Open Inventor

6 - Curve

Claudio Baldazzi *Maurizio Patrignani*

Università degli Studi di Roma Tre
Dip. di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

30 aprile 1998

1 Introduzione

Curve e superfici sono strumenti matematici per descrivere modelli geometrici. Il progettista usa quindi espressioni matematiche per rappresentare gli oggetti, ad esempio per rappresentare le ali di un aereoplano, la carrozzeria di un automobile, parti meccaniche di un motore, etc. Inventor usa un particolare tipo di polinomi parametrici cubici, le **NURBS** (Non-Uniform Rational B-Spline), per rappresentare curve e superfici. Le informazioni, le figure e i listati contenuti in questo documento, sono tratti, salvo diversa indicazione, dal libro "The Inventor Mentor" di Josie Wernecke edito dalla Addison Wesley.

2 Overview sulle curve

In questa sezione vedremo alcuni convetti di base per le curve parametriche e il significato di alcuni termini che si useranno diffusamente nel resto del documento.

Le curve nello spazio possono essere rappresentate matematicamente in diverse forme:

- *Forma Esplicita*, in cui si esprimono y e z in funzione di x, quindi $y = f(x)$ e $z = g(x)$. I problemi di questa rappresentazione sono sostanzialmente tre: (1) è impossibile avere valori multipli di y (o z) data una singola x, per cui non si riesce a rappresentare oggetti come cerchi o ellissi; (2) questa rappresentazione non è rotazionalmente invariante, nel senso che se bisogna ruotare la curva si devono ricalcolare le funzioni; (3) risulta difficile descrivere le parti di curva con tangente perpendicolare all'asse x.
- *Forma Implicita*, quindi $f(x, y, z) = 0$. I problemi che si possono avere sono: (1) un'equazione di questo tipo può portare a più soluzioni di quante se ne desidera (ad esempio se l'equazione è quella di una circonferenza mentre si desidera disegnare una semicirconferenza); (2) se si volesse congiungere due equazioni insieme ci sarebbe un problema nel punto di giunzione perchè è difficile imporre la stessa tangente nel punto di giunzione.

- *Forma Parametrica* di una curva, $x = x(u)$, $y = y(u)$ e $z = z(u)$ con $u \in [0,1]$. Questo tipo di rappresentazione elimina i problemi su esposti ed anzi crea degli altri notevoli vantaggi. Ogni segmento Q della curva è dato dalle tre funzioni che sono dei polinomi, di solito, cubici nel parametro u.

Come si è visto i polinomi scelti per le curve sono cubici e questo per diverse ragioni. Innanzitutto polinomi a grado inferiore sono meno flessibili nel controllo della forma della curva. Ad esempio con polinomi di grado 1 si possono solo disegnare segmenti lineari. Polinomi con un grado maggiore di 3 portano a grossa complessità nei calcoli dei punti e a grossa difficoltà nel controllo poichè è facile generare cicli e cappi. Si noti poi come il grado 3, equazioni parametriche cubiche, è il minimo grado per ottenere nello spazio curve non complanari. Ad esempio equazioni di grado 2, come si è visto a lezione, impongono alla curva di passare per tre punti. Tre punti nello spazio definiscono un piano e la cirva si trova giacente in quel piano, è quindi complanare al piano definito dai tre punti.

I polinomi cubici che definiscono il segmento di curva $Q(u) = [x(u) y(u) z(u)]$ sono:

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x, \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y, \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z, \quad 0 \le u \le 1. \end{aligned} \tag{1}$$

Con $T = [u^3 \, u^2 \, u \, 1]$ si può definire la matrice dei coefficienti nel modo seguente:

$$C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} \tag{2}$$

e scruvere il segmento di curva come:

$$Q(u) = [x(u) \, y(u) \, z(u)] = U \cdot C. \tag{3}$$

La derivata di $Q(u)$ è il vettore tangente della curva dipendente dal parametro u. Dalla formula 3 si può ricavare la derivata:

$$\begin{aligned} &Q'(u) = [x'(u) \, y'(u) \, z'(u)] = \tfrac{d}{du} U \cdot C = \\ &= [3a_x u^2 + 2b_x u + c_x \quad 3a_y u^2 + 2b_y u + c_y \quad 3a_z u^2 + 2b_z u + c_z]. \end{aligned} \tag{4}$$

Se uniamo due curve insieme esse, in generale, potranno avere una discontinuità nel punto di unione (continuità C^0, a meno che non si impongano certe condizioni sulla tangente nel punto. Se la tangente della prima curva nel punto di unione è uguale alla tangente della seconda curva, sempre nel punto di unione, allora la curva totale avrà continuità C^1. Se si impongono condizioni anche sulla derivata seconda, si potrà ottenere una curva con continuità C^2, cioè stessa concavità.

Un segmento di curva $Q(u)$ sarà quindi completamente definito imponendo dei vincoli sui punti iniziali e finali, sui vettori tangenti e sulla continuità tra due segmenti successivi. Ogni polinomio cubico ha quattro coefficienti, cosi che bisognerà specificare quattro vincoli. I tre maggiori tipi di curve conosciute sono:

- *Hermite*, definite dai due punti iniziale e finale e dalle due tangenti a tali punti.
- *Bézier*, definite dai due punti iniziale e finale e da altri due punti che controllano le due tangenti ai punti iniziale e finale.
- Alcuni tipi di *spline*, ognuna definita da quattro punti di controllo. Le spline hanno continuità C^1 e C^2 nei punti di unione dei diversi segmenti di curva, sono vicine ai loro punti di controllo ma generalmente non passano per essi. I tipi di spline più conosciute sono le *Uniform B-Spline* e le *Non Uniform B-Spline* (polinomiali o razionali).

Open Inventor usa al suo interno le B-Spline (in particolare le *NURBS*), per cui in questo paragrafo verrà effettuate una breve analisi delle B-Spline. Per quanto riguarda le curve di Hermite e Bézier, si rimanda alle lezioni di teoria del corso di Informatica Grafica.

Uniform Nonrational B-Spline Le B-Spline consistono in segmenti di curva in cui i coefficienti polinomiali dipendono da pochi punti di controllo (*local control* e non da tutti i punti di controllo della B-Spline completa. Questo fa si che se si sposta un punto di controllo, questo ha effetto solo su una piccola parte della curva totale.

Una B-Spline cubica approssima una serie di $m + 1$ punti di controllo P_0, P_1, ..., P_m, $m \geq 3$ con una curva consistente di $m - 2$ segmenti di curva polinomiale cubica $Q_3, Q_4, ..., Q_m$. Ogni segmento di curva è definito ognuno nel suo dominio $0 \leq u < 1$. Se si cambia variabile del parametro u in $u = u + k$, i domini per le varie curve diventano sequenziali. In questo modo il segmento di curva Q_i risulta definito nel dominio $u_i \leq u < u_{i+1}$, per $3 \leq i \leq m$. Nel caso perticolare di $m = 3$ si ha un singolo segmento di curva Q_3 definito in $u_3 \leq u < u_4$ da quattro punti di controllo P_0, P_1, P_2 e P_3.

Per ogni $i \geq 4$ esiste un punto di giunzione o *knot* tra Q_{i-1} e Q_i al valore del parametro u_i. Il valore di questo parametro viene chiamato *knot value*. I punti iniziali e finali a u_3 e u_{m+1} vengono chiamati anch'essi knot, così si hanno un totale di $m - 1$ knot.

Il termine *uniform* sta ad indicare che i vari knot sono spaziati ad uguali intervalli del parametro u. Si assume che $u_3 = 0$ e l'intervallo $u_{i+1} - u_i = 1$. Il termine *nonrational* è usato perchè le funzioni coordinate sono dei semplici polinomi.

Nonuniform Nonrational B-Spline La non uniformità della sequenza dei valori di knot porta ad avere funzioni di base diverse per ogni intervallo. Questo porta a vantaggi come ridurre la continuità nei punti di giunzione da C^2 a C^1, C^0 o nessuna. Se la continuità viene ridotta a C^0 si ha che la curva interpola il punto di controllo. Nello stesso modo si possono interpolare i punti estremi della curva.

La generalità di questo tipo di curve porta ad una reinterpretazione delle notazioni usate in precedenza. Come prima la spline è composta da pezzi di curve cubiche che approssimano i punti di controllo P_0, P_1, ..., P_m. La sequenza di valori knot è ora una sequenza di valori non decrescenti u_0, u_1, ..., u_{m+4} (esistono ora 4 valori in più) non necessariamente equispaziati. Siccome il minor numero di punti di controllo è 4, il minor numero di valori di knot risulta essere 8, e la curva è definita sull'intervallo parametrico $[u_3, u_4]$. Successivi valori di knot possono quindi essere identici. Quando questo accade il valore del parametro viene chiamato knot multiplo o knot con molteplicità. La curva non è definita al di fuori dell'intervallo u_3, u_{m+1}. L'aumento della molteplicità di un valore knot porta a due effetti. Per prima cosa ogni valore knot u_i automaticamente giace nel guscio convesso formato dai punti P_{i-3}, P_{i-2} e P_{i-1}. Se u_i e u_{i+1} sono identici essi si trovano nel guscio convesso formato dai punti P_{i-3}, P_{i-2}, P_{i-1} e dai punti P_{i-2}, P_{i-1} e P_i. Questo implica che il valore knot giace sul segmento rettilineo che congiunge i punti P_{i-2} e P_{i-1}. Nello stesso modo se $u_i = u_{i+1} = u_{i+2}$ il valore knot si trova su P_{i-1}. Se invece $u_i = u_{i+1} = u_{i+2} = u_{i+3}$ allora il valore knot si trova su P_{i-1} e su P_i e la curva collassa ad un punto. Inoltre la molteplicità riduce la continuità nei punti di giunzione.

Nonuniform Rational B-Spline (NURBS) In questo tipo di curve cubiche i coefficienti delle equazioni sono dei rapporti di polinomi:

$$x(u) = \frac{X(u)}{W(u)}, \quad y(u) = \frac{Y(u)}{W(u)}, \quad z(u) = \frac{Z(u)}{W(u)} \tag{5}$$

in cui $X(u)$, $Y(u)$, $Z(u)$ e $W(u)$ sono curve polinomiali cubiche i cui punti di controllo sono definiti in coordinate omogenee. Possiamo pensare la curva come definita nello spazio delle coordinate omogenee: $Q(u) = [X(u)\,Y(u)\,Z(u)\,W(u)]$. Si noti come qualsiasi curva non razionale può essere posta nella forma delle curve razionali impostando $W(u) = 1$ per qualsiasi valore di u. Questo tipo di spline vengono generalmente chiamate **NURBS**.

Le NURBS sono usate in grafica principalmente per due motivi.

1. Sono invarianti per le rotazioni, le traslazioni, gli scalamenti e per la trasformazione prospettica dei punti di controllo (le curve polinomiali sono invarianti solo per rotazioni, traslazioni e scalamenti). Quindi la trasformazione prospettica può essere applicata solamente ai punti di controllo.

2. Questo tipo di curve permettono di definire in maniera precisa qualsiasi sezione di conica.

Per una teoria più completa ed approfondita si rimanda alle lezioni del corso di Informatica Grafica.

3 Curve NURBS e Inventor

Dopo aver visto in generale cosa sono le curve parametriche e in special modo le spline, vediamo ora come Inventor gestisce i dati per generare tali curve. Innanzitutto conviene rivedere, a livello di nomenclatura, i concetti chiave che Inventor usa.

- *Control Points* — Sono i punti di controllo per ogni segmento di curva. La curva può passare vicino questi punti o interpolarne alcuni. Inventor usa i nodi **SoCoordinate3** e **SoCoordinate4** per memorizzare queste informazioni.

- *Knot Sequence* — Definiscono i valori del parametro u nei punti di giunzione dei vari segmenti di curva. La sequenza di knot è semplicemente una lista di numeri non decrescenti. Inventor usa i nodi **SoNurbsCurve** e **SoIndexedNurbsCurve** per inserire questi valori nel grafo della scena.

- *Ordine di una curva* — Determina la forma delle equazioni parametriche. L'ordine è pari al massimo grado dell'equazione più 1. Per esempio una cubica ha $grado = 3$ e $ordine = 4$. Si può pensare all'ordine come al numero di coefficienti delle equazioni. In Inventor l'ordine della curva non è esplicitamente impostabile, ma è dato da:

$$number\_of\_knots - number\_of\_control\_points.$$

L'ordine di una curva determina il numero minimo di punti di controllo necessari per definire la curva stessa. Per definire una curva bisogna ovviamente avere almeno un numero di punti di controllo pari all'ordine della curva. L'ordine di una curva influenza anche come si modifica la curva quando un punto di controllo viene spostato. In Inventor il massimo ordine possibile per una NURBS è 8, ma di solito si usano NURBS di ordine 4 (quindi cubiche).

Per costruire curve con più punti di controllo, rispetto all'ordine della curva, si connettono insieme più segmenti di curva per formare la spline. Il punto di giunzione tra questi segmenti di curva è chiamato *join point*. In questo punto si possono avere diversi tipi di continuità della curva:

- *Non Continuità* — La curva non è congiunta con nessun'altra.

- *Continuità* C^0 — Gli estremi delle due curve si toccano senza nessuna condizione aggiuntiva.

- *Continuità* C^1 — Gli estremi delle due curve si toccano ed hanno lo stesso vettore tangente.

- *Continuità* C^2 — Gli estremi delle due curve si toccano ed hanno sia lo stesso vettore tangente che la stessa concavità.

L'ordine della curva determina anche la massima continuità possibile, la massima continuità è infatti pari a $ordine - 2$.

I punti di controllo determinano il punto in cui applicare la particolare funzione di base. Per le NURBS di Inventor queste funzioni di base sono le B-Spline basis function. Le funzioni di base B-Spline descrivono le curve nello spazio del parametro u. La curva risultante è pari alla somma dei contributi di ogni punto di controllo. La distribuzione delle funzioni di base nello spazio dei parametri è controllato dalla sequenza di knot. La sequenza di knot è una lista di valori non decrescenti. Ogni knot definisce l'inizio e la fine di una funzione di base. Se questi valori sono singolari, non ci sono valori ripetuti, la curva è una curva B-Spline uniforme. Valori duplicati nella sequenza di knot aumentano il valore della molteplicità e causano la partenza di più di una funzione di base da un unico punto di controllo. Contemporaneamente questo causa una diminuzione della continuità nei punti di giunzione. La relazione tra molteplicità e continuità può essere espressa dalla formula:

$$C^{ORD-(M+1)}$$

in cui ORD è pari all'ordine della curva e M è la molteplicità. La massima molteplicità (il massimo numero di ripetizioni di un knot) è pari all'ordine della curva.

Sequenze di knot particolari Alcune sequenze particolari sono usate nelle applicazioni:

- *Uniform Cubic B-Spline* — I knot sono uniformemente spaziati e a molteplicità unica, ad esempio: $\{0, 1, 2, 3, 4, 5, 6, 7\}$.
- *Cubic Bézier Curve* — Molteplicità pari a 4 sul punto iniziale e su quello finale, totale punti di controllo 4: $\{0, 0, 0, 0, 1, 1, 1, 1\}$.
- *Uniform Cubic B-Spline che passano per i punti estremi* — Molteplicità pari a 4 sul punto iniziale e su quello finale, knot uniformemente spaziati e a molteplicità unica nei punti intermedi. Esempio: $\{0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5\}$.

Sommario delle relazioni nelle NURBS Nelle pagine precedenti si sono stabilite importanti relazioni tra i parametri delle NURBS:

- $order = degree + 1$.
- Per definire una curva sono necessari almeno *order* punti di controllo.
- Massima continuità è pari a $order - 2$.
- Numero di knot è pari a $order + number_control_point$.
- I valori di knot devono essere non decrescenti.
- Massima molteplicità dei knot è pari a *order*.
- Continuità = order-(molteplicità+1).

Per le cubiche si ha quindi $order = 4$, massima continuità pari a C^2, minimo numero di knot pari a 8, massima molteplicità di un knot pari a 4.

Curve Razionali Ogni punto di controllo può avere associato un peso (usando nodi del tipo **SoCoordinate4**) che influenza la forma delle funzioni di base. Per le curve non razionali, questo peso ha valore 1.0, mentre per le curve razionali il peso lo sceglie il programmatore.

4 Nodi di Inventor per le NURBS

Le classi dei nodi che servono a definire una curva NURBS sono:

- **SoCoordinate3** — Questo nodo serve a specificare i punti di controllo della curva è deve essere posizionato prima del nodo **SoNurbsCurve**. Il nodo si usa per le curve non razionali, in cui si assume il peso pari a 1.0.
- **SoCoordinate4** — Questo nodo ha una funzione analoga alla precedente, ma la quarta coordinata serve a specificare un peso, per cui è usato con le curve razionali.
- **SoNurbsCurve** — Questa classe rappresenta una curva NURBS, basata sul vettore di knot e sui punti di controllo specificati dai nodi precedenti. I campi del nodo sono due. Il campo **knotVector** contiene un array di floating-point contenenti i valori di knot, che devono essere valori non decrescenti. La curva userà poi i primi **numControlPoints** (secondo campo del nodo). Se vengono specificati n valori knot allora si possono specificare fino a $n-8$ punti di controllo. Il numero di knot meno il numero di punti di controllo da l'ordine della curva.

5 Examples

Uniform B-Spline Curve Il seguente esempio genera una B-Spline con 7 punti di controllo. Il vettore dei valori di knot contiene quindi 10 valori poichè

$$number\_of\_knots = order + number\_of\_control\_points.$$

La curva ha ordine 3, molteplicità 2 (uno knot è doppio) e continuità C^0.

```
// The control points for this curve
const float pts[7][3] = {
   { 4.0, -6.0,  6.0},
   {-4.0,  1.0,  0.0},
   {-1.5,  5.0, -6.0},
   { 0.0,  2.0, -2.0},
   { 1.5,  5.0, -6.0},
   { 4.0,  1.0,  0.0},
   {-4.0, -6.0,  6.0}};

// The knot vector
float knots[10] = {1, 2, 3, 4, 5, 5, 6, 7, 8, 9};

// Create the nodes needed for the B-Spline curve.
SoSeparator* makeCurve()
{
   SoSeparator* curveSep = new SoSeparator();
   curveSep->ref();

   // Set the draw style of the curve.
   SoDrawStyle* drawStyle  = new SoDrawStyle;
   drawStyle->lineWidth = 4;
   curveSep->addChild(drawStyle);

   // Define the NURBS curve including the control points and a complexity.
   SoComplexity* complexity = new SoComplexity;
   SoCoordinate3* controlPts = new SoCoordinate3;
```

```
   SoNurbsCurve* curve = new SoNurbsCurve;
   complexity->value = 0.8;
   controlPts->point.setValues( 0, 7, pts );
   curve->numControlPoints = 7;
   curve->knotVector.setValues( 0, 10, knots );
   curveSep->addChild(complexity);
   curveSep->addChild(controlPts);
   curveSep->addChild(curve);

   curveSep->unrefNoDelete();
   return curveSep;
}
```

Uniform B-Spline Curve passante per gli estremi Il seguente esempio genera una B-Spline che passa per i punti di controllo iniziale e finale. I punti di controllo sono 13. Il vettore dei valori di knot contiene quindi 17 valori di cui i primi 4 e gli ultimi 4 hanno molteplicità 4 per far passare la curva sui punti di controllo. La curva ha ordine 4.

```
// The control points for this curve
const float pts[13][3] = {
   { 6.0,  0.0,  6.0},
   {-5.5,  0.5,  5.5},
   {-5.0,  1.0, -5.0},
   { 4.5,  1.5, -4.5},
   { 4.0,  2.0,  4.0},
   {-3.5,  2.5,  3.5},
   {-3.0,  3.0, -3.0},
   { 2.5,  3.5, -2.5},
   { 2.0,  4.0,  2.0},
   {-1.5,  4.5,  1.5},
   {-1.0,  5.0, -1.0},
   { 0.5,  5.5, -0.5},
   { 0.0,  6.0,  0.0}};

// The knot vector
float knots[17] = {
   0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10};

// Create the nodes needed for the B-Spline curve.
SoSeparator* makeCurve()
{
   SoSeparator* curveSep = new SoSeparator();
   curveSep->ref();

   // Set the draw style of the curve.
   SoDrawStyle* drawStyle = new SoDrawStyle;
   drawStyle->lineWidth = 4;
   curveSep->addChild(drawStyle);

   // Define the NURBS curve including the control points
   // and a complexity.
   SoComplexity* complexity = new SoComplexity;
   SoCoordinate3* controlPts = new SoCoordinate3;
   SoNurbsCurve* curve = new SoNurbsCurve;
   complexity->value = 0.8;
   controlPts->point.setValues( 0, 13, pts );
   curve->numControlPoints = 13;
   curve->knotVector.setValues( 0, 17, knots );
   curveSep->addChild(complexity);
   curveSep->addChild(controlPts);
```

```
   curveSep->addChild(curve);

   curveSep->unrefNoDelete();
   return curveSep;
}
```

6 LAB: Umbrella

In questa sezione si vuole discutere un piccolo esempio: la generazione dello scheletro di un ombrello. L'esempio é stato giá visto a lezione in PLaSM. L'esempio si basa sulle curve quadratiche di Bézier, quindi per prima cosa conviene costruire una funzione che prende come argomenti i punti di controllo della quadratica e crea un oggetto Inventor che rappresenterá la curva.

```
static SoSeparator* bezier( SbVec3f p0, SbVec3f p1, SbVec3f p2 )
// Bezier Quadratic Curve
{
    // Storage for control points
    SbVec3f* pts = new SbVec3f[3];
    pts[0] = p0; pts[1] = p1; pts[2] = p2;

    // knots value for Bezier Quadratic Curve
    float* knots = new float[6];
    knots[0] = 0.0; knots[1] = 0.0; knots[2] = 0.0;
    knots[3] = 1.0; knots[4] = 1.0; knots[5] = 1.0;

    SoSeparator* curveSep = new SoSeparator();
    curveSep->ref();

    // Define the Bezier curve including the control points and a complexity.
    SoComplexity* complexity = new SoComplexity;
    complexity->value = 0.8;

    SoCoordinate3* controlPts = new SoCoordinate3;    // Control points
    controlPts->point.setValues( 0, 3, pts );

    SoNurbsCurve* curve = new SoNurbsCurve;
    curve->numControlPoints = 3;                      // Control points number
    curve->knotVector.setValues( 0, 6, knots );       // Knot values

    curveSep->addChild(complexity);
    curveSep->addChild(controlPts);
    curveSep->addChild(curve);

    curveSep->unrefNoDelete();
    return curveSep;
}
```

Di seguito si riportano le funzioni per costruire l'ombrello. La funzione **umbrella** richiamerá le altre e vuole come parametri la lunghezza del manico e l'apertura dell'ombrello. La funzione **umbrella_rodpair** crea i raggi dell'ombrello e la funzione **umbrella_handle** crea il manico dell'ombrello.

```
///////////
//
// UMBRELLA
//

// Local Function Prototyping
```

```
static SoSeparator* bezier( SbVec3f p0, SbVec3f p1, SbVec3f p2 );
static SoSeparator* umbrella_rodpair( float h, float alpha );
static SoSeparator* umbrella_handle( float h );


SoSeparator* umbrella( float h, float alpha )
// Make an umbrella
{
    SoSeparator* umbrella = new SoSeparator;
    umbrella->ref();

    umbrella->addChild(umbrella_handle(h));            // Add handle

    SoSeparator* rodpair = umbrella_rodpair(h,alpha);
    SoRotationXYZ* rot = new SoRotationXYZ;
    rot->axis = SoRotationXYZ::Z;
    rot->angle = M_PI/6;
    for( unsigned i=0; i<12; i++ )                     // Generate rots
    {
     umbrella->addChild(rot);
     umbrella->addChild(rodpair);
    }

    umbrella->unrefNoDelete();
    return umbrella;
}


static SoSeparator* umbrella_handle( float h )
// Make an umbrella handle
{
    SoSeparator* pole = bezier(SbVec3f(0,0,0),SbVec3f(0,0,h/2),SbVec3f(0,0,h));
    SoSeparator* hand = bezier(SbVec3f(0,0,0),SbVec3f(0,h/16,h/5),SbVec3f(0,h/8,0));

    SoSeparator* handle = new SoSeparator;
    handle->ref();

    // Pole
    SoTranslation* xt1 = new SoTranslation;
    xt1->translation = SbVec3f( 0, 0, h-h/10 );
    handle->addChild(xt1);
    handle->addChild(pole);

    // Handle
    SoTranslation* xt2 = new SoTranslation;
    xt2->translation = SbVec3f( 0, 0, h );
    handle->addChild(xt2);
    handle->addChild(hand);

    handle->unrefNoDelete();
    return handle;
}

inline SoSeparator* umbrella_rodcurve( float len, float beta )
// Rod Curve
{
    return bezier(SbVec3f(0,0,0),SbVec3f(0,0,len/2),SbVec3f(-sin(beta)*len/2,0,len));
}

static SoSeparator* umbrella_rodpair( float h, float alpha )
// Make an umbrella rod pair
{
```

```
    float  ab = h*4/10;
    float  alphaRad = alpha*M_PI/180;

    SoSeparator* rod10 = bezier(SbVec3f(0,0,0),SbVec3f(0,0,ab/2),SbVec3f(0,0,ab));
    SoSeparator* rod11 = new SoSeparator;
    SoSeparator* rod2 = rod10;

    SoTranslation* xt1 = new SoTranslation;
    xt1->translation = SbVec3f( 0, 0, h );

    SoTranslation* xt2 = new SoTranslation;
    xt2->translation = SbVec3f( 0, 0, ab );

    SoRotation* xr1 = new SoRotation;
    xr1->rotation.setValue( SbVec3f(0.0,1.0,0.0), -alphaRad );

    SoRotation* xr2 = new SoRotation;
    xr2->rotation.setValue( SbVec3f(0.0,1.0,0.0), 2*alphaRad );

    SoSeparator* rodpair = new SoSeparator;
    rodpair->ref();

    // Linear piece of rod
    rodpair->addChild(xt1);
    rodpair->addChild(xr1);
    rodpair->addChild(rod10);

    // Curved piece of rod
    rod11->addChild(xt2);
    rod11->addChild(umbrella_rodcurve( ab, -alphaRad/4 ));
    rodpair->addChild(rod11);

    // Linear back rod
    rodpair->addChild(xt2);
    rodpair->addChild(xr2);
    rodpair->addChild(rod2);

    rodpair->unrefNoDelete();
    return rodpair;
}
```

Per completezza si riporta la funzione main e i file include.

```
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoDrawStyle.h>
#include <Inventor/nodes/SoComplexity.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoNurbsCurve.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoRotation.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoTranslation.h>

#include <math.h>

// Function Prototyping
SoSeparator* umbrella( float h, float alpha );

int main( int, char** argv )
{
    // Initialize Inventor and Xt
```

```
    Widget appWindow = SoXt::init(argv[0]);
    if(appWindow == 0) return 1;

    SoSeparator* root  = new SoSeparator;
    root->ref();

    // Set Matirial
    SoMaterial* material = new SoMaterial;
    material->diffuseColor.setValue( 1.0, 0.0, 0.0 );
    root->addChild(material);

    // Set the draw style of the curve.
    SoDrawStyle* drawStyle  = new SoDrawStyle;
    drawStyle->lineWidth = 2;
    root->addChild(drawStyle);

    // Draw scene
    root->addChild(umbrella(10,80));

    SoTranslation* xt = new SoTranslation;
    xt->translation = SbVec3f( 15, 0, 0 );

    root->addChild(xt);
    root->addChild(umbrella(10,30));

    // Initialize an Examiner Viewer
    SoXtExaminerViewer* viewer = new SoXtExaminerViewer(appWindow);
    viewer->setSceneGraph(root);
    viewer->setTitle("Umbrella");
    viewer->show();

    SoXt::show(appWindow);
    SoXt::mainLoop();

    return 0;
}
```