Applying MATLAB to Computer Graphics and CAGD. Application to a Visualization Problem in the Automotive Industry.

Akemi Gálvez[1], Andrés Iglesias[1,*], Flabio Gutiérrez[2]

[1]Department of Applied Mathematics and Computational Sciences, University of Cantabria
Santander, Spain

[2]Department of Mathematics, National University of Piura, Piura, Peru

[*] Corresponding author.

Abstract

Computer graphics are usually achieved by using some traditional programming languages (Fortran, Pascal, C, etc.). In this paper an extensive use of the general-purpose numerical computation programs (NCPs) in the computer graphics field is proposed instead of. The paper describes the main advantages of this kind of programs, and several examples of how they can be successfully applied to computer graphics and visualization. Moreover, the paper briefly introduces several functions and commands developed by the authors, which will be successfully applied to solve a visualization problem coming from the automotive industry.

Keywords: Computer Graphics, MATLAB, CAGD.

1. INTRODUCTION

Computer graphics play a fundamental role in engineering design, capturing the visual and quantitative aspects of the geometric objects. For example, in the automotive industry, one is interested not only to obtain the curves and surfaces holding some prescribed constrains but also to join all these geometric entities together and to visualize the resulting picture in order to take care the aesthetic features and the general look of the product.

Many of the most important programs for computer graphics have been written in traditional programming languages (Fortran, Pascal, C, etc.) However, in the last recent years, the general-purpose numerical computation programs (NCPs) are gaining more and more popularity. Today, they are well established as a powerful alternative to the traditional programming languages in many different areas, as mechanical engineering, signal processing, quality control, electronic circuits, etc.

In this context, it would seem natural to wonder if the NCPs could be applied, instead of the traditional programming languages, in the computer graphics field. The present paper tries to answer this question by following the next sequence: Section 2 describes the main advantages of this kind of programs. In addition, the main reasons to justify our choice of MATLAB as the NCP to be used in this paper are also discussed in this section. Then, Section 3 introduces some additional commands we need to implement in order to solve some interesting problems related to CAGD and computer graphics. This section also includes a wide description of the main MATLAB graphical commands, options and utilities that will be useful for rendering surfaces. As an application, Section 4 shows how they can be successfully applied to solve a given visualization problem coming from the automotive industry. Today, many industries are concerned about the possibility to transfer their information by Internet, avoiding other possible and slower transference ways. Section 5 discusses such a possibility for the visual information, through the MATLAB-VRML connection. Finally, the paper closes with the main conclusions and remarks of this work.

2. ADVANTAGES OF THE NCP

In this section we show the main advantages of the NCPs, which justify our proposal to apply them to the computer graphics field. Then, we proceed to choose the program to be used along the paper.

2.1 Why to use NCPs for Computer Graphics?

There are many reasons to explain why we propose the NCPs to be used in the computer graphics field. Some of them are listed below:

- The NCPs are easier to use, because:
  - they incorporate many mathematical and programming commands and libraries
  - their algorithms are very optimized
  - they have a powerful and user-friendly interface
- The NCPs are very powerful, because:
  - their programming languages incorporate not only the procedural but also the functional programming including, in several cases, pattern recognition and object-oriented programming.
  - they have a very remarkable graphical capabilities.

Based on these considerations, our research group undertook the ambitious task to apply the NCPs to computer graphics. The following paragraphs are devoted to show how this work has been performed, indicating the main advantages of our approach.

2.2 Choosing the NCP: MATLAB

Evidently, not all the NCPs offer the same advantages and features. Therefore, the first thing to be done in this line is to choose the program to work with. After a careful analysis, our final choice was MATLAB (see the MathWorks Home Page at: *http://www.mathworks.com)*. In this choice we took into account some features as:

- Spreading. MATLAB is used for hundreds of thousands of industrial, government and academic users around the world. Its last versions are available for Microsoft Windows 9x and NT, Macintosh and Linux personal Computers, as well as UNIX workstations from Sun, Hewlett-Packard, IBM, Silicon Graphics and Digital, and Open VMS computers.
- Graphical capabilities, which raise many of the current graphics-oriented programs (see Section 3.2).
- Since MATLAB is based on C, it runs faster than other analyzed symbolic and numerical programs. Moreover, its basic element is an array that does not require dimensioning, so it takes less time to be computed.

It must be noticed that, in spite of our choice, the same results can be obtained by using some other NCPs. For example, SCILAB (see [1] for details) is a free software whose programming and graphical capabilities are very similar (although slightly lower for our purposes) to those of MATLAB. However, we think this last one is more popular and used in academic and industrial environments.

3. APPLYING MATLAB TO CAGD AND COMPUTER GRAPHICS

The aim of this section is twofold: on the one hand, Section 3.1 introduces some additional commands we need to solve a visualization problem described in Section 4 and other interesting problems related to CAGD and computer graphics. On the other hand, Section 3.2 describes the main MATLAB commands, options and utilities that will be useful for rendering surfaces.

3.1 Building numerical libraries for CAGD

Once the program is chosen, the following task to be done is the implementation of an extensive set of numerical libraries for CAGD. By "extensive" we mean the libraries must contain all the relevant geometric entities in the sense that if a given geometric entity is useful in CAGD, it must be incorporated to the system. Of course, libraries must be continuously updated, so the system must be flexible enough to allow the programmer to improve the algorithms and codes in an efficient, quick and easy way.

MATLAB incorporates some useful commands for CAGD. For instance, its kernel includes a basic command for interpolation through cubic splines and some other commands for interpolation in one and several variables. However, the system lacks of many of the most important mathematical entities for CAGD, such as Bézier and B-spline curves and surfaces, which must be implemented. The powerful MATLAB functional programming offers us the possibility to implement these functions in a short, elegant and simple code. As an illustration, the following script calculates and displays the Bézier curve of a given set of two- or three-dimensional points:

```
function Bezier(ptos)      % main function
[n,d]=size(ptos);
n=n-1;
bt=ptos'*mij(n)*ti(n);
if d==2
plot(bt(1,:),bt(2,:),ptos(:,1),ptos(:,2),'r-.p')
else
plot3(bt(1,:),bt(2,:),bt(3,:), ...
      ptos(:,1),ptos(:,2),ptos(:,3),'r-.p')
end
rotate3d

function T=ti(n)           % generating the t^i
m=1;
t=0:0.05:m;                % step=0.05
T=[];
for i=0:n
   T= [T;t.^i];
end

function M = mij(n)
for i=0:n
   for j=0:n
     M(i+1,j+1)=(-1)^(ji)*binom(n,j)*binom(j,i);
   end
end
M=M(1:n+1,1:n+1);

function c=binom(n,i)      % defining the binom function
if i==n | i==0
   c=1;
elseif i<n & i>=0
   c=factorial(n)/(factorial(i)*factorial(n-i));
else
   c=0;
end

function f=factorial(n)    %defining the factorial function
if n==1
   f=1;
else
   f=n*factorial(n-1);
end
```

Table 1: MATLAB code for the Bézier curves.

We remark that this example has been chosen primarily for simplicity, rather than to correspond to a valuable code or a very complicated algorithm. However, some questions deserve to be pointed out: as the reader may appreciate, we use the matrix form for the Bézier curves (see, for example, [2] pag. 58). This is not a chance: MATLAB handles vectors and matrices in a straightforward and intuitive way. Furthermore, there are typically many different ways to formulate a given problem in MATLAB; in almost all cases, however, the best performance is expected when matrix formulation is applied. The simple idea of organizing data in a matrix form yields to programs that are more efficient and easier to understand. Thus, Table 2 shows the corresponding code for Bézier surfaces (which can be easily derived from Table 1):

```
function SupBezier(ptos)
[m,n,o]=size(ptos);
for k=1:3
b(:,:,k)=ti(m-1)'*mij(m-1)'*ptos(:,:,k)...
         *mij(n-1)*ti(n-1);
end
surf(b(:,:,1),b(:,:,2),b(:,:,3)), hold on,
mesh(ptos(:,:,1),ptos(:,:,2),ptos(:,:,3)),
hidden off
plot3(ptos(:,:,1),ptos(:,:,2),ptos(:,:,3),'bp')
rotate3d
```

Table 2: MATLAB code for the Bézier surfaces.

In MATLAB each command or group of them is stored into a file, which is called a *M-file*. When several M-files for solving similar problems of a certain field are written (as it is our case) they can be collected together into special directories (*Toolboxes).* The toolbox for CAGD developed by the authors and described in this paper deals with the following functions and topics:

- **Bézier curves**. As shown before, the toolbox deals with two- and three-dimensional curves. The cases of single and composite Bézier curves are also considered. Curves can take both the rational and non-rational form.
- **Bézier surfaces**. As in the case of curves, Bézier surfaces have been implemented in MATLAB (see Figure 6 for an example).
- **B-spline curves**. Commands work with two- and three-dimensional curves, for any order and knots vector (periodic, non-periodic or non-uniform), and different weights (rational curves). NURBs are therefore considered here as a particular case.
- **B-spline surfaces**. All the options described for B-spline curves are also available here (for example, Figures 1 and 2 corresponds to a B-spline and a NURB surface, respectively).
- **Two- and three-dimensional transformations**. Since all these transformations are not available directly in MATLAB, they were implemented in the toolbox.
- **Projections and perspectives**. MATLAB only supports some kinds of projections and perspectives. The toolbox incorporates all of them.

3.2 MATLAB graphics commands

MATLAB provides a set of powerful high-level graphical routines for displaying both two- and three-dimensional graphics. However, since we are mainly concerned about the three-dimensional pictures, the following description is restricted to this case (which includes the 2-D case for many commands). In the following we describe the most important MATLAB features for computer graphics (the corresponding commands are denoted in courier font style).

(1). **Plotting 3-D data**. They can be displayed as line plots (`plot3` command) or rectangular grids (`mesh`, `surf`). The `mesh` command generates a wireframe view of the surface (as in Figure 2). On the contrary, `surf` shows a colored, faceted view. For example, Figure 1 shows a 3x3-order B-spline surface (obtained by using the commands described in Section 3.1), defined by the *z*-coordinate of points above a grid in the *x*-*y* plane.

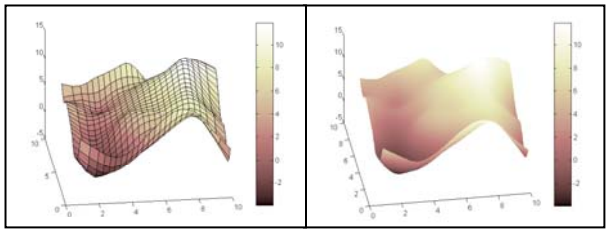

Figure 1: A 3x3-order periodic B-spline surface with: (left) a faceted shading; (right) an interpolated shading.

(2). **Hidden line removal**. In MATLAB, mesh plots remove hidden lines by default. You can disable hidden line removal through the `hidden off` command. A typical example is shown in Figure 2: on the left, a NURB surface is displayed as a wireframe plot. On the right, the hidden line removal is off, so the back part of the surface becomes visible now.

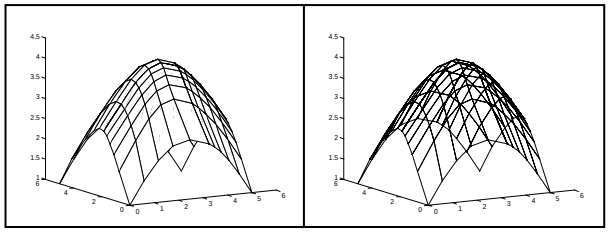

Figure 2: Example of a NURB surface with the hidden line removal: (left) on; (right) off.

(3). **Color**. User may enhance the information content of the surface plots by changing their colors, either using the

RGB triplets or a predefined range of colors called `colormap`. Moreover, colors can be assigned by means of a relevant function. For example, the `del2` command applies the same color to regions exhibiting similar curvature. This effect can be visually established by a simple comparison of Figures 3 (left) and (right).

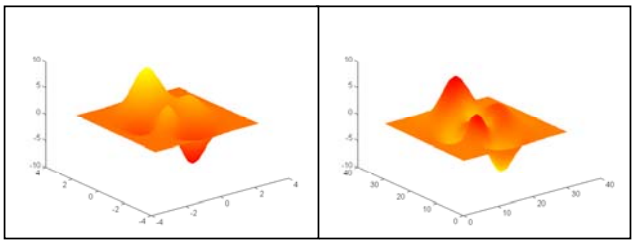

Figure 3: Using color for geometric information. Color indicates:(left) heights; (right) curvature.

(4). **Texture mapping**. By this, we mean a technique for mapping a 2-D image onto a 3-D surface by transforming color data so that it conforms the surface plot. Texture mapping has become a very important topic, allowing to applying a texture, such a wood grain, to a surface. In MATLAB, to apply texture mapping is as easy as setting the `FaceColor` option of the three-dimensional surface to `texturemapping`.

```
f=imread('rusia.jpg');
f2=double(f)/255;
a=SupBSpline(3,3);          %generates a Bspline surface
surface_handle=surf(a(:,:,1),a(:,:,2),a(:,:,3));
set(surface_handle,'EdgeColor','none',...
    'FaceColor','texturemap','cdata',f2);
set(gca,'box','on');
```

Table 3: MATLAB code for the texture mapping.

Table 3 lists a simple code for texture mapping and Figure 4 illustrates this process: the image in the middle is mapped onto the surface, giving the picture on the right. Note that the color data can be any image; in this case, a scanned photograph. Furthermore, you can map any image onto the surface, no matter their sizes. Finally, the mapping process can be total or partial (in this last case, you must indicate the size of the image to be mapped).

(5). **Patches**. Figure 5 shows two typical patches, obtained in MATLAB from the `patch` command. A patch is a graphic object that consists of one or more polygons that may or may not be connected. Patches are useful for modeling real-world objects such as airplanes or automobiles. In MATLAB, a patch is defined by specifying the coordinates of its vertices and some form of color data. Such coordinates can be introduced in two different ways: either indicating the coordinates of the vertices of each polygon (MATLAB connects them to form the patch) or indicating the coordinates of each unique vertex and a matrix specifying how to connect the vertices to form the faces.

Figure 4: Texture mapping.

Coming back to Figure 5, it consists of two pictures: the first one is displayed with the `faceted` option, whereas for the second one the selected option is `interp`, which is based on interpolation, so better quality is expected when using this option (as already appreciated in Figure 1).

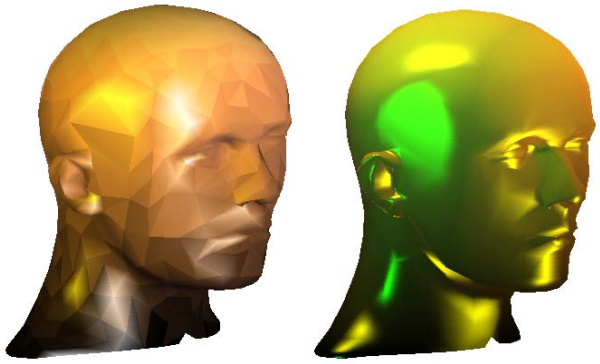

Figure 5: Choosing different options for the patch and the lighting: (left) faceted, gouraud; (right) interpolated, phong.

Finally, we remark that many of the surfaces features (as features (4) and (7)) are shared by the patches too. Other example is the light sources appearing in Figure 5, which are described in the next section.

(6). **Lighting**. This feature adds realism to a graphical scene. MATLAB supports three different ways for lighting calculations (the reader is referred to Chapter 16 of [3] for a more complete description about the shading models):

- `flat`. Produces uniform color across each of the faces of the object. It is specially indicated for faceted surfaces.
- `gouraud`. This algorithm calculates the colors at the vertices and then interpolates color across the faces (see Figure 6(left)). It is ideal for curved surfaces.
- `phong`. This method interpolates the vertex normals across each face and then calculates the reflectance at each pixel (see Fig. 6(right)). This algorithm produces better results than `gouraud` but takes longer to render.

These shading models have been applied to Figure 5, namely, gouraud for the left picture and phong for the right one. In addition of the previous commands, there are others for creating light sources. You only must specify three light properties:

- `Color`. The color of the light.
- `Style`. It can be `infinite` (when the light source is placed at infinity, which means that light shines from the specified direction with parallel rays) or `local` (the light source is a point source, which radiates from the specified position in all directions).
- `Position`. For infinite lights, indicates the direction. For local lights, the position of the light source.

For example, Fig. 5(left) exhibits an infinite light. On the contrary, Fig. 5(right) receives four different lights: a local yellow light, in the front of the face, and three (two yellow and one green) infinite lights. This second figure also exhibits other light effects. For this reason, its corresponding code has been listed in Table 4.

```
load vert_mann -ascii
load faces_mann -ascii
f=faces_mann+1;
v=vert_mann;
p=size(v);
h=superf(f,v);
set(h,'edgecolor','none', ...
'SpecularStrength',4,'DiffuseStrength',2, ...
  'AmbientStrength',1,'SpecularExponent',15, ...
  'SpecularColorReflectance',0.2, ...
  'FaceColor','interp');
colormap(copper(p(1,1)))
lighting phong
light('Style','Local','Color','y', ...
      'Position',[3 -4 0]);
light('Color','y','Position',[1 -2 -1]);
light('Color','g','Position',[-3 -1 1]);
light('Color','y','Position',[0 0 6]);
material shiny
axis vis3d off
rotate3d
```

Table 4: MATLAB code of Figure 5 (right).

MATLAB enables to control the amount of both the specular (`SpecularStrength` command) and the diffuse reflection (`DiffuseStrength` command) from the object. Their values are shown in Table 4.

Another interesting light property is the ambient light, that is, a directionless light that shines uniformly on all objects. In this paper we uses the `AmbientStrength` command, which determines the intensity of the ambient light on the particular object (the head in this case) and the `SpecularExponent` command, which determines the size of the specular highlight spot (the lower the parameter value is, the bigger the spot size). Finally, the `SpecularColorReflectance` command is used to determine the color of the specularly reflected light, ranged from a combination of the color of the object (defined by the `colormap` command in Table 4) and the color of the light source to this last one only.

(7). **Reflectance properties**. The reflectance properties of a object are described by the `material` command, which can take three different values: `shiny`, `dull` and `metal`, meaning that objects are made shiny, dull and metallic, respectively. Figure 5 corresponds to the first case.

(8). **Contouring**. In scientific computing, the contour lines (obtained through intersections between a number of parallel planes and a given surface) are often of great importance. Relevant examples can be found in the medical area, for reconstructing and displaying the external surface of the organ under investigation (see [4] and references), in pattern recognition and computer vision [5], etc. The MATLAB `contour` and `contour3` commands display the 2-D and 3-D isolines generated from values given by a matrix of heights in two and three dimensions, respectively. Figure 6 shows a contouring example: the Bézier surface on the left is intersected with a set of planes $z=z_0$ for 30 different values of z_0. These intersections give a set of curves, which are shown in Figure 6 (right).

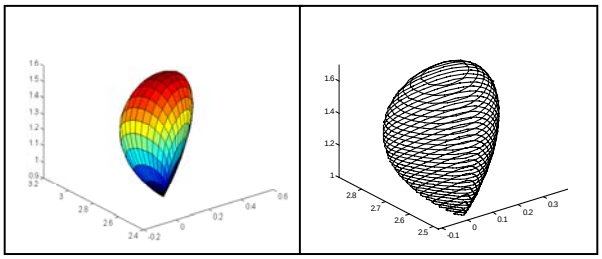

Figure 6: Example of a Bézier surface contouring.

(9). **Files management**. This is a very powerful MATLAB feature. This program reads and writes images data in TIFF, JPEG, BMP, PCX, XWD and HDF formats. Thus, the `imread` command reads an image from a file in any of these formats. You can also save the image data using the `imwrite` function. Finally, the `iminfo` enables you to obtain information about graphic files, including the name of the file and its path, format, version, size (in bytes), width and height and number of bits. Additional information could be obtained, depending on the type of file you have. All these capabilities will be applied to create an IGES-MATLAB converter (see Section 4.1) and to transfer our MATLAB files to VRML (see Section 5).

(10). **Animation**. Animation is one of the most important features in computer graphics. In automotive industry, the design process often requires to visualize the piece to be constructed. Sometimes, projections are enough to perform this task but some of the features of the piece can be more

easily appreciated by animating it. Although this paper cannot show any animation, it is interesting to point out that MATLAB allows to create movies either saving a number of different pictures and then playing them back or by continually erasing and then redrawing the objects on the screen. Of course, the first option is more advisable in situations in which each frame is fairly complex and cannot be redraw rapidly.

Another interesting animation possibility comes from the new virtual reality programming languages. Since MATLAB can read many different file formats, this task becomes now easier for us. It just consists of translating the graphical MATLAB output to some of these languages. It allows a better visualization, since the designer can "navigate" by the graphical environment of the piece. This option is especially valuable when looking for piece defects, after the design process, and will be discussed in the next section.

(11). **Other properties**. Of course, the previous MATLAB graphical commands list is not intended to be exhaustive. Some other interesting and useful properties are also available in MATLAB. For a more complete information about this topic, the reader is referred to [6].

4. APPLICATION TO AN AUTOMOTIVE EXAMPLE

A year ago, our research group, at Cantabria University (Spain) established an agreement with CANDEMAT S.A. (*http://www.candemat.com*), a company devoted to both the automotive and the aerospace industries. The agreement includes the use of the previously described numerical Toolbox based on MATLAB and the implementation of the other ones for solving the problems arising in the company daily work. CANDEMAT builds moulds of pieces of cars and planes, which will be used later for testing by other associated companies. For doing this work, this company receives files (that are electronically transferred from the automotive and aerospace companies) containing the geometric information of the pieces to be shaped. This information is then processed by using the program CISC, developed at CANDEMAT and written in Visual BASIC.

In general, CISC has been successfully used for many different tasks, being able to read these electronic files and apply numerical routines for dealing with the different geometric entities defined therein. However, the program has some strong limitations that can be improved in several directions. One of them refers to the visualization process. They would like to visualize the pieces under the following conditions:

1. Although at the beginning the company worked with UNIX workstations, the software they required became more and more expensive. Today, company's policy is oriented to the use of personal computers (PCs). This imposes the software for visualization to be available for personal computers and cheap enough to be installed in all the computers.
2. The software should incorporate a powerful programming language and almost all the facilities described in the previous sections.
3. If possible, the graphical output should be transferable to Internet, in order to visualize (even manipulate) it without having the same software and/or hardware. This question will be discussed in Section 5.

Fortunately, all these conditions are satisfied when applying MATLAB and the added numerical routines. In the next paragraphs we are going to describe the steps we followed to perform this task.

4.1 IGES-MATLAB Converter

In many industrial areas, geometric information is given by employing different standard formats (IGES, DVA, SET, CATIA, etc.). A format is a way to express such an information as an alphanumeric text, following some well-established rules. Therefore, these formats represent the real-world objects, as the different pieces of a car (doors, bumpers, wings, etc.), in a mathematical formulation and are stored in electronic files (see Figure 7).

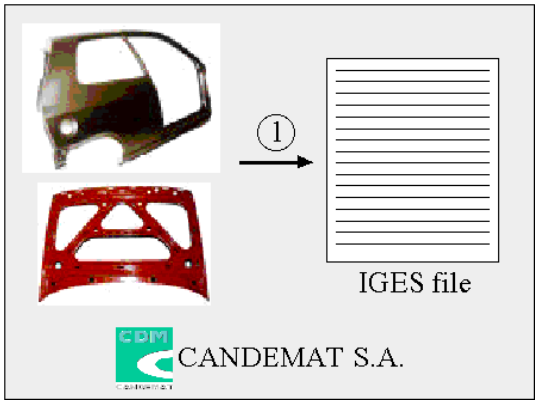

Figure 7. Step I. The IGES files store the geometric information of the real-world objects.

Of course, there are many different formats to be used, although they are reduced, in practice, to some few, that are considered as standards. Each of these standard format systems supports a different representation. Thus, IGES [7] only supports the B-spline representation, whereas VDA [8] uses the monomial one.

Usually, CANDEMAT works with IGES files, so in the following we restrict ourselves to this standard format. This means that for being able to work with IGES files, we firstly need to create an IGES-MATLAB converter (see Figure 8).

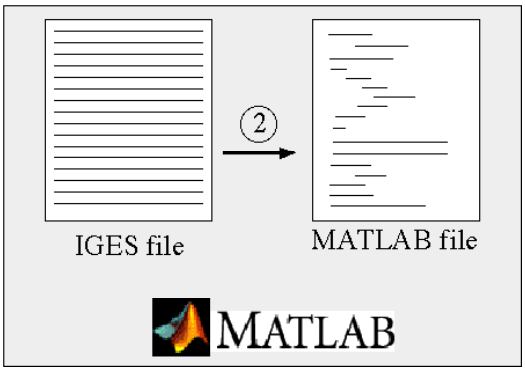

Figure 8. Step II: Now, the IGES file is converted to MATLAB.

This converter is a code written in MATLAB that extracts the useful part of the IGES file, in such a way that only the basic information for the curves and surfaces to be drawn is taken. Thus, as the section S only includes comments and user information, this part of the IGES file is ignored. Some information of the D section can also be removed. For more information about what these sections are and, in general, how a IGES file is organized, we refer the reader to [9] (see also [7] for more details).

4.2 Using the CAGD toolbox for visualization

Once the IGES file is converted to MATLAB, the next step (shown in Figure 9) consists of applying the commands and utilities described in Section 3.

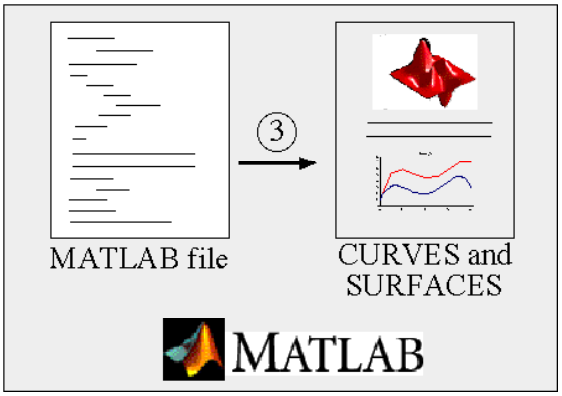

Figure 9. Step III: The commands from the CAGD toolbox are applied for displaying the curves and surfaces.

This will give us a numerical and graphical output of the curves and surfaces forming the different pieces to be built. Figure 10 shows a typical output. In this example, a wireframe model of the back door of a car is obtained. The data file was read with the converter described in the Section 4.1. Then, the commands for B-spline curves defined in the Section 3.1 were applied to the obtained file.

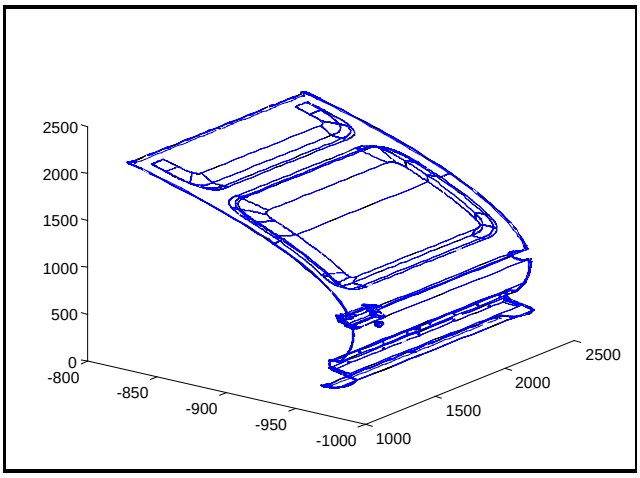

Figure 10. Car door defined by 1727 B-spline curves.

5. VISUALIZATION IN INTERNET

The Virtual Reality Modeling Language (VRML) is a standard language to describe interactive 3-D objects and integrate them into scenes and virtual worlds. In the context of our project, VRML allows us to create interactive simulations and physical movement of the different pieces, looking for defects in the piece under analysis. Moreover, scenes and virtual worlds can be distributed and visualized throughout Internet by means of some plug-ins developed for Web browsers. Following the same way that in Section 4.1, before using VRML we needed a MATLAB-VRML converter to transfer our files to the VRML format. Fortunately, we did not need to do that. There is a free software called **VRMLplot,** from Craig Sayers, (see [10]) for generating VRML files from the graphical MATLAB output (see Figure 11).

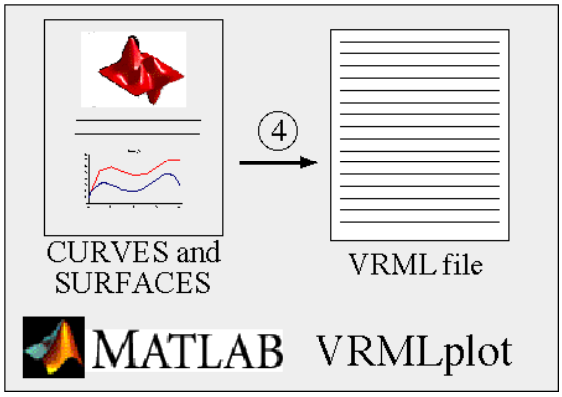

Figure 11. Step IV: Using converter MATLAB-VRML.

These files can be easily read by a Web browser, such as Netscape Navigator, and then, the interactive visualization is available (see Figure 12). Figure 13 shows an example of this visualization process.

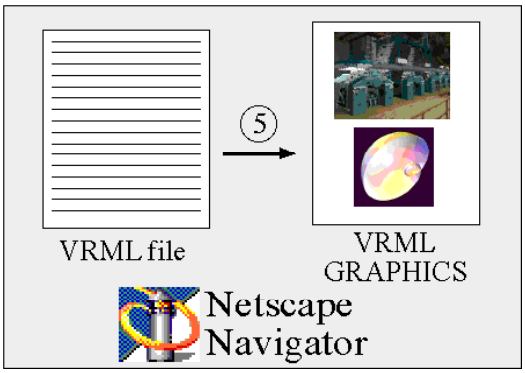

Figure 12. Step V: Using a Web browser, the VRML files are displayed on the screen.

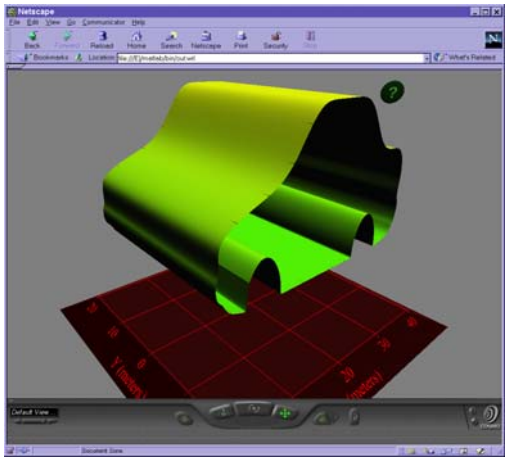

Figure 13. Example of a VRML scene.

6. CONCLUSIONS AND REMARKS

In this paper we propose the use of the NCPs (in particular MATLAB) as a powerful alternative to the traditional programming languages in the computer graphics field. This idea has been supported by recent announcements of the use of MATLAB and Simulink by, among others, DaimlerChrysler and Motor Ford Company (see [11] for details). Other news in this line are arising around the world. Therefore, it is expected, for the next years, a strong growth of similar approaches to the one described in this paper.

7. REFERENCES

[1] SCILAB Home page: *http://www-rocq.inria.fr/scilab/*

[2] G. E. Farin. *Curves and Surfaces for Computer Aided Geometric Design*, 3rd ed., Academic Press, San Diego (1993).

[3] J.D. Foley, A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Massachussetts (1982).

[4] A.B. Ekoule, F.C. Peyrin, C.L. Odet. "A Triangulation Algorithm form Arbitrary Shaped Multiple Planar Contours", *ACM Transactions on Graphics*, Vol. 10, pp. 182-199 (1991).

[5] J. D. Boissonat. "Surface reconstruction from planar cross-sections", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 393-397 (1985).

[6] The MathWorks, Inc. *Using MATLAB Graphics*, (1997).

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

[8] VDA Working Group CAD/CAM. *VDA Surface, Data Interface. (VDAFS) Version 2.0.* Verband der Automobilindustrie. (VDA). Frankfurt (1987).

[9] D. Basu, S. Kumar. "Importing mesh entities through IGES/PDES", *Advances in Engineering Software*. Vol. 23, pp. 151-161 (1995).

[10] *http://www.dsl.whoi.edu/DSL/sayers/VRMLplot/*

[11] *http://www.mathworks.com/company/pressroom*

Acknowledgments:

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

Author(s):

Andrés Iglesias (*corresponding author) holds a Ph.D. in Mathematics and currently is associate profesor at the Department of Applied Mathematics, E.T.S.I. de Caminos. Avda. de los Castros, s/n 39005 University of Cantabria, Spain. Tel: 34 (942) 201723 / Fax: 34 (942) 201703. E-mail: iglesias@ccaix3.unican.es.

Akemi Gálvez is a Ph.D. candidate at the Department of Applied Mathematics, Cantabria University (Spain). She holds a B.Sc. in Chemical Engineering at U.N.T. (Perú) and a M.Sc. in Computation at Cantabria University. E-mail: uc8031@cclx1.unican.es

Flabio Gutiérrez is also a Ph.D. candidate at the Department of Applied Mathematics, Cantabria University (Spain). He also holds a B.Sc. in Mathematics at U.N.T. (Perú) and a M.Sc. in Computation at Cantabria University. He teaches at National University of Piura, (Perú).E-mail: flabio@tallan.unp.edu.pe