TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics and Mechanics
Vol. 55, Issue I, 2012

SOME ASPECTS OF GRAPHICAL PROCESS SIMULATION IN VISUAL BASIC

ANTAL Tiberiu Alexandru

Abstract: *The paper describes how to turn a simulation of a continuous process to a digital one by accepting the simulation model simplification combined with some advantages of the Visual Basic programming language when dealing with graphical processes, starting from graphical file manipulation to structured programming and graphical user interface access.*
Key words: *Visual Basic, GUI, simulation, structured programming.*

1. INTRODUCTION

The process of experimenting with a computerized mathematical model is called simulation. Simulation is about transforming numbers, or the input data, based on the mathematical model, to obtain the output data, or the results. Sometimes we are interested in obtaining the results as a graphical solution because this will help us to establish rapidly if the mathematical model and the computer implementation of the model are correct. Consider the following problem: We are interested the make a simulation of some particle movements through three environments with different densities. The first two environments are traveled by the particles while the third is impenetrable, leading to reflection. The densities of the two environments are not the same and this will influence the speed of the travelling particles. The structure of the environments is loaded from graphical files which are taken of filming devices. At initial state the particles are considered as occurring only in the lower density environment and their moment is random.

2. SOME WORDS ON VISUAL BASIC

BASIC stands for Beginner's All Purpose Symbolic Instruction Code was a programming language developed at Dartmouth College in 1964 under the directory of J. Kemeny and T. Kurtz. The language was meant to be very simple to help students make the leap to other more complex languages such as FORTRAN or ALGOL. At first it was an interpreted, but after the community realized the potential of the language, compilers have been conceived too. The BASIC used to implement this project was the idea of Bill Gates and Paul Allen in the 1970's and was connected to a personal computer named Altair. The name of this “new” language seems to come from the work of Charles Kay Ogden who wrote a series called "Basic English" (a list of 850 English words which would serve to describe any other word in English). Although the language was invented by Kemeny and Kurtz, Bill Gates, with its well-known candor, reinvents it and sells it under the same name and on different platforms. Visual Basic is derived from BASIC [1]. The term of Visual was a bit misleading, as a language called Visual can be programmed in two distinct spaces, a visual one by using graphical symbols and a textual one, by writing classical ASCII code. So in a Visual language the program can be written by simply dragging graphical symbols instead of writing keywords. Again Bill Gates had its way of dealing “differently” from the civilized world. Visual BASIC was a mixture of two concepts, the

interface build with the help of predefined graphical symbols and the code section behind the program that kept the classical keyword programming approach. Visual Basic 1.0 was released by Microsoft in 1991 for Windows and in 1992 for DOS. The first stable released for Windows, Visual Basic 6 appears on the market in 1998 and this is the version that will be used further in the simulation. The language remains modest in terms of possibilities working directly with 2D or 3D graphics or animation but helps without too much effort to build forms and dialogs.

2. Visual Basic PROJECT CONFIGURATION AND SIMULATOR STRUCTURE

Most of the modern IDEs (Integrated Development Environemets) are using the concept of project [1] – [6] and VB6 (Visual Basic 6) is no different. The simulator has two forms: *frmLoadImg* and *frmGraph* (see Figure 1). They are stored in files with the same name having the *.frm* extension. All files of a project are kept together with the help of the *LoadImg.vbp* project file. The first form holds the interface with all the input of the application and the code to run the simulator, the second one is showing the information that is confirming the correctness of the simulation. The *frmLoadImg* contains also two controls, *Timer* and *CommonDialog*, which are necessary because the application must measure time and work with files.

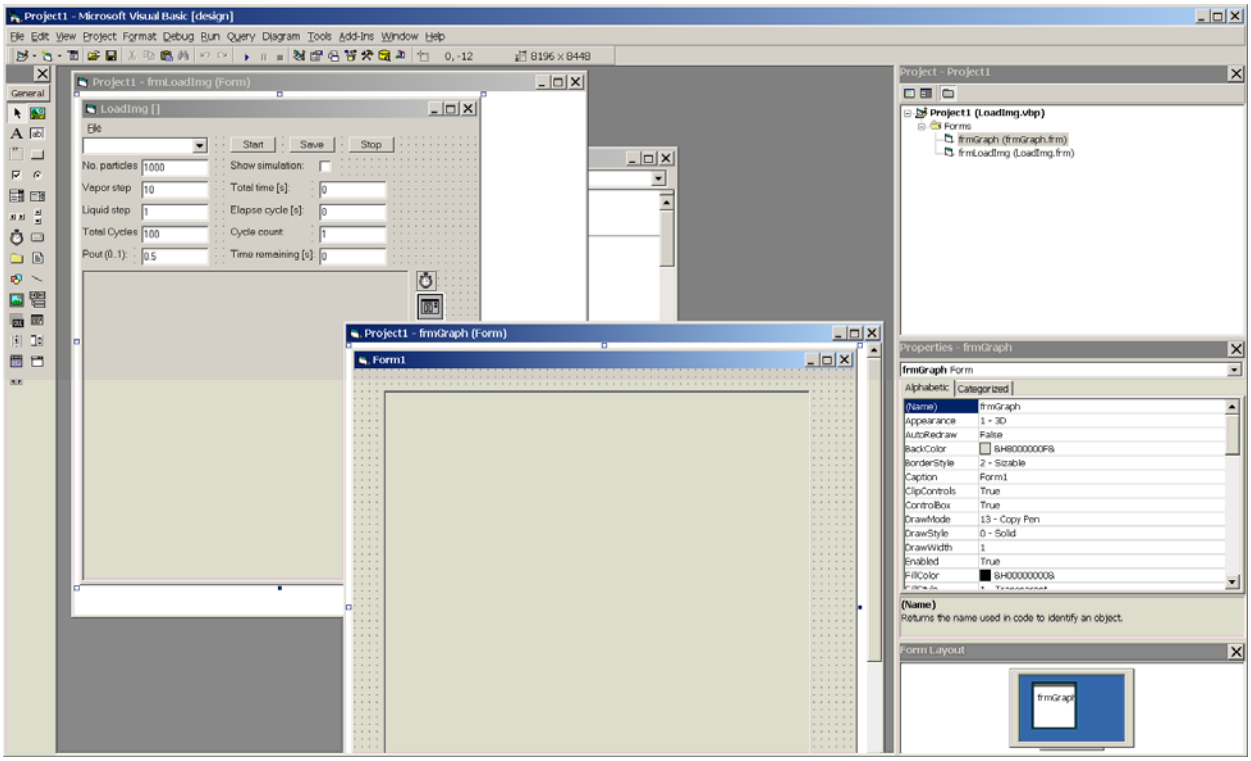

Fig. 1. The VB6 project of the simulator.

3. CONSIDERATIONS ABOUT THE SIMPLIFICATIONS USED IN THE SIMULATOR

The first environment, less dense, is called vapor, the second one that is denser, is called liquid. At initial state all particles appear in the vapor part, their movement is random. The sense and direction of a particle is coded in two variables that can hold three distinct values {-1, 0, 1}, for example the pair (0, 1) means that the particle is moving on the NS direction, with the sense from S to N, while for (0, -1) the particle is moving on the NS direction, with the sense from N to S. In the general case particles move along a straight line, in a plan, with slope and initial random conditions. As this approach is trying to simplify (reduce the computational complexity and time) the process of simulation it passes from the continuous space to the

discrete one where only four directions, each with 2-way, are randomly taken by particles. A particle can pass from the liquid to the vaporous environment and vice versa.

Particle movement in vapor environment is easier, therefore faster, while moving in the liquid medium is more difficult, so more slowly. To simulate the concept of speed this is seen as a variation of space in time. Because particle speeds are constant, speed can be viewed as a step that is bigger when we simulate faster or less when we simulate lover speed in the environment. At this level, another simplification that was considered is related to the selection step in passing from one environment to another. More specifically the current step is kept unchanged following the environment change and only after the next position in the new environment is computed this is updated to the specific step of the current environment. At contact with solid environment the particle is reflected in accordance with classical laws of physics. On contact with the liquid some particles are reflected in the new environment while others pass.

4. DATA STRUCTURES AND CODE MODULES

In order to keep the simulator simple and as fast as possible global arrays are used to store the position, speed and direction for each particle. Initial position of the particles can be only if the environment color is white. Blue is used for liquid, black for solid environment and red for particles initial position and trajectory. Initial data that must be supplied in order to run a simulation is the structure of the environment. This takes the form a picture loaded into the simulator.

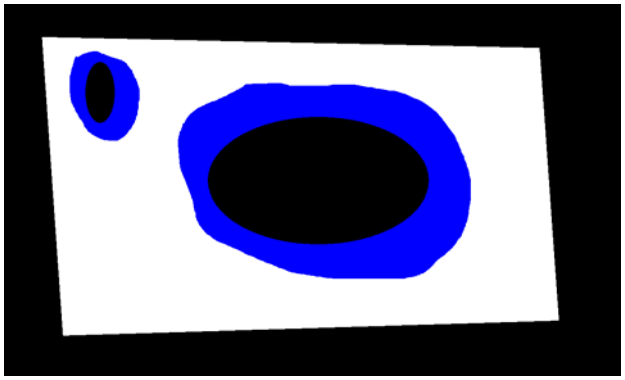

Fig. 2. Simple environment sample for the simulator.

Environment examples are given in Figure 2 to Figure 4. The first two environments are closed to keep the particles inside de viewable window while the last is opened in order to let the particles flow away.

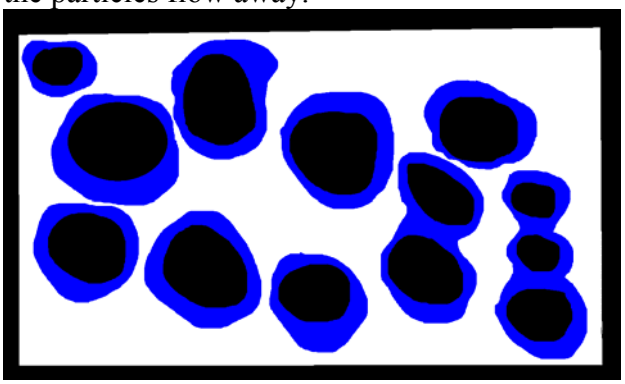

Fig. 3. A complex environment sample for the simulator.

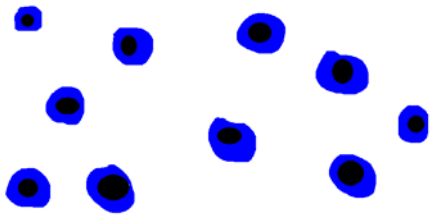

Fig. 4. A complex environment sample for the simulator without boundery.

The images are stored in bitmap files and Visual Basic will load them with a simple `LoadPicture()` function into the `Picture` property of a form, image control or picture box control.

Once the environment is loaded initial points of the particles must be generate. The code used to initialize the simulator is:

```
Public Sub InitPosition()
  Dim i As Long
  ReDim x(1 To NrParticule)
  ReDim y(1 To NrParticule)
  ReDim dx(1 To NrParticule)
  ReDim dy(1 To NrParticule)
  ReDim reflect(1 To NrParticule)
  ReDim delta(1 To NrParticule)
…
  For i = 1 To NrParticule
  'Find an empty point to put a particle
  'This is the start point of the particle
    Do
      Randomize
      x(i) = Rnd * Width
      y(i) = Rnd * Height
```

```
    Loop While (Point color is not white)
    'Set the particle's random trajectory
    traletorie dx(i), dy(i), delta1
    reflect(i) = False
    delta(i) = delta1
    'Particle color is set o red
    Set point x(i), y(i) to red
  Next i
End Sub
```

Initial conditions for a particle are computed by following sequence of code that generates a random sense and direction with the `rx` and `ry` parameters.

```
   Private Sub traletorie (ByRef rx As
Integer, ByRef ry As Integer, delta As
Integer)
       Dim rd As Double
       rd = Rnd
       If (rd < 0.33) Then
           rx = -1 * delta
       ElseIf (rd < 0.66) Then
           rx = 0
       Else
           rx = 1 * delta
       End If

       rd = Rnd
       If (rd < 0.33) Then
           ry = -1 * delta
       ElseIf (rd < 0.66) Then
           ry = 0
       Else
           ry = 1 * delta
       End If
   End Sub
```

In the simulation a set of rules are used to determine how the particles move inside the environment. The physical laws of the universe are simplified to make calculation faster. Because of the simplifications the solution for this problem is based on point processing [2] that is the pixels from the image are processed at a time. The `Point` VB function returns the RGB color (`RGB(255, 255, 255)` is white, `RGB(0, 0, 0)` is black) specification of a point from the image, while the `Line` VB function will show the trajectory of a particle inside the environment. A particle has the `x(i)`, `y(i)` position, while the next position will be `x(i)+dx(i)`, `y(i)+dy(i)`.

The following piece of code gives a pseudo code description of the logic that is driven by the examination a single pixel value in order to compute the output.

```
   For i = 1 To NrParticule

'if point is white
  If (point(x(i), y(i)) = white) Then
    reflect(i) = False

'first point x1, y1
    x1 = x(i)
    y1 = y(i)
'randomly obtain the second point
'characteristics
    dx(i) = Sgn(dx(i)) * delta(i)
    dy(i) = Sgn(dy(i)) * delta(i)
'second point
    x2 = x1 + dx(i)
    y2 = y1 + dy(i)

'if the second point is also white
'draw the line with step delta1
  If (point(x2, y2) = white) Then
    Line x1,y1,x2,y2,red
    delta(i) = delta1
  ElseIf (point(x2, y2) = blue) Then
'the second point is blue
'the step is delta2
    Line x1,y1,x2,y2,red
    delta(i) = delta2
  ElseIf point(x2, y2) = black Then
'second point is black
'reflection is compulsory
    While (point(x2, y2) <> white))
      x2 = x2 - Sgn(dx(i)) * 1
      y2 = y2 - Sgn(dy(i)) * 1
    Wend
    line x1,y1,x2, y2, red
    x(i) = x2
    y(i) = y2

'here the reflexion is computed
    dx(i) = -1 * dx(i)
    dy(i) = -1 * dy(i)
    reflect(i) = True
    delta(i) = delta1

  End If

. . .

Next i
```

The code concerning the simulation is grouped into a single module while the code dealing with the graphical representations is a separate module that is communicating with the first one with the help of global arrays. The role of the graphical module is to check the correctness of the simulation. Choosing a number of particles too low or too short cycles may lead to simulation with bad results. This would lead in the graphical representation of the mean squared displacement of the particles on the x axis to a curve that is not smooth or that is not converging to a constant value in time (cycles).

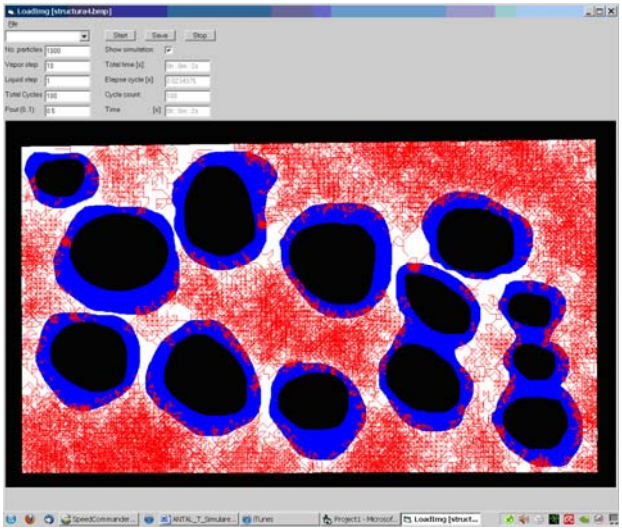

Fig. 5. Complex environment sample for the simulator.

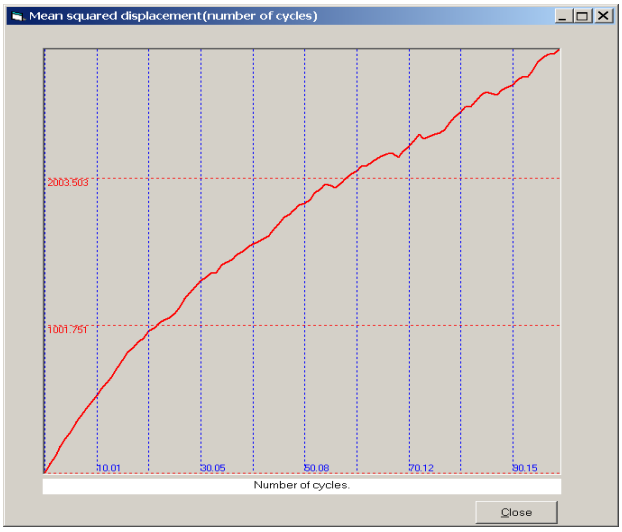

Fig. 6. Mean square displacement for Figure 5.

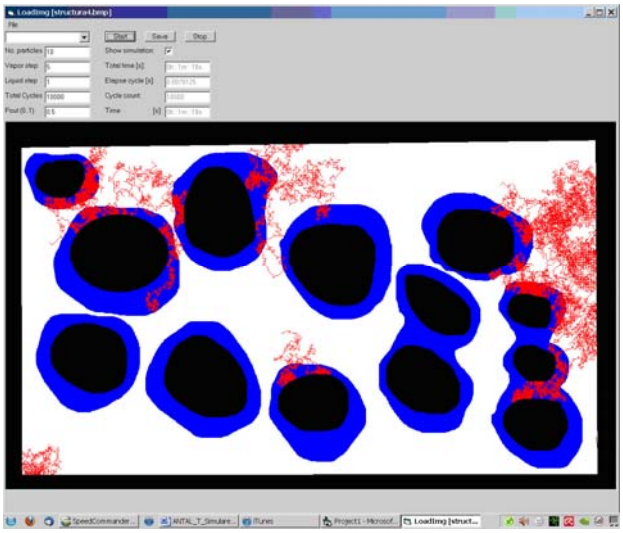

Fig. 7. Complex environment sample for the simulator.

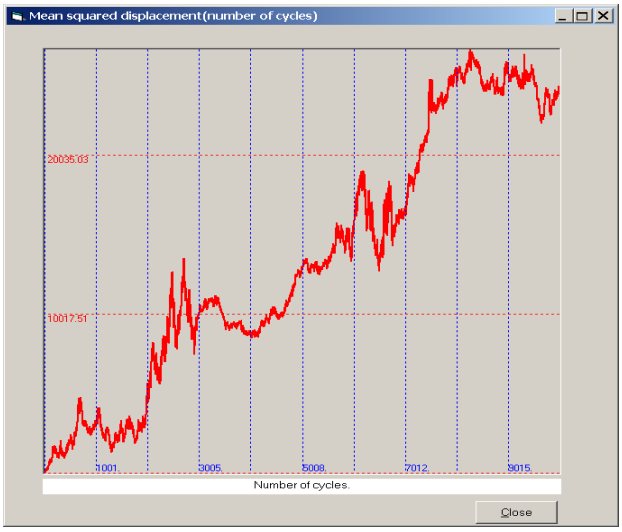

Fig. 8. Mean square displacement for Figure 7.

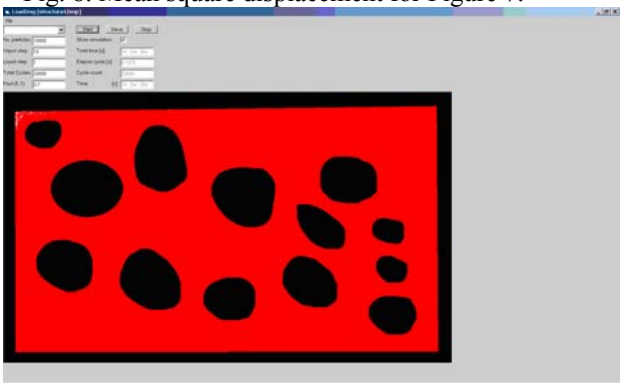

Fig. 9. Complex environment sample for the simulator.

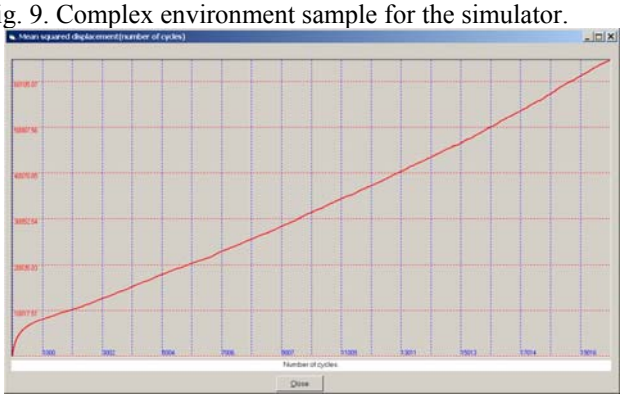

Fig. 10 Mean square displacement for Figure 9.

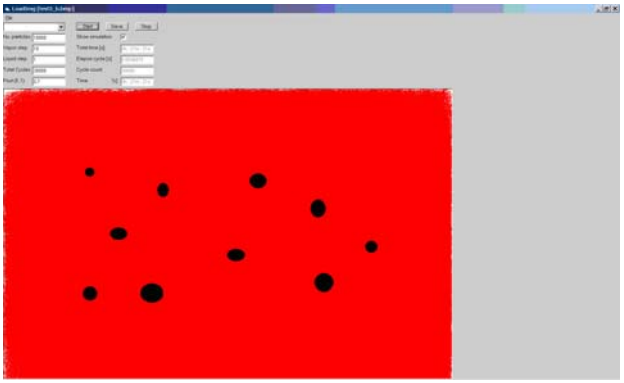

Fig. 11. Complex environment sample for the simulator.

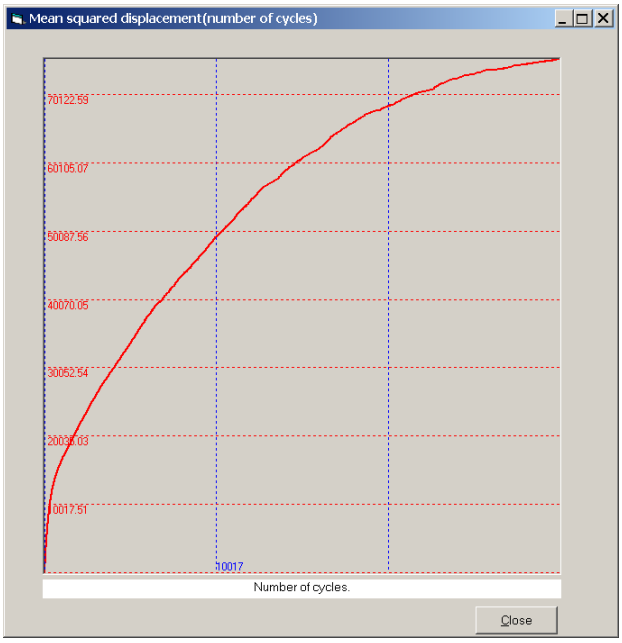

Fig. 12 Mean square displacement for Figure 11.

5. CONCLUSIONS

Figure 5, 7, 9 and 11 are showing results of some simulations, while Figure 6, 8, 10 and 12 are showing the corresponding mean square displacements. For unfinished (early terminated by using the stop button) or to short simulation durations the mean square displacement is not a smooth curve, however if the curve is smooth the simulation is correct. The VB language allows the user to write a structured code organized in reusable modules. The feature of dynamic arrays helps the programmer to work with growable number of cycles without interfering with the source code. Storage and retrieval of the graphical environments is made directly from the language without any special API knowledge as also the point manipulation is supported directly by the language. In conclusion, switching from a continuous process to a discrete one makes the programmer to carry out simple and effective tasks without putting the pressure regarding the graphical knowledge required to work with the Windows operating system.

6. REFERENCES

[1] Antal Tiberiu Alexandru, “Visual BASIC pentru ingineri”, RISOPRINT, Cluj-Napoca, 2003, p. 244, ISBN: 973-656-514-9.
[2] Rod Stephens, “Visual Basic Graphics Programming”, Wiley, 2000, p. 712, ISBN:0-471-35599-2.
[3] Wallace Wang, “Visual Basic 6 for Dummies”, John Wiley & Sons,1998, p. 504, ISBN: 978-0764503702.
[4] Stewe Brown, “Visual Basic 6 Complete”, SYBEX, 1999, p. 1040, ISBN: 978-0782124699
[5] ANTAL, T. A., Visual Basic class for simple surface representation with hidden face removal, Acta Technica Napocensis, Series: Applied Mathemathics and Mechanics, Nr. 46, Vol. 2, 2003, p.15-24, ISSN 1221-5872.
[6] ANTAL, T. A., Surface generation in Catia v5r11 defined by parametrical equations in Visual Basic using ActiveX Automation, Acta Technica Napocensis, Series: Applied Mathemathics and Mechanics, Nr. 46, Vol. 2, 2003, p.7-14, ISSN 1221-5872.

Unele aspecte ale simularii de procese grafice in Visual BASIC

Lucrarea prezintă o modalitate de trecere de la simularea unui proces grafic continuu la unul discret prin acceptarea unor simplificări ce exploatează facilitatile limbajului de programare Visual Basic plecând de la manipularea fişierelor grafice, programarea structurară si pana la accesul la conținutul de grafica prin interfața oferita de limbaj.

ANTAL Tiberiu Alexandru, Associate Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanics and Computer Programming, antaljr@bavaria.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.