1	5	2
2	0	3	2	1	4	2	2	5	2	3	5	2	4	4	2	5	3
3	0	3	3	1	4	3	2	5	3	3	5	3	4	4	3	5	3
4	0	2	4	1	3	4	2	4	4	3	4	4	4	3	4	5	2
5	0	1	5	1	2	5	2	3	5	3	3	5	4	2	5	5	1

Table 4. Set of 3-D points of control.

The following picture shows a typical output for the `BezierSurf` command with the control points given in Table 4 and stored in the variable `pts3d`:

```
In[15]: BezierSurf[pts3d,{u,v}]
```

Out[15]: {4 v,2 u,2 v (3+(-3+u) v)}

The corresponding patch surface is shown in Figure 4(left):

```
In[16]:PlotBezierSurf[ptos3d,{u,v}]
```

Out[16]: Figure 4(left)

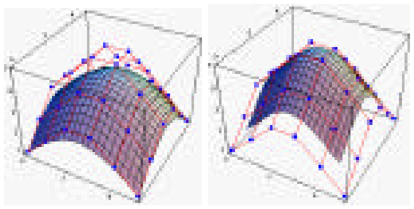
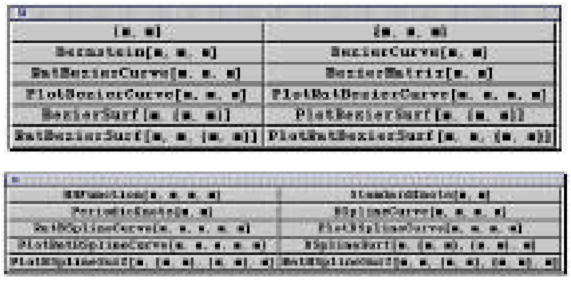


Figure 4. (left) Bézier patch; (right) B-spline patch.

Of course, many other commands (which are not described here for limitations of space) for dealing with the most usual curves and surfaces in CAGD have also been implemented. Figure 5 shows two Mathematica palettes with the most important commands included in the package.



{■, ■}	{■, ■, ■}
Bernstein[■, ■, ■]	BezierCurve[■, ■]
RatBezierCurve[■, ■, ■]	BezierMatrix[■, ■]
PlotBezierCurve[■, ■, ■]	PlotRatBezierCurve[■, ■, ■, ■]
BezierSurf[■, {■, ■}]	PlotBezierSurf[■, {■, ■}]
RatBezierSurf[■, ■, {■, ■}]	PlotRatBezierSurf[■, ■, {■, ■}]

BBFunction[■, ■, ■, ■]	StandardKnots[■, ■]
PeriodicKnots[■, ■]	BSplineCurve[■, ■, ■, ■]
RatBSplineCurve[■, ■, ■, ■, ■]	PlotBSplineCurve[■, ■, ■, ■]
PlotRatBSplineCurve[■, ■, ■, ■, ■]	BSplineSurf[■, {■, ■}, {■, ■}, ■]
PlotBSplineSurf[■, {■, ■}, {■, ■}, ■]	RatBSplineSurf[■, ■, {■, ■}, {■, ■}, ■]

Figure 5. Some CAGD package commands.

In the next section these commands will be applied to solve some interesting and illustrative examples.

3. Some Examples of Application

This section is devoted to show some examples of application of the CAGD Mathematica package described in the previous paragraphs.

3.1. Two-dimensional transformations

In this example we start with an initial square, given by joining the list of points:

```
In[17]: square={{-5.,-5.},{5.,-5.},
 {5.,5.},{-5.,5.},{-5.,-5.}};
```

Now, some two-dimensional transformations are applied to the previous list (see [4], Chapters 2 and 3, for a survey on two- and three-transformations). First of all, we consider a proportional 2D-scaling (given by the `Scaling2D` command) of factor `Sqrt[68]/10`. Then, a rotation (`Rotation2D` command) of angle *arctan(5/3)-* $\pi/4$ is applied. The `NestList` command is finally applied to produce a 40-steps iteration of this process, as follows:

```
In[18]:s=NestList[Rotation2D[
 Scaling2D[#,
 {Sqrt[68]/10,Sqrt[68]/10}],
 ArcTan[5/3]-Pi/4]&,
 square,40];
```

obtaining the Figure 6 by using the graphical Mathematica commands:

```
In[19]:Show[Graphics[ListPlot[#,
 PlotJoined->True,
 Axes->False,PlotRange->All]
 ]& /@ s,AspectRatio->1]
```

Out[19]:

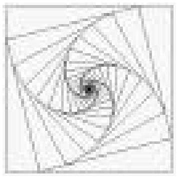


Figure 6. Iterative process of scaling and rotation two-dimensional transformations.

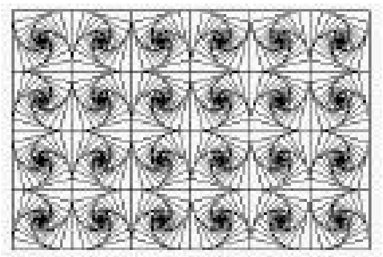


Figure 7. Mosaic obtained by the application of some simple (rotations, scalings and reflections) two-dimensional transformations.

Finally, reflecting and traslating Figure 6 (by using the `Reflection2D` and `Traslation2D` commands) Figure 7 is obtained. As the reader may appreciate, the previous transformations can be achieved in a very easy and intuitive way, by the simple application of the Mathematica and `CAGD` package facilities.

3.2. The Spinning Top Example

Given the list of points:

```
In[20]:spitop={{0,16},{1,16},
 {1.5,16},{1.5,16},{1.5,16},
 {1.5,16},{1.5,14.5},{1.5,14.5},
 {1.5,14.5},{1.5,14.5},{1.5,14.5},
 {2,14.5},{3.5,14},{5,13},{5.5,11},
 {4.5,8.5},{3,7},{1.5,6},{1,4.5},
 {0.5,3},{0,1.5},{0,0}};
```

the mathematical expression for its Bézier curve is given by:

```
In[21]:BezierCurve[spitop,t]
```

Out[21]: {21 t-105 t^2+2992.5 t^4-20349 t^5+81396 t^6-232560 t^7+508725 t^8- 881790 t^9+ 1.2345 10^6 t^{10}-1.23451 10^6 t^{11}+146965 t^{12}+ 2.34013 10^6 t^{13}-4.593 10^6 t^{14}+ 4.17833 10^6 t^{15}-1.30234 10^6 t^{16}- 1.21196 10^6 t^{17}+ 1.67114 10^6 t^{18}-915600 t^{19}+259717 t^{20}-31662.5

t^{21}, 16-81396 t^6+1.04652 10^6 t^7- 6.40993 10^6 t^8 + 2.46901 10^7 t^9- 6.66633 10^7 t^{10}+ 1.33327 10^8 t^{11}- 2.0384 10^8 t^{12}+2.42764 10^8 t^{13}-2.27734 10^8 t^{14} + 1.69277 10^8 t^{15}-9.98322 10^8 t^{16}+4.6437 10^7 t^{17}-1.666 10^7 t^{18}+4.36432 10^6 t^{19}-737383 t^{20}+59270.5 t^{21}}

which is a polynomial of degree 21, obtained from the 22 control points (see Figure 8(left)). A 2D-reflection is then applied in order to obtain a whole spinning top, as shown in Figure 8(middle). Finally, several 2D-rotations can be applied for simulating the movement of the spinning top in a plane (see Figure 8(right)). Finally, Figure 9 shows four different frames of a movie generated from the 3-D transformation commands implemented in the package.

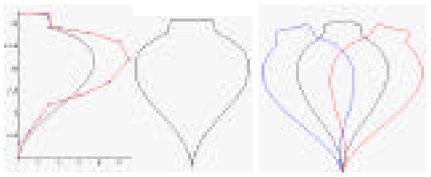


Figure 8. Simulation of the spinning top movement: three different steps.

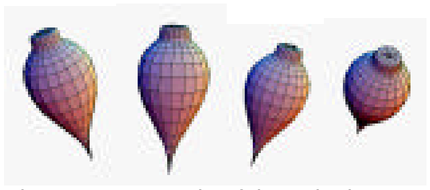


Figure 9. 3-D movie of the spinning top.

3.3. B-spline Curves

In general, Bézier curves are not appropriate for computer design because no local control can be performed, that is, moving the position of one control point alters the shape of the whole curve (see Figure 10(left)). Moreover, the resulting curve degree, *n*, is totally determined by the number of the control points (see [1,3-6]). Hence, it is more interesting for computer graphics to consider B-spline curves, whose degree can be chosen and that exhibit local control, that is, if a control point is moved only the closest parts of the curve to this point are affected, as shown in Figure 10(right).

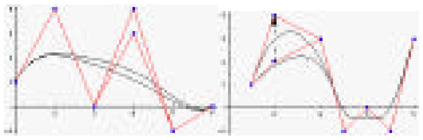


Figure 10. Different control for the curves: (left) Bézier: global control; (right) B-spline: local control.

As another example of B-spline curves, Figure 11 shows three different pictures of a squirrel obtained by using the `PlotBSplineCurve` command. They differ in the options, `ControlPoints` and `ControlLines`, used to display a given picture, which can take the values `True` or `False`, indicating whether or not the picture is displayed with the control points and/or lines.

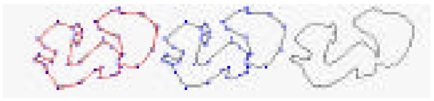


Figure 11. Example of B-spline applied to design.

In addition of this, different choices of the knots vectors, control points, and orders can be taken, for both curves and surfaces. This will be illustrated in the case of a surface in the next example.

3.4. B-spline Surfaces

Figure 4(right) showed a typical example of a B-spline patch, which has been built as a tensor product of two B-spline curves (see [1, 3, 4, 5] for more details). In this case, a periodic knot vector has been chosen for both axes. Compare with the Bézier patch defined by the same control points (Figure 4(left)).

The implemented commands allow us to choose many different options. As an illustration, Figure 13 shows two B-spline surfaces: on the left, a (3,3)-order with two nonperiodic knot vectors surface; on the right, a (5,2)-order with a nonperiodic and a periodic knot vectors. Differences between them can be visually stablished.

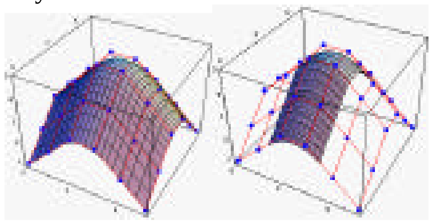


Figure 13. Two different choices of orders and knot vectors for a B-spline surface.

3.5. Blending Surfaces

If a single parametric surface is not able to approximate a given set of points sufficiently well, then we may use several patches which are joined together. Figure 14 illustrates this blending process. Two Bézier surface patches $\mathbf{F}_0$ and $\mathbf{F}_2$ (see Figure 14 (left)) are connected using a Bézier $\mathbf{F}_1$ with C^1 continuity. Following [3], pag. 269, this connection can be achieved by a careful choice of the control points for the Bézier surface $\mathbf{F}_1$. Figure 14(right) shows the resulting blending surface.

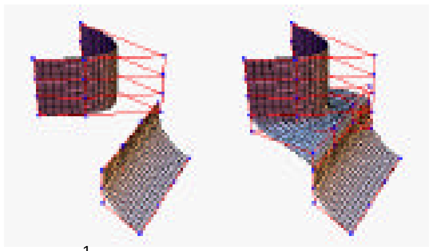


Figure 14. C^1-continuous connection of two Bézier surfaces.

Once again following [3], Figure 15 shows another example of blending surface: in this case, it is given by two different curves, a circle and a ellipse.

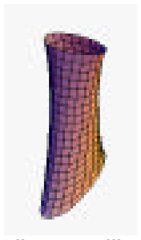


Figure 15. Blending an ellipse and a circle.

3.6. The CAGD Package in Industry

In many industrial areas, geometric information is stored, transmitted and manipulated in different standard formats, like IGES, VDA, CATIA, etc. Each of these standard formats support a different mathematical representation of the objects. For example, VDA uses the monomial representation, whereas IGES supports the B-spline one (see [7] for more details on the IGES format). Since the CAGD package is able to deal with this last description, we have developed an IGES-Mathematica converter. This program allows us to have access to some geometric entities of the electronic files from the automotive or aerospace industries. After reading the files, we can apply all the above described commands to these geometric entities and introduce the students to the industrial world.

4. Interaction with New Computer Graphics Education Resources

Nowadays, the new technologies are changing the usual teachers' resources for computer graphics and visualization education. Among them, we consider the use of the virtual reality languages and Internet.

4.1. Conversion to VRML language

Perhaps one of the most recent innovations in computer graphics education is the appearance of the virtual reality languages. By using these languages, like VRML, you can navigate around and through the graphic and manipulate it in a very easy and intuitive way. In virtual reality, the user "lives" in the scene. Evidently, this opens many possibilities, some of them still unexplored, not only technical but also in terms of students' motivation. E. Donley, at IUP, has developed a Mathematica package that converts Mathematica 3D graphics into VRML format and writes it to a .wrl file. This file can be viewed using a variety of VRML browsers, like Netscape. For further information, we refer the reader to [8]. This program allows the students a better manipulation of the graphics, adding many options not available in the Mathematica outputs.

4.2. Use of Internet

Today, many teachers are concerned about the possibility to take advantage of Internet as a vehicle for communication and education. Indeed, there is a large collection of Java programs and Web pages for computer graphics education. In this sense, Mathematica 3.0 and higher also allow to convert any standard Mathematica notebook to HTML. Unfortunately, pictures are fixed as they are converted to GIF and JPEG formats. However, there is a free Java applet, `LiveGraphics3D`, by M. Kraus [9], for incorporating some kind of interactivity (display and rotation) to these pictures in your Web pages. Some examples can be found in [9].

5. Conclusions

In this paper we have introduced a Mathematica package for learning and teaching CAGD and Computer Graphics. This package constitutes, as far as we know, the first attempt to bring SCPs, and specially Mathematica, to the Computer Graphics community in an extensive way. This work arose from our feeling that SCPs could be successfully applied in computer graphics and visualization education. After using the packages in some regular courses, our experience was so satisfactory that we are convinced now that this is one of the best tools for introducing students to CAGD and computer graphics.

The weakest part to be appreciated in this work is related with the requirements in memory and computation time that Mathematica requires, in comparison with other programming languages. On the contrary, the Mathematica code is shorter and easier to understand, and many mathematical functions are already incorporated. For example, the students can immediately obtain a graphical representation of the problem under analysis and its solution without using any external visualization program.

Acknowledgements

Authors would like to thank the CICYT (Comisión Interministerial de Ciencia y Tecnología) of the Spanish Ministry of Education (project TAP98-0640), the European FEDER funds (project 1FD97-0409) and University of Cantabria for finnancial support of this work.

References

[1] V. B. Anand. *Computer Graphics and Geometric Modeling for Engineers*, John Wiley and Sons, New York, 1993.

[2] S. Wolfram. *The Mathematica Book*. 3rd. ed., Wolfram Media / Cambridge University Press, 1996.

[3] J. Hoscheck and D. Lasser. *Fundamentals of Computer Aided Geometric Design.* A.K. Peters, Massachussetts, 1993.

[4] D.F. Rogers and J.A. Adams. *Mathematical Elements for Computer Graphics*. Second Ed., Mc Graw-Hill, New York, 1989.

[5] L. Piegl and W. Tiller. *The NURBS Book*, Second Ed., Springer-Verlag, Heidelberg, 1997.

[6] G.Farin. *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, Orlando Fl, 1988.

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

[8] E. Donley. *VRMLConvert program*. Web Page: http://www.ma.iup.edu/people/ hedonley. html

[9] M. Kraus. *LiveGraphics3D program*. Web Page: http://theorie3.physik.uni-erlangen.de/ ~mkraus/Live.html