C15

CREATION OF 3D ENVIRONMENTS WITH OPEN-GL AND ECOSIMPRO: APPLICATIONS IN ROBOTICS

Fernando Carbonero Martínez
Universidad de Córdoba. Campus de Rabanales. 14014 Córdoba
p52camaf@uco.es

Francisco Vázquez Serrano
Universidad de Córdoba. Campus de Rabanales. 14014 Córdoba
fvazquez@uco.es

Abstract

The majority of simulation software generates the results in the form of data tables and 2D graphics. To interpret these the designer must have a vivid imagination and good spatial vision. Thanks to the possibility offered by EcosimPro in which C/C++ files are created for each experiment, by means of graphics libraries the data can be presented in a 3D environment like Open-GL. In this environment the user can handle the object that has been created as though it were a real prototype. It is an easy and economical way to detect defects or possible improvements, and to present it to clients or personnel who are not accustomed to technical representation. This environment would be suitable for simulating robots because the dynamics usually present movement on the three spatial axes. The mathematics associated with robotics is closely linked to matrix algebra, so the work has been aimed at facilitating the matrix processing within the scalar environment of EcosimPro.

Key words: Simulation, EcosimPro, Open-GL, 3D, C/C++, Robot, matrix processing.

1. COMMUNICATION WITH ECOSIMPRO THROUGH C/C++

Any model connected with EcosimPro [4] can be connected with any other application based on the C++ class generated. In this way, a program can be written so that it can be reused with any other software (real time, etc). The power of this code will be the same as that of an experiment in EcosimPro. Anything that can be done in an EcosimPro experiment can be achieved with a program written by the user. It is also possible to create lists of variables, exactly as they are created during the experiment.

There are numerous possibilities:

- Creation of a graphics interface adapted for the characteristics of the variables
- Connection with a real-time simulation environment
- Connection of the model with programs written by the user
- Connection with other simulation tools

1.1 REAL-TIME DATA RETRIEVAL

The data necessary for a graphic representation can be retrieved using two methodologies, depending on the moment the readout takes place. The data will configure the robot each time the display is refreshed, so one method would be to capture the data just before each refresh. When the algorithms are complex, the calculation can be prolonged over time, which makes it advisable to gather the data first, store them and later retrieve them as necessary.

EcosimPro can be used as input data to obtain the different values of the variables of a given experiment; for example, the position and orientation of an object. Due to the way that Open-GL works, only the actual data will need to be known, unless some process is required in which a greater number of elements is needed. A significant amount of memory can therefore be saved if only the data at each determined instant is stored. In addition, in a computer with a good accelerator card the display refresh time is of the order of 90 frames per second (fps), and this is reduced to about 15 fps in a normal computer without asking it for much complexity in the rendering of the model. The latter will be more typical so there will be sufficient time to progress in the communication interval (it is assumed that the rendering will take place within a fixed time, defined

by the communication interval) and to store the data in a structure like the example shown in Figure 1.

With this method, we can obtain feedback from EcosimPro and alter the variables of the experiment as necessary. This dynamic model would greatly increase system capacity which, instead of being a simple display, would become a complete environment. As an example we can build a robot with EcosimPro and externally model the industrial cell in which the robot would be located. Thanks to the properties of Open-GL, combined with a mathematical base we can detect collisions, calculate trajectories, etc, giving environmental feedback to EcosimPro so that it can recalculate, for example, the moments of each axis of the robot.

If by having recourse to the system clock an actual real-time system can be built, the perception time of our system may be small (atomic models) or great (astral models). The real-time concept can therefore be extended to the fact that EcosimPro communicates with the program in such a way that when the display is refreshed, the updated data is displayed.

1.2 DATA COLLECTION FROM ARRAYS OR FILES

The time needed to perform the calculations of a communication interval may exceed the time needed to refresh the display. Furthermore, if the mathematical model is very complex the calculation time may be very significant indeed. For this reason, a second method of data retrieval has been used.

This second method constitutes our experiment with EcosimPro. Before continuing with the program, the results are stored with the same structure shown in Figure 1, having already reserved the arrays based on their a priori dimension (we have the initial time, the final time and the communication interval) or, if preferred, through the use of lists. We could also resort to the MFC [10] since they too allow the size to be dynamically increased. In cases where the computation time is very long, a preliminary program could be run in which the data are stored in a file. These data would be retrieved later by another program which could be executed in real time or by arrays.

Of course, nothing stops us from carrying out the same example of the robot with the environment described above. In this case the rendering would go one step further in the mathematical calculation of the model, since the graphics displayed in this phase would not be of much interest (due to the excessive time they would take) whereas they will be later on when, having retrieved all the data, they will be displayed at a suitable speed. Although we lose the idea of Open-GL to a certain extent, we shouldn't forget this alternative with regard to other procedures such as modelling the same environment in a mathematical way.

1.3 EXAMPLE OF DATA RETRIEVAL

Figure 1 shows a ".c" file using the "skier" example, in which the header has been included to shorten to code. The method of data retrieval is based on arrays.

```
/************************************************
Fernando Carbonero Martinez
Francisco Vazquez Serrano
Simulacion Open-GL con datos de EcosimPro
Obtencion de Datos de Ecosim
Esquiadores por una superficie
************************************************/

#include <stdio.h>
// Inclusion de la clase experimento generada por EcosimPro
#include "sky.elige.exp1.h"

struct datos
{
          float time_ini, time_stop, cint;
          float ** posicion_X;
          ...

          int fin;
};

typedef struct datos DATOS;

void DatosEcosim(DATOS *datos);

// Librerias que hay que incluir
#pragma comment (lib, "Wsock32.lib")
#pragma comment (lib, "C:\\Archivos de programa\\Ecosim
\\lib\\Integ_mt.lib")
#pragma comment (lib, "C:\\Archivos de programa\\Ecosim
\\lib\\Dformd.lib")
#pragma comment (lib, "C:\\Archivos de programa\\Ecosim
\\lib\\Dfconsol.lib")
#pragma comment (lib, "C:\\Archivos de programa\\Ecosim
\\lib\\Dfport.lib")

// Hay que ignorar "libcd.lib"

void DatosEcosim(DATOS *datos)
{
   int i=0;
   int estado=0;

   sky_elige_exp1 exp;

   datos->time_ini = 0;
   datos->time_stop = 40;
   datos->cint = 0.1f;

   // Reversa de arrays de datos
   datos->posicion_X = (float**) calloc (6, sizeof(float*));
          ...

   for (i=0; i<6; i++)
   {
      datos->posicion_X[i] = (float*) calloc ((int)((datos-
>time_stop-datos->time_ini)/datos->cint), sizeof(float));
          ...
   }

   initEcosim( &exp);
   g_logProgramme= false;   // disable log files
   g_warnProgramme= false; // disable range checking
   g_traceProgramme= false; //  disable simulation trace

   // Inicializacion de las variables
   exp.setValueReal("x[A]", 0);
          ...

   exp.TIME= datos->time_ini;
   exp.TSTOP= datos->time_stop;
   exp.CINT= datos->cint;

   // Bucle de obtencion de datos
   do
   {
      estado = exp.INTEG_STEP();
      if (estado == IS_CINT )
      {
         // Almacenamiento de datos en arrays procedentes de
Ecosim.
         datos->posicion_X[0][i] = (float)exp.getValueReal
("x[A]");
          ...
         i++;
```

```
    }
  } while (estado != INTEG_END);

  datos->fin = i-1;
}
```

Figure 1. File of the "Skier" Example of data retrieval from EcosimPro

First we have to include the classes which EcosimPro has generated for the experiment to be displayed. We then have to include a series of libraries but ignore "libcd.lib" which has to be dealt with from the Visual C++ menus [10].

Seeing that the data retrieved from EcosimPro are going to by used by the majority of the program's functions, a group structure has been formed which we will define as "type".

The "DatosEcosim" function will be called each time new data are required (the data can be changed in mid-execution to switch between different initial conditions in the experiment or simply because the real-time method is used). The example shows the communication both in the direction of EcosimPro and from it.

In order to simplify the code only one of the variables is shown, and "..." is used to proceed the same way with the others.

2. THE THREE DIMENSIONS (3D)

A computerised three-dimensional figure could be defined as an unlimited number of photographs of an object or a series of objects. But to end the definition here would be to underrate it because, as the scenes are manipulatable we can obtain a result close to reality. In other words, although the object is not tangible, by means of the peripherals we can rotate it, zoom in on it, etc, to make it act as its equivalent would (carrying out the movement, adjusting the characteristics for light, shade, etc), creating an illusion of credibility.

The importance of 3D images lies in the facility with which the designer can "see" the dynamic behaviour of his project before it is put into practice, making it possible to detect very many anomalies and improvements which would be difficult to spot with standard procedures (numerical data, graphs).

It is a very simple tool for presenting a prototype to clients or non-technical personnel who do not have to have a great aptitude for spatial vision, which means that the information required for the project can be transmitted reliably and quickly.

2.1 OPEN-GL

Open-GL [9] can be strictly defined as "graphical device interface". In essence, it is a 3D graphics and portable modelling library, and very fast. With Open-GL we can create lovely elegant 3D graphics with a close-up quality close to that of a ray tracer, but with the great advantage of being quicker than a tracer.

It utilises algorithms developed and optimised by Silicon Graphics, Inc. (SGI). Open-GL is intended for use with hardware designed and optimised for displaying and handling 3D graphics. Generic uses of Open-GL -which only comprise code- are also possible, and Windows-based rendering comes under the same category.

PC hardware companies are now adding graphics accelerators to their products, mainly in response to the demand of the 3D games market which is closely following the evolution of Windows-based 2D graphics. The accelerators optimise operations such as line drawing and the completion and handling of bitmaps.

Open-GL is a procedural rather than a descriptive system. Instead of designing the scene and how it should appear, the programmer describes the necessary steps to achieve a certain appearance or effect. These "steps" involve calls to a highly portable API which includes approximately 120 statements and functions. These are used to draw primitive graphics in three-dimensional points, lines and polygons. In addition, Open-GL supports light and shade, texture mapping, animation and other special effects.

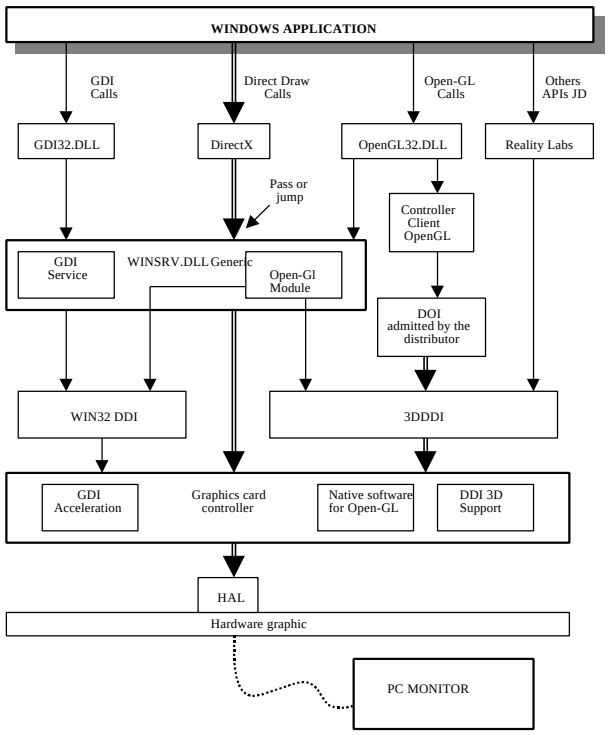

Figure 2: Operation of an Open-GL 3D Graphics Library [9]

Open-GL does not include any functions for managing Windows, for interaction with the user or for managing files. Each host environment (like Microsoft Windows) has its own functions for this purpose and is responsible for providing Open-GL with the means to draw in a window or bitmap.

2.2 MODELLING PROGRAMS IN 3D

3D modelling consists in making a visual representation of an object or a series of objects by means of a computer or any other device with which the final model can be observed from any angle. As an example, Figure 3 shows the graphics model of the PUMA 760 robot and its environment, in this case a laboratory. Figure 4 shows one of the multiple views of the model which coincides with that of the photograph.

There is a wide range of 3D modelling programs on the market. Depending on the requirements, there is a wide choice of applications, many of which are freely available. Some, such as those based on POV-Ray [7] offer many possibilities and are generally simple and easy to use. In addition, there are the large modelling and CAD programs used both in engineering and in the film industry.

These programs normally work either by processing basic figures (polyhedrons, spheres, etc) or by creating mesh systems. One of the main problems that may be encountered is that of optimising the number of polygons in the model so that the graphics card can refresh the display at an acceptable speed. So unless an Open-GL graphics accelerator is installed, the complex modelling systems of programs should not be used because although it may make the presentation more attractive, the system might be subject to excessive delay due to the high number of polygons.

Another advantage of these programs is that materials and textures (images) can be applied to the faces of the objects, which decreases the number of vertices and offers a high level of credibility.

Without any doubt, the “wysiwyg” feature (acronym for *what you see is what you get*) is the most significant characteristic of modelling programs. It is important to be able to view the object as you are creating it. Open-GL has its own statements for creating the basic figures and the meshes. Whatever is finally developed will be interpreted by these statements (although the tedious task of conversion will be carried out by programs specific for that purpose). However, it would be very difficult to limit ourselves just to these statements while we have to compile the program so that we can see if that which has been coded corresponds with what was required.

2.3 CONVERTERS

Each modelling program allows us to store the results in a file with a determined extension. The format in which the data are stored, and even the actual data, may vary considerably between one software package and another. The structure of normal Open-GL data usually consists in a ‘.c’ or ‘.cpp’ file which contains a series of arrays which store the data in a grouped manner. Because of these differences, an intermediate step between the 3D modelling and the introduction of data into our program would be to convert the format of the stored data so that they can be read by our application.

Without these converters, modelling with programs external to Open-GL would not make much sense. As with the previously mentioned software, there is also a wide variety in this case, both from the point of view of cost and of the functions. However, only a limited number of freeware converters are available on the Internet which convert to Open-GL. If you don’t want to spend money in this phase, you could resort to converters which convert from the format in which the model has been built to one that is acceptable and later use another, final converter. There are, of course, good commercial brands available.

Perhaps, in addition to accepting the primitive format and the ‘.c’ or ‘.cpp’ file for Open-GL, the most important aspect of the converter is the way in which it returns the code because they vary from some simple numerical vertex matrices to different matrices (vertices, faces, normal, textures, lights, etc) and the functions to improve them. Although very few export lights, the lights themselves are usually introduced directly by the programmer with Open-GL statements.

3. ROBOTICS

The field of robotics is one in which simulation is being used most as a fundamental design tool. In spite of the fact that robot modelling can be interpreted just as the definition of cinematic and dynamic problems or, depending on the purpose of the calculations, of direct and indirect problems, it could be taken as far as the electrical stage which is very often forgotten by specific robot design programs.

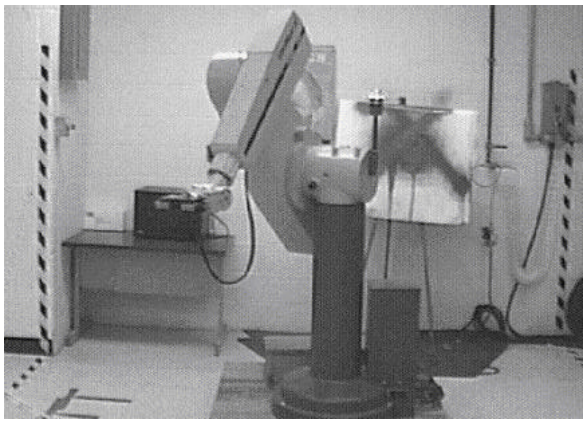

Figure 3. Digital image of the PUMA 760 in the CAE laboratory[6]

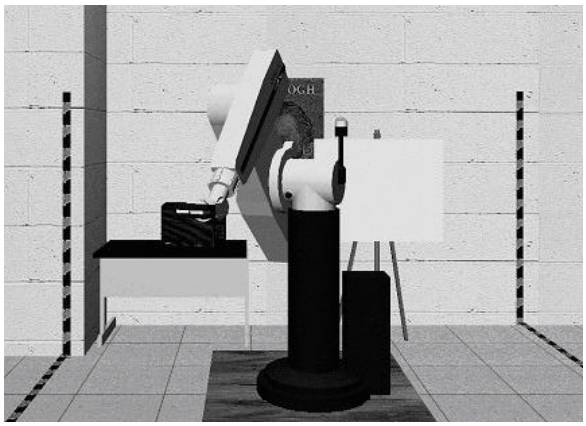

Figure 4. Graphics model of the PUMA 760 in the CAE laboratory [6]

Thanks to the fact that EcosimPro is a general purpose tool, and that it includes an electrical library and a control library, it constitutes an ideal framework for the creation of a robotics library.

3.1 ROBOTICS LIBRARY FOR ECOSIMPRO

One of the current lines of work of authors is the implementation of a robotics library under EcosimPro. The aim is to make it sufficiently complete to be able to carry out all the simulations that could be carried out with a simulator of a specific nature.

It is also intended to create it so that it can connect with the other libraries in EcosimPro; more specifically, those to which it is more closely linked, such as the electrical library and the control library.

The robotics library will contain the following groups of elements:

- Basic construction elements: these are the links of the cinematic chain, the physical part
- Forces and moments
- 2D axes:
  - Revolution
  - Prismatic
  - Screw
- 3D axes:
  - Cylindrical
  - Universal
  - Planar
  - Spherical
  - Cardan

Perhaps the best definition of this library is that it is the most important part of the project since not only the possibilities that are offered depend on it, but also the speed at which the variables are calculated which will subsequently enable 3D display. What's more, we mustn't forget that we're not looking for a nice film scene, but a result as near to reality as possible. In principle, the objective is to use the resulting software in engineering tasks, not in artistic activities (although we can't disregard its possible use in such industries for the construction of games, virtual worlds, etc).

3.2 THE FOUR DIMENSIONS OF ROBOTICS

Seeing that robots are by definition bodies in movement, we will define the fourth dimension as the introduction of time in the Cartesian axes (3D). It is important to be able to represent the four axes as a whole.

Figure 5 represents robot manipulation on the axes *x, y, z*. Let us suppose that at a determined instant in time, the position indicated corresponds to the position of the robot manipulator in Figure 4. Interpretation of the graph unquestionably requires significantly more effort that the simple glance it takes to work it out from the image of the robot.

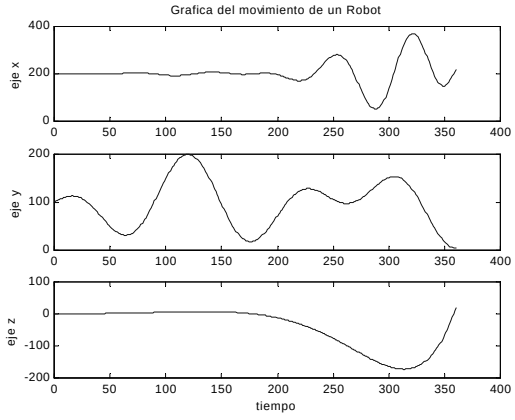

Figure 5. Graph of the movement of a robot

We will add two new parameters: we will first position two robots so that they partially coincide in the area of work, and we will also introduce the fourth dimension. If we now add movement to the

complete assembly, it becomes clear why this configuration has been avoided in the industry.

When the area of work of one of the machines is largely invaded by the other, the design of the movements becomes too complicated. It is also possible that one of the robots might have enough degrees of freedom to enable different configurations of the axes by positioning the manipulator at a determined point.

Being able to display a robot or different robots might not make drawing the routes of the different mobile parts any easier, but we believe it would greatly reduce the time it takes to detect conflicts and give clear ideas as to how they can be solved, thus facilitating the work of the programmer and making it more reliable.

3.3 MATRICES AS A MATHEMATICAL BASE

Because of the spatial nature of robot systems, it is important to group all existing parameters: location and orientation.

$$\begin{bmatrix} \cos(\alpha) & -\text{sen}(\alpha) & 0 & L\cdot\cos(\alpha) \\ \text{sen}(\alpha) & \cos(\alpha) & 0 & L\cdot\text{sen}(\alpha) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The foregoing matrix could correspond to that of the individual transformation of a single torque in which rotation takes place on a single Cartesian axis in accordance with the Hartenberg-Denavit method [1, 5, 8]. If the system had three more articulations, the position and orientation of the manipulator would be given by expression 2, where the superscript indicates to which bar it is referred and the subscript indicates on which bar the operation takes place; ${}^{0}T_{3}$ would be the transformation of bar 3 referred to axis 0, in this case the base.

$${}^{0}T_{3}={}^{0}T_{1}*{}^{1}T_{2}*{}^{2}T_{3} \quad (2)$$

We can already tell intuitively that the biggest advantage of the matrix structure lies in the simplicity and clarity of the resulting expressions. Without this grouping, in the previous case we would have to resort to 12 formulae (we have to take into account that the last column is an addition which doesn't supply information, so it will have to be calculated), and these formulae are very often considerably long.

4. MATRIX ALGEBRA

The fact that EcosimPro does not contemplate matrix processing presents a significant setback because of the great importance of the advantages gained in using this tool.

This problem has been solved in part with the design of a program which partially resolves the problem we are faced with.

The application is basically a text analyser. A series of operations is identified on the basis of a source file and the matrices are then separated into a series of equivalent scalar operations so that EcosimPro can perform timely algorithms (clear, arrange, etc).

4.1 FILES

The program is an executable file which depends on some others:

- File to be created by the user. It must include all the variables with which the program works.
- File again created by the user. It must include the formulae to be transformed from matrix to scalar.
- File furnished by the application. It contains the scalar results along with any data that will assist in the detection of errors.

4.2 OPERATIONS

The operations it detects are: parentheses, transposed matrix (because it is an operation that is not considered, it has been assigned the symbol "#"), derivative "'", product "*", scalar product: "*" (the scalar operator is detected automatically), sum: "+", scalar sum: "+" (the scalar operator is detected automatically), subtract: "-", scalar subtraction: "-" (the scalar operator is detected automatically) and equals "=".

The order of priority is established by the order of appearance in this section, that is:

(), #, ', *, +, -, =

It also detects numbers, and decimals can begin with a number or with '.'. Any sequence which begins with a number will be taken as a numerical expression; therefore, no variable should begin with a number or with '.'.

4.5 EXAMPLE

```
B 1 3
C 1 3
D 1 1
```

Figure 6. "variables.txt" file

```
C + D * (C*.57 - B')
```

Figure 7. "formulas.txt" file

```
**********************************
   MATRICES 2 ESCALARES v.1.1
   Archivo de salida
**********************************


- - - VARIABLES - - -

NOMBRE  FILAS   COLUMNAS
  .57            1       1
  B              1       3
  C              1       3
  D              1       1


- - - FORMULAS - - -

C+D*(C*.57-B')



- - - EXPRESIONES FINALES - - -

Desarrollo de la expresion: C+D*(C*.57-B')


(C[1,1]+D*(C[1,1]*.57-B[1,1]'))

(C[1,2]+D*(C[1,2]*.57-B[1,2]'))

(C[1,3]+D*(C[1,3]*.57-B[1,3]'))
```

Figure 8. "salida.txt" file

5. 3D EXAMPLES

We have chosen two examples to illustrate an experiment in 3D. Each of them contains different possibilities so that we can get an overall idea of the capabilities of this tool. For ease of understanding, we have used the examples published in the EcosimPro web site.

5.1 THREE BODIES

Based on the data gathered in *Demo1: Modeling a three bodies problem using EcosimPro* [2]. In this problem, an attempt is made to integrate a very simple system of gravitational interaction into three bodies. It is assumed that two of them are fixed in space and separated by a given fixed distance while the third rotates around them. It is also assumed that movement takes place on a plane.
The problem is displayed on a fixed Open-GL console, as shown in Figure 9. It has two great qualities:

- Exactly as in the demonstration, three experiments have been carried out. In each of these experiments the view point is situated so that the display is correct.
- Data can be gathered using the two methods described earlier: in real time and by storage in arrays.

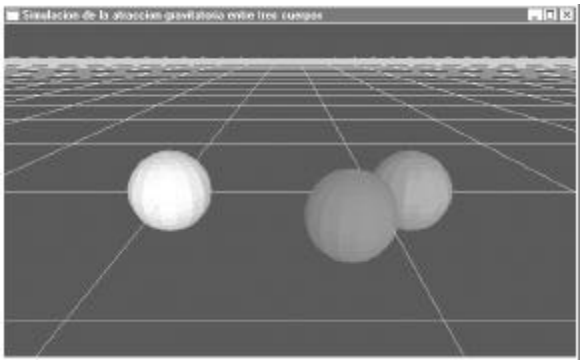

Figure 9. Image captured from the program "3bodies"

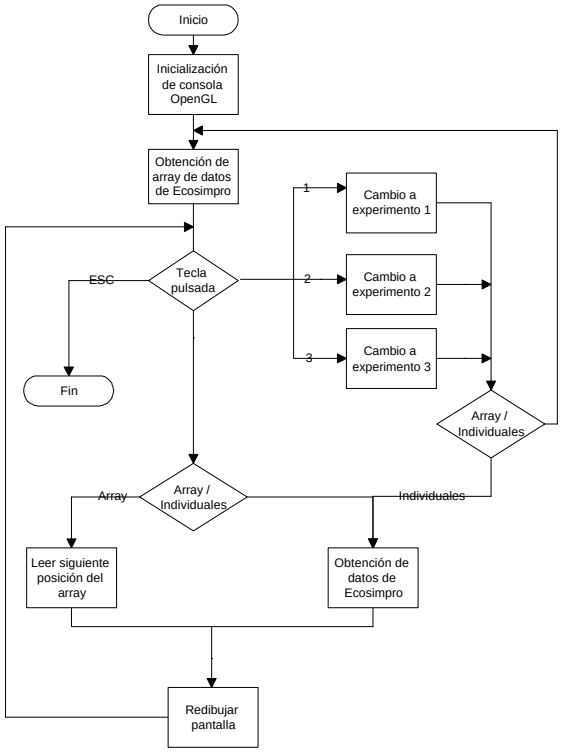

Figure 10. Flow diagram of the program "3bodies"

5.2 SKIER

Based on the data gathered in Demo4: *Simulating sledge slicing by a ski-track* [3]. In this problem, the simulation is of a ski-run on which a skier freely runs at a certain initial speed.

The problem, displayed on a Windows console, is similar to the previous one. On this occasion we simply use the events of Windows (keyboard, timer, etc). It has two great qualities:

- The body is no longer rendered as a simple geometrical figure (the previous one was a sphere) but as a figure modelled by an external

program and a mesh defined by means of an equation.

- Using the keyboard, we can situate the view point anywhere: rotation on all axes as well as zooming in and zooming out. We can also pause the process.

The flow diagram would be very similar to that shown in Figure 10. In this particular case data can be stored only in arrays and so the right-hand branch would be eliminated.

Before the display is refreshed the position of the observer will have changed, whether because the skier has progressed or because the user has pressed a key. The area of "snow" that is visible will also be calculated because if the entire surface is drawn, the number of polygons will increase without providing more information unnecessarily.

Figure 11. Image captured from the "skier" program

This example shows us how easy if would be to create a game with our simulation. We would simply have to contemplate an external force (player) in the EcosimPro model, place the flags on the ski-run and increase the counter each time our skier passed one. And to end the game, a finishing line and a cup for our champion!

References

[1] Craig, J.J. (1986) *Introduction to Robotics, Mechanics and Control*. Addison-Wesley.

[2] EA International, (2000) *Demo1: Modeling a three bodies problem using EcosimPro*. http://www.ecosimpro.com/articles/ecosim_demo1.zip

[3] EA International, (2000) Demo4: *Simulating sledge slicing by a ski-track*. http://www.ecosimpro.com/articles/ecosim_demo4.zip

[4] EA International, (2000) *EcosimPro v3 Manuals*. http://www.ecosimpro.com/download/manuals.htm

[5] Fu, González and Lee, (1989) *Robótica: Control, detección, visión e inteligencia*. McGraw_Hill.

[6] Matthew R. Stein, Shawn Falchetti, (1997). *A New Graphics Simulator for RCCL and its use in Undergraduate Robotics Instruction*. ICAR97. http://yugo.mme.wilkes.edu/~mstein/publications/ICAR97.html

[7] *POV-Ray – the Persistence of Vision Raytracer*, (2001). http://www.povray.org/

[8] Shabana, Ahmed A., (1995) *Computational dynamics*. John Wiley and Sons, New York.

[9] Wright J.R. and Sweet, (1997) *Programming in Open-GL*. Editorial ANAYA.

[10] Zaratian, B, (1999) *Microsoft visual C++ 6.0: manual del programador*. McGraw-Hill, Madrid, Spain.