PRÁCTICA 2: Modelado geométrico curvo

Práctica 2.1 - Visualización de curvas cúbicas de Bezier

Práctica 2.2 - Visualización de superficies bicúbicas de Bezier

Objetivos

- Implementar clases para curvas y superficies de Bezier de grado 3
  - Uso del método de diferencias avanzadas para el cálculo de puntos
  - Cálculo de tangentes
  - Cálculo de normales en superficies
- Implementar un visualizador de curvas de Bezier
- Implementar un visualizador de superficies de Bezier

Planificación

- 1,5 + 1,5 sesiones
- A partir de ahora se evalúan las prácticas
- **Prácticas individuales**
- Puntuación
  - Práctica 2.1 : 0,25 (parte mínima) + 0,25 (parte adicional)
  - Práctica 2.2 : 0,25 (parte mínima) + 0,25 (parte adicional)

2.1 Ficheros para Curvas de Bezier

- Se usan:
  - *Algebra (.h .cpp)*
  - *Primitiva (.h .cpp)*
  - *CurvaBezier.h*: definición de la clase *CurvaBezier*
  - *VerCurvaSimple.cpp*: validación de la implementación
- Se implementan:
  - *CurvaBezier.cpp*: implementación de la clase. Se suministra incompleto
  - *Ver2Curvas.cpp*: Programa de visualización de dos tramos de curva enlazados
- Se entregan:
  - Todos los *(.h .cpp)* necesarios para generar los ejecutables
  - *VerCurvaSimple.exe* y *Ver2Curvas.exe*

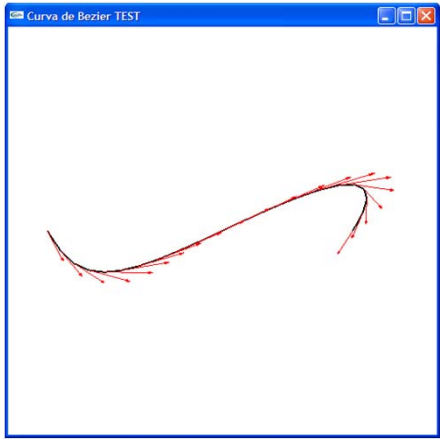


Fig. 1. VerCurvaSimple

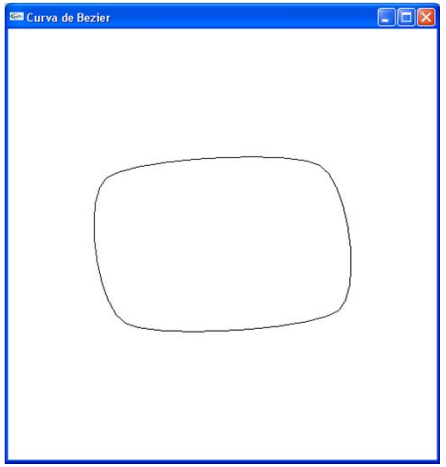
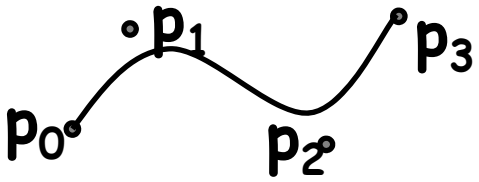


Fig. 2. Curvas enlazas

2.1 Definición de la clase Curva de Bezier

```
class CurvaBezier
{
private:
   static const Matriz MBezier;   //Matriz característica de Bezier
   Punto pControl[4];             //Puntos de control
   Matriz C;                      //Matriz de Coeficientes

   void setC();                   //Calcula la matriz de coeficientes

public:
   CurvaBezier();                 //Constructores
   CurvaBezier(Punto p[4]);
   CurvaBezier(Punto p0, Punto p1, Punto p2, Punto p3);

   void setPoint(int cual, Punto nuevo);
   //Cambia el punto cual [0..3] por el nuevo

   Punto controlPoint(int i)const;
   //Devuelve el punto de control [0..3]

   Vector tangent(float u)const;
   //Devuelve el vector unitario tangente en u

   void getPoints(int n, Punto *puntos)const;
   //Muestrea la curva y devuelve n puntos

   void getTangents(int n, Vector *tangentes)const;
   //Muestrea la curva y devuelve n tangentes

};
```



2.1 Cálculo de puntos en curvas de Bezier

$$Q(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_{Bez} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot C$$

- *setC()*: calcula la matriz de coeficientes multiplicando la matriz característica por la matriz de puntos de control (en filas)
- *getPoints(n,puntos)*: hay que implementar el método de las diferencias avanzadas. El parámetro *n* indica el número de puntos a devolver. La curva pasará por el primer y último punto de control. δ es la distancia paramétrica entre dos puntos seguidos *1/(n-1)*

$$D = E(\delta) \cdot C = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \delta^3 & \delta^2 & \delta & 0 \\ 6\delta^3 & 2\delta^2 & 0 & 0 \\ 6\delta^3 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_x & a_y & a_z & 0 \\ b_x & b_y & b_z & 0 \\ c_x & c_y & c_z & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix}$$

2.1 Método de diferencias en curvas de Bezier.

$$Q(u)$$
$$\Delta Q(u) = Q(u+\delta) - Q(u)$$
$$\Delta^2 Q(u) = \Delta Q(u+\delta) - \Delta Q(u)$$
$$\Delta^3 Q(u) = \Delta^2 Q(u+\delta) - \Delta^2 Q(u)$$

$$Q(u) = au^3 + bu^2 + cu + d$$
$$\Delta Q(u) = 3au^2\delta + u(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta$$
$$\Delta^2 Q(u) = 6a\delta^2 u + 6a\delta^3 + 2b\delta^2$$
$$\Delta^3 Q(u) = 6a\delta^3$$

$$D_{(0)} = E(\delta) \cdot C$$

```
Algoritmo DibujarCurva(n,D)
{D: vector de incrementos en u=0}
  constante inc3P := D[4];
  P= D[1]; incP= D[2]; inc2P=D[3];
  moverA(P.x,P.y,P.z);
  para i:=1 hasta n
    P:= P+incP;
    incP:= incP+inc2P;
    inc2P:= inc2P+inc3P;
    lineaA(P.x,P.y,P.z);
  fin para
fin DibujarCurva
```

Adaptar el algoritmo para muestrear y devolver un vector de puntos

2.1 Cálculo de tangentes en curvas de Bezier

$$Q'(u) = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \cdot M_{Bez} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \cdot C$$

- *tangent(u)*: Calcula el vector unitario tangente en *Q(u)*
- *getTangents(n,tangentes)*: Devuelve *n* tangentes uniformemente espaciadas en *u*. Se puede hacer por simple evaluación de *Q'(u)* dando valores a *u*

2.1 Valoración de la práctica

- La práctica puntúa **0,5 puntos**.
- **Parte mínima**. Se obtienen 0,25 puntos si:
  - Se construye correctamente la clase *CurvaBezier* según requisitos
  - Se visualiza correctamente la curva test *VerCurvaSimple*
  - Se construye *Ver2Curvas* que dibuje dos curvas enlazadas con continuidad C^1. Cada curva será de un color diferente
  - Se dibujan correctamente las tangentes en cada punto calculado
- **Parte adicional**. Se valorará, para los 0,25 puntos restantes, lo siguiente:
  - Dibujo de los puntos de control, el polígono característico y los ejes de coordenadas
  - Interacción mediante ratón para movimiento del gráfico (inspección)
  - Animación basada en la transformación de los puntos de control usando *Algebra*
  - Edición interactiva de los puntos de control

2.2 Ficheros para Superficies de Bezier

- Se usan:
  - *Algebra (.h .cpp)*
  - *Primitiva (.h .cpp)*
  - *SuperficieBezier.h*: definición de la clase *SuperficieBezier*
  - *VerSuperficieAlambrico.cpp*: programa de validación
- Se implementan:
  - *SuperficieBezier.cpp*: implementación de la clase. Se suministra incompleto
  - *VerSuperficie.cpp*: Programa de visualización de una superficie
- Se entregan:
  - Todos los *(.h .cpp)* necesarios para generar los ejecutables
  - *VerSuperficieAlambrico.exe* y *VerSuperficie.exe*

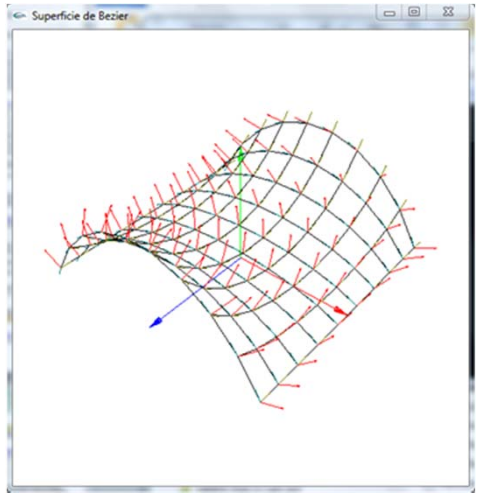


Fig. VerSuperficieAlambrico

2.2 Definición de la clase Superficie Bezier

```
class SuperficieBezier
{
private:
    static const Matriz MBezier;                    //Matriz característica de Bezier
    Bloque pControl;                                //16 Puntos de control
    Bloque C;                                       //Matriz de coeficientes

    void setC();                                    //Calcula la matriz de coeficientes
public:

    SuperficieBezier();                             //Constructores
    SuperficieBezier(Punto p[16]);                  //Ordenación igual que constructor
    siguiente
    SuperficieBezier(Punto p00, Punto p01, Punto p02, Punto p03, //u cte
                     Punto p10, Punto p11, Punto p12, Punto p13,
                     Punto p20, Punto p21, Punto p22, Punto p23,
                     Punto p30, Punto p31, Punto p32, Punto p33);
    Real4 controlPoint(int i, int j)const;      //Devuelve el punto de control Pij
    void setPoint(int i, int j, Punto nuevo);   //Cambia el punto i,j [0..3][0..3]
    void getPoints(int n, Punto *puntos);
    //Devuelve nxn puntos en una malla
    void getNormals(int n, Vector *normales)const;
    //Devuelve nxn normales unif. distribuidas
    void getTangents(int n, Vector *tgU, Vector *tgV)const;
    //Devuelve nxn tangentes en u y v unif. distribuidas
    Vector uTangent(float u, float v)const;
    //Devuelve el vector tangente en dirección u en u,v
    Vector vTangent(float u, float v)const;
    //Devuelve el vector tangente en dirección v en u,v
    Vector normal(float u, float v)const;
    //Devuelve el vector normal en u,v
};
```

2.2 Cálculo de puntos en superficies de Bezier

capa x

$$S^x(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} M_{Bez} \begin{bmatrix} p^x_{0,0} & p^x_{0,1} & p^x_{0,2} & p^x_{0,3} \\ p^x_{1,0} & p^x_{1,1} & p^x_{1,2} & p^x_{1,3} \\ p^x_{2,0} & p^x_{2,1} & p^x_{2,2} & p^x_{2,3} \\ p^x_{3,0} & p^x_{3,1} & p^x_{3,2} & p^x_{3,3} \end{bmatrix} M^T_{Bez} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} = UM_{Bez} G_{Bez} M^T_{Bez} V^T$$

- Las matrices de puntos de control y de coeficientes son 3D (bloques)
- En *getPoints(n,points)* hay que implementar el método de las diferencias avanzadas. Se devuelven nxn puntos. $\delta = 1/(n-1)$ es la distancia paramétrica entre dos puntos seguidos en *u* o en *v*

$$D = E(\delta_u) \cdot C \cdot E(\delta_v)^T$$

$$E(\delta) = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \delta^3 & \delta^2 & \delta & 0 \\ 6\delta^3 & 2\delta^2 & 0 & 0 \\ 6\delta^3 & 0 & 0 & 0 \end{bmatrix}$$

2.2 Diferencias avanzadas para superficies

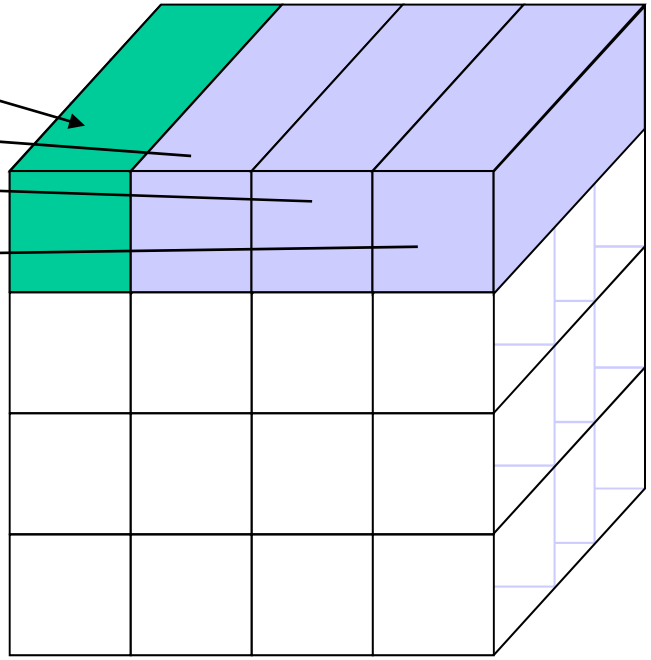
desde u=0 hasta n-1

primer punto curva u

inc1

inc2

inc3



Matriz de diferencias

2.2 Diferencias avanzadas para superficies

desde u=0 hasta n-1

primer punto curva u

inc1

inc2

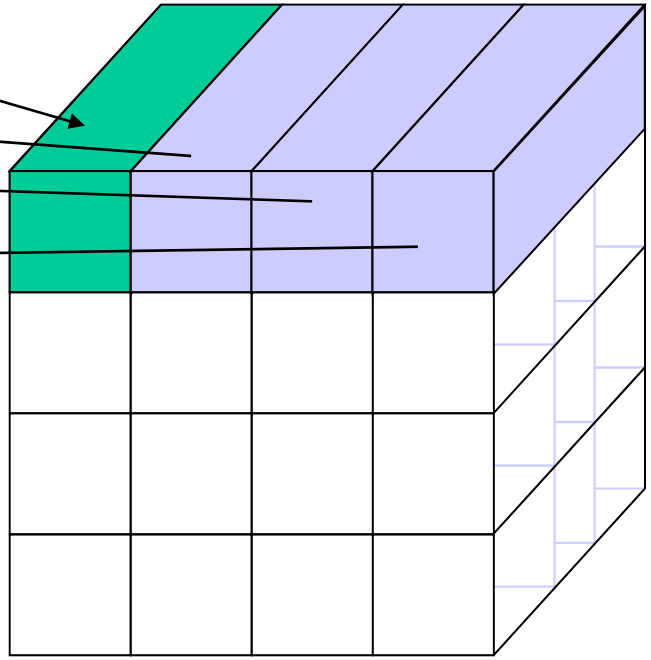
inc3

desde v=1 hasta n-1

siguiente punto=anterior+inc1

inc1+=inc2; inc2+=inc3

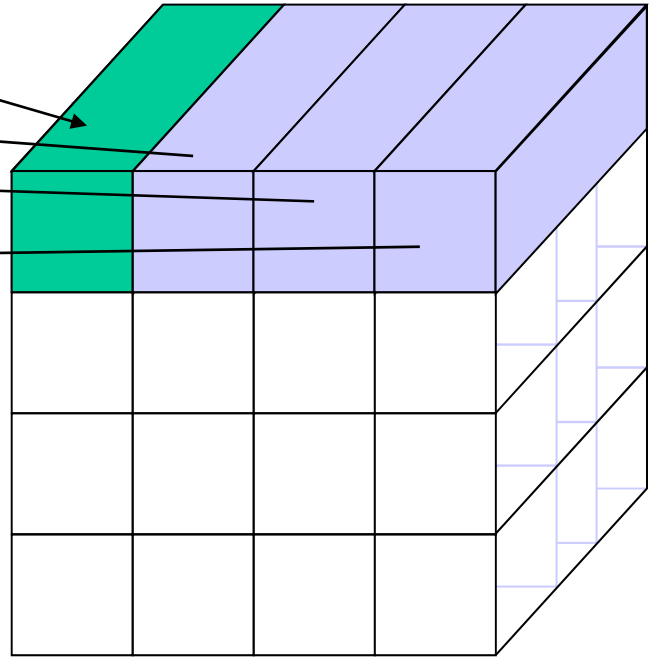
fin desde



Matriz de diferencias

2.2 Diferencias avanzadas para superficies

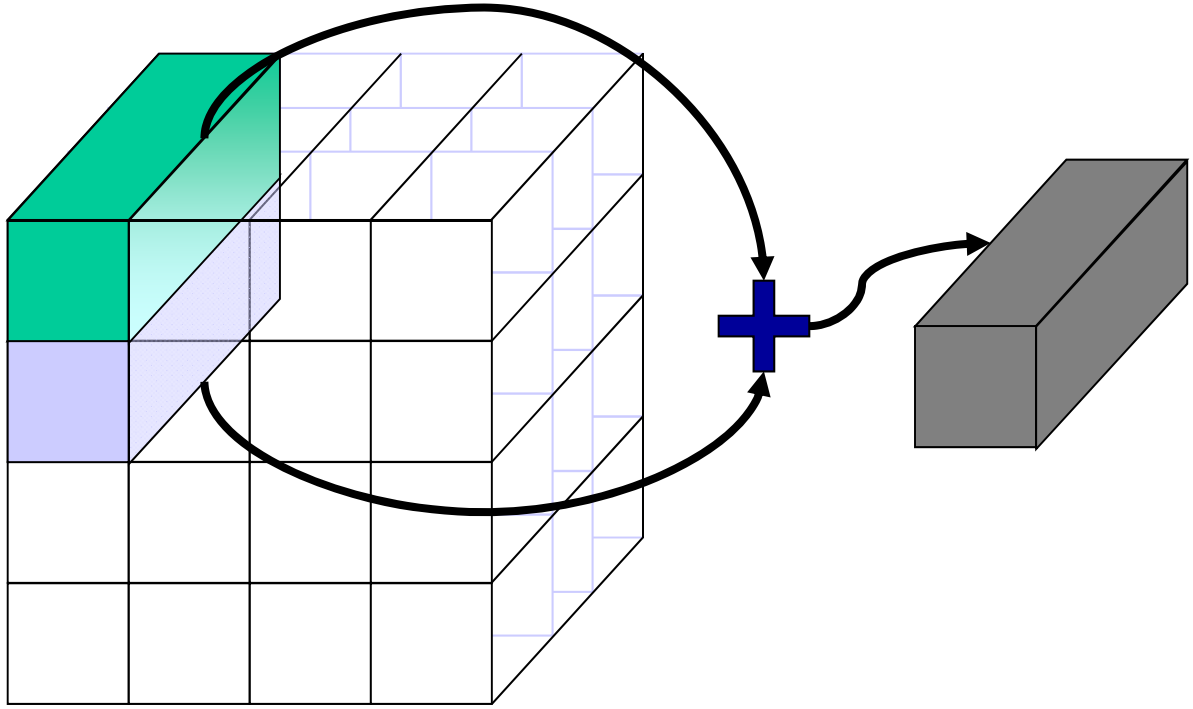
```
desde u=0 hasta n-1
  primer punto curva u
  inc1
  inc2
  inc3
  desde v=1 hasta n-1
    siguiente punto=anterior+inc1
    inc1+=inc2; inc2+=inc3
  fin desde
  actualizar Matriz de diferencias
fin desde
```



Matriz de diferencias

2.2 Diferencias avanzadas para superficies

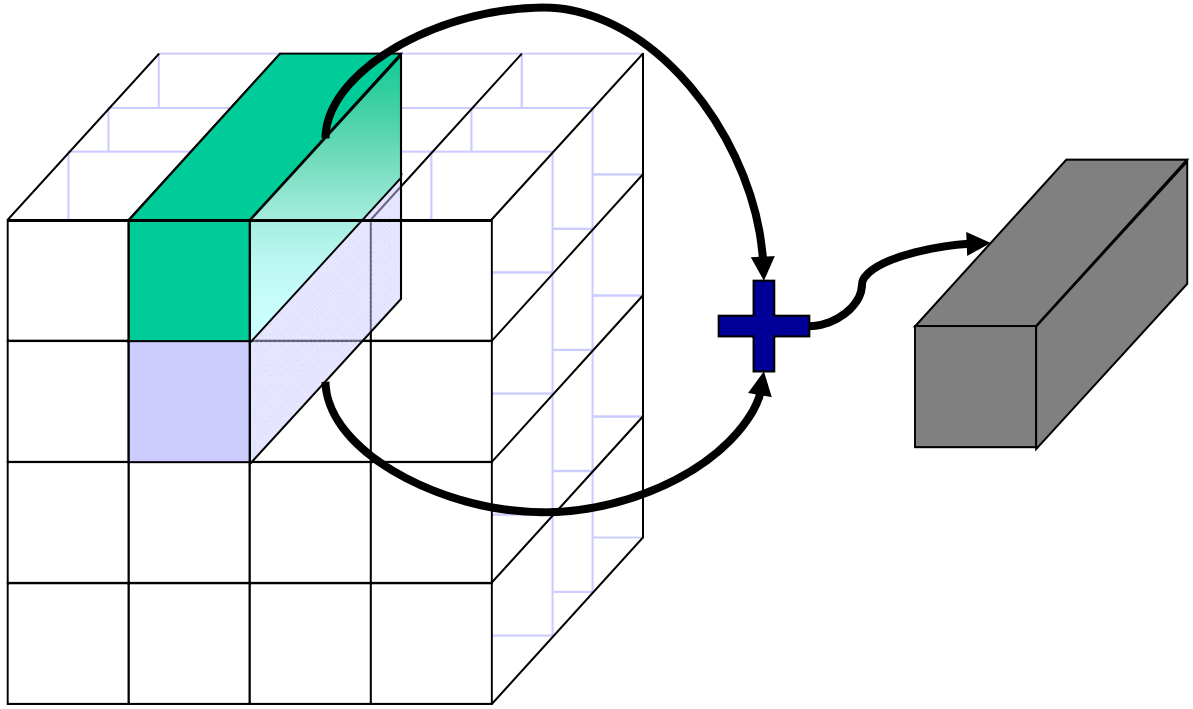
```
desde u=0 hasta n-1
  primer punto curva u
  inc1
  inc2
  inc3
  desde v=1 hasta n-1
    siguiente punto=anterior+inc1
    inc1+=inc2; inc2+=inc3
  fin desde
  actualizar Matriz de diferencias
  fin desde
```



Matriz de diferencias

2.2 Diferencias avanzadas para superficies

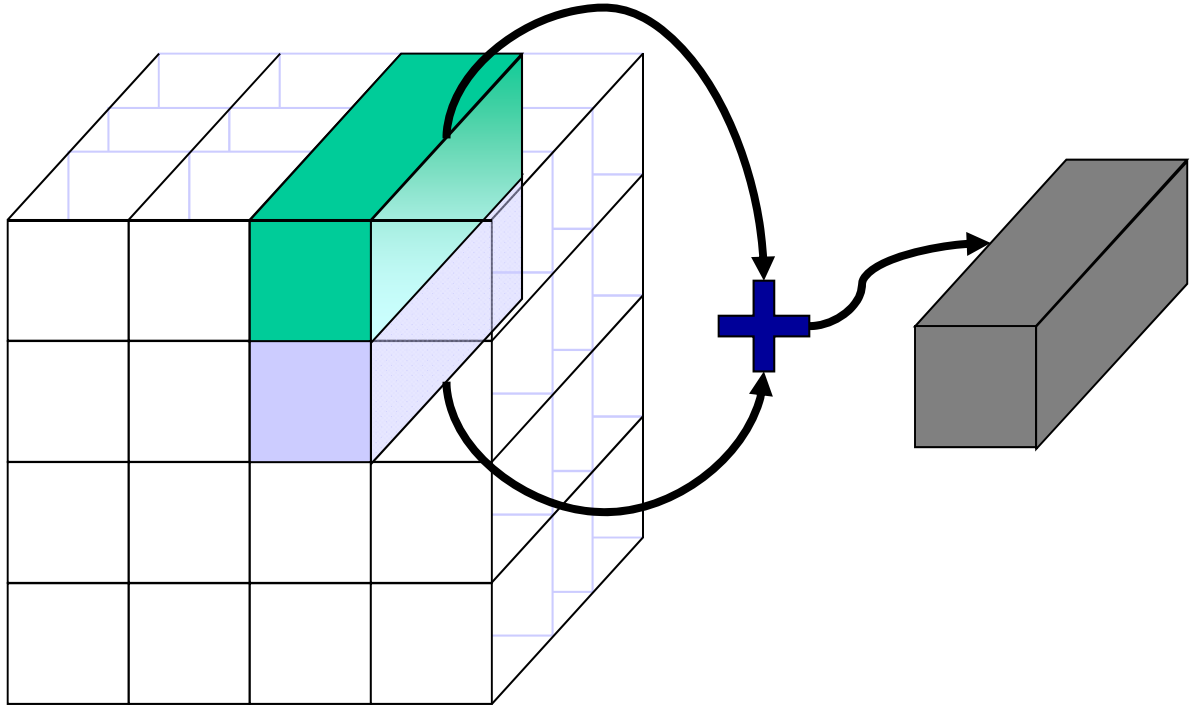
```
desde u=0 hasta n-1
  primer punto curva u
  inc1
  inc2
  inc3
  desde v=1 hasta n-1
    siguiente punto=anterior+inc1
    inc1+=inc2; inc2+=inc3
  fin desde
  actualizar Matriz de diferencias
  fin desde
```



Matriz de diferencias

2.2 Diferencias avanzadas para superficies

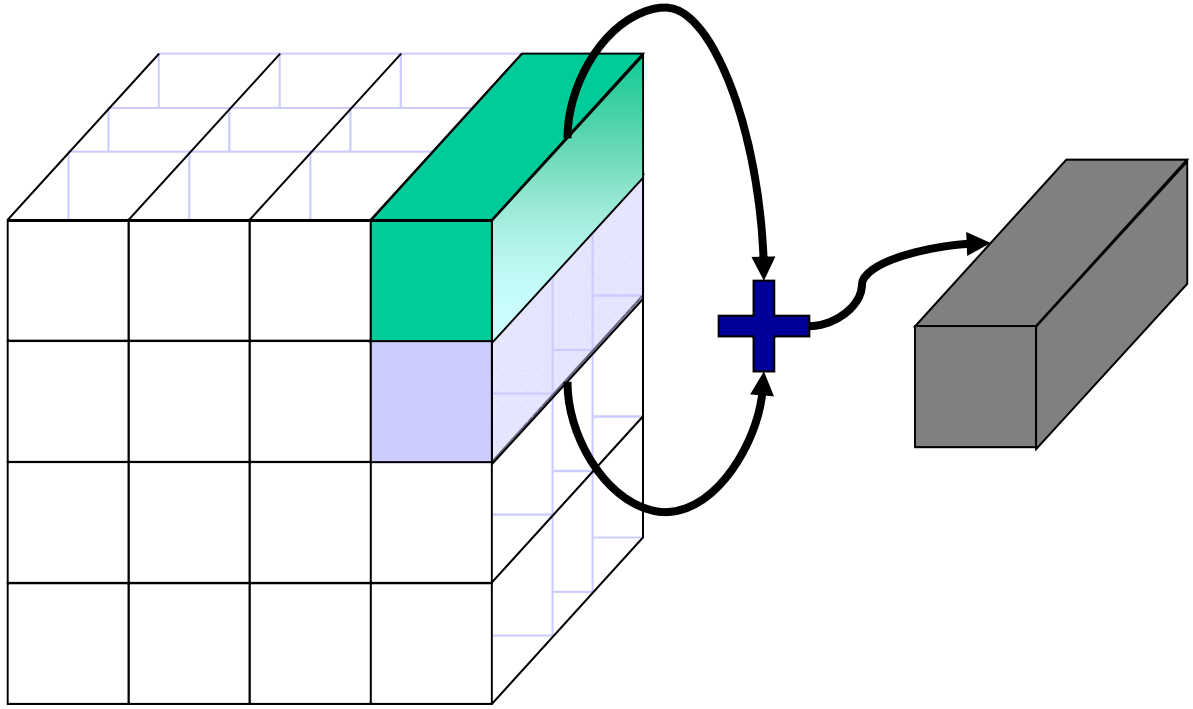
```
desde u=0 hasta n-1
  primer punto curva u
  inc1
  inc2
  inc3
  desde v=1 hasta n-1
    siguiente punto=anterior+inc1
    inc1+=inc2; inc2+=inc3
  fin desde
  actualizar Matriz de diferencias
  fin desde
```



Matriz de diferencias

2.2 Diferencias avanzadas para superficies

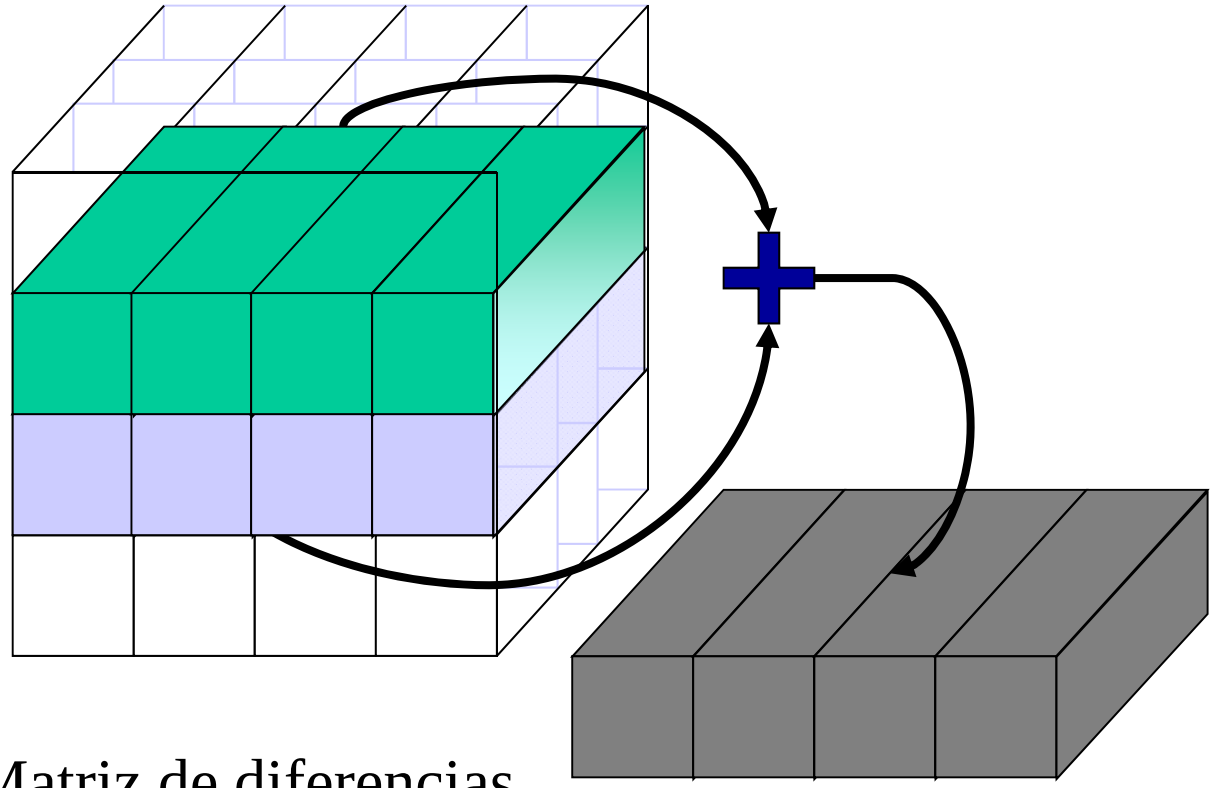
```
desde u=0 hasta n-1
  primer punto curva u
  inc1
  inc2
  inc3
  desde v=1 hasta n-1
    siguiente punto=anterior+inc1
    inc1+=inc2; inc2+=inc3
  fin desde
  actualizar Matriz de diferencias
  fin desde
```



Matriz de diferencias

2.2 Diferencias avanzadas para superficies

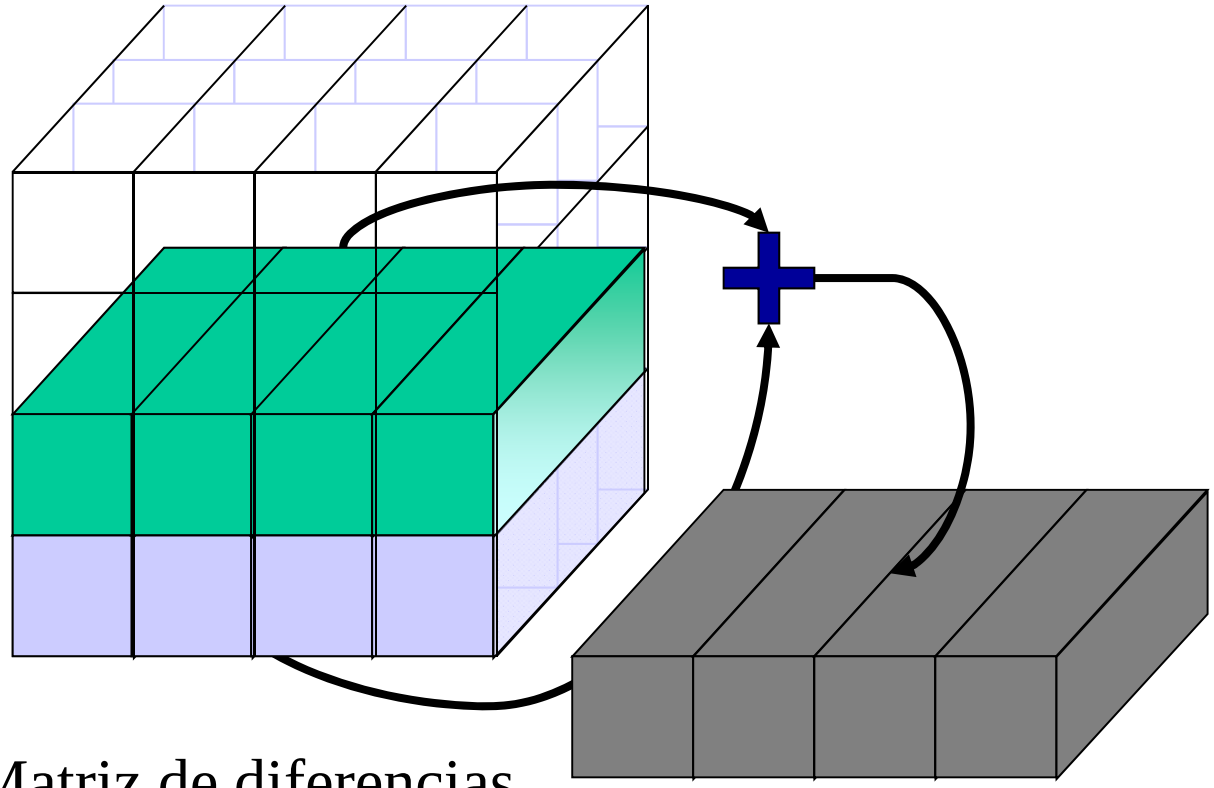
```
desde u=0 hasta n-1
  primer punto curva u
  inc1
  inc2
  inc3
  desde v=1 hasta n-1
    siguiente punto=anterior+inc1
    inc1+=inc2; inc2+=inc3
  fin desde
  actualizar Matriz de diferencias
  fin desde
```



Matriz de diferencias

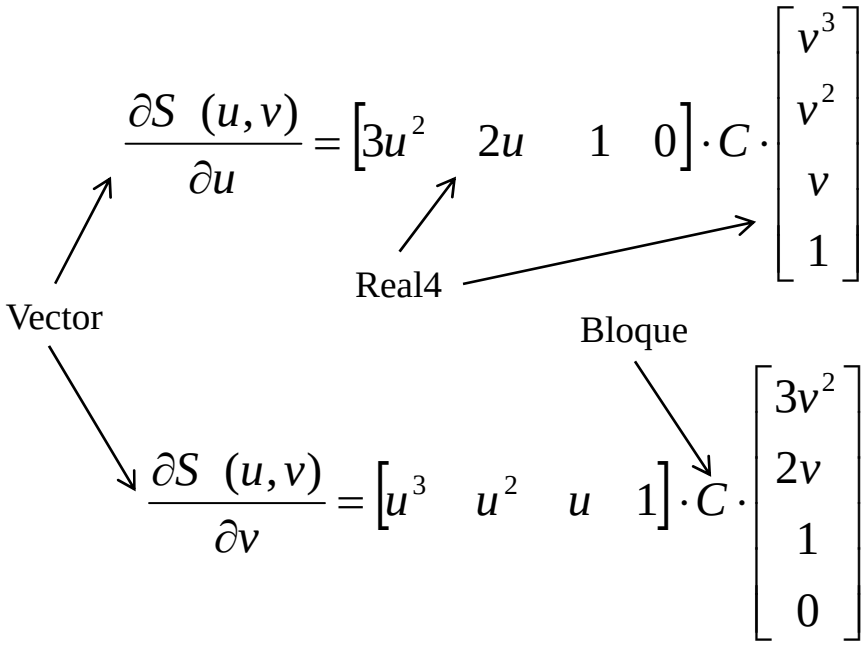
2.2 Diferencias avanzadas para superficies

```
desde u=0 hasta n-1
  primer punto curva u
  inc1
  inc2
  inc3
  desde v=1 hasta n-1
    siguiente punto=anterior+inc1
    inc1+=inc2; inc2+=inc3
  fin desde
  actualizar Matriz de diferencias
  fin desde
```



Matriz de diferencias

2.2 Cálculo de tangentes y normales



$$\vec{n} = \frac{\partial S\ (u,v)}{\partial u} \times \frac{\partial S\ (u,v)}{\partial v}$$

Los vectores deben ser unitarios!!

2.2 Valoración de la práctica

- **La práctica 2.2 vale 0,5 puntos.**

- **Parte mínima**. Para la obtención de 0,25 puntos se requiere:
  - La implementación correcta de la clase SuperficieBezier
  - La representación gráfica correcta de *VerSuperficieAlambrico*
  - El dibujo de las normales en cada punto de la superficie

- **Parte adicional**. Para los restantes 0,25 puntos se valorará:
  - El dibujo de la malla de control y los ejes sobre una superficie propia diferente
  - La iluminación de la superficie como malla poligonal usando OpenGL
  - La interactividad con la aplicación (inspección, luces, resolución de malla, etc.)
  - Animación de los puntos de control usando transformaciones de *Algebra*