C15

CREACIÓN DE ENTORNOS 3D CON OPEN-GL Y ECOSIMPRO: APLICACIONES A LA ROBÓTICA

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

Resumen

La mayoría del software de simulación proporciona como resultado tablas de datos y gráficas 2D. Esto hace que el diseñador realice un gran trabajo de imaginación y la necesidad de poseer una buena visión espacial. Gracias a la posibilidad que EcosimPro ofrece al crear ficheros C/C++ para cada experimento, se pueden presentar los datos en un entorno 3D a través de librerías gráficas como OpenGL. Este entorno permitirá manipular el objeto creado como si de un prototipo real se tratase, pudiendo observarse defectos o posibles mejoras de forma sencilla y económica así como su presentación al cliente o personal no acostumbrado a las representaciones técnicas. Este entorno puede ser adecuado para la simulación de robots debido a que su dinámica suele presentar un movimiento en los tres ejes espaciales. La matemática asociada a la robótica está enormemente ligada al álgebra matricial, por lo que se ha trabajado para facilitar el tratamiento matricial dentro del entorno escalar de EcosimPro.

Palabras Clave: Simulación, EcosimPro, OpenGL, 3D, C/C++, Robot, Tratamiento matricial.

1. LA COMUNICACIÓN CON ECOSIMPRO A TRAVÉS DE C/C++

Cualquier modelo creado con EcosimPro [4] puede ser conectado con cualquier otra aplicación basándose en la clase C++ generada. De esta forma un programa puede ser escrito para ser reutilizado desde cualquier otro software (tiempo real, ...). La potencia de este código será la misma que la de un experimento en ECOSIM. Todo aquello que pueda ser realizado en un experimento de EcosimPro puede lograrse con un programa escrito por el usuario. También es posible crear listados de variables, exactamente como se crean durante el experimento.

Las posibilidades son numerosas:

- Creación de un interfaz gráfico adaptado para las características de las variables.
- Conexión con un entorno de simulación de tiempo real.
- Conexión del modelo con programas escritos por el usuario.
- Conexión con otras herramientas de simulación.

1.1 OBTENCIÓN DE DATOS EN TIEMPO REAL

Los datos necesarios para una representación gráfica se pueden conseguir siguiendo dos metodologías dependiendo del momento en que se realicen las lecturas. Un primer caso sería el obtener los datos que van a configurar al robot en cada refresco de pantalla justo antes de dicho refresco. Cuando los algoritmos son complejos, el cálculo puede prolongarse en el tiempo, siendo más aconsejable obtener en un principio los datos, almacenarlos y después ir recuperándolos a medida que se necesitan.

EcosimPro puede ser utilizado como entrada de datos para obtener los distintos valores de las variables de un experimento dado, por ejemplo la posición y la orientación de un objeto. Por el modo de trabajo de OpenGL, sólo se necesitarán saber los datos actuales, a menos que se quiera hacer algún tratamiento en el que se necesite un número mayor de elementos. Por lo tanto se puede ahorrar una importante cantidad de memoria si sólo se almacenan los datos de cada instante determinado. Además, el tiempo de refresco de pantalla es del orden de 90 frames por segundo (fps) si se posee una buena tarjeta aceleradora, bajado a unos 15 fps para un ordenador normal sin pedirle una gran complejidad en el modelo a renderizar. Este

último será el caso más típico por lo que se tendrá tiempo suficiente de avanzar en el intervalo de comunicación (se supone que la renderización se realizará en un tiempo fijo, definido por el intervalo de comunicación) y almacenar los datos en una estructura como por ejemplo la que se puede observar en el código de la figura 1.
Con esta filosofía se podría realizar un feedback con EcosimPro alterando las variables del experimento que sean precisas. Este modelo dinámico incrementaría enormemente la capacidad del sistema, que pasaría de ser de un simple visualizador a un entorno completo. Como ejemplo se puede realizar un robot con EcosimPro y de forma externa modelar la célula industrial en la que se encontraría situado el brazo robot. Gracias a las propiedades de OpenGL unidas con una base matemática se podría realizar la detección de colisiones, cálculos de trayectorias,... devolviendo los datos de entorno a EcosimPro para que recalcule, por ejemplo, los momentos de cada eje del robot.

Si bien, recurriendo a la lectura del reloj del sistema se podría realizar un verdadero sistema de tiempo real, puede ser que el tiempo de percepción de nuestro sistema sea muy pequeño (modelos atómicos) o muy elevado (modelos astrales) por lo que se extenderá el concepto de tiempo real al hecho de que la comunicación de EcosimPro con el programa se realice de modo que cuando se vaya a realizar un refresco de pantalla ya se tenga a disposición los datos actualizados.

1.2 OBTENCIÓN DE DATOS A PARTIR DE ARRAYS O FICHEROS

El tiempo necesario para que se realicen los cálculos de un intervalo de comunicación puede exceder del tiempo que se necesita entre refresco de pantalla. Es más, si el modelo matemático es muy complejo este podría durar un tiempo realmente importante. Por ello se ha realizado un segundo modo de obtener de datos.

Este segundo método sería el lanzamiento del experimento de EcosimPro. Antes de seguir con el programa se almacenan los resultados en la misma estructura de la figura 1 habiéndose reservado los arrays ya que se conocen su dimensión a priori (se dispone del tiempo inicial, el final y el intervalo de comunicación) o si se prefiere mediante el uso de listas. También se podría recurrir a las MFC [10] ya que permiten incrementar dinámicamente su dimensión. Para los casos en los que el tiempo de computo sea muy largo se podría lanzar un primer programa en el que se almacenen los datos en un fichero, siendo recogidos estos más adelante por otro programa el cual puede seguir el modo de tiempo real o el de array.

Desde luego nada impide realizar el mismo ejemplo del robot con el entorno del apartado anterior. En este caso la renderización sería un paso más en el cálculo matemático del modelo ya que los gráficos presentados no tendrían mucho interés en esta fase (por el excesivo tiempo que tardaría) sino más tarde, cuando, teniendo todos los datos, se representase a una velocidad adecuada. Aunque se pierde un poco la filosofía de OpenGL, no se debería olvidar esta alternativa ante otros procedimientos como modelar el mismo entorno de forma matemática.

1.3 EJEMPLO DE OBTENCIÓN DE DATOS

En la figura 1 se muestra en fichero ".c" utilizado en el ejemplo "esquiador", en el que se ha incluido la cabecera para acortar el código. Se usa el método de obtención de datos a partir de arrays.

```
/************************************************
Fernando Carbonero Martinez
Francisco Vazquez Serrano
Simulacion OpenGL 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;
}
```

Figura 1. Fichero del ejemplo “esquiador” para extraer los datos de EcosimPro

En primer lugar se han de incluir las clases que EcosimPro ha generado para el experimento que se va a visualizar. A continuación hay que incluir una serie de librerías así como ignorar la “libcd.lib” que se ha de hacer desde los menús de Visual C++ [10].

Dado que los datos obtenidos de EcosimPro van a ser utilizados por la mayoría de las funciones del programa, se ha realizado una agrupación en estructura, la cual se definirá como tipo.

La función “DatosEcosim” será llamada cada vez que se requieran nuevos datos (en medio de la ejecución se podrían cambiar los datos para conmutar entre distintas condiciones iniciales del experimento o simplemente porque se utilice el método de tiempo real). En el ejemplo se muestra tanto la comunicación hacia EcosimPro como desde él.

Para simplificar el código sólo se muestra una de las variables, indicándose por “...” que se debe proceder de igual manera con el resto.

2. LAS TRES DIMENSIONES (3D)

Una figura tridimensional por ordenador podría definirse como un número ilimitado de fotografías obtenidas de un objeto o conjunto de ellos. Pero terminar aquí la definición sería un menosprecio ya que debido a las características de manipulabilidad de las escenas se obtiene un resultado cercano a la realidad, esto es, aunque el objeto no sea tangible, sí que por medio de los periféricos se puede girar, acercar,... actuando éste como su homólogo real haría (realizando el movimiento, ajustándose a las características de luces, sombras,...), creándose una ilusión de verosimilitud.

La importancia de las 3D estriba en la facilidad que da al diseñador de “ver” el comportamiento dinámico de su proyecto antes de ser llevado a la práctica, pudiéndose detectar un gran número de anomalías y mejoras que con los procedimientos estándares (datos numéricos, gráficas) serían difícil de observar.

Es una herramienta muy sencilla para presentar un prototipo a clientes o personal no técnico, los cuales no tienen que poseer grandes dotes de visión espacial, con lo que se conseguirá transmitir de forma más fiable y rápida la información requerida del proyecto.

2.1 OPENGL

OpenGL [9] se define estrechamente como "una interfaz software para gráficos por hardware". En esencia, es un librería de gráficos 3D y modelado portátil y muy rápida. Usando OpenGL se pueden crear gráficos 3D elegantes y bellos con una calidad visual cercana a la de un trazador por rayos, con la gran ventaja de ser más rápida que un trazador.

Emplea algoritmos desarrollados y optimizados por Silicon Graphics, Inc. (SGI). OpenGL se propone para el empleo de hardware informático diseñado y optimizado para la visualización y manipulación de gráficos 3D. Las implementaciones genéricas de OpenGL, que sólo constan de código, también son posibles, y en esta categoría entran las implementaciones de Windows por renderización software.

Las empresas de hardware gráfico para PCs están añadiendo aceleración 3D a sus productos, influenciados principalmente por las exigencias del mercado de juegos 3D, hecho este que tiene un paralelismo cercano a la evolución de las aceleradoras gráficas 2D, basadas en Windows, que optimizan operaciones como el trazado de líneas y el relleno y manipulación de mapas de bits.

OpenGL es un lenguaje procedimental más que un lenguaje descriptivo. En lugar de diseñar la escena y cómo debe aparecer, el programador describe los pasos necesarios para conseguir una cierta apariencia o efecto. Estos "pasos" implican llamadas a una API altamente portátil que incluye aproximadamente 120 comandos y funciones. Estos se emplean para dibujar primitivas gráficas como puntos, líneas y polígonos en tres dimensiones. Además, OpenGL soporta iluminación y sombreado, mapeado con texturas, animación y otros efectos especiales.

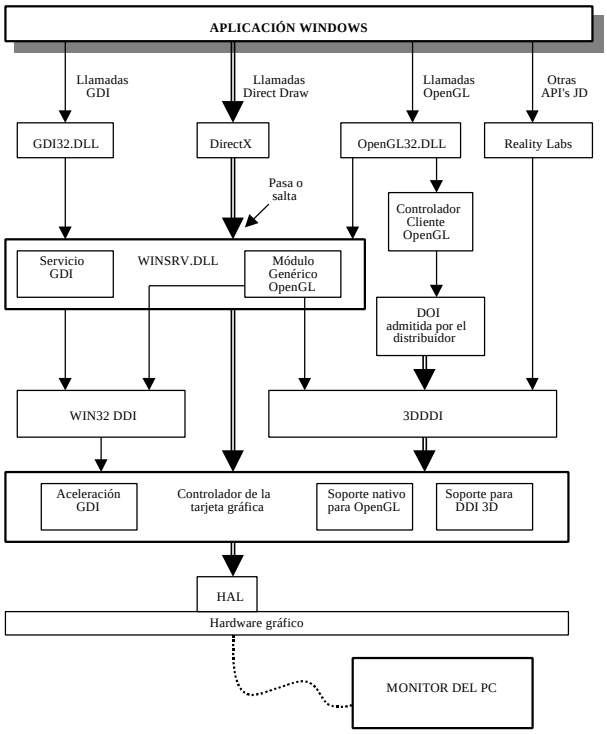

Figura 2: funcionamiento de la librería de gráficos 3D de OpenGL [9]

OpenGL no incluye ninguna función para la gestión de ventanas, interactividad con el usuario ni manejo de ficheros. Cada entorno anfitrión (como Microsoft Windows) tiene sus propias funciones para este propósito y es responsable de implementar alguna manera de facilitar a OpenGL la facultad de dibujar en una ventana o mapa de bits.

2.2 PROGRAMAS DE MODELADO EN 3D

El modelado 3D consiste en la realización de una representación visual de un objeto o conjunto de objetos mediante un ordenador o cualquier otro dispositivo que permita observar el modelo final desde cualquier ángulo. Como ejemplo se puede observar en la figura 3 el modelo gráfico de un brazo de robot PUMA 760 dentro de un entorno, en este caso un laboratorio. En la figura 4 se muestra uno de los múltiples puntos de vista del robot, coincidente con el de la fotografía.

Existen en el mercado una extensa gama de programas de modelado en 3D. Dependiendo de las posibilidades se puede elegir entre una gran variedad de aplicaciones de entre las cuales una buena cantidad son de libre disposición, algunas de ellas llegando a ofrecer amplias posibilidades como puedan ser aquellas basadas en POV Ray [7], éstas en general suelen ser simples y fáciles de manejar. También están los grandes programas de modelado y CAD usados tanto en ingeniería como para la industria cinematográfica.

Normalmente estos programas trabajan o bien por tratamiento de figuras básicas (poliedros, esferas,...) o creando sistemas de mallas. Uno de los principales problemas que se puede encontrar será el de optimizar el número de polígonos del modelo para que la tarjeta gráfica pueda refrescar la pantalla a una velocidad aceptable. Por ello, a menos que se disponga de hardware acelerador OpenGl, no se debe recurrir a los complejos sistemas de modelado de los programas ya que si bien la presentación se hace más atractiva puede que el sistema sufra una excesiva ralentización por el alto número de polígonos.

Otra ventaja que tienen estos programas es que se pueden aplicar materiales y texturas (imágenes) a las caras de los objetos, lo que decrementa el número de vértices y ofrece un alto nivel de credibilidad.

Sin duda alguna es la propiedad WYSYG (del inglés “what you see you get”, lo que ves es lo que obtienes) la característica más significativa de los programas modeladores. Es importante el poder visualizar el objeto a la vez que lo estás creando. OpenGL trae sus propios comandos para la creación de las figuras básicas así como de mallas. Cualquier cosa que finalmente se realice será interpretada por estos comandos (aunque la tediosa labor de conversión la realizarán programas dedicados a esta función), sin embargo sería difícil ceñirnos sólo a estos comandos, teniendo que compilar el programa para ver si aquello que se ha codificado corresponde a lo que se quería realizar.

2.3 CONVERSORES

Cada programa de modelado nos permite almacenar los resultados en un fichero con un extensión determinada. El formato en el que los datos son almacenados, incluso los datos en sí, puede variar de forma considerable de un software a otro. La estructura de datos normal de OpenGL suele consistir en un fichero ‘.c’ o ‘.cpp’ la cual contiene una serie de arrays que almacenan los datos de forma agrupada. Debido a estas diferencias un paso intermedio desde el modelado 3D hasta la introducción de los datos en nuestro programa será la conversión del formato de almacenamiento de datos de forma que puede ser leídos por nuestra aplicación.

Sin estos conversores el modelado a partir de programas externos a OpenGL no tendría mucho sentido. Al igual que ocurría con el software anterior, aquí también existe una gran variedad tanto por su coste como por sus funciones. Sin embargo en Internet sólo se puede encontrar un número limitado de conversores freeware que pasen a OpenGL. Si no se quiere gastar dinero en esta fase se podría acudir a conversores que pasen del formato en el que se ha modelado a uno sí permitido y después aplicar otro

conversor final. Por supuesto, existen buenos conversores comerciales.

Quizás lo más importante del conversor sea, además de que acepte el formato primitivo y el '.c' o '.cpp' para OpenGL, la forma en la que devuelva el código ya que estos varían desde unas simples matrices numéricas de vértices hasta diversas matrices (vértices, caras, normales, texturas, luces,...) y las funciones para manejarlos. Aunque son pocos los que exportan las luces, éstas suelen ser introducidas directamente por el programador con los comandos de OpenGL.

3. ROBÓTICA

Es el campo de la robótica uno en de los que más se está utilizando la simulación como herramienta fundamental de diseño. A pesar de que se pueda interpretar el modelado de robótica sólo como la definición de los problemas cinemáticos y dinámicos o según la finalidad de los cálculos los problemas directos e indirectos se podría bajar a la etapa eléctrica, muchas veces olvidadas por los programas de diseños de robots específicos.

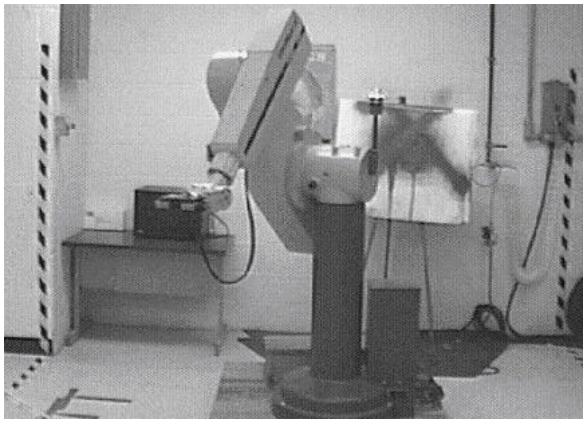

Figura 3. Imagen digital del PUMA 760 en el laboratorio CAE. [6]

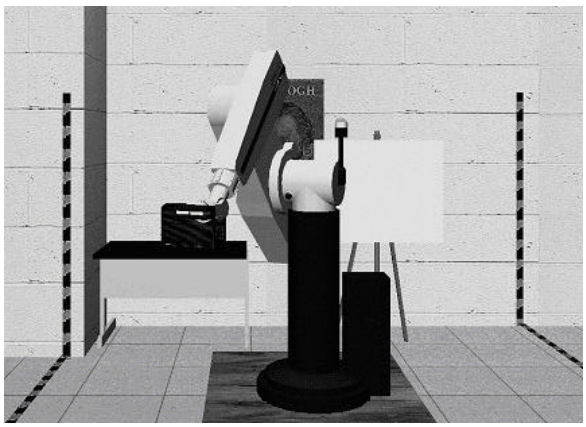

Figura 4. Modelo gráfico del PUMA 760 en el laboratorio CAE. [6]

Gracias a que EcosimPro es un lenguaje de propósito general, y que además posee la librería eléctrica y la de control implementadas se concurre en un marco idóneo para realizar una librería de robótica.

3.1 LIBRERÍA DE ROBÓTICA PARA ECOSIMPRO

Una de las líneas actuales de trabajo de los autores es la implementación de una librería de robótica bajo ECOSIM. Se pretende que ésta sea suficientemente completa y así poder realizar todas aquellas simulaciones que se podrían realizar con un simulador de carácter específico.

Además, se le desea dar a ésta la posibilidad de conectarse al resto de las librerías de EcosimPro, más concretamente a aquellas a las que está más vinculadas como son la librería eléctrica y la librería de control.

Los grupos de elementos de los que se pretende dotar a dicha librería son:

- Elementos básicos de construcción: Serán los eslabones de la cadena cinemática, la parte física.
- Fuerzas y momentos.
- Ejes 2D:
  - De revolución
  - Prismático
  - De tornillo
- Ejes 3D:
  - Cilíndrico
  - Universal
  - Planar
  - Esférico
  - Cardan

Quizás una buena definición de esta librería sea la parte más importante del proyecto, ya que de ella no sólo dependerán las posibilidades que ofrezca, sino también, la velocidad de cálculo de las variables que posteriormente permitirá la visualización 3D. Es más, no se puede olvidar que lo que se persigue no es una bella escena cinematográfica, sino un resultado lo más fidedigno posible a la realidad, ya que el objetivo del software resultante es, en un principio, de uso en la ingeniería y no en actividades artísticas (aunque no se descarta la posibilidad de utilización en dichas industrias para la construcción de juegos, mundos virtuales,...).

3.2 LAS CUATRO DIMENSIONES DE LA ROBÓTICA

Debido a que los robots son por definición, cuerpos en movimiento, se definirá la cuarta dimensión como la introducción del tiempo en los ejes cartesianos (3D). Sería importante el poder representar los cuatro ejes como conjunto.

La figura 5 podría ser la representación en los ejes X/Y/Z del manipular de un robot. Supóngase que en un instante de tiempo determinado, la posición indicada corresponde con la posición del manipulador del robot de la figura 4. Es indudable que la gráfica representa un esfuerzo de interpretación mucho más importante que el simple vistazo que hace falta para situarse en la imagen del robot.

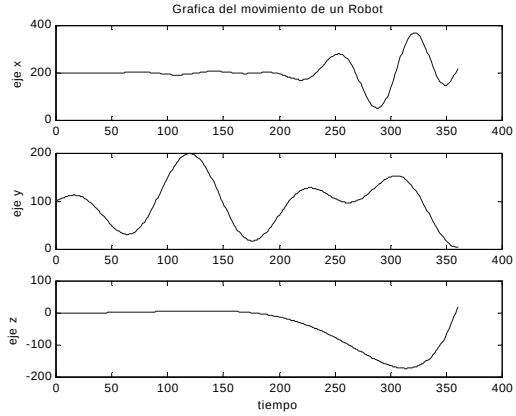

Figura 5. Gráfica del movimiento de un robot

Se añadirán dos nuevos parámetros: primero se situarán dos robots de modo que el área de trabajo sea parcialmente coincidente, y además se introducirá la cuarta dimensión. Si se le da movilidad al conjunto se comprende porqué durante tanto tiempo se ha evitado esta configuración en la industria.

Cuando el área de trabajo de una de las máquinas se ve invadida en gran medida por la otra, el diseño de los movimientos se complica en demasía, además, posiblemente alguno de los robots puede tener los suficientes grados de libertad como para posibilitar distintas configuraciones de los ejes al situar el manipulador en un punto determinado.

Es posible que el poder visualizar al robots o a los distintos robots no haga que sea más fácil el trazado de la rutas de las diferentas partes móviles, pero, en nuestra opinión reduciría enormemente el tiempo de detección de conflictos, dando una idea clara de cómo solucionarlos, facilitando la tarea del programador y haciendo esta mucho más fiable.

3.3 MATRICES COMO BASE MATEMÁTICA

Debido al carácter espacial de los sistemas robóticos, es importante realizar una agrupación de todos los parámetros existentes: localización y orientación.

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

La matriz anterior podría corresponder con la de transformación individual de un solo par en el cual la rotación se realiza sobre un único eje cartesiano según el método de Hartenberg-Denavit [1, 5, 8]. Si el sistema tuviese tres articulaciones más, la posición y orientación del manipulador vendría dada por la expresión 2, donde el superíndice indica a qué barra va referido y el subíndice sobre qué barra se realiza la operación, ${}^{0}T_{3}$ sería la trasformación de la barra 3 referida al eje 0, en este caso la base.

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

Como ya se puede intuir, la mayor ventaja de la estructura matricial es la sencillez y claridad de las expresiones resultantes. Sin esta agrupación, en el caso anterior, se tendría que recurrir a 12 fórmulas (hay que tener en cuenta que la última columna es un añadido que no aporta información, por lo que no habrá que calcularlo), siendo éstas, en muchos casos, de una longitud considerable.

4. ÁLGEBRA MATRICIAL

Dado que EcosimPro no contempla el tratamiento matricial, se presenta un fuerte contratiempo debido a la gran importancia de las ventajas que conlleva el uso de esta herramienta.

Como solución parcial ante este problema se ha diseñado un programa que viene a resolver parcialmente el problema presentado.

La aplicación es, básicamente, un analizador de texto. A partir de un fichero origen se reconocen una serie de operaciones y se procede a separar las matrices en una serie de operaciones escalares equivalentes, de forma que ECOSIM puede realizar los algoritmos oportunos (despejar, ordenar,...).

4.1 FICHEROS

El programa consta de un fichero ejecutable, el cual depende varios más:

- Fichero ser creado por el usuario. En él se deben incluir todas las variables con las que trabaja el programa.

- Fichero también creado por el usuario. En él se deben incluir las fórmulas que se quieren pasar de matricial a escalar.

- Fichero proporcionado por la aplicación. En él que se muestran los resultados escalares, así como algunos datos interesantes para la detección de errores.

4.2 OPERACIONES

Las operaciones que detecta son: paréntesis, matriz traspuesta (ya que es una operación no contemplada se le ha adjudicado el símbolo "#"), derivada "'", producto "*", producto escalar: "*" (el operador escalar es detectado automáticamente), suma: "+", suma escalar: "+" (el operador escalar es detectado automáticamente), resta: "-", resta escalar: "-" (el operador escalar es detectado automáticamente) e igualdad "=".

La relación de prioridad está establecida por el orden de aparición en este apartado, es decir:

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

También realiza una detección de números pudiendo los decimales empezar por un número o por '.'. Cualquier cadena que empiece por un número será tomada como expresión numérica, por lo que ninguna variable debe empezar por una cifra o por '.'.

4.5 EJEMPLO

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

Figura 6. Fichero "variables.txt"

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

Figura 7. Fichero "formulas.txt"

```
**********************************
   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']))
```

Figura 8. Fichero "salida.txt"

5. EJEMPLOS 3D

Para ilustrar la representación 3D de un experimento se han escogido dos ejemplos. En cada uno de ellos se recogen diferentes posibilidades de forma que se pueda completar una idea global de las posibilidades de esta herramienta. Para que la comprensión sea fácil se ha recurrido a los ejemplos que EcosimPro tiene publicado en su web.

5.1 TRES CUERPOS

Basado en los datos obtenidos en la "Demo1: modeling a three bodies problem using EcosimPro" [2]. En este problema se trata de integrar un sistema muy simple de interacción gravitatoria entre tres cuerpos. Para ello se supone que dos de ellos están fijos en el espacio separados por una cierta distancia fija, mientras que el tercero rota a su alrededor. También se supone que el movimiento tiene lugar en un plano.
El problema se visualiza en una consola OpenGL fija tal como se presenta en la figura 9. Presenta dos grandes cualidades:

- Igual que en la demostración, se han desarrollado los tres experimentos. Cada uno de estos experimentos tiene el punto de vista situado de modo que la visualización sea correcta.

- La obtención de datos se puede realizar por los dos métodos descritos anteriormente: tiempo real y por almacenamiento en array.

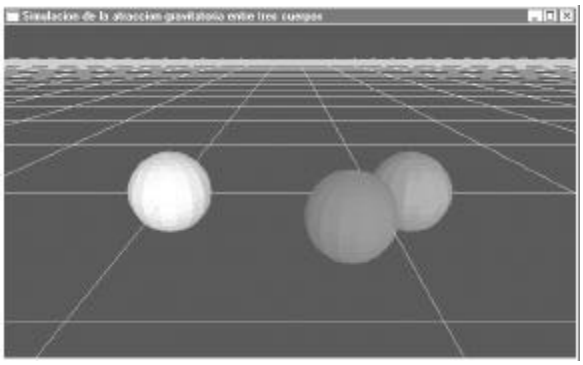

Figura 9. Imagen capturada del programa "3cuerpos"

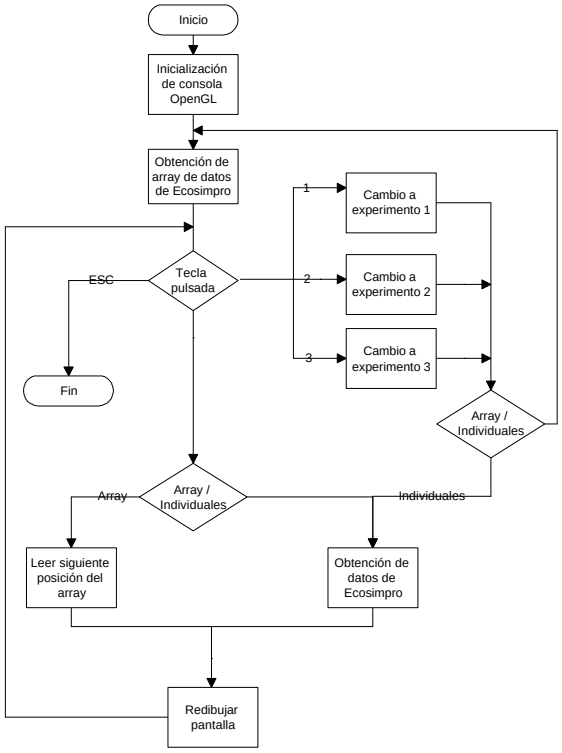

Figura 10. Diagrama de flujo del programa "3cuerpos"

5.2 ESQUIADOR

Basado en los datos obtenidos en la "Demo4: Simulating sledge slicing by a ski-track" [3]. En este problema se simula una pista de esquí por la que se deja caer un esquiador libremente con cierta velocidad inicial.

El problema, que se visualizará en una consola Windows, es similar al anterior. Simplemente en esta ocasión se utilizan los eventos de Windows (teclado, timer,...). Presenta dos grandes cualidades:

- Los cuerpos a renderizar dejan de ser figuras geométricas simples (el anterior era una esfera) para ser una figura modelada mediante un programa exterior y una malla definida mediante una ecuación.

- A través del teclado se puede situar el punto de vista en cualquier parte: rotación en todos los ejes así como alejarse y acercarse. También se puede entrar en un modo de pausa.

El diagrama vendría a ser muy similar al de la figura 10. En este caso sólo se soporta el almacenamiento de datos en array, por lo que se eliminaría la rama de la derecha.

Antes de redibujar la pantalla se realizaría el cambio de posición del observador, ya sea porque el esquiador avanza o porque el usuario ha pulsado alguna tecla. También se calcula el área de "nieve" visible ya que si se dibujase la superficie entera el número de polígonos aumentaría sin aportar más información, por lo tanto, de forma innecesaria.

Figura 11. Imagen capturada del programa "esquiador"

Este ejemplo nos muestra lo fácil que sería realizar un juego con nuestra simulación. Simplemente se tendría que prever una fuerza exterior (jugador) en el modelo de ECOSIM, situar unas banderitas en la pista e ir aumentando un contador cuando nuestro esquiador pasase sobre ellas. Para finalizar una meta y una copa para nuestro campeón.

Referencias

[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) Manuales de EcosimPro v3.1. http://www.ecosimpro.com/ download/manuals.htm

[5] Fu, González, y 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 JR. y Sweet, (1997) Programación en OpenGL. Editorial ANAYA.

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