Representaciones gráficas y Mundos Virtuales infinitos en las Prácticas de *Programación Lógica y Funcional*

Jose Emilio Labra Gayo
Dpto. de Informática
Universidad de Oviedo
CP. 33007 Oviedo, Spain
e-mail: labra@lsi.uniovi.es

Resumen

En el esquema de prácticas de la asignatura de *programación lógica y funcional* se ha incorporado la generación de representaciones gráficas y de *quadtrees* y *octrees* para la enseñanza de las diferentes características de este tipo de lenguajes: funciones de orden superior, evaluación perezosa, polimorfismo, variables lógicas, satisfacción de restricciones, etc. La utilización de vocabularios XML estándar: SVG para gráficos y X3D para realidad virtual permite disponer de numerosas herramientas de visualización. La incorporación de este tipo de ejercicios puede facilitar la enseñanza y motivación de los estudiantes.

1. Introducción

La asignatura *Programación Lógica y Funcional* se imparte como optativa de 6 créditos en las titulaciones de Ingeniero Técnico de Informática de Gestión y de Sistemas en la Universidad de Oviedo. El resto de asignaturas de programación de dicha titulación se centra en lenguajes imperativos: Pascal, C, C++ y Java. Esta asignatura es, por tanto, el primer y en muchas ocasiones, único contacto que los estudiantes de estas titulaciones tienen con los lenguajes declarativos. Dado su carácter optativo, la supervivencia de la asignatura depende del número de alumnos matriculados. Aparte de la dificultad de una asignatura de programación, la elección de los estudiantes se ve condicionada por otros aspectos relacionados con este tipo de lenguajes: entornos de programación rudimentarios, escasez de sistemas gráficos de depuración y traza, carencia de librerías, dificultad para enlazar con librerías escritas en otros lenguajes, etc. Con el fin de ampliar la motivación de los estudiantes y el alcance de la asignatura, desde el curso 2001/02 se ha incorporado al proyecto IDEFIX [14,15] que persigue la enseñanza a través de Internet de lenguajes de programación.

La evaluación de la asignatura se realiza fundamentalmente mediante la realización de trabajos de programación. Los enunciados de estos trabajos siguen un esquema de presentación gradual basado en la taxonomía de objetivos cognitivos [12]: en primer lugar se presenta un programa correcto que los estudiantes deben compilar y ejecutar. Seguidamente, se les pide la realización de modificaciones para que demuestren que comprenden el programa y adquieran por imitación una habilidad básica de construcción de programas. Finalmente, se les solicita a los estudiantes la creación de programas que deben construir por su cuenta.

En este artículo se describe el esquema de prácticas utilizado y que se basa principalmente en la generación de ficheros XML que incluyen gráficos bidimensionales en SVG y mundos virtuales en X3D. El artículo comienza con un repaso a los vocabularios XML utilizados. A continuación se presenta el ejercicio básico que consiste en representaciones gráficas de funciones y permite manipular el concepto de funciones de orden superior. En la sección 4 se estudia la estructura recursiva de *quadtree*. En la sección 5 se presentan los *octrees* y la sección 6 se presenta la generación de mundos virtuales infinitos mediante evaluación perezosa. En la sección 7 se describe la creación de tipos de datos polimórficos. La sección 8 describe la utilización de características propias de la programación lógica (especialmente la utilización de variables lógicas y *no-determinismo*) y la siguiente sección se centra en la utilización de programación lógica con restricciones. Finalmente se resumen los principales trabajos relacionados y se detallan las principales conclusiones y líneas de investigación futuras.

Notación. A lo largo del artículo se utilizan fragmentos de código *Haskell* y *Prolog*. Se supone que el lector tiene ligeros conocimientos de la sintaxis de ambos lenguajes. También se supone cierto conocimiento de vocabularios XML.

2. Vocabularios XML

El lenguaje XML [3] se está convirtiendo en un estándar de intercambio y representación de información en Internet. Puesto que en el actual plan de estudios no se imparte dicho lenguaje en ninguna asignatura, se ha considerado que la utilización de vocabularios XML puede ser una buena oportunidad para que los estudiantes manejen dicho estándar.

Uno de los primeros ejercicios planteados es la construcción de una librería de funciones que permita generar ficheros XML. Esta librería, que los estudiantes construyen, permitirá utilizar una base común en el resto de prácticas que facilita la generación de ficheros con vocabularios XML específicos.

La librería contiene las siguientes funciones básicas:

- *vacio e as* genera un elemento vacío e con atributos as
- *gen e es* genera un elemento *e* con subelementos *es*
- *genAs e as es* genera un elemento e con atributos *as* y subelementos *es*

Uno de los vocabularios XML específicos que se utilizará es el lenguaje SVG (Scalable Vector Graphics) que se está convirtiendo en el principal estándar de representación de gráficos bidimensionales vectoriales en Internet.

Asimismo, el lenguaje VRML (*Virtual Reality Modelling Language*) se puede considerar el principal estándar de representación de mundos virtuales en Internet. Sin embargo, este lenguaje es anterior a XML y no sigue, por tanto, su sintaxis. Desde el año 2000, el consorcio Web3D ha diseñado el vocabulario X3D que puede considerarse una nueva versión de VRML adaptada a XML.

3. Funciones de orden superior: Representaciones gráficas

El segundo trabajo práctico que se plantea consiste en la representación gráfica de funciones. A los estudiantes se les presenta la función plotF que permite almacenar en un fichero SVG la representación gráfica de una función.

```
plotF ::(Double→Double)→String→IO ()
plotF f fn = writeFile fn (plot f)

plot::(Double→Double)→String
plot f = gen “svg” (plotPs ps)
 where
  plotPs       = concat . map mkline
  mkline (x,x')= line (p x) (p x') c
  ps           = zip ls (tail ls)
  ls           = [0..sizeX]
  p x          = (x0+x, sizeY - f x)

line (x,y) (x',y') c =
  vacio "line"
    [(“x1”,show x) ,(“y1”,show y),
     (“x2”,show x'),(“y2”,show y')]

sizeX = 500; sizeY = 500; x0 = 10
```

Obsérvese que plotF realiza acciones de Entrada/Salida. Tradicionalmente, los cursos que enseñan el lenguaje *Haskell* tendían a retrasar la presentación del sistema de Entrada/Salida mediante mónadas. Creemos que una presentación gradual permite evitar malentendidos posteriores, ya que en caso contrario, muchos estudiantes consideraban extraña la posterior introducción de efectos laterales en un lenguaje que consideraban *puro*.

El código de la función plot sirve como ejemplo de utilización de los combinadores recursivos zip, map, foldr, etc. característicos del lenguaje *Haskell*.

En este trabajo práctico, los ejercicios que se proponen a los estudiantes utilizan el concepto de funciones de orden superior, clave del paradigma funcional. Así, por ejemplo, se solicita al estudiante que construya una función plotMedia que escriba en un fichero la representación de dos funciones y la función media de ambas. Por ejemplo, en la figura 1 se representa la media de las funciones (\x -> 10 * sin x) y (\x -> 10 + sqrt x).

La solución del ejercicio en *Haskell* puede ser:

```
media f g = plotF (\x→(f x + g x)/2)
```

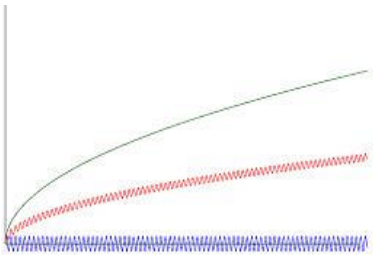

Figura 1. Representación de dos funciones y su media

4. Tipos de datos recursivos: *Quadtrees*

La siguiente unidad didáctica se centra en la presentación de técnicas de definición de tipos recursivos y su manipulación. Tradicionalmente, los trabajos prácticos en esta sección se realizaban con el tipo predefinido lista y con un tipo de datos definido por los alumnos para representar árboles binarios. Los estudiantes tienen serias dificultades para comprender los procesos recursivos y se sienten habitualmente poco motivados por este tipo de ejercicios [9,18]. Para intentar aumentar su motivación se trabajará con *quadtrees* [17] que son estructuras recursivas que permiten representar imágenes y tienen numerosas aplicaciones prácticas. En un *quadtree* las figuras se representan mediante un único color o la subdivisión de cuatro cuadrantes que son a su vez *quadtrees*. La generalización de los *quadtrees* al espacio tridimensional se denomina *octree* ya que cada subdivisión se realiza en ocho *octrees*. Estas estructuras son ampliamente utilizadas en el campo de la informática gráfica ya que permiten optimizar la representación interna de escenas y las bases de datos tridimensionales para sistemas de información geográfica.

En Haskell, un quadtree puede representarse mediante el siguiente tipo de datos:

```
data Color = RGB Int Int Int
data QT = B Color
        | D QT QT QT QT
```

Un ejemplo de quadtree sería

```
ejQT = D r g g (D r g g r)
  where r = B (RGB 255 0 0)
        g = B (RGB 0 255 0)
```

El ejemplo anterior define un *quadtree* formado por la subdivisión en cuatro cuadrantes, dos rojos y dos verdes dispuestos en diagonal. En la figura 2 puede observarse una representación del quadtree `ejQT`.

La visualización de *quadtrees* se realiza mediante una sencilla función que genera un fichero en formato SVG utilizando las funciones *vacio*, *gen*, y *genAs* implementadas por los estudiantes para la generación de documentos XML.

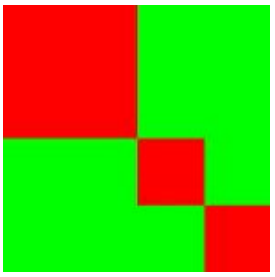

Figura 2. Ejemplo de *Quadtree*

```
type Punto = (Int,Int)
type Dim   = (Punto,Int)

verqt::QT→IO ()
verqt q = writeFile “qtree.svg”
      (gen “svg” (ver ((0,0),500) q))

ver::Dim→QT→String
ver ((x,y),d) (B c) =
 rect (x,y) (x+d+1,y+d+1) c

ver ((x,y),d) (D ul ur dl dr) =
  let d2 = d `div` 2
  in
   if d <= 0 then “”
   else ver ((x,y),d2)       ul ++
        ver ((x+d2,y),d2)    ur ++
        ver ((x,y+d2),d2)    dl ++
        ver ((x+d2,y+d2),d2) dr

rect::Punto→Punto→Color→String
rect (x,y) (x',y') (RGB r g b) =
 vacio "rect"
     [(“x”,show x),(“y”,show y),
      (“height”,show (abs (x - x'))),
      (“width”, show (abs (y - y'))),
      (“fill”, “rgb(”++show r++ “,”++
                       show g ++“,”++
                       show b ++“,”)]
```

5. *Octrees* y mundos virtuales

Un *octrree* es una generalización de un *quadtree* para representaciones tridimensionales: al subdividir cada cara de un cubo en cuatro partes se obtienen ocho cubos. La representación de *octrees* en *Haskell* podría ser la siguiente:

```
data OT = Vacio
        | Cubo Color
        | Esfera Color
        | D OT OT OT OT OT OT OT OT
```

La representación anterior indica que un *octree* puede estar vacío, ser un cubo o una esfera con un determinado color, o una división en ocho *octrees*.

Un ejemplo de *octree* sería:

```
ejOT :: OT
ejOT = D v e e v r v v g
 where v = Vacio
       e = Esfera (RGB 0 0 255)
       r = Cubo   (RGB 255 0 0)
       g = Cubo   (RGB 0 255 0)
```

A los estudiantes se les presenta la función

```
wOT :: OT → FileName → IO ()
```

que toma como argumentos un *octree*, un nombre de fichero y escribe en dicho fichero una representación en X3D del *octree*. En la figura 3 se presenta una pantalla capturada de la representación del oc*tree* ejOT en realidad virtual. Aunque en la figura se representa una versión impresa, el sistema genera un modelo virtual en el que los estudiantes pueden navegar.

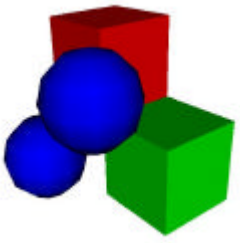

Figura 3. Ejemplo de *Octree*

6. Evaluación *Just-in time:* Mundos infinitos

Una de las principales características del lenguaje Haskell es la evaluación perezosa que permite definir algoritmos que manipulan estructuras potencialmente infinitas. La evaluación perezosa puede también considerarse un tipo de evaluación *Just-in time* en la que el sistema no evalúa los argumentos de una función hasta que realmente necesita su valor. El programador puede definir y manipular *quadtrees* infinitos. Por ejemplo, es posible definir:

```
inf :: OT
inf = D inf v s v v v r inf
 where v = Vacio
       s = Esfera (RGB 0 0 255)
       r = Cubo   (RGB 255 0 0)
```

Obsérvese que el *octree* se define en función de sí mismo. Su visualización se presenta en la figura 4.

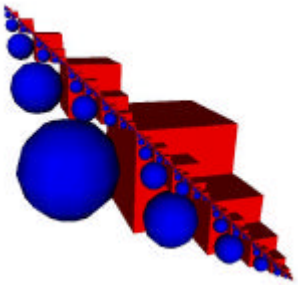

Figura 4. *Octree* infinito

Gracias a la evaluación perezosa, es posible definir funciones que manipulen mundos virtuales infinitos. Por ejemplo, la función `repite` toma como argumento un *octree* y genera un nuevo *octree* x repitiendo en cada cuadrante el *octree* x.

```
repite :: OT → OT
repite x = D x x x x x x x
```

Al aplicar la función repite al *octree* inf se obtendría el resultado de la figura 5.

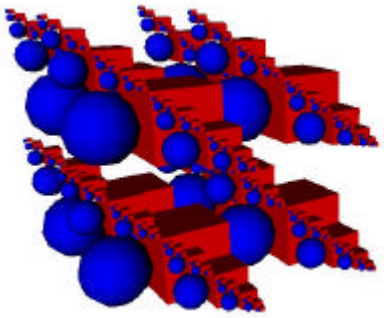

Figura 5. Repetición de un *Octree* infinito

7. Polimorfismo paramétrico: *Quadtrees* de alturas

El sistema de tipos del lenguaje *Haskell* admite la utilización de polimorfismo paramétrico. Las listas son el ejemplo tradicional de tipos de datos polimórfico. Aunque los *quadtrees* tradicionales contienen en cada cuadrante información del color, podría estudiarse una generalización que contuviese en cada cuadrante información de un tipo *a* que se pasa como parámetro. La nueva definición sería:

```
data QT a = B a
          | D (QT a) (QT a)
              (QT a) (QT a)
```

Los *quadtrees* con información de color serían valores de tipo `QT Color`.

La generalización anterior permite definir otros ejemplos de *quadtrees*, como los que contienen en cada cuadrante información de la altura (un valor de tipo `Float`). Estos *quadtrees* pueden utilizarse para la representación de alturas de terrenos. Por ejemplo el siguiente *quadtree*:

```
qta :: QT Float
qta = D x x x x
  where x = D b c a b
        a = B 0
        b = B 10
        c = B 20
```

se representa en la figura 6.

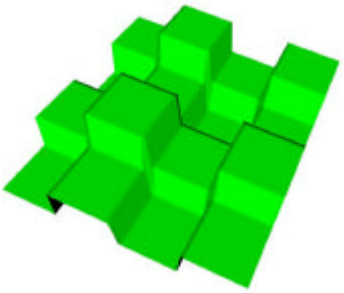

Figura 6. *Quadtree* de alturas

El lenguaje *Haskell* facilita y promueve la utilización de funciones genéricas. La función *foldr* es ejemplo de función genérica predefinida para el caso de las listas. Esta función puede también definirse para el tipo de datos *quadtree:*

```
foldQT::
  (b→b→b→b→b)→(a→b)→QT a→ b
foldQT f g (B x) = g x
foldQT f g (D a b c d) =
 f (foldQT f g a) (foldQT f b)
   (foldQT f g c) (foldQT f d)
```

Es posible definir múltiples funciones a partir de *foldQT*. Por ejemplo, para calcular la lista de valores de un *quadtree* puede definirse:

```
valores :: QT a → [a]
valores = foldQT
    (\a b c d → a ++ b ++ c ++ d)
    (\x → [x])
```

La profundidad de un *quadtree* puede definirse como:

```
profundidad :: QT a → Int
profundidad = foldQT
   (\a b c d → 1 + maximum [a,b,c,d])
   (\_ → 1)
```

La función *foldQT* pertenece al conjunto de funciones que recorren y transforman una estructura recursiva en un valor. Estas funciones se denominan también *catamorfismos* y son estudiadas en el campo de la programación genérica [2].

8. No determinismo: Generación de *quadtrees* en programación lógica

Una de las dificultades de la asignatura es la introducción de los paradigmas funcional y lógico en un breve espacio de tiempo. En la actualidad se emplean, además, dos lenguajes diferentes: *Haskell* y *Prolog*. Para facilitar el cambio entre paradigmas y lenguajes, se ha optado por solicitar prácticas de programación muy similares de forma que los alumnos puedan observar más claramente las diferencias entre ambos.

De esta forma, los primeros trabajos prácticos en la parte de programación lógica vuelven a examinar sobre el tema de los *quadtrees.*

Una característica sobresaliente de la programación lógica es la posibilidad de definir predicados no-deterministas que admiten varias soluciones obtenidas por *backtracking.*

Es posible definir el predicado `col(Xs,Q1,Q2)` que se cumple cuando `Q2` es el *quadtree* formado por rellenar el *quadtree* `Q1` con los colores de la lista `Xs`.

```
col(Xs,b(_),b(X)):-elem(X,Xs).
col(Xs,d(A,B,C,D),d(E,F,G,H)):-
   col(Xs,A,E),col(Xs,B,F),
   col(Xs,C,G),col(Xs,D,H).
```

donde `elem(X,Xs)` es un predicado predefinido que se cumple si X es un elemento de la lista Xs. Obsérvese que al rellenar un quadtree con dos colores se obtienen varias respuestas.

```
?-col([0,1],d(b(_),b(_),b(_),b(_)),V).
V = d(b(0),b(0),b(0),b(0)) ;
V = d(b(0),b(0),b(0),b(1)) ;
V = d(b(0),b(0),b(1),b(0)) ;
V = d(b(0),b(0),b(1),b(1)) ;
. . .
```

Un problema clásico en el campo algorítmico es el de colorear un mapa de regiones con una serie de colores de forma que ninguna región adyacente tenga el mismo color. El problema puede plantearse para colorear *quadtrees.* En la figura 7 se presenta una posible solución al colorear un *quadtree* que representa un rombo.

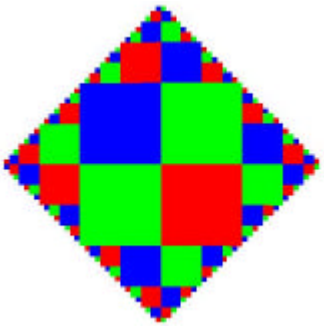

Figura 7. Solución del problema de coloreado

Una solución directa utilizando programación lógica consiste simplemente en generar todos los posibles quadtrees y comprobar la condición de no adyacencia mediante el predicado `noColor`:

```
noColor(b(_)).
noColor(d(A,B,C,D)):-
  noColor(A), noColor(B),
  noColor(C), noColor(D),
  right(A,Ar),left(B,Bl),diff(Ar,Bl),
  right(C,Cr),left(D,Dl),diff(Cr,Dl),
  down(A,Ad),up(C,Cu),diff(Ad,Cu),
  down(B,Bd),up(D,Du),diff(Bd,Du).

up(b(X),l(X)).
up(d(A,B,_,_),f(X,Y)):-up(A,X),
                       up(B,Y).

down(b(X),l(X)).
down(d(_,_,C,D),f(X,Y)):-down(C,X),
                         down(D,Y).

left(b(X),l(X)).
left(d(A,_,C,_),f(X,Y)):-left(A,X),
                         left(C,Y).

right(b(X),l(X)).
right(d(_,B,_,D),f(X,Y)):-right(B,X),
                          right(D,Y).

diff(l(X),l(Y)):-X\=Y.
diff(l(X),f(A,B)):notElem(X,A),
                  notElem(X,B).
diff(f(A,B),l(X)):-notElem(X,A),
                   notElem(X,B).
diff(f(A,B),f(C,D)):-diff(A,C),
                     diff(B,D).

notElem(X,l(Y)):-X\=Y.
notElem(X,f(A,B)):-notElem(X,A),
                   notElem(X,B).
```

9. Programación con restricciones

Los lenguajes declarativos ofrecen un marco ideal para la programación basada en restricciones (*constraint programming*). Aunque una presentación en profundidad se saldría del ámbito de la asignatura, hemos considerado interesante presentar a los estudiantes una breve introducción a esta disciplina ya que para muchos de ellos, será el único acercamiento en toda su carrera universitaria. Continuando con el tema de los *quadtrees*, se presenta una solución del problema de colorear un *quadtree* utilizando las extensiones de programación lógica con dominios finitos soportadas por algunos sistemas Prolog. En concreto, a continuación se presenta un predicado `colC` que rellena un *quadtree* con los valores tomados de un dominio [M,N]. Se utiliza la sintaxis de CLP(FD) implementado en GNU Prolog [5].

```
colC(M,N,b(_),b(X)):-fd_domain(X,M,N).
colC(M,N,d(A,B,C,D),d(E,F,G,H)):-
   col(M,N,A,E),col(M,N,B,F),
   col(M,N,C,G),col(M,N,D,H).
```

La nueva versión de noColor utilizando restricciones sería:

```
noColorC(b(_)).
noColorC(d(A,B,C,D)):-
  noColorC(A), noColorC(B),
  noColorC(C), noColorC(D),
  right(A,Ar),left(B,Bl),diffC(Ar,Bl),
  right(C,Cr),left(D,Dl),diffC(Cr,Dl),
  down(A,Ad),up(C,Cu),diffC(Ad,Cu),
  down(B,Bd),up(D,Du),diffC(Bd,Du).

diffC(l(X),l(Y)):-X #\= Y.
diffC(l(X),f(A,B)):-notElemC(X,A),
                    notElemC(X,B).
diffC(f(A,B),l(X)):-notElemC(X,A),
                    notElemC(X,B).
diffC(f(A,B),f(C,D)):-diffC(A,C),
                      diffC(B,D).

notElemC(X,l(Y)):-X #\= Y.
notElemC(X,f(A,B)):-notElemC(X,A),
                    notElemC(X,B).
```

Obsérvese que la implementación es prácticamente igual a la presentada en la sección anterior ya que solamente se ha sustituido el predicado '\=' por el predicado '#\='. Sin embargo, el algoritmo basado en restricciones obtiene una solución para N=3 en 0.01sg mientras que el algoritmo de la sección anterior tardaba 74.5sg.

10. Trabajo relacionado

La exigua utilización de los lenguajes declarativos [20] ha llevado a varios miembros de esta comunidad a la búsqueda de aplicaciones atractivas que resalten las capacidades de este tipo de lenguajes. Cabe destacar el libro de texto escrito por P. Hudak [10] en el que se presenta una introducción a la programación funcional incluyendo ejemplo relacionados con sistemas multimedia. En el libro se utilizan varias librerías específicas que permiten generar y visualizar los diversos ejercicios propuestos. La estrategia seguida en este artículo es similar en el objetivo, pero difiere en la forma de visualizar las construcciones gráficas. Se ha optado por utilizar vocabularios XML que se están convirtiendo en estándares en sus respectivos campos. Esta técnica supone varias ventajas: existencia de mayor número de herramientas de visualización, portabilidad entre plataformas e independencia de librerías específicas. Además, los estudiantes emplean tecnologías XML estándar que pueden ser beneficiosas en otros campos de su trayectoria profesional.

Las representaciones declarativas de *quadtrees* fueron estudiadas en [4,8,19,21]. Recientemente, C. Okasaki [16] toma como punto de partida la representación en Haskell de un *quadtree* para definir una implementación eficiente de matrices cuadradas mediante tipos anidados. Su implementación mantiene la consistencia de la representación gracias al sistema de tipos de Haskell.

En el campo imperativo existen varios trabajos [7,11,13] que resaltan la utilización de *quadtrees* como buenos ejemplos de prácticas de programación, centrándose fundamentalmente en su aplicación para comprimir imágenes. En [6] se describen diversos algoritmos de coloreado de *quadtrees* y su aplicación práctica en la planificación de computaciones paralelas.

11. Conclusión y Líneas de trabajo

El presente artículo propone un esquema de trabajos prácticos que pretende potenciar la visualización gráfica de los resultados a la vez que se exploran las diversas características de los lenguajes declarativos. Aunque no se ha realizado un estudio sistemático de la reacción de los estudiantes ante este esquema,

puede afirmarse que las primeras impresiones son altamente positivas, con un porcentaje de abandono de la asignatura notablemente inferior al de cursos anteriores. Sin embargo, este tipo de afirmaciones debería contrastarse de una forma rigurosa comprobando, por ejemplo, que los estudiantes expuestos a este tipo de enseñanza realmente resuelven mejor otros problemas de programación.

La generación de mundos virtuales ha supuesto un aliciente en la investigación del grupo IDEFIX [14,15]. Entre las futuras líneas de investigación se encuentra el estudio de comunidades virtuales tipo *ActiveWorlds* [1] donde los estudiantes puedan visitar la comunidad, crear sus propios mundos y conversar con otros estudiantes mediante *chat.*

Referencias

[1] ActiveWorlds, página Web: `http://www.activeworlds.com`

[2] R. Backhouse, P. Jansson, J. Jeuring, L. Meertens, *Generic Programming – An Introduction,* Advanced Functional Programming, Lecture Notes in Computer Science, vol 1608, S. Swierstra, P. Henriques, J. N. Oliveira (Eds), 1999

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, Extensible markup language (1.0), `http://www.w3.org/TR/REC-xml`, Oct 2000

[4] F. W. Burton, J. G. Kollias. *Functional programming with quadtrees.* IEEE Software, 6(1):90-97, Enero, 1989

[5] P. Codognet and D. Diaz. Compiling Constraints in CLP(FD). *Journal of Logic Programming*, Vol. 27, No. 3, June 1996

[6] D. Eppstein, M. W. Bern, B. Hutchings, *Algorithms for coloring quadtrees,* Algorithmica, 32(1), Ene. *2002*

[7] J. B. Fenwick Jr., C. Norris, J. Wilkes, *Scientific Experimentation via the Matching Game*,SIGCSE Bulletin 34(1), 33th SIGCSE Technical Symposium on Computer Science Education, Marzo, 2002

[8] J. D. Frens, D. S. Wise, Matrix inversion using quadtrees implemented in gofer. Technical Report 433, Computer Science Department, Indiana University, Mayo 1995

[9] J. Good, P. Brna, *Novice Difficulties with recursion: Do graphical Representations Hold the solution?*, European Conference on Artificial Intelligence in Education, Lisboa, Portugal, Oct.,1996

[10] P. Hudak, *The Haskell School of Expression: Learning Functional Programming through Multimedia,* Cambridge Univ. Press, 2000

[11] R. Jiménez-Peris, S. Khuri, M. Patiño-Martínez, *Adding breadth to CS1 and CS2 courses through visual and interactive programming projects,* ACM SIGCSE Bulletin 31(1), Marzo, 1999

[12] J. Kaasbøll, *Exploring didactic models for programming,* Norsk Informatikk-Konferanse, Høgskolen I Agder, 1998

[13] S. Khuri, H. Hsu, *Interactive Packages for learning image compression algorithms*, ACM SIGCSE Bulletin 32(3), Sept. 2000

[14] J.E. Labra, J.M. Morales, R. Turrado, *Plataforma de enseñanza de lenguajes de programación a través de Internet: Proyecto Idefix,* Jornadas de Enseñanza Universitaria de Informática, JENUI-2002, Cáceres, Jun. 2002

[15] J.E. Labra, J. M. Morales, A. M. Fernández, H. Sagastegui, *A Generic e-Learning Multiparadigm Programming Language System: IDEFIX Project*, ACM 34th SIGCSE Technical Symposium on Computer Science Education, Reno, Nevada, USA, Febrero 2003

[16] Chris Okasaki, *From Fast exponentiation to Square Matrices: An adventure in Types*, ACM SIGPLAN Notices 34(9), Intl. Conference on Functional Programming, pp. 28-35, 1999

[17] Hanan Samet *The quadtree and related hierarchical data structures*. ACM Computing Surveys, 16(2): 187-260, Junio 1984.

[18] J.Segal, *Empirical studies of functional programming learners evaluating recursive functions,* Instructional Science 22, 385-411, 1995

[19] S. Edelman and E. Shapiro, *Quadtrees in concurrent prolog*, Proc. Intl. Conference on Parallel Processing, 1985, pp. 544-551

[20] P. Wadler, *Why no one uses functional programming languages?,* ACM SIGPLAN Notices 33(8): 23-27, Agosto, 1998

[21] D. Wise, *Matrix algorithms using quadtrees.* Technical Report 357, Computer Science Department, Indiana University, Jun., 1992