COBEM 99
XV Congresso Brasileiro de Engenharia Mecânica
15th Brazilian Congress of Mechanical Engineering
22 - 26 de Novembro de 1999 / November 22 - 26, 1999 Águas de Lindóia, São Paulo,

VISUALIZING 3D VECTORS USING JAVA

Silvana A. Barbosa
Instituto de Aeronáutica e Espaço - Centro Técnico Aeroespacial
12.228-904 - São José dos Campos, SP, Brazil
Marcio L. X. dos Santos
Coordenadoria Central de Informática - Universidade de Taubaté
12.020-270 – Taubaté, SP, Brazil

ABSTRACT. *The purpose of this work was to develop a computer graphics representation of the vector properties of a given three-dimensional field through JAVA computational language. Although this problem had been addressed elsewhere in the past (see references), the new opportunities for dynamic loading and displaying vector fields in the context of Intranets through JAVA language methods seemed worth to explore. The capability of the resulting method to generate fast outputs containing all the information of the vector field makes it attractive for interactive applications.*

Key-words: *Visualization, Vector fields, Java.*

1. INTRODUCTION

Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Richard Hamming observed many years ago that "The purpose of [scientific] computing is insight, not numbers" (Computer Graphics, 1987). The goal of visualization is to leverage existing scientific methods by providing new scientific insight through visual methods.

In this work several techniques of graphics computing were applied in a CFD ("Computational Fluid Dynamics") problem, *the visualization of three-dimensional vector fields* (Barbosa, 1998).

Discrete values of a generating function for a vector field can be obtained by numerical analysis, but the use of a representation which allows quick identification of undesired outputs may save a considerable amount of time and effort.

This work describes a graphical method to visualize vector field distributions in 3D, using a pyramidal form to represent a 3D arrow with fixed proportions to visualize the magnitude and direction of the vector field at each point at which it is known, Fig.1.

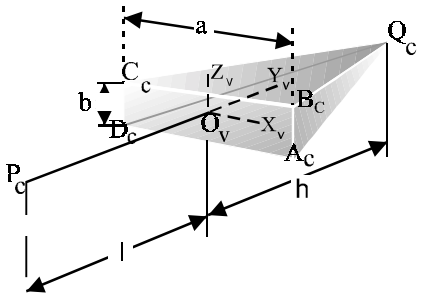


Figure 1 – Geometrical representation of a vector quantity.

The main preoccupation in choosing the computer language to be used in the software development was the question of portability. The wish was that the computer language was the most possible platform independent, to avoid troubles when changing of computational equipment. So the chosen language was JAVA.

2. GEOMETRICAL MODEL

The basic literature of physics sciences in electromagnetism, hydrodynamics, etc points to various methods in visualization of vector fields. Most of them to represent two-dimensional fields.

According to (Dos Santos, 1979), adopting a convenient 3D image for the vector quantity at a point in the 3D space, it is possible to accommodate all the pertinent field information accurately and produce a satisfactory field pattern to represent this distribution

Computer graphics representation of vector fields were also proposed recently: (Leeuw, W. C., and Wijk, J. J., 1993) (Simkin, Vector Fields Limited, 1995) (Boring E., and Pang A, 1996). Dos Santos pioneered the way – see (Nassif, N., and Silvester, P. P., 1980) – with a method restricted to the display techniques available in the 70s while the recent works focus on modern interactive display techniques. The ideas present by all of them, however, concur to stress the importance of computer graphics representations of 3D fields. In this work, we developed a novel approach to this basic problem, trying to make use of the capabilities of the JAVA language and of the computer graphics techniques. Both, the geometrical model and the programming technique will not be discussed in detail in the present paper. The reader who is wishing to understand the details of the geometrical model and those pertaining to the computer program is referred to the works of (Barbosa, 1998) and (Barbosa and Dos Santos, 1999).

3. GENERAL DESCRIPTION OF THE SYSTEM

ViCamp is a software that was developed in JAVA, with the objective of visualizing 3D vector fields.

In the software development, it would be necessary to create routines that doing, among other things, the input data reading, calculate the vector co-ordinates according to the geometrical model presented, the transformation and displaying routines.

The system is called *ViCamp* and it is composed of 6 classes, Fig.2.

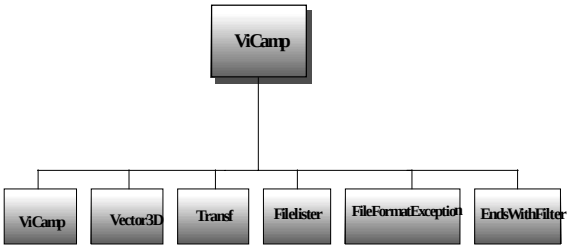


Figure 2 - ViCamp Project and its Classes

3.1 ViCamp

It is the main class of the system and it is composed by various methods. ViCamp is responsible for the painting (displaying) definition and creation, its format, colours, dimensions, options menu.

The system waits for some event and so the *handleEvent* method is actioned. From this method the actions are taken – see Fig. 4.

Figure 3 - Initial Display

Considering that the processing speed (performance) in JAVA is not as high as a compiled language, the use of techniques that became lower the exhibition and images manipulation time were used.

One of the features that JAVA offers, which was required in this work, is the *Double-Buffering*: it creates a white image, associated with this image a *graphics context* that may be compared to a blackboard where the paints are being drawn. This technique reduces the exhibition and manipulation time of images.

After the data file is defined, the *run* method is actived. It is responsible to determine the sequence that the pyramid faces will be drawn, according to the Segment Visibility and Image Degeneration (Barbosa, 1998).

The *repaint – update/paint –* method is called when it is necessary to redraw the image. For the image creation the *paint3D* method that belongs to the Vector3D class is called.

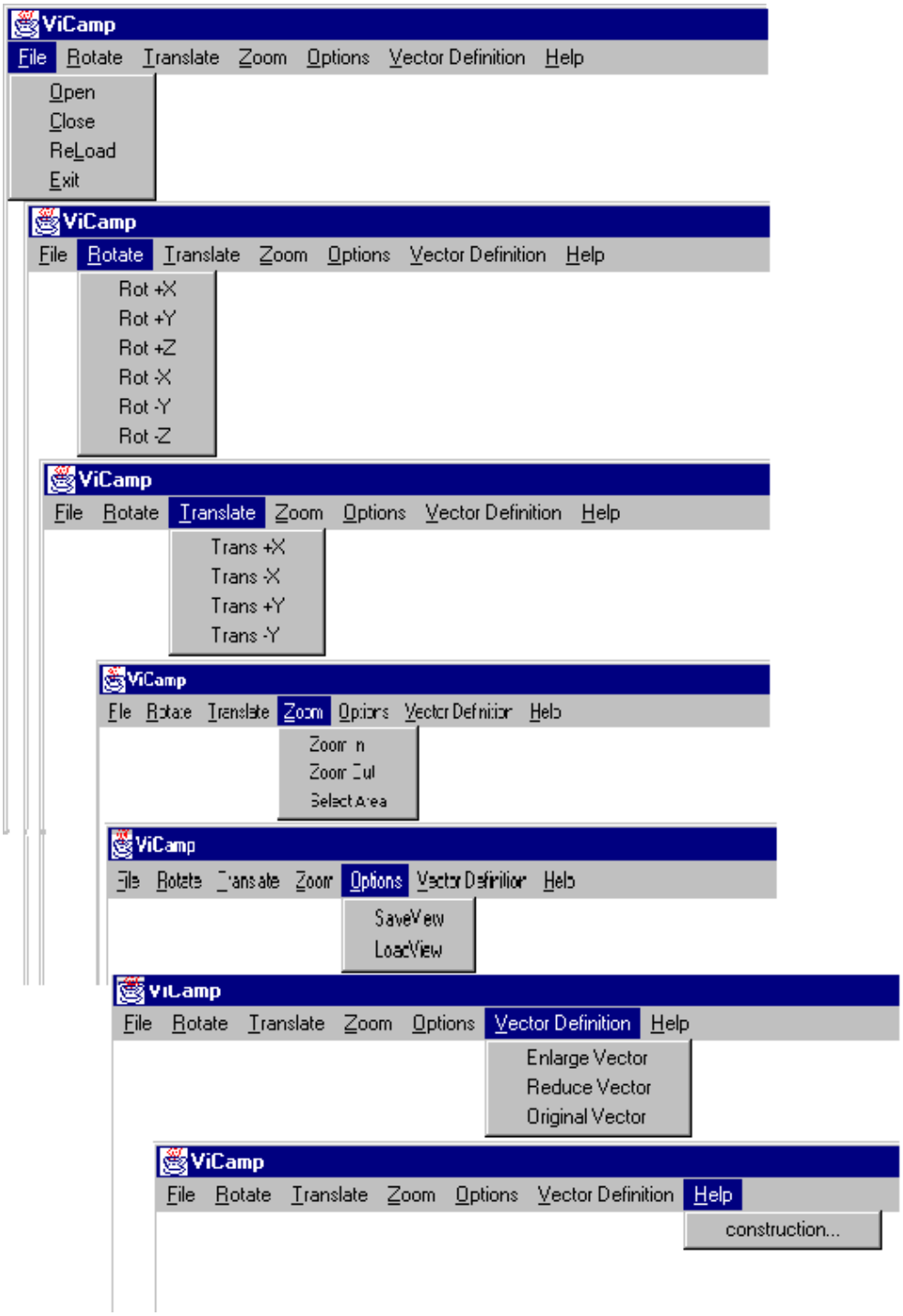


Figure 4 – Menu: Available Options.

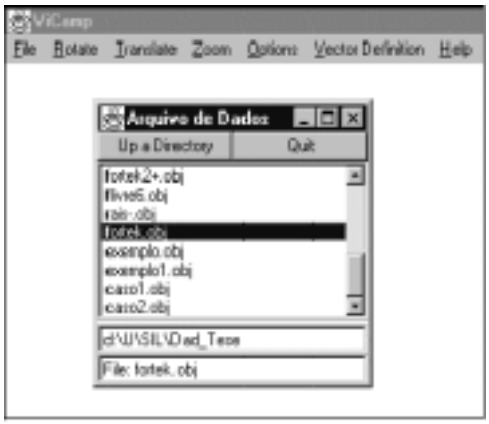


Figure 5 - Menu for Input File Choice.

3.2 Vector3D

This class is composed of the following methods: *Vector3D* (constructor - a method that is called automatically when a new instance of a class is created. It has the same name as the class), *determ, add, addVert, calc, cleanVert, findBB, paint3D, ptTela, transform*.

It is responsible to calculate the three dimensional model. First, through its constructor, it reads an ASCII file containing a set of data in real format, given by the user in this format:

1ª line:
npts $\rightarrow$ total number of mesh points
2ª line to npts^a +1 line :
xi yi zi xii yii zii

where:
xi, yi, zi $\rightarrow$ co-ordinates of the 3D mesh
xii, yii, zii $\rightarrow$ co-ordinates of the vector property in that point

The program will do this data reading, and the treatment of these values to get the graphical representation. The representation form to each field point will be pyramidal – an arrow drawn in a 3D way. The first image created will be according to the read values. After that, the user can manipulate this image through rotations in one or more axis, translation, and the zoom options, being able to observe better the data behaviour in several situations. The right-hand axis system was used according to the geometrical model (Barbosa, 1998).

Vector3d Methods. The methods, which compose the Vector3D class, will be described, except the constructor (commented at once).

Calc. It is called when some image transformation occurs. From the data reading done by the constructor, the vector length, angles and magnitude are defined.

Know that

$$V = \sqrt{\overline{xv}^2 + \overline{yv}^2 + \overline{zv}^2}$$

the vector total length is normalized according to the highest V value, encountered by this class constructor:

$$V_{new} = \frac{V}{V\,\max} * V * def$$

where initially

def = 1

By definition it was considered that

a = V
b = V/3
h = V
l = V
L = l + h

Because the length of each vector is normalized with regard to the maximum magnitude sample, in case of vector fields in structured mesh, this normalization attends to the need of a good visualization of the entire field. But when the mesh is unstructured, sometimes the user perception of all the points becomes difficult. The menu option *VectorDefinition*, enables the changing of the vector total length. In this case, the user has the option to enlarge or reduce the vector total length, through the *VectorDefinition*/*EnlargeVector*, *VectorDefinition*/ *Reduce Vector* options. It is done by changing the *def* variable value. From there the transformations will be made considering this new length vector definition. At any moment the user can return to the original vector length through the menu option *VectorDefinition*/*OriginalVector*.

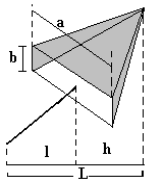


Figure 6 - Vector Length.

Paint3d. It forms the image by drawing the vector field in the *graphics context*. Each pyramid face is a triangle except the base, which is a quadrilateral. In this case *determ* method is called to define the drawing sequence, to avoid line crossing, like in the following example:

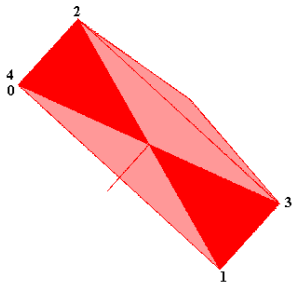


Figure 7 – Wrong Drawing Sequence.

***Determ*.** It avoids line crossing inside the pyramid base. The base is divided into two triangles, and then the determinant of each triangle is calculated thus verifying the normal vector direction.

The triangle area calculus through its determinant is:

$$\pm\frac{1}{2}\begin{vmatrix} x1 & y_1 & 1 \\ x2 & y_2 & 1 \\ x3 & y_3 & 1 \end{vmatrix}$$

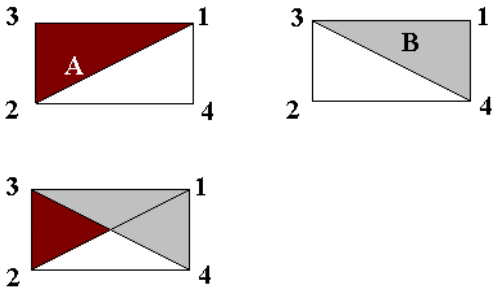
So, for each triangle it is associated a 3x3 matrix, that represents the area double. Through the determinant calculus the normal vector direction is found:

$$\det A = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} \qquad \det B = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_3 & x_4 \\ y_1 & y_3 & y_4 \end{vmatrix}$$

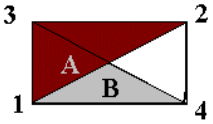
The most interest in this calculation is the resulting signal obtained from the multiplication of the two determinants. If the result is positive it means that the two normal vectors are in the same direction and there is no need to change the drawing sequence. But if the result is negative it will be necessary to establish criteria to define a new drawing sequence. At first the position of the 1 and 2 co-ordinates are changed and the determinant calculus are redone. A new test is done and if the negative result persists, it means that the points 1 and 2 are in the same diagonal and so it will be necessary to change the 2 and 3 co-ordinates positions – see Fig.8.

Findbb. It finds the maximum values for x, y, z.

Pttela. It transforms the edges values calculated to the vector field in display co-ordinates, and keeps the drawing in scale.



changing the 1 and 2 co-ordinates:



changing the 2 and 3 co-ordinates:

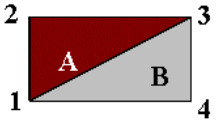


Figure 8 – Co-ordinates change 1 and 2, 2 and 3.

3.3 Transf

The methods, which are responsible by the image transformations, are in this class: *scale*, *translate*, *zrot*, *xrot*, *yrot*, and *ajuste*. These methods were implemented according to (Harrington, 1983). The right-hand axis system was used.

Scale. It is called when the *Zoom* option is chosen. All the co-ordinates are multiplied by the scale factor received like an argument by this method.

Translate. It is called when the *Translate* option is chosen. It is added ± 50 pixels to the axis indicated by the option. This method is too called when the image suffers any other kind of transformation, to adjust the image into the display. The values to be added to the axis are calculated by the *ajuste* method.

Zrot. When rotating Z axis, all the co-ordinates belonging to this axis have no change, while the X and Y co-ordinates behave the same manner as the 2D case. The homogeneous co-ordinates matrix is used.

$$x = x * \cos\theta - y * sin\theta$$
$$y = x * sin\theta + y * \cos\theta$$

Xrot. When rotating X axis, all the co-ordinates belonging to this axis have no change, while the Y and Z co-ordinates suffer some changes. To rotate on the X axis such that Y becomes Z, the homogenous co-ordinates matrix is used.

$$y = y * \cos\theta - z * sin\theta$$
$$z = y * sin\theta + z * \cos\theta$$

Yrot. When rotating Y axis, all the co-ordinates belonging to this axis have no change, while the X and Z co-ordinates suffer some changes. To rotate on the Y axis such that Z becomes X, the homogenous co-ordinates matrix is used.

$$x = x * \cos\theta + z * sin\theta$$
$$z = -x * sin\theta + z * \cos\theta$$

Ajuste – It calculates the necessary adjust to the image into the display. This method is called when some transformation is done.

3.4 Filelister

It is called when the "*Open*" option is activated. It creates the panel that shows the files contained in a directory.

3.5 Endswithfilter

It defines what will be exhibited into the created panel:
. files with *obj* extension and
. the directories list.

4. CONCLUSIONS

The goal of this work was to develop a software to represent graphically vector properties of a certain 3D vector field, using the computational language JAVA. The focus of this work was divided into these two themes: the implementation of the graphical method and exploring this new concept about computational language, JAVA.

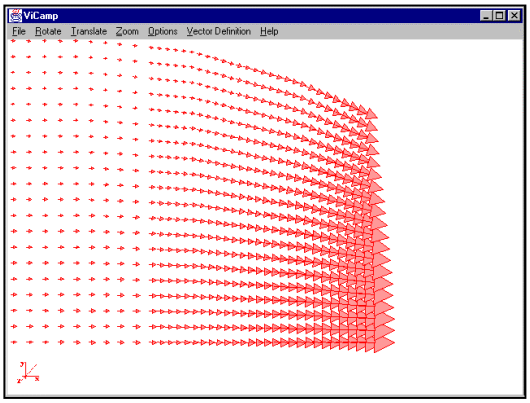
One of the requisites of the software is that it presents user-friendly interface and the results were next to the intuitively wished by the user. For those reasons the Menus were constructed to be simple and self-explained.

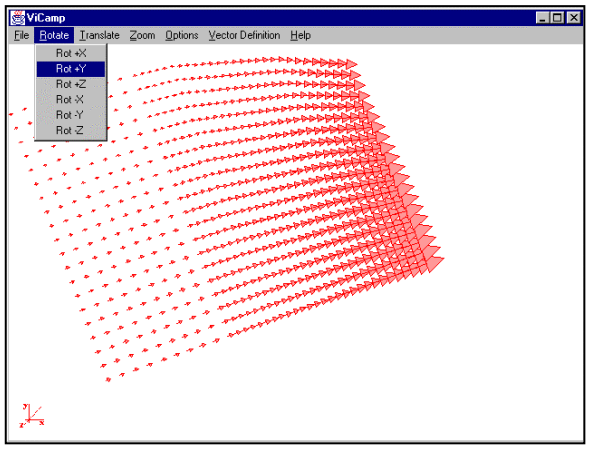
One of the project advantages is about the low disk space required. The necessary files to the system installation are the ones with *class* extension and they require about 40kb. The source files (*.java*) require about 45kb.

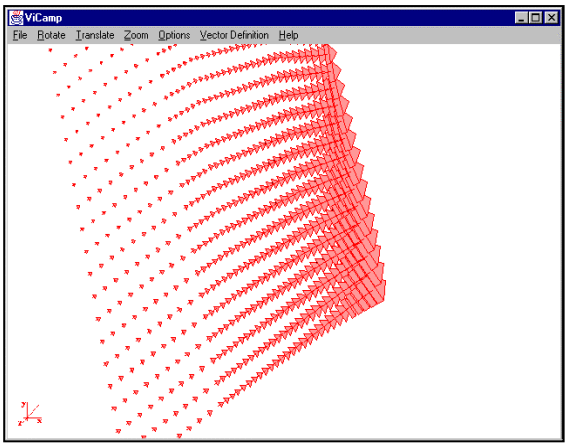
The implementation does not make use of the data reading techniques suggested in (Michaels, C. and Bailey, M., 1997) although efforts had been made in order to reach the five goals proposed in that paper.

5. EXAMPLES

Displays showing the representation of some vector fields with transformations are given below. The Computational Fluid Dynamics group of CTA/IAE generated the data file, that represents the velocity in a region of a Transonic Aerodynamics Tunnel.







REFERENCES

Barbosa, Silvana, A., 1998, Visualização de Campos Vetoriais - Uma Aplicação em JAVA Master of Science Thesis - Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, Brasil.

Barbosa, Silvana, A., and Dos Santos, M. L. X 1999, Visualization of 3D Vector Fields – A JAVA Application, Proceedings of the EuroGraphics UK Chapter 1999.

Boring E., and Pang A., 1996, Directional Flow Visualization of Vector Fields, Proceedings of the IEEE Visualization 96.

Computer Graphics, 1987, Special Issue on Visualization in Scientific Computing, vol 21,nº6.

Dos Santos, M. L. X, 1979, The Computation of Wave Guide Vector Fields and the Generation of Fields Patterns using Computer Graphics"-Ph.D. Thesis , University of Leicester.

Harrington, Steven, 1983, Computer Graphics - A Programming Approach McGraw-Hill Book Company.

JAVA -Mito e Realidade, 1997, Informática Exame, Ano 12, nº 134,pp.24/30.

Leeuw, W. C., and Wijk, J. J., 1993, A Probe for Local Flow Field Visualization, IEEE Visualization93 Conference.

Michaels, C. and Bailey, M., 1997, VizWiz: A JAVA Applet for Interactive 3D Scientific Visualization on the Web, Proceedings of the IEEE Visualization97.

Nassif, N., and Silvester, P. P., 1980, Graphic Representation of Three Component Vector Fields, Computer Aided Design, vol 12.

Simkin, Vector Fields Limited, 1995, Visualization of Three Dimensional Vector Fields and Functions, IEE Colloquium on Visualization of Three Dimensional Fields.