Tutorial de la biblioteca NURBS++.

Daniel A. Cervantes Cabrera.

1. Introducción.

La biblioteca **NURBS++**, es un conjunto de rutinas con un diseño orientado a objetos implementadas con el lenguaje de programación C++ por el Dr.Philippe Lavoie, la cual proporciona clases con las se pueden representar curvas y superficies NURBS y tiene implementadas la mayoría de los algoritmos descritos en libro "The NURBS Book"([PT97]). Además proporciona módulos para la manipulación de matrices, métodos numéricos (como aproximación mínima cuadrada, SVD y funciones estadísticas), y permite el manejo de distintos formatos de imágenes con los que se puede realizar impresiones de curvas y superficies usando **OpenGL** , **VRML** y **Post-Script**.

NURBS++ se puede obtener libremente bajo los términos de la licencia **GNU** en la página de "Internet"(ver figura 1) `http://libnurbs.sourceforge.net/index.shtml`.

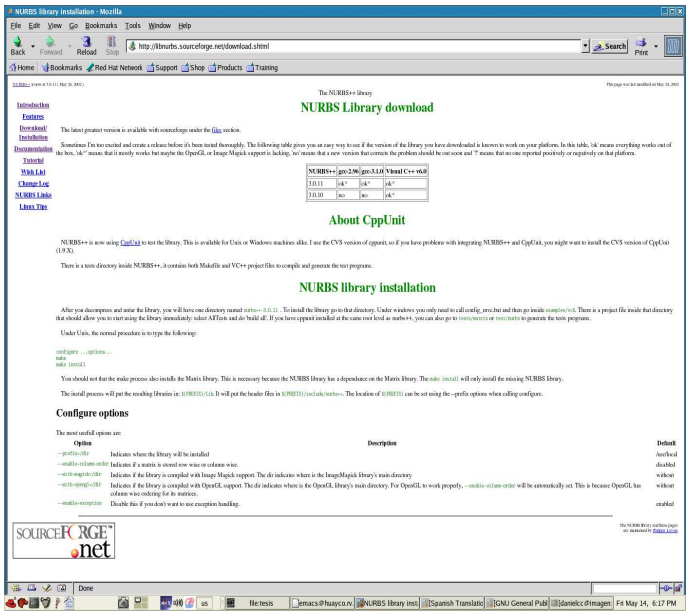

Figura 1: Página electrónica de la biblioteca NURBS++

En ésta, además se puede consultar las instrucciones para instalación de la biblioteca tanto en los sistemas operativos basados en "Unix"(como linux) o"Windows"de Microsoft. Un manual de usuario y de referencia de todas la clases que componente a la biblioteca, e información adicional a este proyecto.

2. Instalación.

Existen varias manera de realizar la instalación dependiendo del "software"preinstalado y el sistema operativo con el que se trabaje. Sin embargo, independientemente de éste, la biblioteca **NURBS++** requiere instaladas las bibliotecas gráficas de **OpenGL** y **GLUT** [1] y un compilador del lenguaje C++[2]

A continuación presentaremos las instrucciones básicas para la instalación de la versión 3.11, suponiendo que se está trabajando bajo sistema operativo linux.

Primeramente se debe bajar el paquete, en la página de **NURBS++** presionando en la liga "files", la cual nos mandará a la base de datos de "sourceforge"[3] para esta biblioteca, en esta página es posible seleccionar el paquete **NURBS++** comprimido en varios formatos, para esta explicación nosotros seleccionaremos el formato `tar.gz`, es decir `nurbs++-3.0.11.tar.gz`. Una vez hecho, se abrirá otra página en donde se puede seleccionar el dominio desde donde se copiará el archivo, en este caso es recomendable seleccionar el más cercano a nuestro lugar de trabajo, por ejemplo nosotros seleccionaremos cualquiera de norte América.

Cuando el paquete esté copiado, debemos proceder a descomprimirlo e instalarlo, en este caso supondremos que esto se realizará en la carpeta `/home/usuario/`, en donde `usuario`, es el "login"de nuestra respectiva cuenta en linux. Entonces tecleemos los siguientes comandos

```
[usuario]$ tar -zxf nurbs++-3.0.11.tar.gz
[usuario]$ cd nurbs++-3.0.11
[usuario]$ ./configure
[usuario]$ make
[usuario]$ su
[root]# make install
```

La primera instrucción descomprimirá la biblioteca, con la segunda entraremos a la carpeta en donde se encuentra **NURBS++** y la siguiente ejecutará al programa `configure` que sirve para configurar el "software"en la plataforma que se va instalar, a este programa es posible pasarle varios parámetros para realizar una instalación personalizada, por ejemplo si escribimos

```
[usuario]$ ./configure --prefix=/home/usuario/nurbs++-3.0.11/lib
```

las bibliotecas ya compiladas se instalarán en la ruta `/home/usuario/nurbs++-3.0.11/lib` en vez de `/usr/local/lib` que es por "default", esto es útil cuando por ejemplo no se conoce el "password"de "root"o se desea realizar una instalación de las biblioteca sólo en nuestra cuenta de Linux. Otros posibles parámetros al programa `configure` se presentan en la página.

La instrucción `make`, compilará todas los bibliotecas de **NURBS++**, su ejecución tardará unos cuantos minutos dependiendo del procesador de nuestra máquina.

La instrucción `su`, permite en un sistema Linux cambiar a modo superusuario o administrador, con esta podremos ejecutar la siguiente `make install`, con la que se copiarán las bibliotecas recién compiladas a la carpeta`/usr/local/lib`, estás dos instrucciones sólo se podrán realizar si se conoce el "password"de "root". En caso de que no sea así, es recomendable como se mencionó anteriormente, realizar la instalación local pasándole al programa `configure` el parámetro `--prefix`, en cuyo caso estas dos últimas instrucciones no se deberán ejecutarse.

[1] Es posible encontrar más información en http://www.opengl.org/.

[2] En la página se indica cuales son la versiones adecuadas.

[3] Para más acerca de este proyecto, ver en

Una manera fácil de comprobar si nuestra instalación funciona, es tratar de correr los ejemplos incluidos en la biblioteca. Por ejemplo probemos los de curvas y superficies NURBS escribiendo

```
[usuario]$ cd /home/usuario/nurbs++-3.0.11/examples/nurbs
[usuario]$ make
```

Así una vez que todos estén compilados ejecutemos por ejemplo

```
[usuario]$  ./tnsSweep
```

este programa creará varias superficies de barrido en el formato **VRML**, para poder verlas es necesario tener un visualizador de esta clase de archivos, uno fácil de instalar es **Rational Reducer**, el cual tiene una versión de evaluación gratuita en la dirección `http://www.sim.no/products/evaluate/`

En algunas versiones de linux, es posible que al momento de hacer la compilación de los ejemplos no se encuentre ciertas bibliotecas de **OpenGL** y GLUT o sus versiones no sean compatibles con las requeridas por **NURBS++**. Una manera fácil de resolver esto, es instalar la última versión de **OpenGL** o su versión libre **Mesa**. A continuación explicaremos la instalación de la versión 6.2 de **Mesa** suponiendo que se realizará en la carpeta `/home/usuario`.

En la página de Mesa `http://www.mesa3d.org/`, bajar los archivos `MesaLib-6.2.1.tar.gz` y `MesaDemos-6.2.1.tar.gz` y descomprimirlos con las instrucciones

```
[usuario]$ tar -zxf MesaLib-6.2.1.tar.gz
[usuario]$ tar -zxf MesaDemos-6.2.1.tar.gz
```

una vez hecho esto se creará la carpeta `Mesa-6.2.1` y entonces tecleemos

```
[usuario]$ cd Mesa-6.2.1
[usuario]$ chmod a+x bin/mklib
[usuario]$ make linux
```

para comprobar que la instalación tuvo éxito, probemos uno de los ejemplos que vienen con la distribución, así escribamos

```
[usuario]$ cd lib
[usuario]$ export LD_LIBRARY_PATH=${PWD}
[usuario]$ cd ../progs/demos
[usuario]$ ./gears
```

si **Mesa** se instaló correctamente, se presentará una animación de varios engranes moviéndose. En caso de tener algún problema es recomendable revisar con cuidado la página de "Internet", en donde se explica con detalle la instalación aquí presentada.

Una vez que hayamos instalado **Mesa** lo único que debemos hacer para que los ejemplos de **NURBS++** funcionen, es indicar al compilador la ruta del las nuevas bibliotecas de **GLUT** y **OpenGL**, las cuales están en la carpeta `/home/usuario/Mesa-6.2.1/lib`, una manera de hacerlo es modificando el Makefile de `/home/usuario/nurbs++-3.0.11/examples/nurbs` remplazando las líneas

```
OPENGL_LIBS = -lglut $(GL_LFLAGS) $(GL_LIBS) $(X_EXTRA_LIBS)
ALLLIBS = -lglut $(packagedir)/nurbs/libnurbsf.la $(MATRIXLIB)
```

por

```
OPENGL_LIBS = -L/home/usuario/Mesa-6.3.1/glut $(GL_LFLAGS) $(GL_LIBS) $(X_EXTRA_LIBS)
ALLLIBS = -L/home/usuario/Mesa-6.3.1/glut $(packagedir)/nurbs/libnurbsf.la $(MATRIXLIB)
```

3. Uso de la biblioteca NURBS++.

En esta sección describiremos brevete el uso de la biblioteca **NURBS++** para la representación y manipulación de curvas y superficies NURBS.

3.1. Curvas

Una curva NURBS de grado $p > 0$ está dada por

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i$$

en donde P_i son puntos en $\mathbb{R}^3$ y las funciones $N_{i,p(u)}$ son funciones B-spline definidas en el vector de nodos

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1} \ldots, u_n, \underbrace{b, \ldots, b}_{p+1}\}$$

La clase principal para la definición de un objeto NURBS en el caso de la curva, es `NurbsCurve` la cual se deriva de la clase `ParaCurve` y está definida en el archivo `nurbs.h`. Los prototipos de sus constructores son

```
NurbsCurve() ;
NurbsCurve(const NurbsCurve<T,N>& nurb) ;
NurbsCurve(const Vector< HPoint_nD<T,N> >& P1, const Vector<T> &U1, int deg=3);
NurbsCurve(const Vector< Point_nD<T,N> >& P1, const Vector<T> &W, const Vector<T> &U1,
           int deg=3);
```

Además se definen los tipos de datos

```
typedef NurbsCurve<float,3> NurbsCurvef;
typedef NurbsCurve<double,3> NurbsCurved;
typedef NurbsCurve<float,2> NurbsCurve_2Df;
typedef NurbsCurve<double,2> NurbsCurve_2Dd;
```

los cuales nos permite por ejemplo, construir una curva NURBS usando el constructor `NurbsCurvef`, en donde `<float,3>` nos indica que la curva manipulará números representados en punto flotante y puntos de tres dimensiones, análogamente con los otros tipos de datos.

El siguiente programa muestra cómo se puede definir una curva NURBS cúbica con 5 puntos de control, vector de nodos $U = \{0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1\}$ y pesos $W = \{1, 1, 1, 2, 1, 3\}$. La curva resultante se muestra en 2.

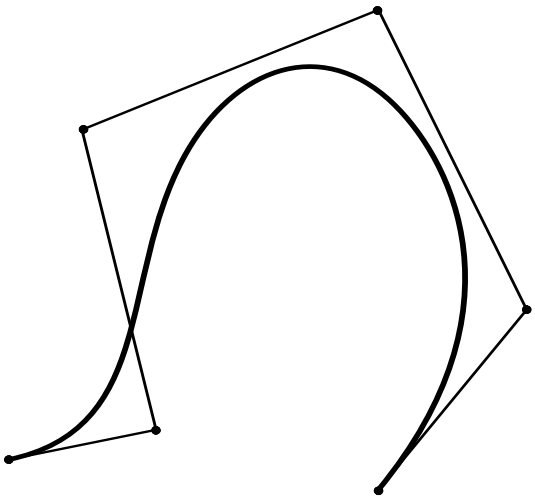

Figura 2: Curva NURBS cúbica.

```
//definida la clase NurbsCurve
#include <nurbs.h>
int main(){

 //Permiten escribir NurbsCurvef
 //en vez de PLib::NurbsCurvef
  using namespace PLib ;


//-----------------------------------------------
//Grado y número de puntos de la curva
//-----------------------------------------------

 int p = 3 ; //  Grado de la curva.
 int n = 5;  //  n+1 es el número de puntos.
 int i,j;    //  Variables auxiliares.

//-----------------------------------------------
//  Definición de los puntos de control y pesos
//-----------------------------------------------

  //Vector de puntos de tres dimensiones
  // en donde se almacenan los puntos
  // de control de la curva.

 Vector_Point3Df P(n+1);
  P[0] = Point3Df(2,3,0);
  P[1] = Point3Df(6,4,0);
  P[2] = Point3Df(4,14,0);
  P[3] = Point3Df(12,18,0);
  P[4] = Point3Df(16,8,0);
  P[5] = Point3Df(12,2,0);
```

```
  //Vector de números de punto flotante
  //en el que almacenaremos los pesos de
  // la curva.

  Vector_FLOAT W(n+1);
  W[0]=1.0;
  W[1]=1.0;
  W[2]=1.0;
  W[3]=2.0;
  W[4]=2.0;
  W[5]=1.0;

  //También es posible usar la versión de
  //los puntos en coordenadas homogéneas
  //haciendo

  Vector_HPoint3Df PH(n+1);
  PH[0] = HPoint3Df(2,3,0,1);
  PH[1] = HPoint3Df(6,4,0,1);
  PH[2] = HPoint3Df(4,14,0,1);
  PH[3] = HPoint3Df(12,18,0,2);
  PH[4] = HPoint3Df(16,8,0,2);
  PH[5] = HPoint3Df(12,2,0,1);

//-----------------------------------------------
//Definición del vector de nodos
//-----------------------------------------------

  // subíndice del último vector de nodos.
  int m = n+p+1;

  // Vector de nodos
  Vector_FLOAT U(m+1) ;
  for(i=0;i<=p;++i)
    U[i] = 0;
  for(i=p+1,j=1;i<=n ;i++,j++)
    U[i] = (float)j/float(n-p+1);
  for(i=n+1;i<=m;++i)
    U[i] = 1;

//-------------------------------------------
// Declaración de la curva nurbs
//-------------------------------------------

  NurbsCurvef curva_nurbs1(PH,U,p);
  NurbsCurvef curva_nurbs2(P,W,U,p);

//-------------------------------------------------
//Impresión en formato ps usando el método writePS
//-------------------------------------------------
  // Parámetros:

  //El primer argumento es el nombre del
  //archivo postscrip.

  //cp: sirve, para indicar si
  //se desea imprimir los puntos de control
```

```
  //en donde si cp=0 no y cp=1 si.
  int cp = 1;

  //magFact: sirve, para indicar el factor
  //de tama~o de la imagen postscrip.
  float magFact = 40.5;

  //dash: sirve para determinar los espacios
  //en blanco de las líneas que unen los puntos
  //de control de la curva.
  //Si este valor es menor o igual que 0
  //la línea no tendrá espacios.
  float dash =0.0;

  curva_nurbs1.writePS("cnurbs1.ps",cp,magFact,dash) ;
  curva_nurbs2.writePS("cnurbs2.ps",cp,magFact,dash) ;
//-------------------------------------------------------
//Impresión en formato VRML usando el método writeVRML
//-------------------------------------------------------

 // Parámetros:

  //El primer argumento es el nombre del
  //archivo en formato VRML.

  //radio: Radio de la línea.
  float radio = 2.0;

  //K: Número mínimo de interpolación.
  int K=18;

  //color: Color de la línea.
  Color color = blueColor;

  // Nu: Número de puntos en el circulo.
  int Nu =20;

  //Nv: Número de puntos a través del camino.
  int Nv = 30;

 curva_nurbs1.writeVRML("cnurbs1.wrl",radio,K,color,Nu,Nv) ;
 curva_nurbs2.writeVRML("cnurbs2.wrl",radio,K,color,Nu,Nv) ;

  return 0 ;
}
```

Es posible probar la curva NURBS, se convierte en una curva B-spline si $w_i = c$ con c constante para toda i, y una curva de Bézier si $n = p$ y

$$U = \{\underbrace{a, \ldots, a}_{p+1}, \underbrace{b, \ldots, b}_{p+1}\}$$

en particular si $a = 0$ y $b = 1$. En el siguiente programa creamos dos objetos, en donde uno representa una curva B-spline y otro una curva de Bézier. Las curvas resultantes se muestran en la figura 3

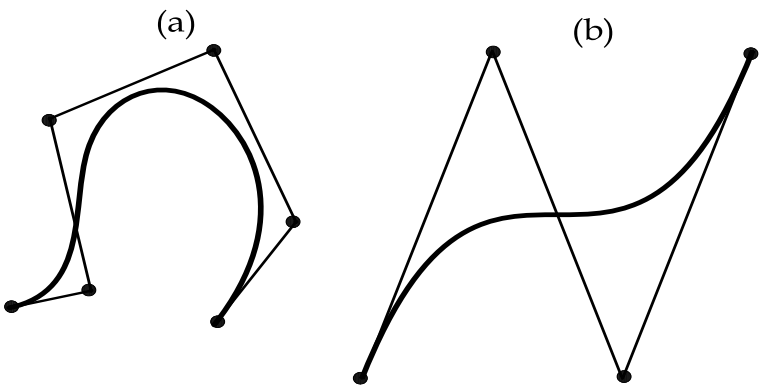

Figura 3: (a) Curva Bspline cúbica. (b) Curva de Bézier cúbica.

```
#include <nurbs.h>
int main(){

  int p  = 3; // Grado de las dos curvas.
  int n1 = 5; //  n1+1 es el número de
              // puntos de la curva B-spline.
  int n2 = 3; //  Número de puntos de la curva de Bézier.
  int i,j;    //  Variables auxiliares.

  using namespace PLib;

  //Puntos de control
  Vector_Point3Df P1(n1+1);
  P1[0] = Point3Df(2,3,0);
  P1[1] = Point3Df(6,4,0);
  P1[2] = Point3Df(4,14,0);
  P1[3] = Point3Df(12,18,0);
  P1[4] = Point3Df(16,8,0);
  P1[5] = Point3Df(12,2,0);

  //Pesos
  Vector_Point3Df Pd(n1+1);
  Vector_FLOAT W1(n1+1);
  W1[0]=1.0;
  W1[1]=1.0;
  W1[2]=1.0;
  W1[3]=1.0;
  W1[4]=1.0;
  W1[5]=1.0;

  // subíndice del último vector de nodos.
  int m1 = n1+p+1;

  // Vector de nodos
  Vector_FLOAT U1(m1+1) ;
  for(i=0;i<=p;++i)
    U1[i] = 0;
  for(i=p+1,j=1;i<=n1 ;i++,j++)
```

```
    U1[i] = (float)j/float(n1-p+1);
  for(i=n1+1;i<=m1;++i)
    U1[i] = 1;

 // Declaración de la curva B-spline
 PlNurbsCurvef curva_bspline(P1,W1,U1,p);
 curva_bspline.writePS("curva_bspline.ps",1,25.0,0) ;


/*********************************************************************************************
 *********************************************************************************************/



 //Puntos de control
  Vector_Point3Df P2(n2+1);
  P2[0] = Point3Df(0,-20,0) ;
  P2[1] = Point3Df(15,25,0) ;
  P2[2] = Point3Df(30,-20,0) ;
  P2[3] = Point3Df(45,25,0) ;


  //Pesos

  Vector_FLOAT W2(n2+1);
  W2[0]=1.0;
  W2[1]=1.0;
  W2[2]=1.0;
  W2[3]=1.0;


  // subíndice del último vector de nodos.
  int m2 = n2+p+1;

  // Vector de nodos
  Vector_FLOAT U2(m2+1) ;
  for(i=0;i<=p;++i)
    U2[i] = 0;
  for(i=n2+1;i<=m2;++i)
    U2[i] = 1;

 // Declaración de la curva de Bézier
 PlNurbsCurvef curva_de_bezier(P2,W2,U2,p);
 curva_de_bezier.writePS("curva_de_bezier.ps",1,25.0,0) ;

  return 0;
}
```

Como mencionamos al inicio de este apéndice, la biblioteca **NURBS++**, tiene implementados muchos de los algoritmos descritos en el libro "The NURBS Book", uno útil es el de interpolación global, el cual. Esta biblioteca crea una curva B-spline de interpolación usando el método `globalInterp`. En el siguiente programa usaremos este método para obtener las curvas de interpolación que se muestran en la figura 4.

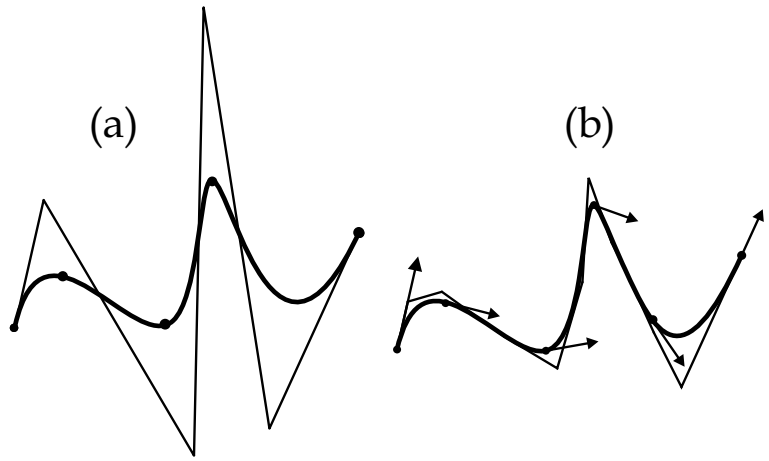

Figura 4: (a) Curva B-spline de interpolación. (b) Curva B-spline de interpolación de puntos y derivadas.

```
#include <nurbs.h>

using namespace PLib ;

//Definición de funciones auxiliares
double normaEuclidiana(Point3Df P1, Point3Df P2){
  return sqrt((P2.x()-P1.x())*(P2.x()-P1.x())+
              (P2.y()-P1.y())*(P2.y()-P1.y())+
              (P2.z()-P1.z())*(P2.z()-P1.z()));
}
double longitudArco(const Vector_Point3Df & P){
  int n = P.size();
  double larc=0.0;

  for(int i=0;i<n-1;i++)
    larc+= normaEuclidiana(P[i],P[i+1]);

  return larc;
}

int main(){

  int i ;
  int n = 5;

  // Puntos de control
  Vector_Point3Df points(n+1) ;
  points[0] = Point3Df(0,0,0) ;
  points[1] = Point3Df(5,5,0) ;
  points[2] = Point3Df(15,0,0) ;
```

```
points[3] = Point3Df(20,15,0) ;
points[4] = Point3Df(25,5,0) ;
points[5] = Point3Df(35,10,0) ;

// Cálculo del vector Ubar, usando la
// longitud de arco. Ver sección 4.4 del
// capítulo 4.

double larc = longitudArco(points);
std::cout << larc << std::endl;

Vector_FLOAT Ubar(n+1);
Ubar[0] = 0.0;
Ubar[n] = 1.0;

for(int i=1;i<n;i++)
  Ubar[i]= Ubar[i-1]+normaEuclidiana(points[i-1],points[i])/larc;


// Declaración de las curvas NURBS
NurbsCurvef curva1,curva2;

Vector_Point3Df deriv(n+1) ;
Vector_Point3Df empty ;

//Creación de la curva de interpolación
//cúbica, usando el vector Ubar.
curva1.globalInterp(points,Ubar,3);

//Cálculo de los vectores de derivadas
//en los puntos de interpolación.
Vector_Point3Df dtemp(2);
for(int i=0;i<=n;i++){
  curva1.deriveAt(Ubar[i],1,dtemp);
  deriv[i]= dtemp[1];
}

//Creación de la curva de interpolación
//cúbica, la cual además de interpolar los
//puntos, también lo hace con las derivadas.
//Ver sección 4.4 del capítulo 4.
curva2.globalInterpD(points,deriv,3,0) ;

//Impresión de la curva, en la cual también se
//presentan los puntos de interpolación.
curva1.writePSp("nurbsinterp1.ps",points,empty,1,25.0,0) ;

//Impresión de la curva, en la cual también se
//presentan los puntos de interpolación y sus
//derivadas.
```

```
  curva2.writePSp("nurbsinterp2.ps",points,deriv,1,25.0,0) ;

  return 0;
}
```

3.2. Superficies

Una superficie NURBS de grado p en dirección u y grado q en dirección v es una función racional por piezas bivariada de la forma

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \quad 0 \le u,v \le 1 \tag{1}$$

Los puntos $\{P_{i,j}\}$ forman una red bidireccional de puntos de control, los $\{w_{i,j}\}$ son los pesos y $\{N_{i,p}(u)\}y\{N_{j,q}(v)\}$ son dos sucesiones de funciones base B-spline sobre los vectores

$$U = \{\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,u_{r-p-1}, \underbrace{b,\ldots,b}_{p+1}\}$$
$$V = \{\underbrace{a,\ldots,a}_{q+1}, v_{q+1},\ldots,v_{s-q-1}, \underbrace{b,\ldots,b}_{q+1}\}$$

donde $r = n+p+1$ y $s = m+q+1$, respectivamente.

En este caso, la clase para su representación es `NurbsSurface`, la cual se deriva de `ParaSurface` y se encuentra en el archivo `nurbsS.h`. Tiene como constructores

```
NurbsSurface() ;
NurbsSurface(const NurbsSurface<T,N>& nS) ;
NurbsSurface(int DegU, int DegV, const Vector<T>& Uk,
             const Vector<T>& Vk, const Matrix< HPoint_nD<T,N> >& Cp) ;
NurbsSurface(int DegU, int DegV, Vector<T>& Uk,
             Vector<T>& Vk, Matrix< Point_nD<T,N> >& Cp, Matrix<T>& W) ;
```

Los tipos de datos definidos para la declaración de los objetos son

```
typedef NurbsSurface<float,3> NurbsSurfacef ;
typedef NurbsSurface<double,3> NurbsSurfaced ;
```

con éstos, se pueden definir superficies NURBS manejando datos de punto flotante y de doble precisión respectivamente.

El siguiente programa define una superficie NURBS de grados $p = 3$ y $q = 2$, puntos de control

Puntos de control					
i/j					
	(20,-10,10)	(10,-10,10)	(0,-10,10)	(-5,-10,10)	(-10,-10,10)
	(20,0,0)	(10,0,10)	(0,0,25)	(-5,0,10)	(-10,0,10)
	(20,5,10)	(10,5,10)	(0,5,10)	(-5,5,10)	(-10,5,10)
	(20,10,10)	(10,10,10)	(0,10,10)	(-5,10,10)	(-10,10,10)

pesos

$$W = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 \\ 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

y vectores de nodos

$$U = \{0, 0, 0, 0, 1, 1, 1, 1\}$$
$$V = \{0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1\}$$

las superficies resultantes para cada método de impresión se muestran en la figura 5.

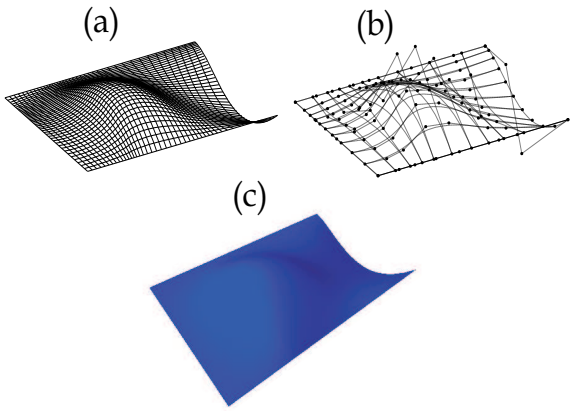

Figura 5: (a) Superficie NURBS. (b) Superficie NURBS, con los polígonos de control de sus isocurvas. (c) Superficie NURBS en el formato VRML.

```
#include <nurbsS.h>

int main()
{

  // Supuerficie B-spline

  int i,j;

  using namespace PLib ;

  int p = 3, q = 2;
  int n = 3, m = 4;  // Número de puntos, 4(0,...,3) en dirección u y 5 (0,...,4) en dirección v
  int a = 0, b = 1; // intervalo de los nodos

  Matrix_Point3Df Pts(n+1,m+1);
  Pts(0,0) = Point3Df(20,-10,10);
  Pts(1,0) = Point3Df(20,0,0);
  Pts(2,0) = Point3Df(20,5,10);
  Pts(3,0) = Point3Df(20,10,10);

  Pts(0,1) = Point3Df(10,-10,10);
  Pts(1,1) = Point3Df(10,0,10);
  Pts(2,1) = Point3Df(10,5,10);
  Pts(3,1) = Point3Df(10,10,10);

  Pts(0,2) = Point3Df(0,-10,10);
  Pts(1,2) = Point3Df(0,0,25);
  Pts(2,2) = Point3Df(0,5,10);
  Pts(3,2) = Point3Df(0,10,10);

  Pts(0,3) = Point3Df(-5,-10,10);
  Pts(1,3) = Point3Df(-5,0,10);
  Pts(2,3) = Point3Df(-5,5,10);
  Pts(3,3) = Point3Df(-5,10,10);

  Pts(0,4) = Point3Df(-10,-10,10);
  Pts(1,4) = Point3Df(-10,0,10);
  Pts(2,4) = Point3Df(-10,5,10);
  Pts(3,4) = Point3Df(-10,10,10);

  //Pesos
  Matrix_FLOAT W(n+1,m+1);
  for(i=0;i<n+1;i++)
    for(j=0;j<m+1;j++)
      W[i][j]=1.0;

    W[1][1] = 2;
    W[2][4] = 2;
    W[0][1] = 2;
    W[3][0] = 2;


  // Vector de nodos U y V
  int r = p+n+1;
  Vector_FLOAT U(r+1) ;
```

```
  for(i=0;i<=p;i++)
    U[i] = a;
  for(i=p+1,j=1; i<=r-p-1;i++,j++)
    U[i]=a+j*(float(b-a)/(r-2*p) );
  for(i=r-p;i<=r;i++)
    U[i] = b ;

  int s = q+m+1;
  Vector_FLOAT V(s+1);

  for(i=0;i<=q;i++)
    V[i]=a;
  for(i=q+1,j=1;i<=s-q-1;i++,j++)
    V[i] = a+j*(float(b-a)/(s-2*q));
  for(i=s-q;i<=s;i++)
    V[i] = b ;

   PlNurbsSurfacef sbspline(p,q,U,V,Pts,W);
   sbspline.writePS("SupBspline.ps",40,40,Point3Df(0,0,0),Point3Df(1.5,1.5,1.5),false,15,5);
   sbspline.writePS("SupBspline2.ps",10,10,Point3Df(0,0,0),Point3Df(1.5,1.5,1.5),true,15,5);
   sbspline.writeVRML("SupBspline.wrl",blueColor);
}
```

De forma análoga a las curvas, del capítulo 3 sección 3.5, sabemos que una superficie de Bézier Integral es un caso particular de una superficie NURBS, cuando

$$
\begin{aligned}
n &= p \\
m &= q \\
U &= \{\underbrace{0,\ldots,0}_{p+1},\underbrace{1,\ldots,1}_{p+1}\} \\
V &= \{\underbrace{0,\ldots,0}_{q+1},\underbrace{1,\ldots,1}_{q+1}\}
\end{aligned}
$$

El siguiente programa define una superficie de Bézier bicúbica con puntos de control

Puntos de control				
i/j				
	(15,0,10)	(5,0,15)	(-5,0,15)	(-15,0,10)
	(15,5,15)	(5,5,25)	(-5,5,25)	(-15,5,15)
	(15,10,15)	(5,10,25)	(-5,10,25)	(-15,10,15)
	(15,15,10)	(5,15,15)	(-5,15,15)	(-15,15,10)

Las superficies resultantes de los tres métodos de impresión, se presenta en la figura 6.

```
#include <nurbsS.h>

//Programa que define una superficie de Bézier usando
int main(){

  //Grados de la superficie de Bézier.
  int p   = 3, q   = 3;
  int npr = npc =p+1;

  int i,j,k; // variables auxiliares
```

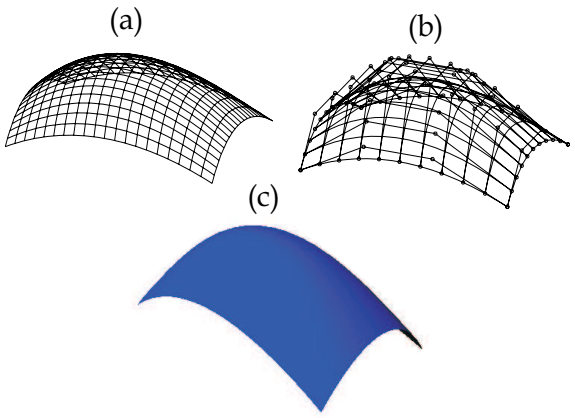

Figura 6: (a) Superficie de Bézier bicúbica. (b) Isocurvas con sus respectivos polígonos de control. (c) Superficie de Bézier en formato VRML.

```
using namespace PLib ;


//
Matrix_Point3Df Pts(npr,npc);

Pts(0,0) = Point3Df(15,0,10);
Pts(1,0) = Point3Df(15,5,15);
Pts(2,0) = Point3Df(15,10,15);
Pts(3,0) = Point3Df(15,15,10);


Pts(0,1) = Point3Df(5,0,15);
Pts(1,1) = Point3Df(5,5,25);
Pts(2,1) = Point3Df(5,10,25);
Pts(3,1) = Point3Df(5,15,15);


Pts(0,2) = Point3Df(-5,0,15);
Pts(1,2) = Point3Df(-5,5,25);
Pts(2,2) = Point3Df(-5,10,25);
Pts(3,2) = Point3Df(-5,15,15);

Pts(0,3) = Point3Df(-15,0,10);
Pts(1,3) = Point3Df(-15,5,15);
Pts(2,3) = Point3Df(-15,10,15);
Pts(3,3) = Point3Df(-15,15,10);



//Pesos
Matrix_FLOAT W(npr,npc);
```

```
  for(i=0;i<npr;i++)
    for(j=0;j<npc;j++)
      W[i][j]=1.0;


  // Vector de nodos U y V

  Vector_FLOAT U(2*p+2) ;
  for(i=0;i<p+1;i++)
    U[i] = 0;
  for(i=p+1 ;i<U.n();i++)
    U[i] = 1.0 ;

  Vector_FLOAT V(2*q+2);
  for(i=0;i<q+1;i++)
    V[i]=0;
  for(i=q+1;i<V.n();i++)
    V[i] = 1.0;

  NurbsSurfacef s(p,q,U,V,Pts,W);

  s.writePS("SupBezInt1.ps",10,10,Point3Df(10,10,10),Point3Df(0,0,0),true,15,5);
  s.writePS("SupBezInt2.ps",20,20,Point3Df(10,10,10),Point3Df(0,0,0),false,15,5);
  s.writeVRML("SupBezInt.wrl",blueColor);

  return 0;
}
```

El método de interpolación global usando una superficie B-spline, se presentó en el capítulo 5 sección 4.5, a continuación se presenta un programa que construye una superficie de este tipo, interpolando los puntos de control de la superficie NURBS que presentamos al inicio de la sección. Las superficie resultantes se presentan en la figura 7.

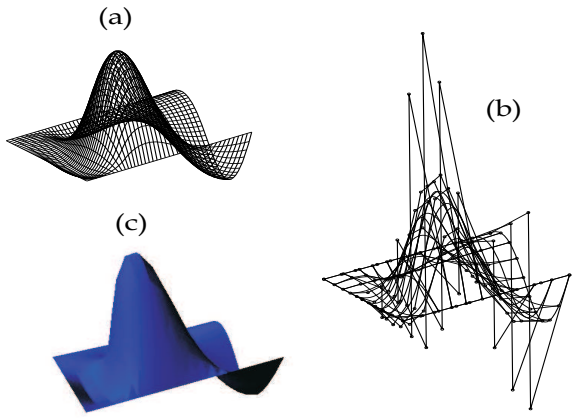

Figura 7: (a) Superficie B-spline de interpolación global. (b) Isocurvas con sus respectivos polígonos de control.(c) Superficie B-spline de interpolación global en el formato VRML.

```
#include <nurbsS.h>

int main()
{

  // Supuerficie B-spline

  int i,j;

  using namespace PLib ;

  int p = 3, q = 2;
  int n = 3, m = 4;  // Número de puntos, 4(0,...,3) en dirección u y 5 (0,...,4) en dirección v
  int a = 0, b = 1; // intervalo de los nodos



  Matrix_Point3Df Pts(n+1,m+1);

  Pts(0,0) = Point3Df(20,-10,10);
  Pts(1,0) = Point3Df(20,0,0);
  Pts(2,0) = Point3Df(20,5,10);
  Pts(3,0) = Point3Df(20,10,10);


  Pts(0,1) = Point3Df(10,-10,10);
  Pts(1,1) = Point3Df(10,0,10);
  Pts(2,1) = Point3Df(10,5,10);
  Pts(3,1) = Point3Df(10,10,10);


  Pts(0,2) = Point3Df(0,-10,10);
  Pts(1,2) = Point3Df(0,0,25);
  Pts(2,2) = Point3Df(0,5,10);
  Pts(3,2) = Point3Df(0,10,10);

  Pts(0,3) = Point3Df(-5,-10,10);
  Pts(1,3) = Point3Df(-5,0,10);
  Pts(2,3) = Point3Df(-5,5,10);
  Pts(3,3) = Point3Df(-5,10,10);

  Pts(0,4) = Point3Df(-10,-10,10);
  Pts(1,4) = Point3Df(-10,0,10);
  Pts(2,4) = Point3Df(-10,5,10);
  Pts(3,4) = Point3Df(-10,10,10);

    NurbsSurfacef sbspline;
   sbspline.globalInterp(Pts,3,2);
   sbspline.writePS("SupInterp1.ps",40,40,Point3Df(0,0,0),Point3Df(1.5,1.5,1.5),false,15,5);
   sbspline.writePS("SupInterp2.ps",10,10,Point3Df(0,0,0),Point3Df(1.5,1.5,1.5),true,15,5);
   sbspline.writeVRML("SupInterp.wrl",blueColor);


return 0;
```

Referencias

[Blom91] **Approximation of Sweep Surfaces by Tensor Product NURBS**, *Bloomenthal M. y Riesenfeld R.F.*, Curves and Surfaces in Computer Vision and Graphics II, Society of Photo-Optical Instrumentation Enginneers, Vol. 1610 páginas 132-144, 1991.

[Boehm80] **Inserting New Knots Into B-splines Curves**,*Wolfgang Boehm*, Computer Aided Design (CAD), Vol. 12, No. 4, páginas 199-202, 1980.

[Boehm85] **The insertion algorithm.** *Boehm, W., y Prautzsch, H.* Computer Aided Design (CAD), Vol. 17, No. 2, páginas 58-59, 1985.

[Cohen] **Discrete B-splines and subdivison techniques in computer-aided geometric design and computer graphics**, *Cohen E., Lyche T. y Riesenfeld R.* Computer Graphics and Image Processing, Vol. 14 páginas 87-111, 1980.

[Cohen85] **Algorithms for degree-raising of splines**, *Cohen E., Lyche, T., and Schumaker, L.L.,* ACM TOG, Vol. 4, No. 3, pp. 171-181,1985.

[Coqu87] **A Control Point Based Sweeping Technique**,*Coquillart S.*, IEEE Comput. Graph and Appl., Vol 7, No.11, páginas 36-45, 1987.

[Cox72] **The Numerical Evaluation of B-splines** *M.G. Cox.*, Jour. Inst. Math. Aplic., Vol. 10, pp. 134 − 149, 1972.

[Cox82] **Practical Spline Aproximation**, *M.G. Cox.*, NPL Report Division of Information Technology and Computing, 1982.

[DeBoor72] **On calculating with B-splines**, *De Boor, C.*, Jour. Approx. Theory, Vol. 6, pág.50-62, 1972.

[DeBoor78] **A Practical Guide to Splines**, *De Boor, C.*, New York: Springer-Verlag, 1978.

[DoCa] **Differential Geometry of Curves and Surfaces**, *Englewood Cliffs,* NJ: Prentice-Hall, 1976.

[Far90] **Curves and Surfaces for CAGD, a practical guide**, *Gerald Farin*, Academic Press, Inc. Third Edition, 1990.

[Far91] **NURBS for curve and surface design**,*Editado por Gerald Faring*, SIAM Activity Group on Geometric Design,1991.

[GGM95] **Cálculo Numérico I** *Gasca Gonzáles Mariano*, Universidad Nacional de Educación a Distancia, Madrid 1995.

[HL97] **Computer Aided Geometric Design**, *Josef Hoschek, Dieter Lasser*, A. K. Peters, Wellesley, Massachusetts, 1997.

[Klok] **Two moving coordinate frames for sweeping along a 3D trajectory**, *Klok,F.*, Computer Aided Geometry Desing, Vol. 3, pág. 217-229, 1986.

[Lee87] **Rational quadratic Bézier representation for conics, in *Geometric Modeling: Algorithms and New Trends***, *Eugene T. Y. Lee.* Faring, G.E.,Ed., Philadelphia:SIAM, pp. 3-19,1987.

[Marsh] **Applied Geometry for Computer Graphics and CAD.** , *Marsh Duncan.* Springer-Verlag London Berling Heidelberg.

[Pieg94] **Sofware engineering approach to degree elevation of B-splines curves.** *Piegl, L. y Tiller, W.* Computer Aided Design, Vol. 26, No. 1, páginas: 17-28, 1994.

[Prau84] **Degree elevation of B-spline curves.** *Prautzsch H.* Computer Aided Design, Vol. 1, No. 1, páginas: 193-198, 1984.

[Prau91] **A fast algorithm to raise the degree of spline curves.** *Prautzsch, H., y Piper, B.* Computer Aided Geometry Design, Vol. 8, páginas: 253-265,1991.

[PT97] **The NURBS Book** *Les Piegl, Wayne Tiller*, Springer-Verlang 1997.

[Ram02] **Introducción a la geometría avanzada.** *Ana Irene Ramírez Galarza, José Seade Kuri*, Facultad de Ciencias UNAM, 2002.

[Rog01] **An Introducction to NURBS With Historical Perspective**, *David F. Rogers*,Morgan Kaufmann Pubishers 2001.

[Scho46] **Contributions to the problem of approximation of equidistant data by analytic functions**, *Schoenberg, I.J.*, Quart. Appl. Math, Vol. 4, pág. 45-99, 1946.

[Lang] **Algebra**, *Serge Lang.* , Addison-Wesly Publishing Company, 1993.

[Silt] **Normal orientation methods for 3D offset curves, sweep surfaces and skinning**, *Siltanen, P., and Woodward, C.*, Proc. Eurographics 92, Vol. 11, No.3, pág. C-449-C-457, 1992.

[tesis] **Diseño de Curvas en Forma Libre con B-splines, Tesis de Licenciatura**, *Cervantes Cabrera Daniel.* Facultad de Ciencias UNAM, 2000.

[Wood87] **Cross-Sectional Design of B-splines Surfaces.** *Charles D. Woodward*, Computer and Graphics, Vol. 11, No. 2, páginas 193-201, 1987.